

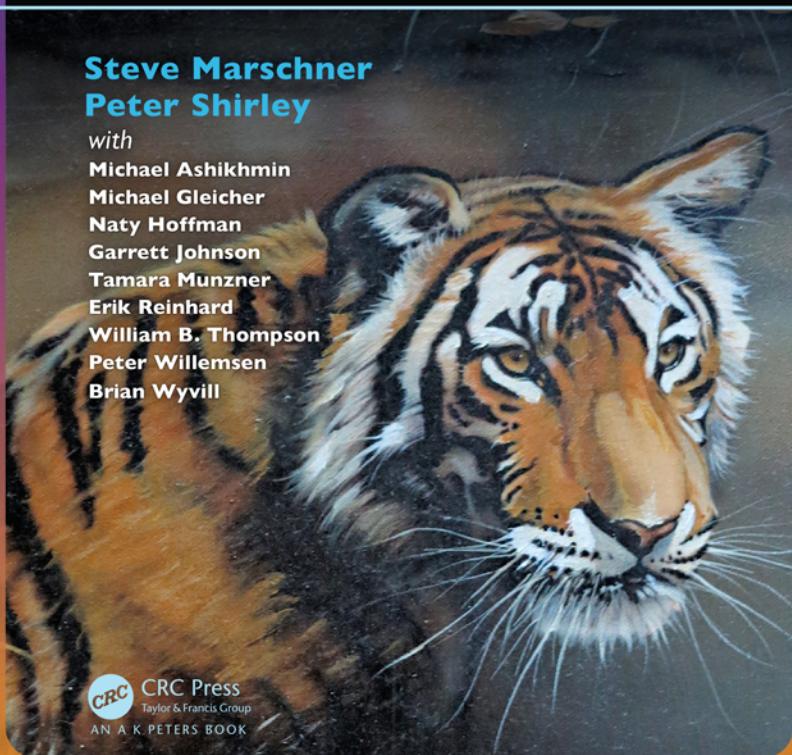
Fundamentals of Computer Graphics

FOURTH EDITION

Steve Marschner
Peter Shirley

with

Michael Ashikhmin
Michael Gleicher
Naty Hoffman
Garrett Johnson
Tamara Munzner
Erik Reinhard
William B. Thompson
Peter Willemsen
Brian Wyvill



 CRC Press
Taylor & Francis Group
AN A K PETERS BOOK

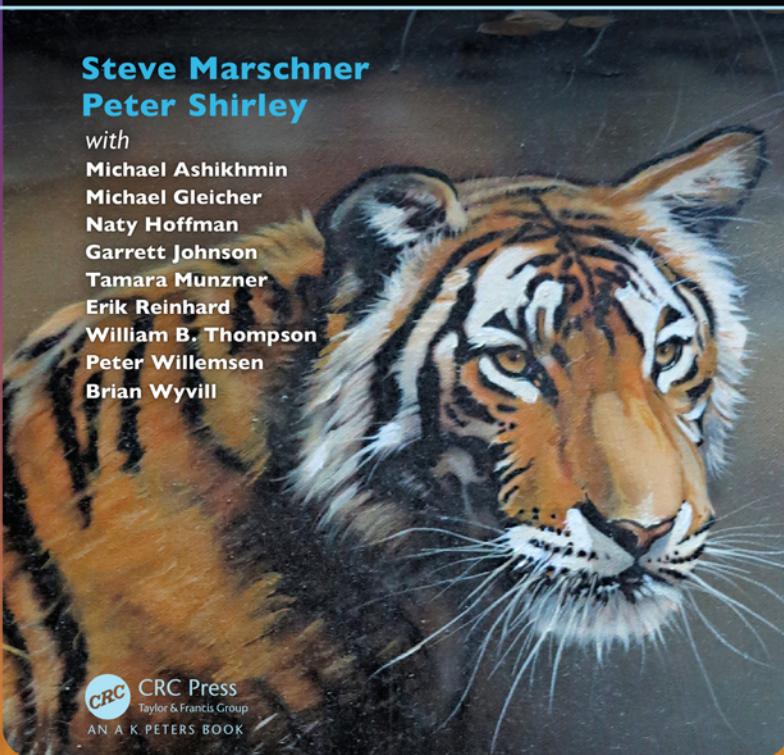
Fundamentals of Computer Graphics

FOURTH EDITION

Steve Marschner
Peter Shirley

with

Michael Ashikhmin
Michael Gleicher
Naty Hoffman
Garrett Johnson
Tamara Munzner
Erik Reinhard
William B. Thompson
Peter Willemsen
Brian Wyvill



 CRC Press
Taylor & Francis Group
AN A K PETERS BOOK

Fundamentals of Computer Graphics

FOURTH EDITION

计算机图形学基础

FOURTH EDITION

Fundamentals of Computer Graphics

FOURTH EDITION

Steve Marschner

Cornell University

Peter Shirley

Purity, LLC

with
Michael Ashikhmin
Michael Gleicher
Naty Hoffman
Garrett Johnson
Tamara Munzner
Erik Reinhard
William B. Thompson
Peter Willemsen
Brian Wyvill

计算机图形学基础

FOURTH EDITION

Steve Marschner

康奈尔大学

彼得·雪莉

Purity, LLC

with
Michael Ashikhmin
迈克尔·阿希克敏
纳蒂·霍夫曼
加勒特·约翰逊
玛拉·蒙兹纳
Erik Reinhard
威廉·B·汤普森
Peter Willemsen
Brian Wyvill



CRC Press

Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN A K PETERS BOOK



CRC Press

Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN A K PETERS BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2016 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works
Version Date: 20151012

International Standard Book Number-13: 978-1-4822-2941-7 (eBook - PDF)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Visit the Taylor & Francis Web site at
<http://www.taylorandfrancis.com>

and the CRC Press Web site at
<http://www.crcpress.com>

泰勒和弗朗西斯集团6000破碎的声音公园
道NW, 套房300博卡拉顿, FL33487-274
2

©2016由泰勒和弗朗西斯集团, LLCCRC出版社是泰勒和弗朗西斯集团的
印记, 一个Informa业务

没有要求美国政府的原始作品

国际标准书号-13:978-1-4822-2941-7(电子书PDF)

这本书包含了从真实和高度重视的来源获得的信息。已做出合理努力来发布可靠的数据和信息，但作者和出版商不能对所有材料的有效性或其使用的后果承担责任。作者和出版商试图追踪本出版物中复制的所有材料的版权所有者，如果未获得以此形式发布的许可，请向版权所有者道歉。如果任何版权材料没有被承认，请写信告诉我们，以便我们可以纠正任何未来的转载。

除美国版权法许可外，未经出版书面许可，不得以任何电子、机械或其他方式（包括影印、缩微胶卷和录音）重印、复制、传送或以任何形式使用本书的任何部分。

如欲以电子方式影印或使用本作品的资料，请浏览www.copyright.com (<http://www.copyright.com/>) 或联系版权清算中心, Inc. (CCC) 222红木驱动器 丹弗斯 MA 01923 978-750-8400.CCC是一个非盈利性组织，为各种用户提供许可证和注册。对于获得CCC颁发复印许可证的组织，已安排单独的支付系统。

商标声明：产品或企业名称可能是商标或注册商标，仅用于识别和解释，无意侵权。

访问泰勒和弗朗西斯网站<http://www.taylorandfrancis.com>

和CRC新闻网站<http://www.crcpress.com>

Contents

Preface	xi
1 Introduction	1
1.1 Graphics Areas	2
1.2 Major Applications	3
1.3 Graphics APIs	4
1.4 Graphics Pipeline	4
1.5 Numerical Issues	5
1.6 Efficiency	7
1.7 Designing and Coding Graphics Programs	8
2 Miscellaneous Math	13
2.1 Sets and Mappings	13
2.2 Solving Quadratic Equations	17
2.3 Trigonometry	18
2.4 Vectors	21
2.5 Curves and Surfaces	30
2.6 Linear Interpolation	44
2.7 Triangles	44
3 Raster Images	53
3.1 Raster Devices	54
3.2 Images, Pixels, and Geometry	59
3.3 RGB Color	64
3.4 Alpha Compositing	65
4 Ray Tracing	69
4.1 The Basic Ray-Tracing Algorithm	70
4.2 Perspective	71
4.3 Computing Viewing Rays	73
4.4 Ray-Object Intersection	76
4.5 Shading	81

Contents

Preface	xi
1 Introduction	1
1.1 图形区域。	2
1.2 主要应用。	3
1.3 Graphics APIs	4
1.4 图形管道。	4
1.5 数字问题。	5
1.6 Efficiency	7
1.7 设计和编码图形程序	8
2 杂项数学	13
2.1 集和映射。	13
2.2 求解二次方程。	17
2.3 Trigonometry	18
2.4 Vectors	21
2.5 曲线和曲面。	30
2.6 线性插值	44
2.7 Triangles	44
3 Raster Images	53
3.1 Raster Devices	54
3.2 图像、像素和几何。	59
3.3 RGB Color	64
3.4 阿尔法合成。	65
4 光线追踪	69
4.1 基本的光线追踪算法。	70
4.2 Perspective	71
4.3 计算观察射线	73
4.4 Ray-Object Intersection	76
4.5 Shading	81

4.6	A Ray-Tracing Program	84
4.7	Shadows	86
4.8	Ideal Specular Reflection	87
4.9	Historical Notes	87

5 Linear Algebra 89

5.1	Determinants	89
5.2	Matrices	91
5.3	Computing with Matrices and Determinants	96
5.4	Eigenvalues and Matrix Diagonalization	101

6 Transformation Matrices 109

6.1	2D Linear Transformations	109
6.2	3D Linear Transformations	123
6.3	Translation and Affine Transformations	128
6.4	Inverses of Transformation Matrices	132
6.5	Coordinate Transformations	133

7 Viewing 139

7.1	Viewing Transformations	140
7.2	Projective Transformations	146
7.3	Perspective Projection	149
7.4	Some Properties of the Perspective Transform	153
7.5	Field-of-View	154

8 The Graphics Pipeline 159

8.1	Rasterization	160
8.2	Operations Before and After Rasterization	171
8.3	Simple Antialiasing	177
8.4	Culling Primitives for Efficiency	179

9 Signal Processing 183

9.1	Digital Audio: Sampling in 1D	184
9.2	Convolution	187
9.3	Convolution Filters	201
9.4	Signal Processing for Images	208
9.5	Sampling Theory	217

4.6	A Ray-Tracing Program	84
4.7	Shadows	86
4.8	理想的镜面反射。	87
4.9	历史笔记	87

5 线性代数 89

5.1	Determinants	89
5.2	Matrices	91
5.3	用矩阵和行列式计算。	96
5.4	特征值和矩阵对角化。	101

6 变换矩阵 109

6.1	2D线性变换。	109
6.2	3D线性变换。	123
6.3	平移和仿射变换。	128
6.4	变换矩阵的反转。	132
6.5	协调变换。	133

7 Viewing 139

7.1	查看变换。	140
7.2	射影变换。	146
7.3	透视投影	149
7.4	透视变换的一些属性。	153
7.5	Field-of-View	154

8 图形管道 159

8.1	Rasterization	160
8.2	光栅化之前和之后的操作。	171
8.3	Simple Antialiasing	177
8.4	效率的剔除基元。	179

9 信号处理 183

9.1	数字音频：1D采样。	184
9.2	Convolution	187
9.3	卷积滤波器。	201
9.4	图像的信号处理。	208
9.5	抽样理论。	217



10 Surface Shading	233
10.1 Diffuse Shading	233
10.2 Phong Shading	236
10.3 Artistic Shading	239
11 Texture Mapping	243
11.1 Looking Up Texture Values	244
11.2 Texture Coordinate Functions	246
11.3 Antialiasing Texture Lookups	260
11.4 Applications of Texture Mapping	267
11.5 Procedural 3D Textures	273
12 Data Structures for Graphics	281
12.1 Triangle Meshes	282
12.2 Scene Graphs	295
12.3 Spatial Data Structures	297
12.4 BSP Trees for Visibility	309
12.5 Tiling Multidimensional Arrays	317
13 More Ray Tracing	323
13.1 Transparency and Refraction	324
13.2 Instancing	327
13.3 Constructive Solid Geometry	328
13.4 Distribution Ray Tracing	329
14 Sampling	337
14.1 Integration	337
14.2 Continuous Probability	342
14.3 Monte Carlo Integration	346
14.4 Choosing Random Points	349
15 Curves	359
15.1 Curves	359
15.2 Curve Properties	365
15.3 Polynomial Pieces	368
15.4 Putting Pieces Together	375
15.5 Cubics	378
15.6 Approximating Curves	385
15.7 Summary	402



10表面阴影	233
10.1 漫射阴影	233
10.2 Phong Shading	236
10.3 艺术阴影	239
11纹理映射	243
11.1 查找纹理值	244
11.2 纹理坐标函数	246
11.3 Antialiasing Texture Lookups	260
11.4 纹理映射的应用	267
11.5 程序化的3d纹理。	273
12图形的数据结构	281
12.1 三角形网格	282
12.2 场景图。	295
12.3 空间数据结构。	297
12.4 能见度的BSP树	309
12.5 平铺多维数组。	317
13更多光线追踪	323
13.1 透明度和折射。	324
13.2 Instancing	327
13.3 建设性的实体几何。	328
13.4 分布光线追踪。	329
14 Sampling	337
14.1 Integration	337
14.2 续概率。	342
14.3 蒙特卡罗积分。	346
14.4 选择随机点。	349
15 Curves	359
15.1 Curves	359
15.2 曲线属性。	365
15.3 多项式片	368
15.4 把碎片拼在一起。	375
15.5 Cubics	378
15.6 近似曲线。	385
15.7 Summary	402

16 Computer Animation	405
16.1 Principles of Animation	406
16.2 Keyframing	410
16.3 Deformations	418
16.4 Character Animation	419
16.5 Physics-Based Animation	426
16.6 Procedural Techniques	428
16.7 Groups of Objects	431
17 Using Graphics Hardware	437
17.1 Hardware Overview	437
17.2 What Is Graphics Hardware	437
17.3 Heterogeneous Multiprocessing	439
17.4 Graphics Hardware Programming: Buffers, State, and Shaders	441
17.5 State Machine	443
17.6 Basic OpenGL Application Layout	444
17.7 Geometry	445
17.8 A First Look at Shaders	447
17.9 Vertex Buffer Objects	450
17.10 Vertex Array Objects	452
17.11 Transformation Matrices	455
17.12 Shading with Per-Vertex Attributes	457
17.13 Shading in the Fragment Processor	461
17.14 Meshes and Instancing	467
17.15 Texture Objects	469
17.16 Object-Oriented Design for Graphics Hardware Programming	475
17.17 Continued Learning	476
18 Light	479
18.1 Radiometry	479
18.2 Transport Equation	488
18.3 Photometry	489
19 Color	493
19.1 Colorimetry	495
19.2 Color Spaces	504
19.3 Chromatic Adaptation	510
19.4 Color Appearance	514

16 电脑动画	405
16.1 动画原理	406
16.2 Keyframing	410
16.3 Deformations	418
16.4 角色动画。	419
16.5 Physics-Based Animation	426
16.6 程序技术。	428
16.7 对象组	431
17 使用图形硬件	437
17.1 硬件概述	437
17.2 什么是图形硬件	437
17.3 Heterogeneous Multiprocessing	439
17.4 图形硬件编程：缓冲区、状态和着色器。	441
17.5 状态机。	443
17.6 基本OpenGL应用程序布局	444
17.7 Geometry	445
17.8 初看着色器	447
17.9 顶点缓冲区对象。	450
17.10 顶点数组对象	452
17.11 变换矩阵。	455
17.12 片段处理器中的着色。	457
17.13 网格和实例化	461
17.14 纹理对象	469
17.15 图形硬件编程的面向对象设计。	475
17.16 持续学习	476
18 Light	479
18.1 Radiometry	479
18.2 运输方程。	488
18.3 Photometry	489
19 Color	493
19.1 Colorimetry	495
19.2 色彩空间。	504
19.3 半音适应	510
19.4 颜色外观	514

**20 Visual Perception****515**

20.1 Vision Science	516
20.2 Visual Sensitivity	517
20.3 Spatial Vision	534
20.4 Objects, Locations, and Events	547
20.5 Picture Perception	556

21 Tone Reproduction**559**

21.1 Classification	562
21.2 Dynamic Range	563
21.3 Color	565
21.4 Image Formation	567
21.5 Frequency-Based Operators	567
21.6 Gradient-Domain Operators	569
21.7 Spatial Operators	570
21.8 Division	572
21.9 Sigmoids	573
21.10 Other Approaches	578
21.11 Night Tonemapping	581
21.12 Discussion	582

22 Implicit Modeling**585**

22.1 Implicit Functions, Skeletal Primitives, and Summation Blending	586
22.2 Rendering	594
22.3 Space Partitioning	595
22.4 More on Blending	601
22.5 Constructive Solid Geometry	602
22.6 Warping	604
22.7 Precise Contact Modeling	606
22.8 The BlobTree	608
22.9 Interactive Implicit Modeling Systems	610

23 Global Illumination**613**

23.1 Particle Tracing for Lambertian Scenes	614
23.2 Path Tracing	617
23.3 Accurate Direct Lighting	619

**20视觉感知****515**

20.1 视觉科学。	516
20.2 视觉灵敏度。	517
20.3 空间视觉。	534
20.4 对象、位置和事件。	547
20.5 图片感知	556

21音再现**559**

21.1 Classification	562
21.2 动范围。	563
21.3 Color	565
21.4 图像形成。	567
21.5 Frequency-Based Operators	567
21.6 Gradient-Domain Operators	569
21.7 空间运算符。	570
21.8 Division	572
21.9 乙状结肠。	573
21.10 其他方法	573
21.11 Night Tonemapping	581
21.12 Discussion	582

22隐式建模**585**

22.1 隐式函数、骨架基元和求和 Blending	586
22.2 Rendering	594
22.3 空间分区	595
22.4 更多关于混合	601
22.5 构造实体几何	602
22.6 Warping	604
22.7 精确的接触建模	606
22.8 The BlobTree	608
22.9 交互式隐式建模系统	610

23全局照明**613**

23.1 朗伯场景的粒子追踪	614
23.2 路径跟踪	617
23.3 精确的直接照明。	619



x

Contents

24 Reflection Models **627**

24.1 Real-World Materials	627
24.2 Implementing Reflection Models	629
24.3 Specular Reflection Models	631
24.4 Smooth-Layered Model	632
24.5 Rough-Layered Model	635

25 Computer Graphics in Games **643**

25.1 Platforms	643
25.2 Limited Resources	646
25.3 Optimization Techniques	649
25.4 Game Types	650
25.5 The Game Production Process	653

26 Visualization **665**

26.1 Background	667
26.2 Data Types	668
26.3 Human-Centered Design Process	670
26.4 Visual Encoding Principles	672
26.5 Interaction Principles	680
26.6 Composite and Adjacent Views	681
26.7 Data Reduction	687
26.8 Examples	692

References **701**

x

Contents

24 反射模型 **627**

24.1 Real-World Materials	627
24.2 实现反射模型。	629
24.3 镜面反射模型	631
24.4 Smooth-Layered Model	632
24.5 Rough-Layered Model	635

25 游戏中的计算机图形学 **643**

25.1 Platforms	643
25.2 资源有限。	646
25.3 优化技术。	649
25.4 游戏类型	650
25.5 的游戏制作过程。	653

26 Visualization **665**

26.1 Background	667
26.2 数据类型。	668
26.3 以人为本的设计过程	670
26.4 视觉编码原则。	672
26.5 互动原则。	680
26.6 复合视图和相邻视图	681
26.7 数据减少。	687
26.8 Examples	692

References **701**

Preface

This edition of *Fundamentals of Computer Graphics* includes substantial rewrites of the chapters on textures and graphics hardware, as well as many corrections throughout. The figures are now in color throughout the book.

The organization of the book remains substantially similar to the third edition. In our thinking, Chapters 2 through 8 constitute the “core core,” taking the straight and narrow path through what is absolutely required for understanding how images get onto the screen using the complementary approaches of ray tracing and rasterization. Ray tracing is covered first, since it is the simplest way to generate images of 3D scenes, followed by the mathematical machinery required for the graphics pipeline, then the pipeline itself. After that, the “outer core” covers other topics that would commonly be included in an introductory class, such as sampling theory, texture mapping, spatial data structures, and splines. Starting with Chapter 15 is a number of contributed chapters, authored by contributors we have chosen both for their expertise and for their clear way of expressing ideas.

As we have revised this book over the years, we have endeavored to retain the informal, intuitive style of presentation that characterizes the earlier editions, while at the same time improving consistency, precision, and completeness. We hope the reader will find the result is an appealing platform for a variety of courses in computer graphics.

About the Cover

The cover image is from *Tiger in the Water* by J. W. Baker (brushed and air-brushed acrylic on canvas, 16" by 20", www.jwbart.com).

The subject of a tiger is a reference to a wonderful talk given by Alain Fournier (1943–2000) at a workshop at Cornell University in 1998. His talk was an evocative verbal description of the movements of a tiger. He summarized his point:

Even though modelling and rendering in computer graphics have been improved tremendously in the past 35 years, we are still not at the point where we can model automatically a tiger swimming in

Preface

这个版本的计算机图形学基础包括大量重写的章节纹理和图形硬件，以及许多修正贯穿始终。这些数字现在全书都是彩色的。

这本书的组织与第三个edition基本相似。在我们的思想中，第2章到第8章构成了“核心核心”，通过使用射线跟踪和光栅化的互补方法来理解图像是如何进入屏幕的，这是绝对必要的。首先介绍光线追踪，因为它是生成3D场景图像的最简单方法，其次是图形管道所需的数学机械，然后是管道本身。之后，“外核”涉及其他通常会包含在入门课中的主题，例如采样理论，纹理映射，空间数据结构和样条。从第15章开始是一些贡献的章节，由我们选择的贡献者撰写，因为他们的专业知识和表达想法的清晰方式。随着我们多年来对这本书的修订，我们努力保持早期版本的非正式、直观的展示风格，与此同时提高了一致性、准确性和完整性。我们希望读者会发现结果是一个吸引人的平台，为各种计算机图形课程。

关于封面

封面图片来自TigerinTheWaterbyJ.W.Baker（帆布上的拉丝和空气拉丝丙烯酸，16"by20"，www.jwbart.com）。

老虎的主题是参考阿兰·福尼尔（AlainFournier，1943–2000）在1998年康奈尔大学的一个研讨会上发表的精彩演讲。他的演讲是对老虎动作的口头描述。他总结了他的观点：

尽管在过去的35年里，计算机图形学的建模和渲染已经有了很大的改进，但我们还没有达到可以自动为游来游去的老虎建模的地步

the river in all its glorious details. By automatically I mean in a way that does not need careful manual tweaking by an artist/expert.

The bad news is that we have still a long way to go.

The good news is that we have still a long way to go.

Online Resources

The website for this book is <http://www.cs.cornell.edu/~srm/fcg4/>. We will continue to maintain a list of errata and links to courses that use the book, as well as teaching materials that match the book's style. Most of the figures in this book are in Adobe Illustrator format, and we would be happy to convert specific figures into portable formats on request. Please feel free to contact us at shirley@cs.utah.edu or srm@cs.cornell.edu.

Acknowledgments

The following people have provided helpful information, comments, or feedback about the various editions of this book: Ahmet Oğuz Akyüz, Josh Andersen, Zeferino Andrade, Kavita Bala, Adam Berger, Adeel Bhutta, Solomon Boulos, Stephen Chenney, Michael Coblenz, Greg Coombe, Frederic Cremer, Brian Curtin, Dave Edwards, Jonathon Evans, Karen Feinauer, Amy Gooch, Eunghyoung Han, Chuck Hansen, Andy Hanson, Razen Al Harbi, Dave Hart, John Hart, John "Spike" Hughes, Helen Hu, Vicki Interrante, Wenzel Jakob, Doug James, Henrik Wann Jensen, Shi Jin, Mark Johnson, Ray Jones, Revant Kapoor, Kristin Kerr, Erum Arif Khan, Mark Kilgard, Dylan Lacewell, Mathias Lang, Philippe Laval, Marc Levoy, Howard Lo, Joann Luu, Ron Metoyer, Keith Morley, Eric Mortensen, 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, Hannah Story, Tony Tahbaz, Jan-Phillip Tiesel, Bruce Walter, Alex Williams, Amy Williams, Chris Wyman, and Kate Zebrose.

Ching-Kuang Shene and David Solomon allowed us to borrow their examples. Henrik Wann Jensen, Eric Levin, Matt Pharr, and Jason Waltman generously provided images. Brandon Mansfield helped improve the discussion of hierarchical bounding volumes for ray tracing. Philip Greenspun (philip.greenspun.com)

这条河充满了辉煌的细节.通过自动我的意思是在一种方式，不需要仔细手动调整由艺术家专家。

坏消息是，我们还有很长的路要走。

好消息是，我们还有很长的路要走。

网上资源

这本书的网站是<http://www.cs.cornell.edu/~srm/fcg4/>.我们将继续维护使用该书的勘误表和课程链接的列表，以及与该书风格相匹配的教材。本书中的大多数数字都是Adobe Illustrator格式，我们很乐意根据要求将特定数字转换为便携式格式。请随时与我们联络shirley@cs.utah.edu或srm@cs.cornell.edu...

Acknowledgments

以下人士提供了有关本书各版本的有用信息、评论或反馈：AhmetOguzAkyuz、JoshAndersen、ZeferinoAndrade、KavitaBala、AdamBerger、AdeelBhutta、SolomonBoulos、StephenChenney、MichaelCoblenz、GregCoombe、FredericCremer、BrianCurtin、DaveEdwards、JonathonEvans、KarenFeinauer、AmyGooch、EunghyoungHan、ChuckHans、AndyHanson、DaveRazenAlHarbi、HartJohnhartjohnn"spike" Hughes、HelenHu、VickiInterrante、WenzelJakob、DougJames、HenrikWannJensen、ShiJin、MarkJohnson、RayJones、RevantKapoor、KristinKerr、ErumArifKhan、MarkKilgard、DylanLacewell、MathiasLang、PhilippeLaval、MarcLevoy、HowardLo、JoannLuu、RonMetoyer、KeithMorley、EricMortensen、KojiNakamaru、MicahNeilson、BlakeNelson、MichaelNikelsky、JamesO'Brien、SteveParker、SumantaPattanaik、MattPharr、PeterPoulos、ShaunRamsey、RichRiesenfeld、NateRobins、NanSchaller、ChrisSchryvers、TomSederberg、RichardSharp、SarahShirley、Peter-PikeSloan、HannahStory、TonyTahbaz、Jan-PhillipTiesel、BruceWalter、AlexWilliams、AmyWilliams

克里斯·怀曼和凯特·泽布罗斯。

清光舍内和大卫所罗门允许我们借用他们的考试成绩。HenrikWannJensen、EricLevin、MattPharr和JasonWaltman慷慨地提供了图像。BrandonMansfield帮助改进了对用于光线追踪的hierarchical边界体积的讨论。菲利普·格林斯彭 (philip.greenspun.com)

kindly allowed us to use his photographs. John "Spike" Hughes helped improve the discussion of sampling theory. Wenzel Jakob's *Mitsuba* renderer was invaluable in creating many figures. We are extremely thankful to J. W. Baker for helping create the cover Pete envisioned. In addition to being a talented artist, he was a great pleasure to work with personally.

Many works that were helpful in preparing this book are cited in the chapter notes. However, a few key texts that influenced the content and presentation deserve special recognition here. These include the two classic computer graphics texts from which we both learned the basics: *Computer Graphics: Principles & Practice* (Foley, Van Dam, Feiner, & Hughes, 1990) and *Computer Graphics* (Hearn & Baker, 1986). Other texts include both of Alan Watt's influential books (Watt, 1993, 1991), Hill's *Computer Graphics Using OpenGL* (Francis S. Hill, 2000), Angel's *Interactive Computer Graphics: A Top-Down Approach Using OpenGL* (Angel, 2002), Hugues Hoppe's University of Washington dissertation (Hoppe, 1994), and Rogers' two excellent graphics texts (D. F. Rogers, 1985, 1989).

We would like to especially thank Alice and Klaus Peters for encouraging Pete to write the first edition of this book and for their great skill in bringing a book to fruition. Their patience with the authors and their dedication to making their books the best they can be has been instrumental in making this book what it is. This book certainly would not exist without their extraordinary efforts.

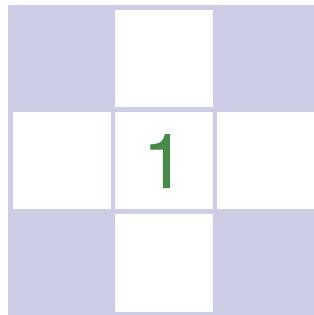
Salt Lake City, Utah
Ithaca, New York
February 2015

请允许我们使用他的照片。John "Spike" Hughes帮助改进了抽样理论的讨论。Wenzel Jakob的Mitsuba渲染器能够创建许多数字。我们非常感谢J.W. Baker帮助荷兰国际集团创造皮特设想的封面。除了是一个有才华的艺术家，他是一个非常愉快的工作与个人。

第一章注释中引用了许多有助于编写本书的著作。但是，影响内容和演示文稿的一些关键文本值得在这里得到特别的认可。其中包括两个经典的计算机图形ics文本，我们都从中学习了基础知识：计算机图形学：原理与实践（Foley, VanDam, Feiner, & Hughes, 1990）和计算机图形ics（Hearn & Baker, 1986）。其他文本包括Alan Watt的两本有影响力的书（Watt, 1993, 1991），Hill的ComputerGraphicsUsingOpenGL（Francis S. Hill, 2000），Angel的InteractiveComputerGraphics: A Top-Down Approach UsingOpenGL（Angel, 2002），Hugues Hoppe的UniversityOfWashington dissertation（Hoppe, 1994）和Rogers的两本优秀图形文本（D.F. Rogers, 1985, 1989）。

我们要特别感谢Alice和Klaus Peters的鼓励
皮特写了这本书的第一版，并感谢他们在使一本书取得成果方面的高超技巧。他们对作者的耐心和他们的奉献精神，使他们的书最好，他们可以帮助使这本书是什么。如果没有他们非凡的努力，这本书肯定不会存在。

犹他州盐湖城
伊萨卡，纽约
February 2015



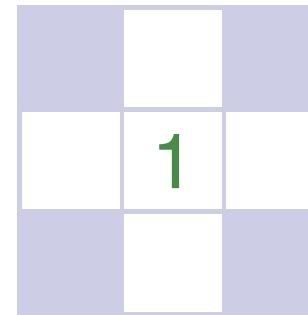
Introduction

The term *computer graphics* describes any use of computers to create and manipulate images. This book introduces the algorithmic and mathematical tools that can be used to create all kinds of images—realistic visual effects, informative technical illustrations, or beautiful computer animations. Graphics can be two- or three-dimensional; images can be completely synthetic or can be produced by manipulating photographs. This book is about the fundamental algorithms and mathematics, especially those used to produce synthetic images of three-dimensional objects and scenes.

Actually doing computer graphics inevitably requires knowing about specific hardware, file formats, and usually a graphics API (see Section 1.3) or two. Computer graphics is a rapidly evolving field, so the specifics of that knowledge are a moving target. Therefore, in this book we do our best to avoid depending on any specific hardware or API. Readers are encouraged to supplement the text with relevant documentation for their software and hardware environment. Fortunately, the culture of computer graphics has enough standard terminology and concepts that the discussion in this book should map nicely to most environments.

This chapter defines some basic terminology and provides some historical background, as well as information sources related to computer graphics.

API: application program interface.



Introduction

计算机图形学一词描述了任何使用计算机来创建和绘制图像。本书介绍了可用于创建各种图像的算法和数学工具—逼真的视觉效果，信息丰富的技术插图或精美的计算机动画。图形可以是两个或三维的；图像可以是完全合成的，或者可以通过拍摄照片来产生。这本书是关于基本的算法和数学仿真，特别是那些用于产生三维物体和场景的合成图像。

实际上做计算机图形学不可避免地需要了解特定的硬件，文件格式，通常还有一个图形API（见第1.3节）或两个。计算机图形学是一个快速发展的领域，所以这些知识的具体细节

边缘是一个运动目标。因此，在本书中，我们尽最大努力避免任何特定硬件或API上的de挂起。我们鼓励读者为他们的软件和硬件环境补充相关的文档。幸运的是，计算机图形文化有足够的标准术语和概念，本书的讨论应该很好地映射到大多数环境。

API: 应用程序接口。

1.1 Graphics Areas

Imposing categories on any field is dangerous, but most graphics practitioners would agree on the following major areas of computer graphics:

- **Modeling** deals with the mathematical specification of shape and appearance properties in a way that can be stored on the computer. For example, a coffee mug might be described as a set of ordered 3D points along with some interpolation rule to connect the points and a reflection model that describes how light interacts with the mug.
- **Rendering** is a term inherited from art and deals with the creation of shaded images from 3D computer models.
- **Animation** is a technique to create an illusion of motion through sequences of images. Animation uses modeling and rendering but adds the key issue of movement over time, which is not usually dealt with in basic modeling and rendering.

There are many other areas that involve computer graphics, and whether they are core graphics areas is a matter of opinion. These will all be at least touched on in the text. Such related areas include the following:

- **User interaction** deals with the interface between input devices such as mice and tablets, the application, feedback to the user in imagery, and other sensory feedback. Historically, this area is associated with graphics largely because graphics researchers had some of the earliest access to the input/output devices that are now ubiquitous.
- **Virtual reality** attempts to *immerse* the user into a 3D virtual world. This typically requires at least stereo graphics and response to head motion. For true virtual reality, sound and force feedback should be provided as well. Because this area requires advanced 3D graphics and advanced display technology, it is often closely associated with graphics.
- **Visualization** attempts to give users insight into complex information via visual display. Often there are graphic issues to be addressed in a visualization problem.
- **Image processing** deals with the manipulation of 2D images and is used in both the fields of graphics and vision.
- **3D scanning** uses range-finding technology to create measured 3D models. Such models are useful for creating rich visual imagery, and the processing of such models often requires graphics algorithms.

1.1 图形区域

在任何领域强加类别都是危险的，但大多数图形从业者会同意计算机图形学的以下主要领域：

*建模以可以存储在计算机上的方式处理形状和外观属性的数学规范。例如，一个咖啡杯可能被描述为一组有序的3D点，以及一些插值规则来连接这些点和一个反射模型，该模型描述了光如何与杯子相互作用。

*渲染是从艺术继承的一个术语，涉及从3d计算机模型创建阴影图像。

*动画是一种通过图像序列创造运动幻觉的技术。动画使用建模和渲染，但增加了随着时间的推移移动的关键问题，这在基本建模和渲染中通常不会处理。

还有许多其他领域涉及计算机图形学，它们是否是核心图形领域是一个意见问题。这些都将至少在文本中触及。这些相关领域包括：

*用户交互处理鼠标和平板电脑等输入设备之间的接口、应用程序、图像中对用户的反馈以及其他感官反馈。从历史上看，这一领域与图形相关，很大程度上是因为图形研究人员对现在无处不在的输入输出设备拥有最早的访问权限。

*虚拟现实试图让用户沉浸在3D虚拟世界中。这通常至少需要立体图形和对头部运动的响应。对于真正的虚拟现实，还应该提供声音和力反馈。由于这一领域需要先进的3d图形和先进的显示技术，它往往与图形密切相关。

*可视化试图通过可视化显示让用户洞察复杂的信息。通常在可视化问题中需要解决图形问题。

*图像处理处理2D图像的操作，并用于图形和视觉领域。

*3D扫描使用测距技术创建测量的3D模型。此类模型对于创建丰富的视觉图像非常有用，并且此类模型的处理通常需要图形算法。

- Computational photography is the use of computer graphics, computer vision, and image processing methods to enable new ways of photographically capturing objects, scenes, and environments.

1.2 Major Applications

Almost any endeavor can make some use of computer graphics, but the major consumers of computer graphics technology include the following industries:

- Video games increasingly use sophisticated 3D models and rendering algorithms.
- Cartoons are often rendered directly from 3D models. Many traditional 2D cartoons use backgrounds rendered from 3D models, which allow a continuously moving viewpoint without huge amounts of artist time.
- Visual effects use almost all types of computer graphics technology. Almost every modern film uses digital compositing to superimpose backgrounds with separately filmed foregrounds. Many films also use 3D modeling and animation to create synthetic environments, objects, and even characters that most viewers will never suspect are not real.
- Animated films use many of the same techniques that are used for visual effects, but without necessarily aiming for images that look real.
- CAD/CAM stands for *computer-aided design* and *computer-aided manufacturing*. These fields use computer technology to design parts and products on the computer and then, using these virtual designs, to guide the manufacturing process. For example, many mechanical parts are designed in a 3D computer modeling package and then automatically produced on a computer-controlled milling device.
- Simulation can be thought of as accurate video gaming. For example, a flight simulator uses sophisticated 3D graphics to simulate the experience of flying an airplane. Such simulations can be extremely useful for initial training in safety-critical domains such as driving, and for scenario training for experienced users such as specific fire-fighting situations that are too costly or dangerous to create physically.
- Medical imaging creates meaningful images of scanned patient data. For example, a computed tomography (CT) dataset is composed of a large 3D

*计算摄影是利用计算机图形学、计算机视觉和图像处理方法，以新的方式捕捉物体、场景和环境。

1.2 主要应用

几乎任何努力都可以利用计算机图形学，但计算机图形学技术的主要消费者包括以下行业：

- *视频游戏越来越多地使用复杂的3D模型和渲染算法。
- *卡通通常直接从3D模型渲染。许多传统的2d卡通使用从3d模型渲染的背景，这允许一个连续移动的观点没有大量的艺术家时间。
- *视觉效果使用几乎所有类型的计算机图形技术。几乎每部现代电影都使用数字合成将背景与单独拍摄的前景叠加在一起。许多电影还使用3D建模和动画来创建合成环境，对象，甚至大多数观众永远不会怀疑的角色都不是真实的。
- *动画电影使用许多用于视觉效果的相同技术，但不一定瞄准看起来真实的图像。
- *CADCAM代表计算机辅助设计和计算机辅助制造。这些领域使用计算机技术在计算机上设计零件和产品，然后使用这些虚拟设计来指导制造过程。例如，许多机械零件设计在3D计算机建模包中，然后在计算机控制的铣削装置上自动生产。
- *模拟可以被认为是准确的视频游戏。例如，飞行模拟器使用复杂的3d图形来模拟飞行飞机的体验。这种模拟对于驾驶等安全关键领域的初始培训非常有用，对于有经验的用户的场景培训也非常有用，例如成本太高或难以物理创建的特定消防情况。
- *医学成像创建扫描患者数据的有意义的图像。例如，计算机断层扫描(CT)数据集由大型3D组成

rectangular array of density values. Computer graphics is used to create shaded images that help doctors extract the most salient information from such data.

- **Information visualization** creates images of data that do not necessarily have a “natural” visual depiction. For example, the temporal trend of the price of ten different stocks does not have an obvious visual depiction, but clever graphing techniques can help humans see the patterns in such data.

1.3 Graphics APIs

A key part of using graphics libraries is dealing with a *graphics API*. An *application program interface* (API) is a standard collection of functions to perform a set of related operations, and a graphics API is a set of functions that perform basic operations such as drawing images and 3D surfaces into windows on the screen.

Every graphics program needs to be able to use two related APIs: a graphics API for visual output and a user-interface API to get input from the user. There are currently two dominant paradigms for graphics and user-interface APIs. The first is the integrated approach, exemplified by Java, where the graphics and user-interface toolkits are integrated and portable *packages* that are fully standardized and supported as part of the language. The second is represented by Direct3D and OpenGL, where the drawing commands are part of a software library tied to a language such as C++, and the user-interface software is an independent entity that might vary from system to system. In this latter approach, it is problematic to write portable code, although for simple programs it may be possible to use a portable library layer to encapsulate the system specific user-interface code.

Whatever your choice of API, the basic graphics calls will be largely the same, and the concepts of this book will apply.

1.4 Graphics Pipeline

Every desktop computer today has a powerful 3D *graphics pipeline*. This is a special software/hardware subsystem that efficiently draws 3D primitives in perspective. Usually these systems are optimized for processing 3D triangles with shared vertices. The basic operations in the pipeline map the 3D vertex locations to 2D screen positions and shade the triangles so that they both look realistic and appear in proper back-to-front order.

密度值的矩形阵列。计算机图形用于创建阴影图像，帮助医生从这些数据中提取最显着的信息。

*信息可视化创建不一定具有“自然”视觉描绘的数据图像。例如，十种不同股票价格的时间趋势没有明显的视觉描绘，但巧妙的图形技术可以帮助人类看到这些数据中的模式。

1.3 Graphics APIs

使用图形库的一个关键部分是处理图形API。Application程序接口(API)是执行一组相关操作的标准函数集合，图形API是执行基本操作的一组函数，例如将图像和3d表面绘制到屏幕上的窗口中。每个图形程序都需要能够使用两个相关的Api：一个图形

用于可视化输出的API和用于从用户获取输入的用户界面API。目前有两种主要的图形和用户界面Api范例。第一种是集成方法，以Java为例，其中图形和用户界面工具包是集成和可移植的包，作为语言的一部分完全标准化和支持。第二个由Direct3D和OpenGL表示，其中绘图命令是与c++等语言绑定的软件库的一部分，用户界面软件是一个独立的实体，可能因系统而异。在后一种方法中，编写可移植代码是有问题的，尽管对于简单的程序，可能可以使用可移植库层来封装系统特定的用户界面代码。

无论您选择哪种API，基本的图形调用都将大致相同，并且本书的概念将适用。

1.4 图形管道

今天的每台台式电脑都有一个强大的3d图形管道。这是一个特殊的软件硬件子系统，可以在透视中有效地绘制3D图元。通常这些系统被优化用于处理具有共享顶点的3d三角形。管道中的基本操作将3D顶点位置映射到2d屏幕位置，并为三角形着色，使它们看起来逼真，并以正确的前后顺序显示。



Although drawing the triangles in valid back-to-front order was once the most important research issue in computer graphics, it is now almost always solved using the *z-buffer*, which uses a special memory buffer to solve the problem in a brute-force manner.

It turns out that the geometric manipulation used in the graphics pipeline can be accomplished almost entirely in a 4D coordinate space composed of three traditional geometric coordinates and a fourth *homogeneous* coordinate that helps with perspective viewing. These 4D coordinates are manipulated using 4×4 matrices and 4-vectors. The graphics pipeline, therefore, contains much machinery for efficiently processing and composing such matrices and vectors. This 4D coordinate system is one of the most subtle and beautiful constructs used in computer science, and it is certainly the biggest intellectual hurdle to jump when learning computer graphics. A big chunk of the first part of every graphics book deals with these coordinates.

The speed at which images can be generated depends strongly on the number of triangles being drawn. Because interactivity is more important in many applications than visual quality, it is worthwhile to minimize the number of triangles used to represent a model. In addition, if the model is viewed in the distance, fewer triangles are needed than when the model is viewed from a closer distance. This suggests that it is useful to represent a model with a varying *level of detail* (LOD).

1.5 Numerical Issues

Many graphics programs are really just 3D numerical codes. Numerical issues are often crucial in such programs. In the “old days,” it was very difficult to handle such issues in a robust and portable manner because machines had different internal representations for numbers, and even worse, handled exceptions in different and incompatible ways. Fortunately, almost all modern computers conform to the *IEEE floating-point* standard (IEEE Standards Association, 1985). This allows the programmer to make many convenient assumptions about how certain numeric conditions will be handled.

Although IEEE floating-point has many features that are valuable when coding numeric algorithms, there are only a few that are crucial to know for most situations encountered in graphics. First, and most important, is to understand that there are three “special” values for real numbers in IEEE floating-point:

1. **Infinity (∞)**. This is a valid number that is larger than all other valid numbers.



虽然以前后顺序绘制三角形曾经是计算机图形学中最重要的研究问题，但现在几乎总是使用z缓冲区来解决，z缓冲区使用特殊的内存缓冲区以蛮力

事实证明，在图形管道中使用的几何操纵几乎完全可以在由三个原始几何坐标和有助于透视观看的第四齐次坐标组成的4D坐标空间中完成。这些4d坐标使用 4×4 矩阵和4矢量进行操作。因此，图形管道包含许多用于高效处理和组合此类矩阵和矢量的机器。这个4d坐标系是计算机科学中使用的最微妙和美丽的构造之一，它肯定是学习计算机图形学时跳跃的最大智力障碍。每本图形书的第一部分都有很大一部分涉及这些坐标。

生成图像的速度很大程度上取决于绘制的三角形的数量。因为交互性在许多应用阳离子中比视觉质量更重要，所以最小化用于表示模型的三角形的数量是值得的。此外，如果在远处观看模型，则需要比从更近的距离观看模型时更少的三角形。这表明表示具有不同细节级别(LOD)的模型是有用的。

1.5 数字问题

许多图形程序实际上只是3D数字代码。数字问题在这些项目中往往是至关重要的。在“旧时代”，以健壮和便携的方式处理这些问题是非常困难的，因为机器对数字有不同的内部表示，更糟糕的是，以不同的和不兼容的方式处理异常。幸运的是，几乎所有现代计算机都符合IEEE浮点标准（IEEE 标准协会，1985）。这也使得程序员对某些数字条件将如何处理做出许多方便的假设。

虽然IEEE浮点有许多功能在编码ing数字算法时很有价值，但对于图形中遇到的大多数情况来说，只有少数功能是至关重要的。首先，也是最重要的是要理解IEEE浮点中实数有三个“特殊”值：

- 1.无穷大(∞)。这是一个大于所有其他有效数字的有效数字。

2. **Minus infinity ($-\infty$)**. This is a valid number that is smaller than all other valid numbers.
3. **Not a number (NaN)**. This is an invalid number that arises from an operation with undefined consequences, such as zero divided by zero.

The designers of IEEE floating-point made some decisions that are extremely convenient for programmers. Many of these relate to the three special values above in handling exceptions such as division by zero. In these cases an exception is logged, but in many cases the programmer can ignore that. Specifically, for any positive real number a , the following rules involving division by infinite values hold:

$$\begin{aligned} +a/(+\infty) &= +0, \\ -a/(+\infty) &= -0, \\ +a/(-\infty) &= -0, \\ -a/(-\infty) &= +0. \end{aligned}$$

Other operations involving infinite values behave the way one would expect. Again for positive a , the behavior is as follows:

$$\begin{aligned} \infty + \infty &= +\infty, \\ \infty - \infty &= \text{NaN}, \\ \infty \times \infty &= \infty, \\ \infty/\infty &= \text{NaN}, \\ \infty/a &= \infty, \\ \infty/0 &= \infty, \\ 0/0 &= \text{NaN}. \end{aligned}$$

The rules in a Boolean expression involving infinite values are as expected:

1. All finite valid numbers are less than $+\infty$.
2. All finite valid numbers are greater than $-\infty$.
3. $-\infty$ is less than $+\infty$.

The rules involving expressions that have NaN values are simple:

1. Any arithmetic expression that includes NaN results in NaN.
2. Any Boolean expression involving NaN is false.

2. **减去无穷大 ($-\infty$)**。这是一个小于所有其他有效数字的有效数字。
3. **不是数字 (NaN)**。这是一个无效的数字，由具有未定义后果的操作产生，例如零除以零。

IEEE浮点的设计者做出了一些对程序员来说非常方便的决定。其中许多与上述处理异常（如除零）的三个特殊值有关。在这些情况下，会记录异常，但在许多情况下，程序员可以忽略它。具体来说，对于任何正实数 a ，以下规则涉及除以无限值

hold:

$$\begin{aligned} +a/(+\infty) &= +0, \\ -a/(+\infty) &= -0, \\ +a/(-\infty) &= -0, \\ -a/(-\infty) &= +0. \end{aligned}$$

其他涉及无限值的操作的行为方式是人们所期望的。
再次对于正 a ，行为如下：

$$\begin{aligned} \infty + \infty &= +\infty, \\ \infty - \infty &= \text{NaN}, \\ \infty \times \infty &= \infty, \\ \infty/\infty &= \text{NaN}, \\ \infty/a &= \infty, \\ \infty/0 &= \infty, \\ 0/0 &= \text{NaN}. \end{aligned}$$

涉及无限值的布尔表达式中的规则如预期的那样：

1. 所有有限有效数都小于 $+\infty$ 。
2. 所有有限有效数都大于 $-\infty$ 。
3. $-\infty$ 小于 $+\infty$ 。

涉及具有NaN值的表达式的规则很简单：

1. 任何包含NaN的算术表达式都会导致NaN。
2. 任何涉及NaN的布尔表达式都是false。

Perhaps the most useful aspect of IEEE floating-point is how divide-by-zero is handled; for any positive real number a , the following rules involving division by zero values hold:

$$\begin{aligned} +a / +0 &= +\infty, \\ -a / +0 &= -\infty. \end{aligned}$$

There are many numeric computations that become much simpler if the programmer takes advantage of the IEEE rules. For example, consider the expression:

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

Such expressions arise with resistors and lenses. If divide-by-zero resulted in a program crash (as was true in many systems before IEEE floating-point), then two *if* statements would be required to check for small or zero values of b or c . Instead, with IEEE floating-point, if b or c is zero, we will get a zero value for a as desired. Another common technique to avoid special checks is to take advantage of the Boolean properties of NaN. Consider the following code segment:

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

Here, the function f may return “ugly” values such as ∞ or NaN, but the *if* condition is still well-defined: it is false for $a = \text{NaN}$ or $a = -\infty$ and true for $a = +\infty$. With care in deciding which values are returned, often the *if* can make the right choice, with no special checks needed. This makes programs smaller, more robust, and more efficient.

1.6 Efficiency

There are no magic rules for making code more efficient. Efficiency is achieved through careful tradeoffs, and these tradeoffs are different for different architectures. However, for the foreseeable future, a good heuristic is that programmers should pay more attention to memory access patterns than to operation counts. This is the opposite of the best heuristic of two decades ago. This switch has occurred because the speed of memory has not kept pace with the speed of processors. Since that trend continues, the importance of limited and coherent memory access for optimization should only increase.

A reasonable approach to making code fast is to proceed in the following order, taking only those steps which are needed:

也许IEEE浮点最有用的方面是如何处理除以零;对于任何正实数a, 以下涉及除以零值的规则成立:

$$\begin{aligned} +a / +0 &= +\infty, \\ -a / +0 &= -\infty. \end{aligned}$$

如果可能出现负零 (-0) , 则必须小心。

如果programmer利用IEEE规则, 许多数字计算变得更加简单。例如, 考虑expression:

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

这样的表达与电阻器和透镜一起出现。如果除以零导致程序崩溃 (在IEEE浮点之前的许多系统中也是如此), 那么需要两个if语句来检查b或c的小值或零值。相反, 对于IEEE浮点, 如果b或c为零, 我们将根据需要获得a的零值。避免特殊检查的另一种常用技术是利用NaN的布尔属性。考虑以下代码段:

```
a=f(x)if(a>0)t
hendosomet
hing
```

这里, 函数f可能会返回“丑陋”的值, 如 ∞ 或NaN, 但if条件仍然定义良好: 对于 $a=\text{NaN}$ 或 $a=-\infty$ 为false, 对于 $a=+\infty$ 为true。在决定返回哪些值时要小心, 通常if可以做出正确的选择, 而不需要特殊的检查。这使得程序更小, 更健壮, 更高效。

1.6 Efficiency

没有神奇的规则可以使代码更有效率。效率是通过仔细的权衡来实现的, 而这些权衡对于不同的架构是不同的。然而, 在可预见的未来, 一个好的启发式是程序员应该更多地关注内存访问模式而不是操作计数。这与二十年前最好的启发式相反。这个开关有occurred, 因为内存的速度没有跟上processors的速度。由于这种趋势仍在继续, 有限和一致的内存访问对于优化的重要性只会增加。

快速编写代码的合理方法是按照以下顺序进行, 只采取所需的步骤:

1. Write the code in the most straightforward way possible. Compute intermediate results as needed on the fly rather than storing them.
2. Compile in optimized mode.
3. Use whatever profiling tools exist to find critical bottlenecks.
4. Examine data structures to look for ways to improve locality. If possible, make data unit sizes match the cache/page size on the target architecture.
5. If profiling reveals bottlenecks in numeric computations, examine the assembly code generated by the compiler for missed efficiencies. Rewrite source code to solve any problems you find.

The most important of these steps is the first one. Most “optimizations” make the code harder to read without speeding things up. In addition, time spent upfront optimizing code is usually better spent correcting bugs or adding features. Also, beware of suggestions from old texts; some classic tricks such as using integers instead of reals may no longer yield speed because modern CPUs can usually perform floating-point operations just as fast as they perform integer operations. In all situations, profiling is needed to be sure of the merit of any optimization for a specific machine and compiler.

1.7 Designing and Coding Graphics Programs

Certain common strategies are often useful in graphics programming. In this section we provide some advice that you may find helpful as you implement the methods you learn about in this book.

I believe strongly in the KISS (“keep it simple, stupid”) principle, and in that light the argument for two classes is not compelling enough to justify the added complexity.
—P.S.

I like keeping points and vectors separate because it makes code more readable and can let the compiler catch some bugs.
—S.M.

1.7.1 Class Design

A key part of any graphics program is to have good classes or routines for geometric entities such as vectors and matrices, as well as graphics entities such as RGB colors and images. These routines should be made as clean and efficient as possible. A universal design question is whether locations and displacements should be separate classes because they have different operations, e.g., a location multiplied by one-half makes no geometric sense while one-half of a displacement does (Goldman, 1985; DeRose, 1989). There is little agreement on this question, which can spur hours of heated debate among graphics practitioners, but for the sake of example let’s assume we will not make the distinction.

- 1.以最直接的方式编写代码。根据需要实时计算中间结果，而不是存储它们。
- 2.在优化模式下编译。
- 3.使用现有的分析工具来查找关键瓶颈。
- 4.检查数据结构以寻找提高局部性的方法。如果可能，请使数据单元大小与目标体系结构上的缓存页大小相匹配。
- 5.如果分析揭示了数值计算中的瓶颈，请检查编译器生成的汇编代码是否漏掉了效率。重写源代码以解决您发现的任何问题。

这些步骤中最重要的是第一个。大多数“优化”使代码更难阅读，而不会加快速度。此外，优化代码所花费的时间通常更好地用于纠正错误或添加功能。此外，请注意旧文本的建议：一些经典技巧，如使用整数而不是实数，可能不再产生速度，因为现代CPU通常可以执行浮点运算，就像它们执行整数运算一样快。在所有情况下，都需要进行性能分析，以确保针对特定机器和编译器进行任何优化的优点。

1.7 设计和编码图形程序

某些常用策略在图形编程中通常很有用。在本节中，我们提供了一些建议，您可能会发现在实现本书中了解的方法时会有所帮助。

1.7.1 班级设计

我坚信
这是简单的，“愚蠢的”)
原则，从这个角度来看
，两个类的论点不足以
证明增加的复杂性。-附
：

我喜欢保持点和向量分
开，因为它使代码更具
可读性，并且可以让编
译器捕获一些错误。-S.
M.

可靠的。一个普遍的设计问题是，位置和位移是否应该是单独的类，因为它们具有不同的操作，例如，位置乘以二分之一没有几何意义，而位移的二分之一则没有（Goldman, 1985; DeRose, 1989）。在这个问题上几乎没有达成一致，这可能会刺激图形从业者之间数小时的激烈辩论，但为了举例，让我们假设我们不会做出区分。



This implies that some basic classes to be written include:

- **vector2**. A 2D vector class that stores an x - and y -component. It should store these components in a length-2 array so that an indexing operator can be well supported. You should also include operations for vector addition, vector subtraction, dot product, cross product, scalar multiplication, and scalar division.
- **vector3**. A 3D vector class analogous to vector2.
- **hvector**. A homogeneous vector with four components (see Chapter 7).
- **rgb**. An RGB color that stores three components. You should also include operations for RGB addition, RGB subtraction, RGB multiplication, scalar multiplication, and scalar division.
- **transform**. A 4×4 matrix for transformations. You should include a matrix multiply and member functions to apply to locations, directions, and surface normal vectors. As shown in Chapter 6, these are all different.
- **image**. A 2D array of RGB pixels with an output operation.

In addition, you might or might not want to add classes for intervals, orthonormal bases, and coordinate frames.

1.7.2 Float vs. Double

Modern architecture suggests that keeping memory use down and maintaining coherent memory access are the keys to efficiency. This suggests using single-precision data. However, avoiding numerical problems suggests using double-precision arithmetic. The tradeoffs depend on the program, but it is nice to have a default in your class definitions.

1.7.3 Debugging Graphics Programs

If you ask around, you may find that as programmers become more experienced, they use traditional debuggers less and less. One reason for this is that using such debuggers is more awkward for complex programs than for simple programs. Another reason is that the most difficult errors are conceptual ones where the wrong thing is being implemented, and it is easy to waste large amounts of time stepping through variable values without detecting such cases. We have found several debugging strategies to be particularly useful in graphics.



这意味着要编写的一些基本类包括:

*vector2。存储x和y分量的2D矢量类。它应该将这些组件存储在length-2数组中，以便索引运算符可以得到很好的支持。您还应该包括向量加法，向量减法，点积，叉积，标量乘法和标量除法的操作。

*vector3。类似于vector2的3D向量类。

*hvector。具有四个分量的齐次向量（参见第7章）。

*rgb。存储三个分量的RGB颜色。您还应该包括RGB加法，RGB减法，RGB乘法，标量乘法和标量除法的操作。

*变换。变换的 4×4 矩阵。您应该包括矩阵乘法和成员函数以应用于位置，方向和表面法向量。如第6章所示，这些都是不同的。

*图像。输出操作的RGB像素的2D阵列。

此外，您可能希望也可能不希望为间隔、正交基和坐标系添加类。

You might also consider a special class for unit-length vectors, although I have found them more pain than they are worth. —P.S.

I suggest using doubles for geometric computation and floats for color computation. For data that occupies a lot of memory, such as triangle meshes, I suggest storing float data, but converting to double when data is accessed through member functions. —P.S.

I advocate doing all computations with floats until you find evidence that double precision is needed in a particular part of the code. —S.M.

你也可以考虑一个单位长度向量的特殊类
虽然我发现他们的痛苦比他们的价值还要多。-附：

我建议使用双打进行几何计算，并使用浮点数进行颜色计算。对于占用大量内存的数据，如三角形网格，我建议存储float数据，但在通过成员函数访问数据时转换为double。-附：

我主张用浮点数做所有的计算，直到你发现在代码的特定部分需要双精度的证据。-S.M.

1.7.2 浮动与双

现代体系结构表明，保持内存使用和保持一致的内存访问是效率的关键。这建议使用单精度数据。然而，避免数值问题建议使用doubleprecision算术。权衡取决于程序，但在类定义中有一个默认值是很好的。

1.7.3 调试图形程序

如果你问周围，你可能会发现，随着程序员变得更有经验，他们使用传统的调试器越来越少。这样做的一个原因是，使用这样的调试器对于复杂的程序比简单的程序更尴尬。另一个原因是，最困难的错误是概念上的错误，其中错误的东西正在实现，并且很容易浪费大量的时间通过变量值而没有检测到这种情况。我们发现几种调试策略在图形中特别有用。

The Scientific Method

In graphics programs there is an alternative to traditional debugging that is often very useful. The downside to it is that it is very similar to what computer programmers are taught not to do early in their careers, so you may feel “naughty” if you do it: we create an image and observe what is wrong with it. Then, we develop a hypothesis about what is causing the problem and test it. For example, in a ray-tracing program we might have many somewhat random looking dark pixels. This is the classic “shadow acne” problem that most people run into when they write a ray tracer. Traditional debugging is not helpful here; instead, we must realize that the shadow rays are hitting the surface being shaded. We might notice that the color of the dark spots is the ambient color, so the direct lighting is what is missing. Direct lighting can be turned off in shadow, so you might hypothesize that these points are incorrectly being tagged as in shadow when they are not. To test this hypothesis, we could turn off the shadowing check and recompile. This would indicate that these are false shadow tests, and we could continue our detective work. The key reason that this method can sometimes be good practice is that we never had to spot a false value or really determine our conceptual error. Instead, we just narrowed in on our conceptual error experimentally. Typically only a few trials are needed to track things down, and this type of debugging is enjoyable.

Images as Coded Debugging Output

In many cases, the easiest channel by which to get debugging information out of a graphics program is the output image itself. If you want to know the value of some variable for part of a computation that runs for every pixel, you can just modify your program temporarily to copy that value directly to the output image and skip the rest of the calculations that would normally be done. For instance, if you suspect a problem with surface normals is causing a problem with shading, you can copy the normal vectors directly to the image (x goes to red, y goes to green, z goes to blue), resulting in a color-coded illustration of the vectors actually being used in your computation. Or, if you suspect a particular value is sometimes out of its valid range, make your program write bright red pixels where that happens. Other common tricks include drawing the back sides of surfaces with an obvious color (when they are not supposed to be visible), coloring the image by the ID numbers of the objects, or coloring pixels by the amount of work they took to compute.

Using a Debugger

There are still cases, particularly when the scientific method seems to have led to a contradiction, when there’s no substitute for observing exactly what is going

在图形程序中，有一个传统调试的替代方案，通常非常有用。它的缺点是它与计算机程序员在职业生涯早期被教导不要做的事情非常相似，所以如果你这样做，你可能会感到“顽皮”：我们创建一个图像并观察它有什么然后，我们制定一个关于导致问题的假设并对其进行测试。例如，在光线跟踪程序中，我们可能会有许多看起来有些随机的暗像素。这是大多数人在写射线追踪器时遇到的经典“阴影痤疮”问题。传统的调试在这里没有帮助；相反，我们必须意识到阴影射线正在击中被阴影的表面。我们可能会注意到，暗点的颜色是环境颜色，所以直接照明是缺少的。可以在阴影中关闭直接照明，因此您可以假设这些点在未被标记为阴影时被错误地标记为阴影。为了验证这个假设，我们可以关闭阴影检查并重新编译。这表明这些是虚假的阴影测试，我们可以继续我们的侦探工作。这种方法有时可以成为良好实践的关键原因是从未发现错误值或真正确定我们的概念错误。相反，我们只是通过实验缩小了我们的概念错误。通常只需要几个试验来跟踪事情，这种类型的调试是令人愉快的。

图像作为编码调试输出

在许多情况下，从图形程序中获取调试信息的最简单渠道是输出图像本身。如果您想知道某个变量对于每个像素运行的部分计算的值，您可以暂时修改您的程序，将该值直接复制到输出图像，并跳过其余的计算，通常会完成。例如，如果您怀疑表面法线的问题导致着色问题，则可以将法向量直接复制到图像（ x 转为红色， y 转为绿色， z 转为蓝色），从而生成计算中实际使用的矢量的或者，如果您怀疑某个特定值有时超出其有效范围，请让您的程序在发生这种情况的地方写入鲜红色像素。其他常见的技巧包括用明显的颜色绘制表面的背面（当它们不应该是可见的时候），用对象的ID号给图像着色，或者用计算的工作量给像素着色。

使用调试器

还有一些情况，特别是当科学方法似乎导致了矛盾时，当没有任何替代物可以准确地观察正在发生的事情时

on. The trouble is that graphics programs often involve many, many executions of the same code (once per pixel, for instance, or once per triangle), making it completely impractical to step through in the debugger from the start. And the most difficult bugs usually only occur for complicated inputs.

A useful approach is to “set a trap” for the bug. First, make sure your program is deterministic—run it in a single thread and make sure that all random numbers are computed from fixed seeds. Then, find out which pixel or triangle is exhibiting the bug and add a statement before the code you suspect is incorrect that will be executed only for the suspect case. For instance, if you find that pixel (126, 247) exhibits the bug, then add:

```
if x = 126 and y = 247 then
    print "blarg!"
```

If you set a breakpoint on the print statement, you can drop into the debugger just before the pixel you’re interested in is computed. Some debuggers have a “conditional breakpoint” feature that can achieve the same thing without modifying the code.

In the cases where the program crashes, a traditional debugger is useful for pinpointing the site of the crash. You should then start backtracking in the program, using asserts and recompiles, to find where the program went wrong. These asserts should be left in the program for potential future bugs you will add. This again means the traditional step-through process is avoided, because that would not be adding the valuable asserts to your program.

Data Visualization for Debugging

Often it is hard to understand what your program is doing, because it computes a lot of intermediate results before it finally goes wrong. The situation is similar to a scientific experiment that measures a lot of data, and one solution is the same: make good plots and illustrations for yourself to understand what the data means. For instance, in a ray tracer you might write code to visualize ray trees so you can see what paths contributed to a pixel, or in an image resampling routine you might make plots that show all the points where samples are being taken from the input. Time spent writing code to visualize your program’s internal state is also repaid in a better understanding of its behavior when it comes time to optimize it.

上。麻烦的是，图形程序通常涉及同一代码的许多执行（例如，每个像素一次或每个三角形一次），因此从一开始就在调试器中单步执行完全不切实际。而最困难的bug通常只发生在复杂的输入上。

一个有用的方法是为bug“设置陷阱”。首先，确保您的程序是确定性的—在单个线程中运行它，并确保所有随机数都是从固定种子计算的。然后，找出哪个像素或三角形正在显示

该错误并在您怀疑的代码之前添加一条语句不正确，该语句将仅针对可疑情况执行。例如，如果您发现像素 (126 247) 显示错误，则添加：

如果x=126和y=247然后打
印"blarg!"

如果您在print语句上设置断点，则可以在计算您感兴趣的像素之前将其放入调试器。一些调试器有一个“conditional断点”功能，可以在不修改代码的情况下实现同样的事情。

在程序崩溃的情况下，传统的调试器对于精确定位崩溃的站点非常有用。然后，您应该在program中开始回溯，使用断言和重新编译，以查找程序出错的位置。这些断言应该留在程序中，以便您将来添加潜在的错误。这再次意味着避免了传统的逐步过程，因为这不会为您的程序添加有价值的断言。

A special debugging mode
that uses fixed random-
number seeds is useful.

一种特殊的调试模式，
使用固定的随机数种子
是有用的。

用于调试的数据可视化

通常很难理解你的程序在做什么，因为它在最终出错之前计算了很多中间结果。情况类似于测量大量数据的科学实验，一种解决方案是相同的：为自己制作好的情节和插图，以了解数据意味着什么。例如，在光线追踪器中，您可能会编写代码来可视化光线树，以便您

I like to format debugging
print statements so that the
output happens to be a Mat-
lab or Gnuplot script that
makes a helpful plot.
—S.M.

我喜欢格式化调试打印
语句，以便输出恰好是M
atlab或Gnuplot脚本，可
以制作有用的绘图。-S.
M.

可以看到哪些路径对像素有贡献，或者在图像重采样例程中，您可以绘制图来显示从输入中提取样本的所有点。在编写代码以可视化程序的内部状态所花费的时间也可以在优化程序时更好地理解程序的行为。

Notes

The discussion of software engineering is influenced by the *Effective C++* series (Meyers, 1995, 1997), the *Extreme Programming* movement (Beck & An-

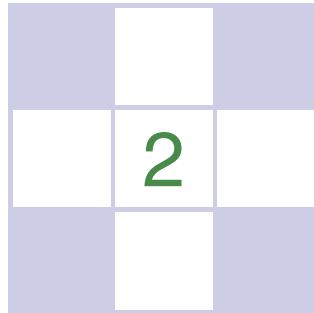
Notes

软件工程的讨论受到有效的C++系列 (Meyers, 1995, 1997)，极限编程运动 (Beck & An-

dres, 2004), and *The Practice of Programming* (Kernighan & Pike, 1999). The discussion of experimental debugging is based on discussions with Steve Parker.

There are a number of annual conferences related to computer graphics, including ACM SIGGRAPH and SIGGRAPH Asia, Graphics Interface, the Game Developers Conference (GDC), Eurographics, Pacific Graphics, High Performance Graphics, the Eurographics Symposium on Rendering, and IEEE VisWeek. These can be readily found by web searches on their names.

dres, 2004) , 以及编程的实践 (Kernighan & Pike, 1999) 。实验调试的讨论基于与SteveParker的讨论。有许多与计算机图形相关的年度会议，包括ACMSIGGRAPH和SIGGRAPHAsia, GraphicsInterface, 游戏开发者大会 (Gdc) , Eurographics, PacificGraphics, HighPerformanceGraphics , EurographicsSymposiumonRendering和IEEEVisWeek。这些可以通过网络搜索他们的名字很容易找到。

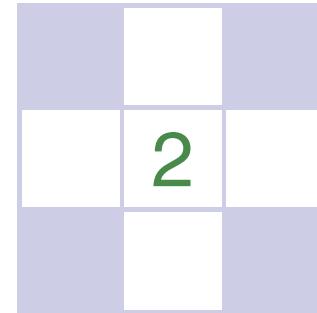


Miscellaneous Math

Much of graphics is just translating math directly into code. The cleaner the math, the cleaner the resulting code; so much of this book concentrates on using just the right math for the job. This chapter reviews various tools from high school and college mathematics and is designed to be used more as a reference than as a tutorial. It may appear to be a hodge-podge of topics and indeed it is; each topic is chosen because it is a bit unusual in “standard” math curricula, because it is of central importance in graphics, or because it is not typically treated from a geometric standpoint. In addition to establishing a review of the notation used in the book, the chapter also emphasizes a few points that are sometimes skipped in the standard undergraduate curricula, such as barycentric coordinates on triangles. This chapter is not intended to be a rigorous treatment of the material; instead intuition and geometric interpretation are emphasized. A discussion of linear algebra is deferred until Chapter 5 just before transformation matrices are discussed. Readers are encouraged to skim this chapter to familiarize themselves with the topics covered and to refer back to it as needed. The exercises at the end of the chapter may be useful in determining which topics need a refresher.

2.1 Sets and Mappings

Mappings, also called *functions*, are basic to mathematics and programming. Like a function in a program, a mapping in math takes an argument of one *type* and



杂项数学

大部分图形只是将数学直接翻译成代码。数学越干净，生成的代码就越干净；这本书的大部分内容都集中在使用正确的数学来完成工作。本章回顾了高中和大学数学的各种工具，旨在更多地用作参考而不是教程。它可能看起来是一个主题的大杂烩，实际上它是；选择每个主题是因为它在“标准”数学课程中有点不寻常，因为它在图形中具有核心重要性，或者因为它不除了对书中使用的符号进行回顾之外，该章还强调了标准本科课程中有时会跳过的几点，例如三角形上的重心坐标。本章并不是要对材料进行严格的处理，而是强调直觉和几何解释。线性代数的讨论推迟到第5章讨论变换矩阵之前。我们鼓励读者浏览本章，以熟悉所涵盖的主题，并根据需要重新参考。本章末尾的练习可能有助于确定哪些主题需要复习。

2.1 集和映射

映射，也称为函数，是数学和编程的基础。就像程序中的函数一样，数学中的映射需要一个类型的参数和

maps it to (returns) an object of a particular type. In a program we say “type”; in math we would identify the set. When we have an object that is a member of a set, we use the \in symbol. For example,

$$a \in S,$$

can be read “ a is a member of set S .” Given any two sets A and B , we can create a third set by taking the *Cartesian product* of the two sets, denoted $A \times B$. This set $A \times B$ is composed of all possible ordered pairs (a, b) where $a \in A$ and $b \in B$. As a shorthand, we use the notation A^2 to denote $A \times A$. We can extend the Cartesian product to create a set of all possible ordered triples from three sets and so on for arbitrarily long ordered tuples from arbitrarily many sets.

Common sets of interest include

- \mathbb{R} —the real numbers;
- \mathbb{R}^+ —the nonnegative real numbers (includes zero);
- \mathbb{R}^2 —the ordered pairs in the real 2D plane;
- \mathbb{R}^n —the points in n -dimensional Cartesian space;
- \mathbb{Z} —the integers;
- S^2 —the set of 3D points (points in \mathbb{R}^3) on the unit sphere.

Note that although S^2 is composed of points embedded in three-dimensional space, they are on a surface that can be parameterized with two variables, so it can be thought of as a 2D set. Notation for mappings uses the arrow and a colon, for example:

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

which you can read as “There is a function called f that takes a real number as input and maps it to an integer.” Here, the set that comes before the arrow is called the *domain* of the function, and the set on the right-hand side is called the *target*. Computer programmers might be more comfortable with the following equivalent language: “There is a function called f which has one real argument and returns an integer.” In other words, the set notation above is equivalent to the common programming notation:

$$\text{integer } f(\text{real}) \leftarrow \text{equivalent} \rightarrow f : \mathbb{R} \mapsto \mathbb{Z}.$$

So the colon-arrow notation can be thought of as a programming syntax. It’s that simple.

The point $f(a)$ is called the *image* of a , and the image of a set A (a subset of the domain) is the subset of the target that contains the images of all points in A . The image of the whole domain is called the *range* of the function.

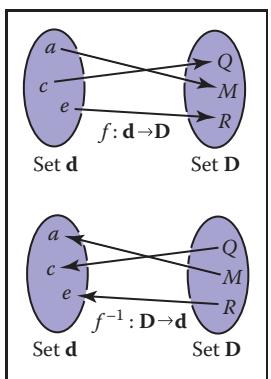


Figure 2.1. A bijection f and the inverse function f^{-1} . Note that f^{-1} is also a bijection.

将其映射到（返回）特定类型的对象。在程序中，我们说“类型”;在数学中，我们会识别集合。当我们有一个对象是一个集合的成员时，我们使用 \in 符号。例如

$$a \in S,$$

可读“ a 是集合 S 的成员。”给定任何两个集合 A 和 B ，我们可以通过取两个集合的笛卡尔积来创建第三个集合，表示 $A \times B$ 。这个集合 $A \times B$ 由所有可能的有序对 (a, b) 组成，其中 $a \in A$ 和 $b \in B$ 。作为简写，我们使用符号 A^2 表示 $A \times A$ 。我们可以扩展笛卡尔积，从三个集合中创建一组所有可能的有序三元组，以此类推，用于来自任意多个集合的任意长的有序元组。常见的兴趣包括

- \mathbb{R} —the real numbers;
- * \mathbb{R}^+ —非负实数（包括零）；
- * \mathbb{R}^2 —真实2D平面中的有序对；
- * \mathbb{R}^n — n 维笛卡尔空间中的点；
- \mathbb{Z} —the integers;
- * S^2 —单位球体上的3D点集（ \mathbb{R}^3 中的点）。

请注意，虽然 S^2 由嵌入三维空间中的点组成，但它们位于可以用两个变量参数化的表面上，因此可以将其视为2D集合。映射的符号使用箭头和冒号，例如：

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

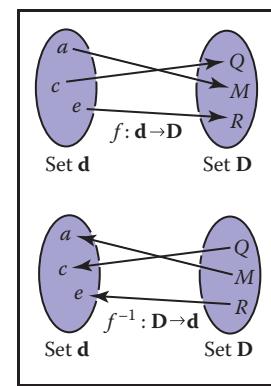


图2.1。一个双射 f 和反函数 f^{-1} 。
○. 请注意， f^{-1} 也是双射。

$$\text{整数 } f(\text{实数}) \leftarrow \text{equivalent} \rightarrow f : \mathbb{R} \mapsto \mathbb{Z}.$$

因此，冒号-箭头表示法可以被认为是一种编程语法。就这么简单。

点 $f(a)$ 称为 a 的图像，集合 A （域的子集）的像是包含 A 中所有点图像的目标子集。整个域的像是函数的范围。



2.1.1 Inverse Mappings

If we have a function $f : A \rightarrow B$, there may exist an *inverse function* $f^{-1} : B \rightarrow A$, which is defined by the rule $f^{-1}(b) = a$ where $b = f(a)$. This definition only works if every $b \in B$ is an image of some point under f (that is, the range equals the target) and if there is only one such point (that is, there is only one a for which $f(a) = b$). Such mappings or functions are called *bijections*. A bijection maps every $a \in A$ to a unique $b \in B$, and for every $b \in B$, there is exactly one $a \in A$ such that $f(a) = b$ (Figure 2.1). A bijection between a group of riders and horses indicates that everybody rides a single horse, and every horse is ridden. The two functions would be *rider(horse)* and *horse(rider)*. These are inverse functions of each other. Functions that are not bijections have no inverse (Figure 2.2).

An example of a bijection is $f : \mathbb{R} \rightarrow \mathbb{R}$, with $f(x) = x^3$. The inverse function is $f^{-1}(x) = \sqrt[3]{x}$. This example shows that the standard notation can be somewhat awkward because x is used as a dummy variable in both f and f^{-1} . It is sometimes more intuitive to use different dummy variables, with $y = f(x)$ and $x = f^{-1}(y)$. This yields the more intuitive $y = x^3$ and $x = \sqrt[3]{y}$. An example of a function that does not have an inverse is $sqr : \mathbb{R} \rightarrow \mathbb{R}$, where $sqr(x) = x^2$. This is true for two reasons: first $x^2 = (-x)^2$, and second no members of the domain map to the negative portions of the target. Note that we can define an inverse if we restrict the domain and range to \mathbb{R}^+ . Then \sqrt{x} is a valid inverse.

2.1.2 Intervals

Often we would like to specify that a function deals with real numbers that are restricted in value. One such constraint is to specify an *interval*. An example of an interval is the real numbers between zero and one, not including zero or one. We denote this $(0, 1)$. Because it does not include its endpoints, this is referred to as an *open interval*. The corresponding *closed interval*, which does contain its endpoints, is denoted with square brackets: $[0, 1]$. This notation can be mixed, i.e., $[0, 1)$ includes zero but not one. When writing an interval $[a, b]$, we assume that $a \leq b$. The three common ways to represent an interval are shown in Figure 2.3. The Cartesian products of intervals are often used. For example, to indicate that a point x is in the unit cube in 3D, we say $x \in [0, 1]^3$.

Intervals are particularly useful in conjunction with set operations: *intersection*, *union*, and *difference*. For example, the intersection of two intervals is the set of points they have in common. The symbol \cap is used for intersection. For example, $[3, 5] \cap [4, 6] = [4, 5]$. For unions, the symbol \cup is used to denote points in either interval. For example, $[3, 5] \cup [4, 6] = [3, 6]$. Unlike the first two operators, the difference operator produces different results depending on argument order.

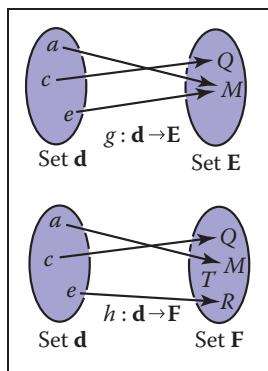


Figure 2.2. The function g does not have an inverse because two elements of d map to the same element of E . The function h has no inverse because element T of F has no element of d mapped to it.

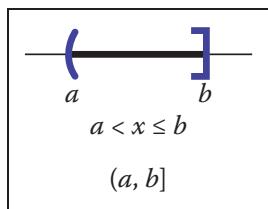


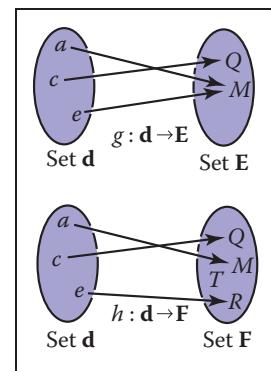
Figure 2.3. Three equivalent ways to denote the interval from a to b that includes b but not a .

2.1.1 Inverse Mappings

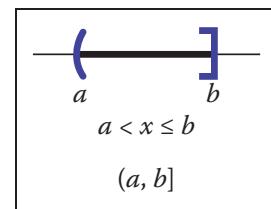
如果我们有一个函数 $f: A \rightarrow B$, 则可能存在一个反函数 $f^{-1}: B \rightarrow A$, 它由规则 $f^{-1}(b) = a$ 定义, 其中 $b = f(a)$ 。只有当每个 $b \neq B$ 是 f 下某个点的图像 (即范围等于目标) 并且只有一个这样的点 (即只有一个 $f(a) = b$) 时, 这个定义才有效。这种映射或函数称为双射。双射将每个 $a \in A$ 映射到唯一的 $b \in B$, 对于每个 $b \in B$, 正好有一个 $a \in A$ 使得 $f(a) = b$ (图2.1)。一群骑手和马之间的双射表明每个人都骑一匹马, 每匹马都被骑。这两个功能将是骑手 (马) 和马 (骑手)。这些是彼此的反函数。不是双射的函数没有逆 (图2.2)。

双射的一个例子是 $f: 3\sqrt{x}$. 这个例子表明, 标准符号可以是

有点尴尬, 因为 x 在 f 和 f^{-1} 中都被用作虚拟变量。有时使用不同的虚拟变量更直观, $v=f(x)$ 和 $x=f^{-1}(v)$ 。这产生了更直观的 $v=x^3$ 和 $x=3\sqrt{y}$. 不具有逆的函数的示例是 $sqr: \mathbb{R} \rightarrow \mathbb{R}$, 其中 $sqr(x)=x^2$ 。有两个原因: 第一个 $x^2=(-x)^2$, 第二个域没有成员映射到目标的负部分。请注意, 如果我们将域和范围限制为 \mathbb{R}^+ , 我们可以定义逆。然后 \sqrt{x} 是一个有效的逆。



函数 g 不具有逆, 因为 d 的两个元素映射到 E 的相同元素。函数 h 没有逆, 因为 F 的元素 T 没有映射到它的 d 的元素。



表示间隔的三种等价方式
从 a 到包括 b 但不包括 a 的 b 。

2.1.2 Intervals

通常我们想指定一个函数处理值受限制的实数。一个这样的约束是指定一个间隔。间隔的一个例子是零到一之间的实数, 不包括零或一。我们表示 this $(0, 1)$ 。因为它不包括其端点, 所以这被称为开放间隔。相应的闭区间包含其端点, 用方括号表示: $[0, 1]$ 。这种表示法可以混合, 即 $[0, 1]$ 包括零但不包括一。当写入区间 $[a, b]$ 时, 我们假设 $a \leq b$ 。表示区间的三种常用方式如图2.3所示。经常使用间隔的笛卡尔积。例如, 为了表示点 x 在 3D 中的单位立方体中, 我们说 $x \in [0, 1]^3$ 。

间隔与集合操作一起特别有用: 交集、并集和差。例如, 两个区间的交集是它们共有的点集。符号 \cap 用于相交。例如 $[3, 5] \cap [4, 6] = [4, 5]$ 。对于联合, 符号 \cup 用于表示任一区间中的点。例如 $[3, 5] \cup [4, 6] = [3, 6]$ 。与前两个运算符不同, 差异运算符根据参数顺序产生不同的结果。

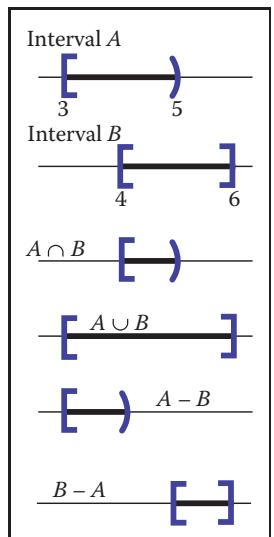


Figure 2.4. Interval operations on [3,5) and [4,6].

The minus sign is used for the difference operator, which returns the points in the left interval that are not also in the right. For example, $[3, 5) - [4, 6] = [3, 4)$ and $[4, 6] - [3, 5) = [5, 6]$. These operations are particularly easy to visualize using interval diagrams (Figure 2.4).

2.1.3 Logarithms

Although not as prevalent today as they were before calculators, *logarithms* are often useful in problems where equations with exponential terms arise. By definition, every logarithm has a *base a*. The “log base *a*” of *x* is written $\log_a x$ and is defined as “the exponent to which *a* must be raised to get *x*,” i.e.,

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

Note that the logarithm base *a* and the function that raises *a* to a power are inverses of each other. This basic definition has several consequences:

$$\begin{aligned} a^{\log_a(x)} &= x; \\ \log_a(a^x) &= x; \\ \log_a(xy) &= \log_a x + \log_a y; \\ \log_a(x/y) &= \log_a x - \log_a y; \\ \log_a x &= \log_b b \log_b x. \end{aligned}$$

When we apply calculus to logarithms, the special number $e = 2.718\dots$ often turns up. The logarithm with base e is called the *natural logarithm*. We adopt the common shorthand \ln to denote it:

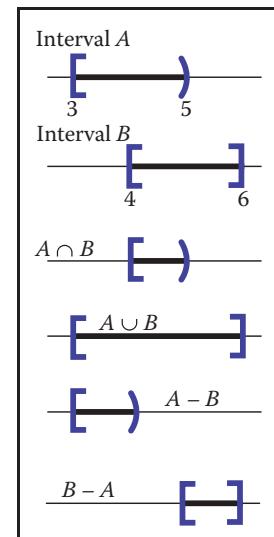
$$\ln x \equiv \log_e x.$$

Note that the “ \equiv ” symbol can be read “is equivalent by definition.” Like π , the special number e arises in a remarkable number of contexts. Many fields use a particular base in addition to e for manipulations and omit the base in their notation, i.e., $\log x$. For example, astronomers often use base 10 and theoretical computer scientists often use base 2. Because computer graphics borrows technology from many fields we will avoid this shorthand.

The derivatives of logarithms and exponents illuminate why the natural logarithm is “natural”:

$$\begin{aligned} \frac{d}{dx} \log_a x &= \frac{1}{x \ln a}; \\ \frac{d}{dx} a^x &= a^x \ln a. \end{aligned}$$

The constant multipliers above are unity only for $a = e$.



间隔歌剧在[3 5]和[4 6]上。

减号用于差运算符，它返回左侧区间中也不在右侧的点。例如 $[3 5) - [4 6] = [3 4)$ 和 $[4 6] - [3 5) = [5 6]$ 。这些操作特别容易使用间隔图可视化（图2.4）。

2.1.3 Logarithms

虽然今天不像计算器之前那样普遍，但对数在出现指数项方程的问题中通常很有用。根据定义，每个对数都有一个底*a*。X的“logbase*a*”被写入log*ax*，并被定义为“*a*必须被提升以获得*x*的指数”，即

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

请注意，对数底*a*和将*a*提升为幂的函数是相互颠倒的。这个基本定义有几个后果：

$$\begin{aligned} a^{\log_a(x)} &= x; \\ \log_a(a^x) &= x; \\ \log_a(xy) &= \log_a x + \log_a y; \\ \log_a(x/y) &= \log_a x - \log_a y; \\ \log_a x &= \log_b b \log_b x. \end{aligned}$$

当我们将微积分应用于对数时，特殊数e=2.718...经常出现。以e为底的对数称为自然对数。我们采用常用的速记ln来表示：

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

请注意，“ \equiv ”符号可以被读取“根据定义是等价的。”像 π 一样，特殊数字e在大量的上下文中出现。除了e之外，许多字段还使用particularbase进行操作，并在其符号中省略该base，即log*x*。例如，天文学家经常使用基数10，理论计算机科学家经常使用基数2。由于计算机图形学借鉴了许多领域的技术，我们将避免这种速记。

对数和指数的导数阐明了为什么自然logarithm是“自然的”：

$$\begin{aligned} \frac{d}{dx} \log_a x &= \frac{1}{x \ln a}; \\ \frac{d}{dx} a^x &= a^x \ln a. \end{aligned}$$

上面的常数乘数仅对于a=e是统一的。



2.2 Solving Quadratic Equations

A *quadratic equation* has the form

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

where x is a real unknown, and A , B , and C are known constants. If you think of a 2D xy plot with $y = Ax^2 + Bx + C$, the solution is just whatever x values are “zero crossings” in y . Because $y = Ax^2 + Bx + C$ is a parabola, there will be zero, one, or two real solutions depending on whether the parabola misses, grazes, or hits the x -axis (Figure 2.5).

To solve the quadratic equation analytically, we first divide by A :

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

Then we “complete the square” to group terms:

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

Moving the constant portion to the right-hand side and taking the square root gives

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

Subtracting $B/(2A)$ from both sides and grouping terms with the denominator $2A$ gives the familiar form:¹

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}. \quad (2.1)$$

Here the “ \pm ” symbol means there are two solutions, one with a plus sign and one with a minus sign. Thus 3 ± 1 equals “two or four.” Note that the term that determines the number of real solutions is

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

which is called the *discriminant* of the quadratic equation. If $D > 0$, there are two real solutions (also called *roots*). If $D = 0$, there is one real solution (a “double” root). If $D < 0$, there are no real solutions.

For example, the roots of $2x^2 + 6x + 4 = 0$ are $x = -1$ and $x = -2$, and the equation $x^2 + x + 1$ has no real solutions. The discriminants of these equations are $D = 4$ and $D = -3$, respectively, so we expect the number of solutions given. In programs, it is usually a good idea to evaluate D first and return “no roots” without taking the square root if D is negative.

¹A robust implementation will use the equivalent expression $2C/(-B \mp \sqrt{B^2 - 4AC})$ to compute one of the roots, depending on the sign of B (Exercise 7).

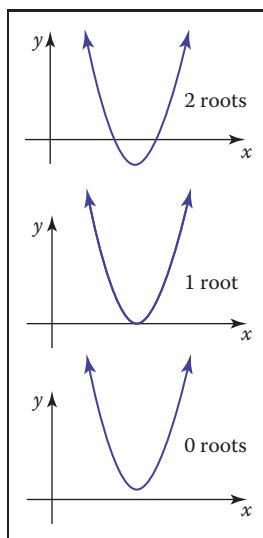


Figure 2.5. The geometric interpretation of the roots of a quadratic equation is the intersection points of a parabola with the x -axis.



2.2 求解二次方程

二次方程的形式

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

其中 x 是一个真正的未知， A ， B 和 C 是已知的常量。如果您认为 $Y=Ax^2+Bx+C$ 的二维 xy 图，则解决方案就是 y 中 x 值的“零交叉”。由于 $y=Ax^2+Bx+C$ 是抛物线，因此根据抛物线是否错过，擦伤或撞击 x 轴（图2.5），将有零个，一个或两个实解。

为了解析地求解二次方程，我们首先除以：

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

然后我们“完成广场”以分组术语：

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

将常量部分移动到右侧并取平方根给出

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

从两侧减去 $B/(2A)$ 并用分母 $2a$ 分组术语给出了熟悉的形式：1

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}. \quad (2.1)$$

这里的“ \pm ”符号意味着有两个解决方案，一个带有加号，一个带有减号。因此 3 ± 1 等于“二或四。”请注意，确定实际解决方案数量的术语是

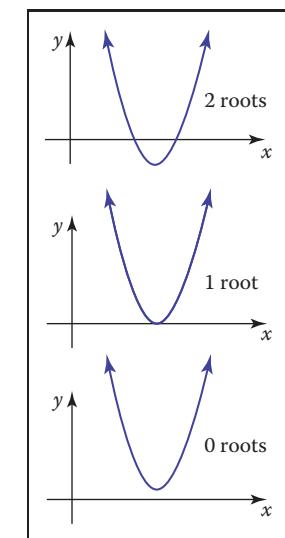
$$D \equiv B^2 - 4AC,$$

这被称为二次方程的判别式。如果 $D>0$ ，则有两个真正的解决方案（也称为根）。如果 $D=0$ ，则有一个真正的解决方案（“双”根）。如果 $D<0$ ，则没有真正的解决方案。

例如， $2x^2+6x+4=0$ 的根是 $x= -1$ 和 $x= -2$ ，方程 x^2+x+1 没有实解。这些方程的判别式分别为 $D=4$ 和 $D= -3$ ，因此我们期望给出的解的数量。在程序中，如果 D 为负，通常最好先评估 D 并返回“无根”而不取平方根。

一个健壮的实现将使用等价的表达式 $2c(-B\sqrt{B^2 - 4AC})$ 至 com

根据 b 的符号（练习7）将其中一个根捣碎。



二次方程根的几何解释是抛物线与 x 轴的相交点。

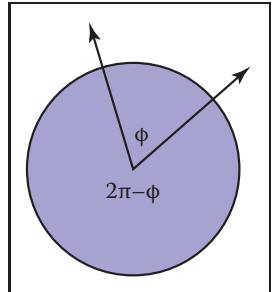


Figure 2.6. Two half-lines cut the unit circle into two arcs. The length of either arc is a valid angle “between” the two half-lines. Either we can use the convention that the smaller length is the angle, or that the two half-lines are specified in a certain order and the arc that determines angle ϕ is the one swept out counterclockwise from the first to the second half-line.

$$\text{degrees} = \frac{180}{\pi} \text{ radians};$$

$$\text{radians} = \frac{\pi}{180} \text{ degrees}.$$

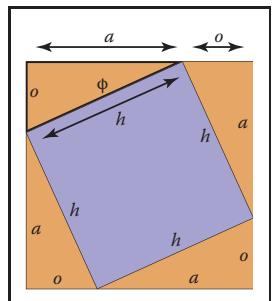


Figure 2.7. A geometric demonstration of the Pythagorean theorem.

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

You can see that this is true from Figure 2.7, where the big square has area $(a+o)^2$, the four triangles have the combined area $2ao$, and the center square has area h^2 .

Because the triangles and inner square subdivide the larger square evenly, we have $2ao + h^2 = (a+o)^2$, which is easily manipulated to the form above.

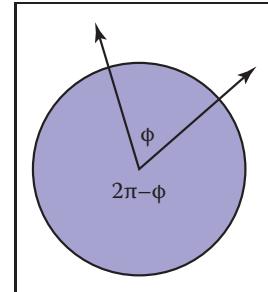
2.3 Trigonometry

In graphics we use basic trigonometry in many contexts. Usually, it is nothing too fancy, and it often helps to remember the basic definitions.

2.3.1 Angles

Although we take angles somewhat for granted, we should return to their definition so we can extend the idea of the angle onto the sphere. An angle is formed between two half-lines (infinite rays stemming from an origin) or directions, and some convention must be used to decide between the two possibilities for the angle created between them as shown in Figure 2.6. An *angle* is defined by the length of the arc segment it cuts out on the unit circle. A common convention is that the smaller arc length is used, and the sign of the angle is determined by the order in which the two half-lines are specified. Using that convention, all angles are in the range $[-\pi, \pi]$.

Each of these angles is the *length of the arc of the unit circle that is “cut” by the two directions*. Because the perimeter of the unit circle is 2π , the two possible angles sum to 2π . The unit of these arc lengths is *radians*. Another common unit is *degrees*, where the perimeter of the circle is 360 degrees. Thus, an angle that is π radians is 180 degrees, usually denoted 180° . The conversion between degrees and radians is



两条半线将单位圆切割成两条弧。

弧的长度是两条半线的有效角度“补间”。Either我们可以使用的convention，即较小的长度是角度，或者两个半线以一定的顺序指定，并且阻止角度φ的弧是从第一个到第二个半线逆时针扫出的一个。

2.3 Trigonometry

在图形中，我们在许多情况下使用基本三角学。通常情况下，它没有什么太花哨，它往往有助于记住基本的定义。

2.3.1 Angles

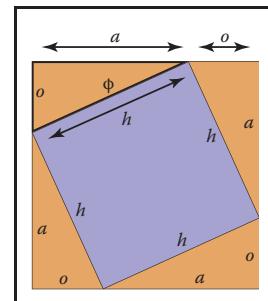
虽然我们在某种程度上认为角度是理所当然的，但我们应该回到它们的定义，这样我们就可以将角度的概念扩展到球体上。两条半线（源自原点的无限射线）或方向之间形成一个角度，并且必须使用一些约定来决定它们之间产生的角度的两种可能性，如图2.6所示。一个角度是由它在单位圆上切出的弧段的长度定义的。一个常见的惯例是使用较小的弧长，并且角度的符号由

两个半线被指定的顺序。使用该约定，所有角度都在 $[\pi, \pi]$ 范围内。

这些角度中的每一个都是被两个方向“切割”的单位圆的弧的长度。因为单位圆的周长是 2π ，所以两个可能的角度总和为 2π 。这些弧长的单位是弧度。另一个常见的单位是度，其中圆的周长是360度。因此， π 弧度的角度为180度，通常表示 180° 。度和弧度之间的转换为

$$\text{degrees} = \frac{180}{\pi} \text{ radians};$$

$$\text{radians} = \frac{\pi}{180} \text{ degrees}.$$



勾股定理的地理度量演示。

2.3.2 Trigonometric Functions

给定一个直角三角形，边长为 a ， o 和 h ，其中 h 是最长边的长度（总是与直角相对）或斜边，一个重要的关系由勾股定理描述：

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

你可以从图2.7中看到这是真的，其中大广场有面积 $(a+o)^2$ 四个三角形具有组合面积 $2ao$ ，中心正方形具有面积 h^2 。因为三角形和内正方形均匀地细分了较大的正方形，所以我们有 $2ao + h^2 = (a+o)^2$ ，这很容易被操纵到上面的形式。



We define *sine* and *cosine* of ϕ , as well as the other ratio-based trigonometric expressions:

$$\begin{aligned}\sin \phi &\equiv o/h; \\ \csc \phi &\equiv h/o; \\ \cos \phi &\equiv a/h; \\ \sec \phi &\equiv h/a; \\ \tan \phi &\equiv o/a; \\ \cot \phi &\equiv a/o.\end{aligned}$$

These definitions allow us to set up *polar coordinates*, where a point is coded as a distance from the origin and a signed angle relative to the positive x -axis (Figure 2.8). Note the convention that angles are in the range $\phi \in (-\pi, \pi]$, and that the positive angles are counterclockwise from the positive x -axis. This convention that counterclockwise maps to positive numbers is arbitrary, but it is used in many contexts in graphics so it is worth committing to memory.

Trigonometric functions are periodic and can take any angle as an argument. For example, $\sin(A) = \sin(A + 2\pi)$. This means the functions are not invertible when considered with the domain \mathbb{R} . This problem is avoided by restricting the range of standard inverse functions, and this is done in a standard way in almost all modern math libraries (e.g., (Plauger, 1991)). The domains and ranges are:

$$\begin{aligned}\text{asin} : [-1, 1] &\mapsto [-\pi/2, \pi/2]; \\ \text{acos} : [-1, 1] &\mapsto [0, \pi]; \\ \text{atan} : \mathbb{R} &\mapsto [-\pi/2, \pi/2]; \\ \text{atan2} : \mathbb{R}^2 &\mapsto [-\pi, \pi].\end{aligned}\tag{2.2}$$

The last function, $\text{atan2}(s, c)$ is often very useful. It takes an s value proportional to $\sin A$ and a c value that scales $\cos A$ by the same factor and returns A . The factor is assumed to be positive. One way to think of this is that it returns the angle of a 2D Cartesian point (s, c) in polar coordinates (Figure 2.9).

2.3.3 Useful Identities

This section lists without derivation a variety of useful trigonometric identities.

Shifting identities:

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

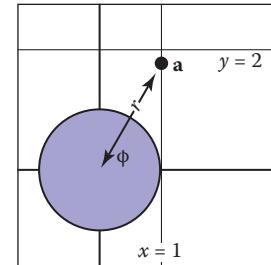


Figure 2.8. Polar coordinates for the point $(x_a, y_a) = (1, \sqrt{3})$ is $(r_a, \phi_a) = (2, \pi/3)$.

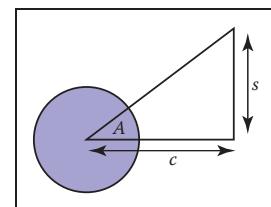


Figure 2.9. The function $\text{atan2}(s, c)$ returns the angle A and is often very useful in graphics.

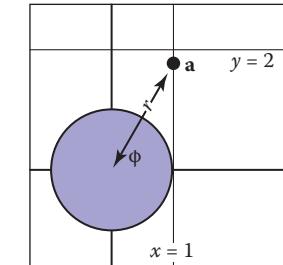


我们定义了 ϕ 的正弦和余弦，以及其他基于比率的三角表达式：

$$\begin{aligned}\csc \phi &\equiv h/o; \\ \cos \phi &\equiv a/h; \\ \sec \phi &\equiv h/a; \\ \tan \phi &\equiv o/a; \\ \cot \phi &\equiv a/o.\end{aligned}$$

这些定义允许我们设置极坐标，其中一个点被编码为与原点的距离和相对于正x轴的带符号角度（图2.8）。请注意角度在 $\phi \in [\pi, \pi]$ 范围内的约定，以及

正角是从正x轴逆时针方向的逆时针映射到正数的这个convention是任意的，但它在图形中的许多上下文中使用，因此值得提交到内存中。



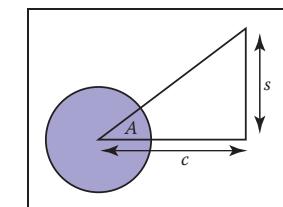
点的极坐标 $(x_a, y_a) = (1, \sqrt{3})$

三角函数是周期性的，可以采用任何角度作为参数。

例如， $\sin(A) = \sin(A + 2\pi)$ 。这意味着当与域 \mathbb{R} 一起考虑时，函数是不可反转的。通过限制标准反函数的范围来避免这个问题，并且这在几乎所有现代数学库中都以标准方式完成（例如（Plauger, 1991））。域和范围是：

$$\begin{aligned}\text{asin} : [-1, 1] &\mapsto [-\pi/2, \pi/2]; \\ \text{acos} : [-1, 1] &\mapsto [0, \pi]; \\ \text{atan} : \mathbb{R} &\mapsto [-\pi/2, \pi/2]; \\ \text{atan2} : \mathbb{R}^2 &\mapsto [-\pi, \pi].\end{aligned}\tag{2.2}$$

最后一个函数， $\text{atan2}(s, c)$ 通常非常有用。它按比例取一个 s 值将 a 和 c 值按相同的因子缩放 $\cos A$ 并返回 A 。该因子被假定为正数。想到这一点的一种方法是它返回极坐标中的2D笛卡尔点 (s, c) 的角度（图2.9）。



Function $\text{atan2}(s, c)$ 返回 angle A 并且通常在图形中非常有用。

2.3.3 有用的身份

本节在没有推导的情况下列出了各种有用的三角恒等式。

Shifting identities:

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

Pythagorean identities:

$$\sin^2 A + \cos^2 A = 1$$

$$\sec^2 A - \tan^2 A = 1$$

$$\csc^2 A - \cot^2 A = 1$$

Addition and subtraction identities:

$$\sin(A + B) = \sin A \cos B + \sin B \cos A$$

$$\sin(A - B) = \sin A \cos B - \sin B \cos A$$

$$\sin(2A) = 2 \sin A \cos A$$

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

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

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

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

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

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

Half-angle identities:

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

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

Product identities:

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

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

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

The following identities are for arbitrary triangles with side lengths a , b , and c , each with an angle opposite it given by A , B , C , respectively (Figure 2.10):

$$\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c} \quad (\text{Law of sines})$$

$$c^2 = a^2 + b^2 - 2ab \cos C \quad (\text{Law of cosines})$$

$$\frac{a+b}{a-b} = \frac{\tan(\frac{A+B}{2})}{\tan(\frac{A-B}{2})} \quad (\text{Law of tangents})$$

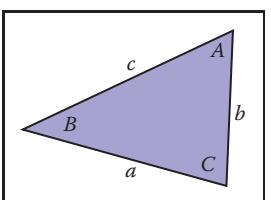


Figure 2.10. Geometry for triangle laws.

$$\text{triangle area} = \frac{1}{4} \sqrt{(a+b+c)(-a+b+c)(a-b+c)(a+b-c)}.$$

Pythagorean identities:

$$\sin^2 A + \cos^2 A = 1$$

$$\sec^2 A - \tan^2 A = 1$$

—

加法和减法身份:

$$\sin(A+B) = \sin A \cos B + \sin B \cos A$$

$$\sin(A-B) = \sin A \cos B - \sin B \cos A$$

$$\sin(2A) = 2 \sin A \cos A$$

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

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

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

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

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

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

Half-angle identities:

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

Product identities:

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

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

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

以下身份是针对边长为 a , b 和 c 的任意三角形, 每个三角形的角度分别由 A , B , C 给出 (图2.10) :

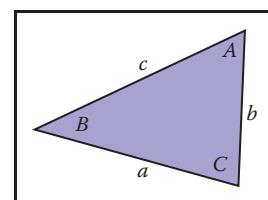


Figure 2.10. 几何形状
三角定律。

$$\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c} \quad (\text{Law of sines})$$

$$c^2 = a^2 + b^2 - 2ab \cos C \quad (\text{Law of cosines})$$

$$\frac{a+b}{a-b} = \frac{\tan(\frac{A+B}{2})}{\tan(\frac{A-B}{2})} \quad (\text{Law of tangents})$$

三角形的面积也可以根据这些边长来计算:

$$\text{三角形面积} = \frac{1}{4} \sqrt{(a+b+c)(-a+b+c)(a-b+c)(a+b-c)}.$$



2.4 Vectors

A *vector* describes a length and a direction. It can be usefully represented by an arrow. Two vectors are equal if they have the same length and direction even if we think of them as being located in different places (Figure 2.11). As much as possible, you should think of a vector as an arrow and not as coordinates or numbers. At some point we will have to represent vectors as numbers in our programs, but even in code they should be manipulated as objects and only the low-level vector operations should know about their numeric representation (DeRose, 1989). Vectors will be represented as bold characters, e.g., \mathbf{a} . A vector's length is denoted $\|\mathbf{a}\|$. A *unit vector* is any vector whose length is one. The *zero vector* is the vector of zero length. The direction of the zero vector is undefined.

Vectors can be used to represent many different things. For example, they can be used to store an *offset*, also called a *displacement*. If we know “the treasure is buried two paces east and three paces north of the secret meeting place,” then we know the offset, but we don’t know where to start. Vectors can also be used to store a *location*, another word for *position* or *point*. Locations can be represented as a displacement from another location. Usually there is some understood *origin* location from which all other locations are stored as offsets. Note that locations are not vectors. As we shall discuss, you can add two vectors. However, it usually does not make sense to add two locations unless it is an intermediate operation when computing weighted averages of a location (Goldman, 1985). Adding two offsets does make sense, so that is one reason why offsets are vectors. But this emphasizes that a location is not an offset; it is an offset from a specific origin location. The offset by itself is not the location.

2.4.1 Vector Operations

Vectors have most of the usual arithmetic operations that we associate with real numbers. Two vectors are equal if and only if they have the same length and direction. Two vectors are added according to the *parallelogram rule*. This rule states that the sum of two vectors is found by placing the tail of either vector against the head of the other (Figure 2.12). The sum vector is the vector that “completes the triangle” started by the two vectors. The parallelogram is formed by taking the sum in either order. This emphasizes that vector addition is commutative:

$$\mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a}.$$

Note that the parallelogram rule just formalizes our intuition about displacements. Think of walking along one vector, tail to head, and then walking along the other.

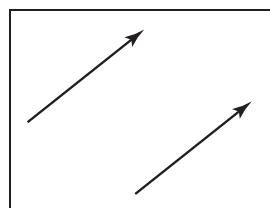


Figure 2.11. These two vectors are the same because they have the same length and direction.

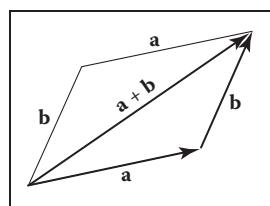


Figure 2.12. Two vectors are added by arranging them head to tail. This can be done in either order.



2.4 Vectors

矢量描述长度和方向。它可以有用地用箭头表示。如果两个向量具有相同的长度和方向，即使我们认为它们位于不同的地方，它们也是相等的（图2.11）。尽可能地，您应该将矢量视为箭头，而不是坐标或数字。在某些时候，我们将不得不在我们的程序中将向量表示为数字，但即使在代码中，它们也应该作为对象进行操作，只有低级向量操作应该知道它们的数字表示（DeRose, 1989）。向量将表示为粗体字符，例如 \mathbf{a} 。向量的长度表示 $\|\mathbf{a}\|$ 。单位向量是长度为1的任何向量。零向量是零长度的向量。零向量的方向未定义。

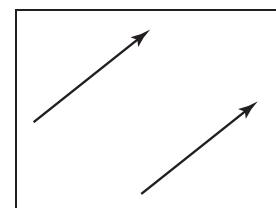


图2.11。这两个向量是相同的，因为它们具有相同的长度和方向。

向量可以用来表示许多不同的东西。例如，它们可用于存储偏移量，也称为位移。如果我们知道“宝藏埋在秘密聚会地点东两步，北三步”，那么我们知道偏移量，但我们不知道从哪里开始。向量也可以用来存储一个位置，另一个词的位置或点。位置可以表示为从另一个位置的位移。通常有一些理解的原点位置，所有其他位置都存储为偏移量。请注意，位置不是向量。正如我们将要讨论的，您可以添加两个向量。但是，添加两个位置通常没有意义，除非它是计算一个位置的加权平均值时的中间操作（Goldman, 1985）。添加两个偏移确实是有趣的，所以这就是偏移量是向量的一个原因。但这强调了一个位置不是偏移量；它是从特定原点位置的偏移量。偏移本身不是位置。

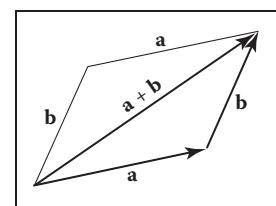
2.4.1 向量运算

向量具有我们与实数相关联的大多数常用算术运算。当且仅当它们具有相同的长度和方向时，两个矢量是相等的。根据平行四边形规则将两个向量相加。这条规则规定

两个向量的总和是通过将任一向量的尾部放在另一个向量的头部来找到的（图2.12）。总和向量是由两个向量开始的“完成三角形”的向量。平行四边形是通过以任一顺序取和而形成的。这强调向量加法是交换的：

$$\mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a}.$$

请注意，平行四边形规则只是形式化了我们关于位移的直觉。想想沿着一个向量走，尾巴到头，然后沿着另一个向量走。



两个向量通过将它们从头到尾排列来添加。这可以按任一顺序完成。

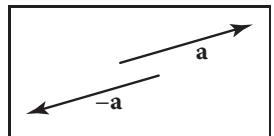


Figure 2.13. The vector $\mathbf{-a}$ has the same length but opposite direction of the vector \mathbf{a} .

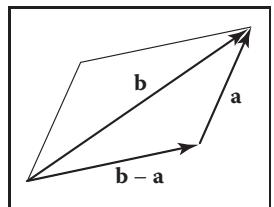


Figure 2.14. Vector subtraction is just vector addition with a reversal of the second argument.

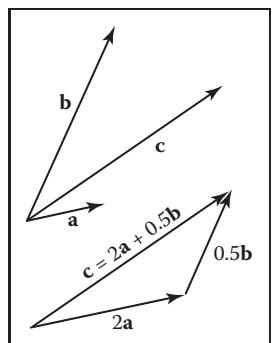


Figure 2.15. Any 2D vector \mathbf{c} is a weighted sum of any two nonparallel 2D vectors \mathbf{a} and \mathbf{b} .

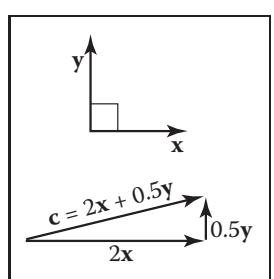


Figure 2.16. A 2D Cartesian basis for vectors.

$$\mathbf{b} - \mathbf{a} \equiv -\mathbf{a} + \mathbf{b}.$$

You can visualize vector subtraction with a parallelogram (Figure 2.14). We can write

$$\mathbf{a} + (\mathbf{b} - \mathbf{a}) = \mathbf{b}.$$

Vectors can also be multiplied. In fact, there are several kinds of products involving vectors. First, we can *scale* the vector by multiplying it by a real number k . This just multiplies the vector's length without changing its direction. For example, $3.5\mathbf{a}$ is a vector in the same direction as \mathbf{a} but it is 3.5 times as long as \mathbf{a} . We discuss two products involving two vectors, the dot product and the cross product, later in this section, and a product involving three vectors, the determinant, in Chapter 5.

2.4.2 Cartesian Coordinates of a Vector

A 2D vector can be written as a combination of any two nonzero vectors which are not parallel. This property of the two vectors is called *linear independence*. Two linearly independent vectors form a 2D *basis*, and the vectors are thus referred to as *basis vectors*. For example, a vector \mathbf{c} may be expressed as a combination of two basis vectors \mathbf{a} and \mathbf{b} (Figure 2.15):

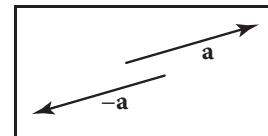
$$\mathbf{c} = a_c \mathbf{a} + b_c \mathbf{b}. \quad (2.3)$$

Note that the weights a_c and b_c are unique. Bases are especially useful if the two vectors are *orthogonal*, i.e., they are at right angles to each other. It is even more useful if they are also unit vectors in which case they are *orthonormal*. If we assume two such “special” vectors \mathbf{x} and \mathbf{y} are known to us, then we can use them to represent all other vectors in a *Cartesian coordinate system*, where each vector is represented as two real numbers. For example, a vector \mathbf{a} might be represented as

$$\mathbf{a} = x_a \mathbf{x} + y_a \mathbf{y},$$

where x_a and y_a are the real Cartesian coordinates of the 2D vector \mathbf{a} (Figure 2.16). Note that this is not really any different conceptually from Equation (2.3), where the basis vectors were not orthonormal. But there are several advantages to a Cartesian coordinate system. For instance, by the Pythagorean theorem, the length of \mathbf{a} is

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



矢量 $\mathbf{-a}$ 具有与矢量 \mathbf{a} 相同的长度但方向相反。

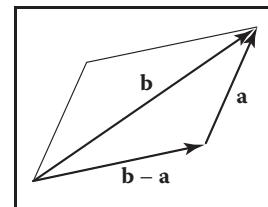
净位移只是平行四边形对角线。您还可以为矢量创建一元减号 $\mathbf{-a}$ (图2.13) 是与 \mathbf{a} 长度相同但方向相反的矢量。这允许我们也定义减法:

$$\mathbf{b} - \mathbf{a} \equiv -\mathbf{a} + \mathbf{b}.$$

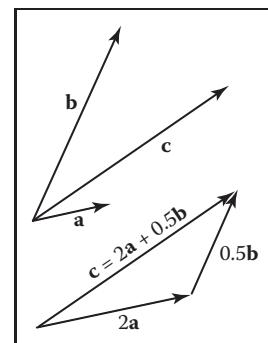
您可以使用平行四边形可视化矢量减法 (图2.14)。我们可以写

$$\mathbf{a} + (\mathbf{b} - \mathbf{a}) = \mathbf{b}.$$

向量也可以相乘。事实上，有几种产品。-ing 矢量。首先，我们可以通过将向量乘以实数 k 来缩放向量。这只是乘以向量的长度而不改变其方向。例如， $3.5\mathbf{a}$ 是与 \mathbf{a} 相同方向的矢量，但它是 \mathbf{a} 长的 3.5 倍。我们将在本节后面讨论涉及两个向量的两个乘积，即点积和叉积，以及涉及三个向量的乘积，即行列式，在第 5 章中。



矢量 \mathbf{a} 牵引只是矢量加法与第二个参数的反转。



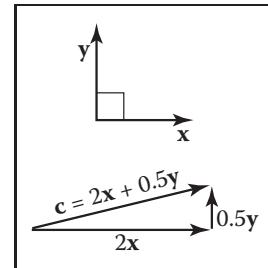
任何 2D 向量 \mathbf{c} 是任何两个非平行 2D 向量 \mathbf{a} 和 \mathbf{b} 的加权和。

2.4.2 矢量的笛卡尔坐标

一个 2D 向量可以写成任意两个不平行的非零向量的组合。两个向量的这种性质称为线性独立性。二 线性独立的向量形成 2D 基，并且向量因此被称为基向量。例如，向量 \mathbf{c} 可以表示为两个基向量 \mathbf{a} 和 \mathbf{b} 的组合 (图2.15) :

$$\mathbf{c} = a_c \mathbf{a} + b_c \mathbf{b}. \quad (2.3)$$

请注意，权重 a_c 和 b_c 是唯一的。如果两个矢量是正交的，即它们彼此成直角，则基特别有用。如果它们也是单位向量，则更有用，在这种情况下它们是正交的。如果我们假设我们知道两个这样的“特殊”向量 \mathbf{x} 和 \mathbf{y} ，那么我们可以用它们来表示笛卡尔坐标系中的所有其他向量，其中每个向量表示为两个实数。例如，矢量 \mathbf{a} 可能表示为



矢量的 2D 点菜ian 基础。

$$\mathbf{a} = x_a \mathbf{x} + y_a \mathbf{y},$$

其中 x_a 和 y_a 是 2D 矢量 \mathbf{a} 的真实笛卡尔坐标 (图2.16)。请注意，这在概念上与公式 (2.3) 没有任何不同，其中基向量不是正交的。但是有几个

笛卡尔坐标系的优点。例如，通过勾股定理， \mathbf{a} 的长度为

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



It is also simple to compute dot products, cross products, and coordinates of vectors in Cartesian systems, as we'll see in the following sections.

By convention we write the coordinates of \mathbf{a} either as an ordered pair (x_a, y_a) or a column matrix:

$$\mathbf{a} = \begin{bmatrix} x_a \\ y_a \end{bmatrix}.$$

The form we use will depend on typographic convenience. We will also occasionally write the vector as a row matrix, which we will indicate as \mathbf{a}^T :

$$\mathbf{a}^T = [x_a \ y_a].$$

We can also represent 3D, 4D, etc., vectors in Cartesian coordinates. For the 3D case, we use a basis vector \mathbf{z} that is orthogonal to both \mathbf{x} and \mathbf{y} .

2.4.3 Dot Product

The simplest way to multiply two vectors is the *dot product*. The dot product of \mathbf{a} and \mathbf{b} is denoted $\mathbf{a} \cdot \mathbf{b}$ and is often called the *scalar product* because it returns a scalar. The dot product returns a value related to its arguments' lengths and the angle ϕ between them (Figure 2.17):

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \phi, \quad (2.4)$$

The most common use of the dot product in graphics programs is to compute the cosine of the angle between two vectors.

The dot product can also be used to find the *projection* of one vector onto another. This is the length $\mathbf{a} \rightarrow \mathbf{b}$ of a vector \mathbf{a} that is projected at right angles onto a vector \mathbf{b} (Figure 2.18):

$$\mathbf{a} \rightarrow \mathbf{b} = \|\mathbf{a}\| \cos \phi = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{b}\|}. \quad (2.5)$$

The dot product obeys the familiar associative and distributive properties we have in real arithmetic:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= \mathbf{b} \cdot \mathbf{a}, \\ \mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) &= \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c}, \\ (k\mathbf{a}) \cdot \mathbf{b} &= \mathbf{a} \cdot (k\mathbf{b}) = k\mathbf{a} \cdot \mathbf{b}. \end{aligned} \quad (2.6)$$

If 2D vectors \mathbf{a} and \mathbf{b} are expressed in Cartesian coordinates, we can take advantage of $\mathbf{x} \cdot \mathbf{x} = \mathbf{y} \cdot \mathbf{y} = 1$ and $\mathbf{x} \cdot \mathbf{y} = 0$ to derive that their dot product



在笛卡尔系统中计算点积、叉积和vector坐标也很简单，我们将在下面的部分中看到。

按照惯例，我们将 \mathbf{a} 的坐标写成有序对 (x_a, y_a) 或列矩阵:

$$\mathbf{a} = \begin{bmatrix} x_a \\ y_a \end{bmatrix}.$$

我们使用的形式将取决于印刷的方便性。我们偶尔也会将向量写为行矩阵，我们将其表示为 T :

$$\mathbf{a}^T = [x_a \ y_a].$$

我们也可以表示3D, 4D等。, 笛卡尔坐标中的向量。对于3D情况，我们使用与 \mathbf{x} 和 \mathbf{y} 正交的基向量 \mathbf{z} 。

2.4.3 点积

将两个向量相乘的最简单方法是点积。A和b的点积表示为 $\mathbf{a} \cdot \mathbf{b}$ ，通常称为标量积，因为它返回标量。点积返回一个与其参数长度和它们之间的角度 ϕ 相关的值（图2.17）：

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \phi, \quad (2.4)$$

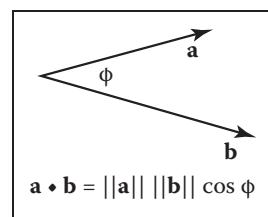


Figure 2.17. The dot product is related to length and angle and is one of the most important formulas in graphics.

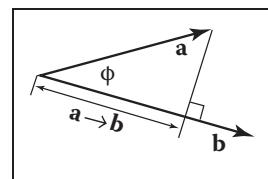


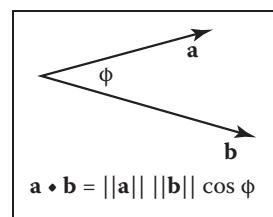
Figure 2.18. The projection of \mathbf{a} onto \mathbf{b} is a length found by Equation (2.5).

$$\mathbf{a} \rightarrow \mathbf{b} = \|\mathbf{a}\| \cos \phi = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{b}\|}. \quad (2.5)$$

点积服从我们在实际算术中熟悉的联想和分布属性:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= \mathbf{b} \cdot \mathbf{a}, \\ \mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) &= \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c}, \\ (k\mathbf{a}) \cdot \mathbf{b} &= \mathbf{a} \cdot (k\mathbf{b}) = k\mathbf{a} \cdot \mathbf{b}. \end{aligned} \quad (2.6)$$

如果2D向量 \mathbf{a} 和 \mathbf{b} 用笛卡尔坐标表示，我们可以利用 $x \cdot x = y \cdot y = 1$ 和 $x \cdot y = 0$ 推导出它们的点积



点product与长度和angle有关，是图形中最重要的公式之一。

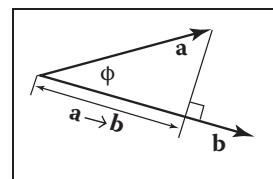


图2.18。A到b上的投影是由等式(2.5)找到的长度。

is

$$\begin{aligned}\mathbf{a} \cdot \mathbf{b} &= (x_a \mathbf{x} + y_a \mathbf{y}) \cdot (x_b \mathbf{x} + y_b \mathbf{y}) \\ &= x_a x_b (\mathbf{x} \cdot \mathbf{x}) + x_a y_b (\mathbf{x} \cdot \mathbf{y}) + x_b y_a (\mathbf{y} \cdot \mathbf{x}) + y_a y_b (\mathbf{y} \cdot \mathbf{y}) \\ &= x_a x_b + y_a y_b.\end{aligned}$$

Similarly in 3D we can find

$$\mathbf{a} \cdot \mathbf{b} = x_a x_b + y_a y_b + z_a z_b.$$

2.4.4 Cross Product

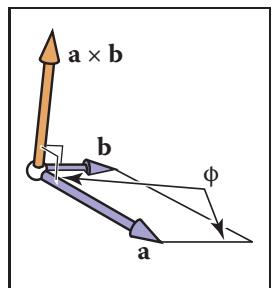


Figure 2.19. The cross product $\mathbf{a} \times \mathbf{b}$ is a 3D vector perpendicular to both 3D vectors \mathbf{a} and \mathbf{b} , and its length is equal to the area of the parallelogram shown.

The cross product $\mathbf{a} \times \mathbf{b}$ is usually used only for three-dimensional vectors; generalized cross products are discussed in references given in the chapter notes. The cross product returns a 3D vector that is perpendicular to the two arguments of the cross product. The length of the resulting vector is related to $\sin \phi$:

$$\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\| \|\mathbf{b}\| \sin \phi.$$

The magnitude $\|\mathbf{a} \times \mathbf{b}\|$ is equal to the area of the parallelogram formed by vectors \mathbf{a} and \mathbf{b} . In addition, $\mathbf{a} \times \mathbf{b}$ is perpendicular to both \mathbf{a} and \mathbf{b} (Figure 2.19). Note that there are only two possible directions for such a vector. By definition, the vectors in the direction of the x -, y - and z -axes are given by

$$\begin{aligned}\mathbf{x} &= (1, 0, 0), \\ \mathbf{y} &= (0, 1, 0), \\ \mathbf{z} &= (0, 0, 1),\end{aligned}$$

and we set as a convention that $\mathbf{x} \times \mathbf{y}$ must be in the plus or minus \mathbf{z} direction. The choice is somewhat arbitrary, but it is standard to assume that

$$\mathbf{z} = \mathbf{x} \times \mathbf{y}.$$

All possible permutations of the three Cartesian unit vectors are

$$\begin{aligned}\mathbf{x} \times \mathbf{y} &= +\mathbf{z}, \\ \mathbf{y} \times \mathbf{x} &= -\mathbf{z}, \\ \mathbf{y} \times \mathbf{z} &= +\mathbf{x}, \\ \mathbf{z} \times \mathbf{y} &= -\mathbf{x}, \\ \mathbf{z} \times \mathbf{x} &= +\mathbf{y}, \\ \mathbf{x} \times \mathbf{z} &= -\mathbf{y}.\end{aligned}$$

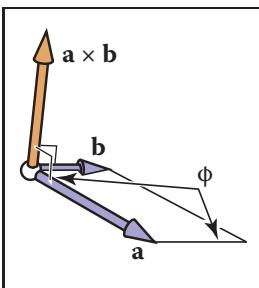
is

$$\begin{aligned}\mathbf{a} \cdot \mathbf{b} &= (x_a \mathbf{x} + y_a \mathbf{y}) \cdot (x_b \mathbf{x} + y_b \mathbf{y}) \\ &= x_a x_b (\mathbf{x} \cdot \mathbf{x}) + x_a y_b (\mathbf{x} \cdot \mathbf{y}) + x_b y_a (\mathbf{y} \cdot \mathbf{x}) + y_a y_b (\mathbf{y} \cdot \mathbf{y}) \\ &= x_a x_b + y_a y_b.\end{aligned}$$

同样在3D中我们可以找到

$$\mathbf{a} \cdot \mathbf{b} = x_a x_b + y_a y_b + z_a z_b.$$

2.4.4 交叉产品



叉积 $\mathbf{a} \times \mathbf{b}$ 是垂直于3D向量 \mathbf{a} 和 \mathbf{b} 的3D向量，其长度等于所示平行四边形的面积。

交叉乘积 $\mathbf{a} \times \mathbf{b}$ 通常仅用于三维向量；广义交叉乘积在章节注释中给出的参考文献中讨论。叉积返回垂直于叉积的两个参数的3d向量。所得向量的长度与 $\sin \phi$ 相关：

$$\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\| \|\mathbf{b}\| \sin \phi.$$

量值 $\triangle \mathbf{a} \times \mathbf{b} \triangle$ 等于矢量 \mathbf{a} 和 \mathbf{b} 形成的平行四边形的面积。此外， $\mathbf{a} \times \mathbf{b}$ 垂直于 \mathbf{a} 和 \mathbf{b} （图2.19）。注

这样的矢量只有两个可能的方向。根据定义， x ， y 和 z axes方向的矢量由下式给出

$$\begin{aligned}\mathbf{x} &= (1, 0, 0), \\ \mathbf{y} &= (0, 1, 0), \\ \mathbf{z} &= (0, 0, 1),\end{aligned}$$

我们设定一个约定， $\mathbf{x} \times \mathbf{y}$ 必须在正负 \mathbf{z} 方向上。选择有点武断，但它是标准的假设，

$$\mathbf{z} = \mathbf{x} \times \mathbf{y}.$$

三个笛卡尔单位向量的所有可能排列是

$$\begin{aligned}\mathbf{x} \times \mathbf{y} &= +\mathbf{z}, \\ \mathbf{y} \times \mathbf{x} &= -\mathbf{z}, \\ \mathbf{y} \times \mathbf{z} &= +\mathbf{x}, \\ \mathbf{z} \times \mathbf{y} &= -\mathbf{x}, \\ \mathbf{z} \times \mathbf{x} &= +\mathbf{y}, \\ \mathbf{x} \times \mathbf{z} &= -\mathbf{y}.\end{aligned}$$

Because of the $\sin \phi$ property, we also know that a vector cross itself is the zero-vector, so $\mathbf{x} \times \mathbf{x} = \mathbf{0}$ and so on. Note that the cross product is *not* commutative, i.e., $\mathbf{x} \times \mathbf{y} \neq \mathbf{y} \times \mathbf{x}$. The careful observer will note that the above discussion does not allow us to draw an unambiguous picture of how the Cartesian axes relate. More specifically, if we put \mathbf{x} and \mathbf{y} on a sidewalk, with \mathbf{x} pointing east and \mathbf{y} pointing north, then does \mathbf{z} point up to the sky or into the ground? The usual convention is to have \mathbf{z} point to the sky. This is known as a *right-handed* coordinate system. This name comes from the memory scheme of “grabbing” \mathbf{x} with your *right* palm and fingers and rotating it toward \mathbf{y} . The vector \mathbf{z} should align with your thumb. This is illustrated in Figure 2.20.

The cross product has the nice property that

$$\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c},$$

and

$$\mathbf{a} \times (k\mathbf{b}) = k(\mathbf{a} \times \mathbf{b}).$$

However, a consequence of the right-hand rule is

$$\mathbf{a} \times \mathbf{b} = -(\mathbf{b} \times \mathbf{a}).$$

In Cartesian coordinates, we can use an explicit expansion to compute the cross product:

$$\begin{aligned} \mathbf{a} \times \mathbf{b} &= (x_a \mathbf{x} + y_a \mathbf{y} + z_a \mathbf{z}) \times (x_b \mathbf{x} + y_b \mathbf{y} + z_b \mathbf{z}) \\ &= x_a x_b \mathbf{x} \times \mathbf{x} + x_a y_b \mathbf{x} \times \mathbf{y} + x_a z_b \mathbf{x} \times \mathbf{z} \\ &\quad + y_a x_b \mathbf{y} \times \mathbf{x} + y_a y_b \mathbf{y} \times \mathbf{y} + y_a z_b \mathbf{y} \times \mathbf{z} \\ &\quad + z_a x_b \mathbf{z} \times \mathbf{x} + z_a y_b \mathbf{z} \times \mathbf{y} + z_a z_b \mathbf{z} \times \mathbf{z} \\ &= (y_a z_b - z_a y_b) \mathbf{x} + (z_a x_b - x_a z_b) \mathbf{y} + (x_a y_b - y_a x_b) \mathbf{z}. \end{aligned} \quad (2.7)$$

So, in coordinate form,

$$\mathbf{a} \times \mathbf{b} = (y_a z_b - z_a y_b, z_a x_b - x_a z_b, x_a y_b - y_a x_b). \quad (2.8)$$

2.4.5 Orthonormal Bases and Coordinate Frames

Managing coordinate systems is one of the core tasks of almost any graphics program; the key to this is managing *orthonormal bases*. Any set of two 2D vectors \mathbf{u} and \mathbf{v} form an orthonormal basis provided that they are orthogonal (at right angles) and are each of unit length. Thus,

$$\|\mathbf{u}\| = \|\mathbf{v}\| = 1,$$

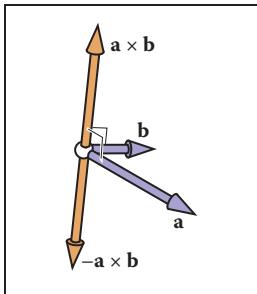


Figure 2.20. The “right-hand rule” for cross products. Imagine placing the base of your right palm where \mathbf{a} and \mathbf{b} join at their tails, and pushing the arrow of \mathbf{a} toward \mathbf{b} . Your extended right thumb should point toward $\mathbf{a} \times \mathbf{b}$.

由于 $\sin \phi$ 属性，我们也知道一个向量交叉本身就是零向量，所以 $\mathbf{x} \times \mathbf{x} = \mathbf{0}$ 等等。请注意，叉积不是交换的，即 $\mathbf{x} \times \mathbf{y} = \mathbf{y} \times \mathbf{x}$ 。细心的观察者会注意到，上面的讨论不允许我们明确地描绘笛卡尔轴的关系。更具体地说，如果我们把 x 和 y 放在人行道上， x 指向东方， y 指向北方，那么 z 是指向天空还是指向地面？通常的惯例是让 z 指向天空。这被称为右手坐标系。这个名字来自用右手掌和手指“抓住” x 并向 y 旋转的记忆方案。矢量 z 应与拇指对齐。这在图 2.20 中说明。

交叉产品具有很好的特性，

$$\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c},$$

and

$$\mathbf{a} \times (k\mathbf{b}) = k(\mathbf{a} \times \mathbf{b}).$$

然而，右手法则的结果是

$$\mathbf{a} \times \mathbf{b} = -(\mathbf{b} \times \mathbf{a}).$$

在笛卡尔坐标中，我们可以使用显式展开来计算交叉积：

$$\begin{aligned} \mathbf{a} \times \mathbf{b} &= (x_a \mathbf{x} + y_a \mathbf{y} + z_a \mathbf{z}) \times (x_b \mathbf{x} + y_b \mathbf{y} + z_b \mathbf{z}) \\ &= x_a x_b \mathbf{x} \times \mathbf{x} + x_a y_b \mathbf{x} \times \mathbf{y} + x_a z_b \mathbf{x} \times \mathbf{z} \\ &\quad + y_a x_b \mathbf{y} \times \mathbf{x} + y_a y_b \mathbf{y} \times \mathbf{y} + y_a z_b \mathbf{y} \times \mathbf{z} \\ &\quad + z_a x_b \mathbf{z} \times \mathbf{x} + z_a y_b \mathbf{z} \times \mathbf{y} + z_a z_b \mathbf{z} \times \mathbf{z} \\ &= (y_a z_b - z_a y_b) \mathbf{x} + (z_a x_b - x_a z_b) \mathbf{y} + (x_a y_b - y_a x_b) \mathbf{z}. \end{aligned} \quad (2.7)$$

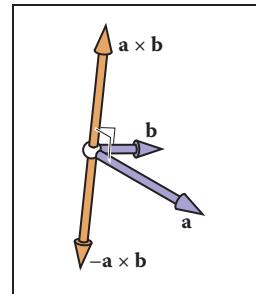
所以，以坐标形式

$$\mathbf{a} \times \mathbf{b} = (y_a z_b - z_a y_b, z_a x_b - x_a z_b, x_a y_b - y_a x_b). \quad (2.8)$$

2.4.5 正交基和坐标系

管理坐标系是几乎所有图形程序的核心任务之一；关键是管理正交基。任何一组两个二维向量 u 和 v 形成一个正交基础，前提是它们是正交的（直角）和每个单位长度。因此

$$\|\mathbf{u}\| = \|\mathbf{v}\| = 1,$$



交叉产品的“右手法则”。想象一下，把你的右手掌的底部放在 a 和 b 的尾巴上，然后把 a 的箭头推向 b 。你伸出的右手拇指应该指向 $a \times b$ 。

and

$$\mathbf{u} \cdot \mathbf{v} = 0.$$

In 3D, three vectors \mathbf{u} , \mathbf{v} , and \mathbf{w} form an orthonormal basis if

$$\|\mathbf{u}\| = \|\mathbf{v}\| = \|\mathbf{w}\| = 1,$$

and

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{w} = \mathbf{w} \cdot \mathbf{u} = 0.$$

This orthonormal basis is *right-handed* provided

$$\mathbf{w} = \mathbf{u} \times \mathbf{v},$$

and otherwise it is left-handed.

Note that the Cartesian canonical orthonormal basis is just one of infinitely many possible orthonormal bases. What makes it special is that it and its implicit origin location are used for low-level representation within a program. Thus, the vectors \mathbf{x} , \mathbf{y} , and \mathbf{z} are never explicitly stored and neither is the canonical origin location \mathbf{o} . The global model is typically stored in this canonical coordinate system, and it is thus often called the *global coordinate system*. However, if we want to use another coordinate system with origin \mathbf{p} and orthonormal basis vectors

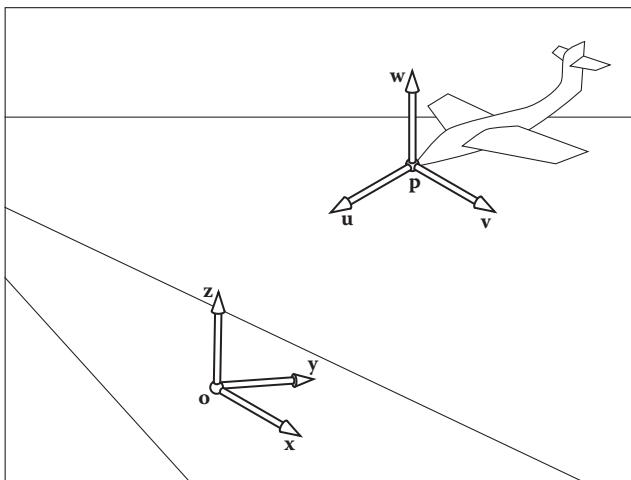


Figure 2.21. There is always a master or “canonical” coordinate system with origin \mathbf{o} and orthonormal basis \mathbf{x} , \mathbf{y} , and \mathbf{z} . This coordinate system is usually defined to be aligned to the global model and is thus often called the “global” or “world” coordinate system. This origin and basis vectors are never stored explicitly. All other vectors and locations are stored with coordinates that relate them to the global frame. The coordinate system associated with the plane are explicitly stored in terms of global coordinates.

and

$$\mathbf{u} \cdot \mathbf{v} = 0.$$

在3D中，三个向量 \mathbf{u} , \mathbf{v} 和 \mathbf{w} 形成正交基，如果

$$\|\mathbf{u}\| = \|\mathbf{v}\| = \|\mathbf{w}\| = 1,$$

and

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{w} = \mathbf{w} \cdot \mathbf{u} = 0.$$

这种正畸基础是右手提供的

$$\mathbf{w} = \mathbf{u} \times \mathbf{v},$$

否则它是左撇子。

请注意，笛卡尔规范正交基只是无限多可能的正交基中的一个。它的特殊之处在于它及其隐式原点位置用于程序中的低级表示。因此，向量 \mathbf{x} 、 \mathbf{y} 和 \mathbf{z} 从不被显式存储，规范原点位置也不是 \mathbf{o} 。全局模型通常存储在这个规范坐标系中，因此通常称为全局坐标系。但是，如果我们要使用另一个具有原点 \mathbf{p} 和正交基向量的坐标系

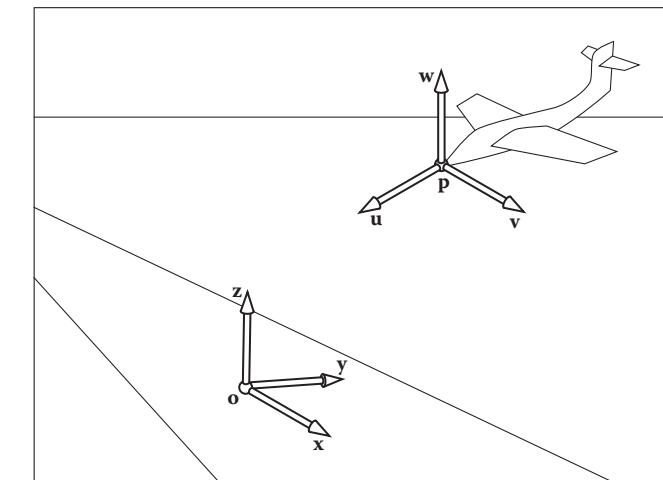


图2.21。总是有一个主坐标系或“规范”坐标系，其原点 \mathbf{o} 和正交基 \mathbf{x} ， \mathbf{y} 和 \mathbf{z} 。该坐标系通常被定义为与全局模型对齐，因此通常被称为“全局”或“世界”坐标系。此原点和基向量从不显式存储。所有其他矢量和位置都存储有与全局帧相关的坐标。与平面相关联的坐标系以全局坐标来明确地存储。



u, **v**, and **w**, then we *do* store those vectors explicitly. Such a system is called a *frame of reference* or *coordinate frame*. For example, in a flight simulator, we might want to maintain a coordinate system with the origin at the nose of the plane, and the orthonormal basis aligned with the airplane. Simultaneously, we would have the master canonical coordinate system (Figure 2.21). The coordinate system associated with a particular object, such as the plane, is usually called a *local coordinate system*.

At a low level, the local frame is stored in canonical coordinates. For example, if **u** has coordinates (x_u, y_u, z_u) ,

$$\mathbf{u} = x_u \mathbf{x} + y_u \mathbf{y} + z_u \mathbf{z}.$$

A location implicitly includes an offset from the canonical origin:

$$\mathbf{p} = \mathbf{o} + x_p \mathbf{x} + y_p \mathbf{y} + z_p \mathbf{z},$$

where (x_p, y_p, z_p) are the coordinates of **p**.

Note that if we store a vector **a** with respect to the **u-v-w** frame, we store a triple (u_a, v_a, w_a) which we can interpret geometrically as

$$\mathbf{a} = u_a \mathbf{u} + v_a \mathbf{v} + w_a \mathbf{w}.$$

To get the canonical coordinates of a vector **a** stored in the **u-v-w** coordinate system, simply recall that **u**, **v**, and **w** are themselves stored in terms of Cartesian coordinates, so the expression $u_a \mathbf{u} + v_a \mathbf{v} + w_a \mathbf{w}$ is already in Cartesian coordinates if evaluated explicitly. To get the **u-v-w** coordinates of a vector **b** stored in the canonical coordinate system, we can use dot products:

$$u_b = \mathbf{u} \cdot \mathbf{b}; \quad v_b = \mathbf{v} \cdot \mathbf{b}; \quad w_b = \mathbf{w} \cdot \mathbf{b}.$$

This works because we know that for *some* u_b , v_b , and w_b ,

$$u_b \mathbf{u} + v_b \mathbf{v} + w_b \mathbf{w} = \mathbf{b},$$

and the dot product isolates the u_b coordinate:

$$\begin{aligned} \mathbf{u} \cdot \mathbf{b} &= u_b(\mathbf{u} \cdot \mathbf{u}) + v_b(\mathbf{u} \cdot \mathbf{v}) + w_b(\mathbf{u} \cdot \mathbf{w}) \\ &= u_b. \end{aligned}$$

This works because **u**, **v**, and **w** are orthonormal.

Using matrices to manage changes of coordinate systems is discussed in Sections 6.2.1 and 6.5.



u, **v**和**w**, 然后我们明确地存储这些向量。这样的系统称为参照系或坐标系。例如, 在飞行模拟器中, 我们可能希望保持一个坐标系, 其原点位于飞机的机头, 正交基与飞机对齐。同时, 我们将拥有主规范坐标系(图2.21)。与特定对象(例如平面)相关联的坐标系通常称为局部坐标系。

在低级别, 局部帧被存储在规范坐标中。例如, 如果**u**具有坐标 (x_u, y_u, z_u)

$$\mathbf{u} = x_u \mathbf{x} + y_u \mathbf{y} + z_u \mathbf{z}.$$

位置隐式包含与规范原点的偏移量:

$$\mathbf{p} = \mathbf{o} + x_p \mathbf{x} + y_p \mathbf{y} + z_p \mathbf{z},$$

其中 (x_p, y_p, z_p) 是 **p** 的坐标。

请注意, 如果我们存储相对于uvw帧的矢量a, 我们存储一个三重 (u_a, v_a, w_a) , 我们可以将其几何解释为

$$\mathbf{a} = u_a \mathbf{u} + v_a \mathbf{v} + w_a \mathbf{w}.$$

要获得存储在uvw坐标系中的矢量a的规范坐标, 只需回想一下u, v和w本身以笛卡尔坐标存储, 因此如果显式计算, 表达式 $u_a \mathbf{u} + v_a \mathbf{v} + w_a \mathbf{w}$ 已要获得存储在规范坐标系中的矢量b的uvw坐标, 我们可以使用点积:

$$u_b = \mathbf{u} \cdot \mathbf{b}; \quad v_b = \mathbf{v} \cdot \mathbf{b}; \quad w_b = \mathbf{w} \cdot \mathbf{b}.$$

这是有效的, 因为我们知道对于一些 u_b , v_b 和 w_b

$$u_b \mathbf{u} + v_b \mathbf{v} + w_b \mathbf{w} = \mathbf{b},$$

和点积隔离 u_b 坐标:

$$\begin{aligned} \mathbf{u} \cdot \mathbf{b} &= u_b(\mathbf{u} \cdot \mathbf{u}) + v_b(\mathbf{u} \cdot \mathbf{v}) + w_b(\mathbf{u} \cdot \mathbf{w}) \\ &= u_b. \end{aligned}$$

这是有效的, 因为 **u**, **v** 和 **w** 是正交的。

在第6.2.1和6.5节中讨论了使用矩阵来管理坐标系的变化。

2.4.6 Constructing a Basis from a Single Vector

Often we need an orthonormal basis that is aligned with a given vector. That is, given a vector \mathbf{a} , we want an orthonormal \mathbf{u} , \mathbf{v} , and \mathbf{w} such that \mathbf{w} points in the same direction as \mathbf{a} (Hughes & Möller, 1999), but we don't particularly care what \mathbf{u} and \mathbf{v} are. One vector isn't enough to uniquely determine the answer; we just need a robust procedure that will find any one of the possible bases.

This can be done using cross products as follows. First make \mathbf{w} a unit vector in the direction of \mathbf{a} :

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

Then choose any vector \mathbf{t} not collinear with \mathbf{w} , and use the cross product to build a unit vector \mathbf{u} perpendicular to \mathbf{w} :

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

This same procedure can, of course, be used to construct the three vectors in any order; just pay attention to the order of the cross products to ensure the basis is right-handed.

If \mathbf{t} is collinear with \mathbf{w} the denominator will vanish, and if they are nearly collinear the results will have low precision. A simple procedure to find a vector sufficiently different from \mathbf{w} is to start with \mathbf{t} equal to \mathbf{w} and change the smallest magnitude component of \mathbf{t} to 1. For example, if $\mathbf{w} = (1/\sqrt{2}, -1/\sqrt{2}, 0)$ then $\mathbf{t} = (1/\sqrt{2}, -1/\sqrt{2}, 1)$. Once \mathbf{w} and \mathbf{u} are in hand, completing the basis is simple:

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}.$$

An example of a situation where this construction is used is surface shading, where a basis aligned to the surface normal is needed but the rotation around the normal is often unimportant.

2.4.7 Constructing a Basis from Two Vectors

The procedure in the previous section can also be used in situations where the rotation of the basis around the given vector is important. A common example is building a basis for a camera: it's important to have one vector aligned in the direction the camera is looking, but the orientation of the camera around that vector is *not* arbitrary, and it needs to be specified somehow. Once the orientation is pinned down, the basis is completely determined.

A common way to fully specify a frame is by providing two vectors \mathbf{a} (which specifies \mathbf{w}) and \mathbf{b} (which specifies \mathbf{v}). If the two vectors are known to be perpendicular it is a simple matter to construct the third vector by $\mathbf{u} = \mathbf{b} \times \mathbf{a}$.

$\mathbf{u} = \mathbf{a} \times \mathbf{b}$ also produces an orthonormal basis, but it is left-handed.

2.4.6 从单个向量构建基础

通常我们需要一个与给定向量对齐的正交基。也就是说，给定一个向量 \mathbf{a} ，我们想要一个正交的 \mathbf{u} , \mathbf{v} 和 \mathbf{w} ，使得 \mathbf{w} 指向与 \mathbf{a} 相同的方向 (Hughes & Möller, 1999)，但我们并不特别关心 \mathbf{u} 和 \mathbf{v} 是什么。一个向量不足以唯一确定答案；我们只需要一个稳健的过程，可以找到任何一个可能的基础。

这可以使用交叉产品来完成，如下所示。首先使 \mathbf{w} 成为 \mathbf{a} 方向的单位向量：

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

然后选择任何与 \mathbf{w} 不共线的向量 \mathbf{t} ，并使用叉积构建垂直于 \mathbf{w} 的单位向量 \mathbf{u} ：

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

当然，同样的过程可以用于以任何顺序构造三个向量；只需注意交叉乘积的顺序，以确保基础是右撇子。

如果 \mathbf{t} 与 \mathbf{w} 共线，分母将消失，如果它们几乎共线，结果将具有低精度。找到与 \mathbf{w} 完全不同的矢量的一个简单过程是从等于 \mathbf{w} 的 \mathbf{t} 开始，并将 \mathbf{t} 的最小幅度分量更改为 1。例如，如果 $\mathbf{w} = (1, 0, 0)$

$\mathbf{t} = (1/\sqrt{2}, -1/\sqrt{2}, 1)$ 。一旦 \mathbf{w} 和 \mathbf{u} 在手，完成基础是简单的：

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}.$$

使用这种结构的情况的一个例子是表面阴影，其中需要与表面法线对齐的基础，但围绕法线的旋转通常不重要。

2.4.7 从两个向量构建基础

上一节中的过程也可以用于围绕给定矢量旋转基础很重要的情况。一个常见的例子是为相机建立一个基础：使一个矢量与相机所看的方向对齐是很重要的，但是相机围绕该矢量的方向不是任意的，并且需要以某种方式指定它。一旦定向被固定下来，基础就完全确定了。

$\mathbf{u} = \mathbf{a} \times \mathbf{b}$ 也产生正交基，但它是左撇子。

完全指定帧的常用方法是通过提供两个向量 \mathbf{a} (其指定 \mathbf{w}) 和 \mathbf{b} (其指定 \mathbf{v})。如果两个向量已知为 perpendicular 通过 $\mathbf{u} = \mathbf{b} \times \mathbf{a}$ 构造第三个向量是一件简单的事情。



To be sure that the resulting basis really is orthonormal, even if the input vectors weren't quite, a procedure much like the single-vector procedure is advisable:

$$\begin{aligned}\mathbf{w} &= \frac{\mathbf{a}}{\|\mathbf{a}\|}, \\ \mathbf{u} &= \frac{\mathbf{b} \times \mathbf{w}}{\|\mathbf{b} \times \mathbf{w}\|}, \\ \mathbf{v} &= \mathbf{w} \times \mathbf{u}.\end{aligned}$$

In fact, this procedure works just fine when \mathbf{a} and \mathbf{b} are not perpendicular. In this case, \mathbf{w} will be constructed exactly in the direction of \mathbf{a} , and \mathbf{v} is chosen to be the closest vector to \mathbf{b} among all vectors perpendicular to \mathbf{w} .

This procedure *won't* work if \mathbf{a} and \mathbf{b} are collinear. In this case \mathbf{b} is of no help in choosing which of the directions perpendicular to \mathbf{a} we should use: it is perpendicular to all of them.

In the example of specifying camera positions (Section 4.3), we want to construct a frame that has \mathbf{w} parallel to the direction the camera is looking, and \mathbf{v} should point out the top of the camera. To orient the camera upright, we build the basis around the view direction, using the straight-up direction as the reference vector to establish the camera's orientation around the view direction. Setting \mathbf{v} as close as possible to straight up exactly matches the intuitive notion of "holding the camera straight."

2.4.8 Squaring Up a Basis

Occasionally you may find problems caused in your computations by a basis that is supposed to be orthonormal but where error has crept in—due to rounding error in computation, or to the basis having been stored in a file with low precision, for instance.

The procedure of the previous section can be used; simply constructing the basis anew using the existing \mathbf{w} and \mathbf{v} vectors will produce a new basis that is orthonormal and is close to the old one.

This approach is good for many applications, but it is not the best available. It does produce accurately orthogonal vectors, and for nearly orthogonal starting bases the result will not stray far from the starting point. However, it is asymmetric: it "favors" \mathbf{w} over \mathbf{v} and \mathbf{v} over \mathbf{u} (whose starting value is thrown away). It chooses a basis close to the starting basis but has no guarantee of choosing *the* closest orthonormal basis. When this is not good enough, the SVD (Section 5.4.1) can be used to compute an orthonormal basis that *is* guaranteed to be closest to the original basis.



为了确保得到的基础确实是正交的，即使输入向量不完全是，建议使用类似于单向量过程的过程：

$$\begin{aligned}\mathbf{w} &= \frac{\mathbf{a}}{\|\mathbf{a}\|}, \\ \mathbf{u} &= \frac{\mathbf{b} \times \mathbf{w}}{\|\mathbf{b} \times \mathbf{w}\|}, \\ \mathbf{v} &= \mathbf{w} \times \mathbf{u}.\end{aligned}$$

事实上，当 \mathbf{a} 和 \mathbf{b} 不垂直时，这个过程工作得很好。在这种情况下， \mathbf{w} 将恰好在 \mathbf{a} 的方向上构造，并且 \mathbf{v} 被选择为在垂直于 \mathbf{w} 的所有向量中最接近 \mathbf{b} 的向量。

如果 \mathbf{a} 和 \mathbf{b} 共线，此过程将不起作用。在这种情况下， \mathbf{b} 对于选择我们应该使用的垂直于 \mathbf{a} 的方向没有任何帮助：它垂直于所有方向。

在指定相机位置的例子（第4.3节）中，我们希望构造一个 \mathbf{w} 平行于相机所看的方向的帧，并且 \mathbf{v} 应该指出相机的顶部。为了使摄像机垂直定向，我们围绕视图方向构建基础，使用直线向上的方向作为参考矢量来建立摄像机围绕视图方向的定向。设置 \mathbf{v} 尽可能接近直线上升完全符合直观的概念“保持相机直线。”

如果你想让我将 \mathbf{w} 和 \mathbf{v} 设置为两个非弯曲方向，那么必须给出一些东西—通过这个方案，我会按照你想要的方式设置所有内容，除了我会对 \mathbf{v} 进行最小的

如果你想让我将 \mathbf{w} 和 \mathbf{v} 设置为两个非弯曲方向，那么必须给出一些东西—通过这个方案，我会按照你想要的方式设置所有内容，除了我会对 \mathbf{v} 进行最小的

If you want me to set \mathbf{w} and \mathbf{v} to two nonperpendicular directions, something has to give—with this scheme I'll set everything the way you want, except I'll make the smallest change to \mathbf{v} so that it is in fact perpendicular to \mathbf{w} .

What will go wrong with the computation if \mathbf{a} and \mathbf{b} are parallel?

2.4.8 建立一个基础

有时，您可能会发现在计算中由一个应该是正交的基础引起的问题，但是由于计算中的舍入误差，或者由于基础被存储在一个低精度的文件中而出现错误。

可以使用上一节的过程；简单地使用现有的 \mathbf{w} 和 \mathbf{v} 向量重新构建基础将产生一个新的基础，该基础是正交的并且接近旧的基础。

这种方法对许多应用程序都很好，但它不是最好的。它确实产生精确的正交矢量，并且对于几乎正交的起始基，结果不会偏离起点很远。然而，它是一个度量：它“赞成” \mathbf{w} 超过 \mathbf{v} 和 \mathbf{v} 超过 \mathbf{u} （其起始值被扔掉）。它选择一个接近起始基础的基础，但不能保证选择最接近的正交基础。当这还不够好时，SVD（第5.4.1节）可用于计算保证最接近原始基础的正交基。

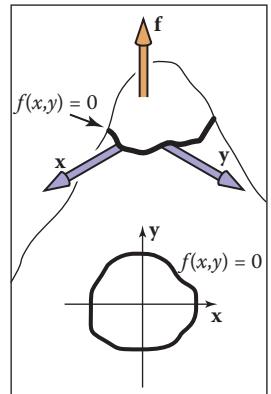


Figure 2.22. An implicit function $f(x,y) = 0$ can be thought of as a height field where f is the height (top). A path where the height is zero is the implicit curve (bottom).

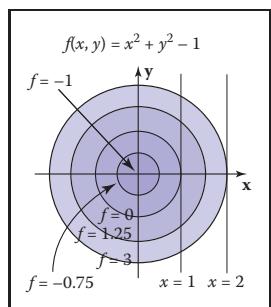


Figure 2.23. An implicit function $f(x,y) = 0$ can be thought of as a height field where f is the height (top). A path where the height is zero is the implicit curve (bottom).

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

This equation, if expanded algebraically, will yield Equation (2.9), but it is easier to see that this is an equation for a circle by “reading” the equation geometrically. It reads, “points \mathbf{p} on the circle have the following property: the vector from \mathbf{c} to

2.5.1 2D Implicit Curves

Intuitively, a *curve* is a set of points that can be drawn on a piece of paper without lifting the pen. A common way to describe a curve is using an *implicit equation*. An implicit equation in two dimensions has the form

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

The function $f(x, y)$ returns a real value. Points (x, y) where this value is zero are on the curve, and points where the value is nonzero are not on the curve. For example, let’s say that $f(x, y)$ is

$$f(x, y) = (x - x_c)^2 + (y - y_c)^2 - r^2, \quad (2.9)$$

where (x_c, y_c) is a 2D point and r is a nonzero real number. If we take $f(x, y) = 0$, the points where this equality holds are on the circle with center (x_c, y_c) and radius r . The reason that this is called an “implicit” equation is that the points (x, y) on the curve cannot be immediately calculated from the equation and instead must be determined by solving the equation. Thus, the points on the curve are not generated by the equation *explicitly*, but they are buried somewhere *implicitly* in the equation.

It is interesting to note that f does have values for all (x, y) . We can think of f as a terrain, with sea level at $f = 0$ (Figure 2.22). The shore is the implicit curve. The value of f is the altitude. Another thing to note is that the curve partitions space into regions where $f > 0$, $f < 0$, and $f = 0$. So you evaluate f to decide whether a point is “inside” a curve. Note that $f(x, y) = c$ is a curve for any constant c , and $c = 0$ is just used as a convention. For example, if $f(x, y) = x^2 + y^2 - 1$, varying c just gives a variety of circles centered at the origin (Figure 2.23).

We can compress our notation using vectors. If we have $\mathbf{c} = (x_c, y_c)$ and $\mathbf{p} = (x, y)$, then our circle with center \mathbf{c} and radius r is defined by those position vectors that satisfy

2.5 Curves and Surfaces

The geometry of curves, and especially surfaces, plays a central role in graphics, and here we review the basics of curves and surfaces in 2D and 3D space.

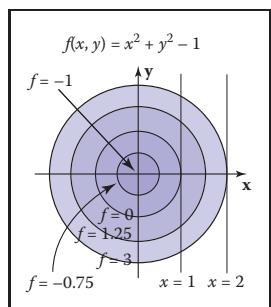


Figure 2.23. An implicit function $f(x,y) = 0$ can be thought of as a height field where f is the height (top). A path where the height is zero is the implicit curve (bottom).

2.5 曲线和曲面

曲线，特别是曲面的几何形状在图形中起着核心作用，在这里我们回顾了二维和三维空间中曲线和曲面的基础知识。

2.5.1 2D隐式曲线

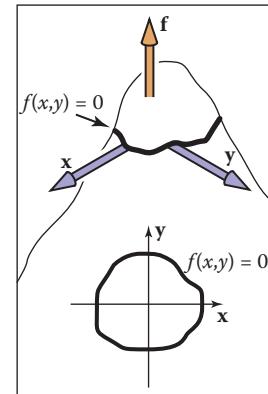
直观地说，曲线是一组可以在一张纸上绘制而无需提起笔的点。描述曲线的常用方法是使用隐式方程。二维的隐式方程具有以下形式

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

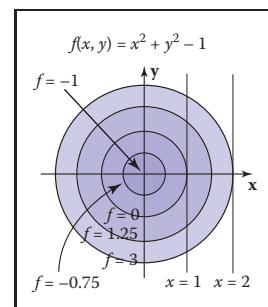
函数 $f(x, y)$ 返回一个实数值。此值为零的点 (x, y) 位于曲线上，值非零的点不在曲线上。例如，假设 $f(x, y)$ 是

$$f(x, y) = (x - x_c)^2 + (y - y_c)^2 - r^2, \quad (2.9)$$

其中 (x_c, y_c) 是 2D 点， r 是非零实数。如果我们取 $f(x, y) = 0$ ，这个相等成立的点在圆心 (x_c, y_c) 和 radius r 的圆上。这被称为“隐式”方程的原因是曲线上的点 (x, y) 不能立即从方程计算，而是必须通过求解方程来确定。因此，曲线上的点并不是由方程明确地定义的，而是隐式地埋在方程中的某个地方。



一个 implicit 函数 $f(x, y) = 0$ 可以被认为是一个高度字段，其中 f 是高度（顶部）。高度为零的路径是隐式曲线（底部）。



一个 implicit 函数 $f(x, y) = 0$ 可以被认为是一个高度字段，其中 f 是高度（顶部）。高度为零的路径是隐式曲线（底部）。

有趣的是， f 确实具有所有 (x, y) 的值。我们可以将 f 视为地形，海平面在 $f=0$ （图 2.22）。岸为隐式曲线。 f 的值是海拔高度。另外需要注意的是，曲线将空间划分为 $f>0$ 、 $f<0$ 和 $f=0$ 的区域。所以你评估 f 来决定一个点是否在曲线的“内部”。请注意， $f(x, y) = c$ 是任何常数 c 的曲线， $c=0$ 只是用作约定。例如，如果 $f(x, y) = x^2+y^2-1$ ，变化的 c 只是给出了以原点为中心的各种圆（图 2.23）。我们可以使用向量压缩我们的符号。如果我们有 $\mathbf{c} = (x_c, y_c)$ 和 $\mathbf{p} = (x, y)$ ，那么我们的圆心 \mathbf{c} 和半径 r 的圆由这些位置定义

满足的向量

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

这个方程，如果代数扩展，将产生方程 (2.9)，但更容易看到这是一个圆的方程，通过几何“读取”方程。它读取，“圆上的点 \mathbf{p} 具有以下属性：从 \mathbf{c} 到

\mathbf{p} when dotted with itself has value r^2 ." Because a vector dotted with itself is just its own length squared, we could also read the equation as, "points \mathbf{p} on the circle have the following property: the vector from \mathbf{c} to \mathbf{p} has squared length r^2 ."

Even better, is to observe that the squared length is just the squared distance from \mathbf{c} to \mathbf{p} , which suggests the equivalent form

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

and, of course, this suggests

$$\|\mathbf{p} - \mathbf{c}\| = r = 0.$$

The above could be read "the points \mathbf{p} on the circle are those a distance r from the center point \mathbf{c} ," which is as good a definition of circle as any. This illustrates that the vector form of an equation often suggests more geometry and intuition than the equivalent full-blown Cartesian form with x and y . For this reason, it is usually advisable to use vector forms when possible. In addition, you can support a vector class in your code; the code is cleaner when vector forms are used. The vector-oriented equations are also less error prone in implementation: once you implement and debug vector types in your code, the cut-and-paste errors involving x , y , and z will go away. It takes a little while to get used to vectors in these equations, but once you get the hang of it, the payoff is large.

2.5.2 The 2D Gradient

If we think of the function $f(x, y)$ as a height field with height = $f(x, y)$, the *gradient* vector points in the direction of maximum upslope, i.e., straight uphill. The gradient vector $\nabla f(x, y)$ is given by

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

The gradient vector evaluated at a point on the implicit curve $f(x, y) = 0$ is perpendicular to the *tangent* vector of the curve at that point. This perpendicular vector is usually called the *normal vector* to the curve. In addition, since the gradient points uphill, it indicates the direction of the $f(x, y) > 0$ region.

In the context of height fields, the geometric meaning of partial derivatives and gradients is more visible than usual. Suppose that near the point (a, b) , $f(x, y)$ is a plane (Figure 2.24). There is a specific uphill and downhill direction. At right angles to this direction is a direction that is level with respect to the plane. Any intersection between the plane and the $f(x, y) = 0$ plane will be in the direction that is level. Thus the uphill/downhill directions will be perpendicular to the line of intersection $f(x, y) = 0$. To see why the partial derivative has something to do

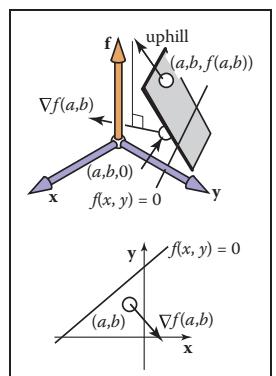


Figure 2.24. A surface height = $f(x, y)$ is locally planar near $(x, y) = (a, b)$. The gradient is a projection of the uphill direction onto the height = 0 plane.

\mathbf{p} 当用自身点划线时具有值r2。"因为一个点本身的矢量只是它自己的长度平方，我们也可以将方程读为，"圆上的点p具有以下属性：从c到p的矢量具有平方长度r2。"

更好的是，观察平方长度只是从c到p的平方距离，这表明等价形式 $\|\mathbf{p} - \mathbf{c}\|^2 = r^2 = 0$

当然，这表明

$$\|\mathbf{p} - \mathbf{c}\| = r = 0.$$

上面可以读成"圆上的点p是离中心点c的距离r"，这是圆的定义。这说明了方程的矢量形式通常比具有x和y的等效完全笛卡尔形式暗示更多的几何和直觉。出于这个原因，通常建议在可能的情况下使用矢量形式。此外，您可以在代码中支持vector类；当使用向量表单时，代码更清晰。面向矢量的方程在实现中也不容易出错：一旦在代码中实现和调试矢量类型，涉及x、y和z的剪切和粘贴错误就会消失。需要一段时间才能习惯这些方程中的向量，但是一旦你掌握了它，回报就会很大。

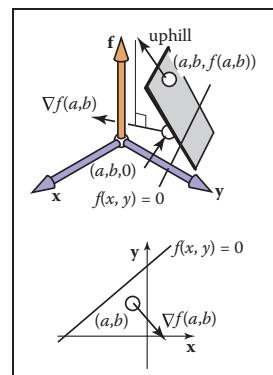
2.5.2 2D梯度

如果我们将函数 $f(x, y)$ 视为高度= $f(x, y)$ 的高度场，则梯度向量指向最大上坡的方向，即直线上坡。梯度向量 $\nabla f(x, y)$ 由下式给出

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

在隐式曲线 $f(x, y) = 0$ 上的点处评估的梯度矢量垂直于该点处曲线的切向矢量。这种垂直矢量通常称为曲线的法向量。另外，由于梯度指向上坡，所以表示 $f(x, y) > 0$ 区域的方向。

在高度场的背景下，偏导数和梯度的几何意义比通常更明显。假设在点 (a, b) 附近， $f(x, y)$ 是一个平面（图2.24）。有特定的上坡和下坡方向。与该方向成直角的方向是相对于平面水平的方向。平面与 $f(x, y) = 0$ 平面之间的任何交点都将处于水平方向。因此，上坡下坡方向将垂直于交点 $f(x, y) = 0$ 的线。看看为什么偏导数有什么关系



表面高度=f(x,y) 在(x,y)=(a,b)附近局部planar。Gradient是上山方向在高度=0平面上的投影。

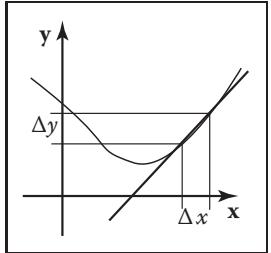


Figure 2.25. The derivative of a 1D function measures the slope of the line tangent to the curve.

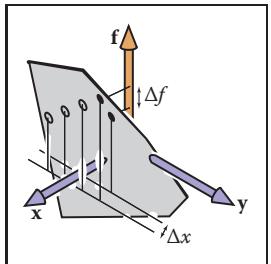


Figure 2.26. The partial derivative of a function f with respect to x must hold y constant to have a unique value, as shown by the dark point. The hollow points show other values of f that do not hold y constant.

with this, we need to visualize its geometric meaning. Recall that the conventional derivative of a 1D function $y = g(x)$ is

$$\frac{dy}{dx} \equiv \lim_{\Delta x \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{g(x + \Delta x) - g(x)}{\Delta x}. \quad (2.10)$$

This measures the *slope* of the *tangent* line to g (Figure 2.25).

The partial derivative is a generalization of the 1D derivative. For a 2D function $f(x, y)$, we can't take the same limit for x as in Equation (2.10), because f can change in many ways for a given change in x . However, if we hold y constant, we can define an analog of the derivative, called the *partial derivative* (Figure 2.26):

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

Why is it that the partial derivatives with respect to x and y are the components of the gradient vector? Again, there is more obvious insight in the geometry than in the algebra. In Figure 2.27, we see the vector a travels along a path where f does not change. Note that this is again at a small enough scale that the surface height $(x, y) = f(x, y)$ can be considered locally planar. From the figure, we see that the vector $a = (\Delta x, \Delta y)$.

Because the uphill direction is perpendicular to a , we know the dot product is equal to zero:

$$(\nabla f) \cdot a \equiv (x_\nabla, y_\nabla) \cdot (x_a, y_a) = x_\nabla \Delta x + y_\nabla \Delta y = 0. \quad (2.11)$$

We also know that the change in f in the direction (x_a, y_a) equals zero:

$$\Delta f = \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial y} \Delta y \equiv \frac{\partial f}{\partial x} x_a + \frac{\partial f}{\partial y} y_a = 0.$$

Given any vectors (x, y) and (x', y') that are perpendicular, we know that the angle between them is 90 degrees, and thus their dot product equals zero (recall that the dot product is proportional to the cosine of the angle between the two vectors). Thus, we have $xx' + yy' = 0$. Given (x, y) , it is easy to construct valid vectors whose dot product with (x, y) equals zero, the two most obvious being $(y, -x)$ and $(-y, x)$; you can verify that these vectors give the desired zero dot product with (x, y) . A generalization of this observation is that (x, y) is perpendicular to $k(y, -x)$ where k is any nonzero constant. This implies that

$$(x_a, y_a) = k \left(\frac{\partial f}{\partial y}, -\frac{\partial f}{\partial x} \right). \quad (2.12)$$

Combining Equations (2.11) and (2.12) gives

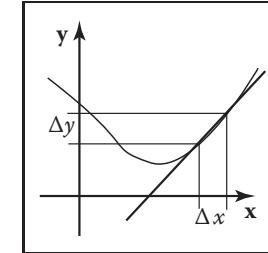
$$(x_\nabla, y_\nabla) = k' \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right),$$

有了这个，我们需要可视化它的几何意义。回想一下，一维函数 $y=g(x)$ 的常规导数是

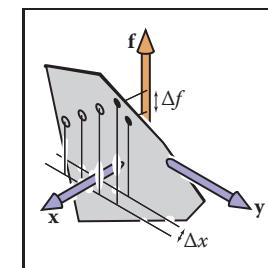
$$\frac{dy}{dx} \equiv \lim_{\Delta x \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{g(x + \Delta x) - g(x)}{\Delta x}. \quad (2.10)$$

这测量了切线到 g 的斜率（图2.25）。

偏导数是一维导数的泛化。对于2Dfunction $f(x, y)$ 我们不能对 x 采取与等式(2.10)中相同的限制因为对于给定的 x 变化 f 可以以多种方式变化。但是，如果我们保持 y 常数，我们可以定义导数的模拟，称为偏导数（图2.26）：



一维函数的导数测量与曲线相切的线的斜率。



函数 f 相对于 x 的偏导数必须保持 y constant以具有唯一值，如暗点所示。空心点显示 f 的其他values不持有 y constant。

为什么相对于 x 和 y 的偏导数是梯度向量的分量？同样，在几何中比在代数中有更明显的洞察力。在图2.27中，我们看到矢量 a 沿着 f 不改变的路径行进。请注意，这再次是在足够小的尺度下，表面高度 $(x, y) = f(x, y)$ 可以被认为是局部平面的。从图中，我们看到矢量 $a = (\Delta x, \Delta y)$ 。

因为上坡方向垂直于 a ，我们知道点积等于零：

$$(\nabla f) \cdot a \equiv (x_\nabla, y_\nabla) \cdot (x_a, y_a) = x_\nabla \Delta x + y_\nabla \Delta y = 0. \quad (2.11)$$

我们也知道 f 在方向 (x_a, y_a) 上的变化等于零：

$$\Delta f = \frac{\partial f}{\partial x} \Delta x + \frac{\partial f}{\partial y} \Delta y \equiv \frac{\partial f}{\partial x} x_a + \frac{\partial f}{\partial y} y_a = 0.$$

给定任何垂直的向量 (x, y) 和 (x', y') ，我们知道它们之间的角度是90度，因此它们的点积等于零（回想一下点积与两个向量之间的角度的余弦成正比）。因此，我们有 $xx' + yy' = 0$ 。给定 (x, y) ，很容易构造有效向量，其点积与 (x, y) 等于零，两个最明显的是 $(y, -x)$ 和 $(-y, x)$ ；您可以验证这些向量给出了所需的零点积与 (x, y) 。这个观察的一个概括是 (x, y) 垂直于 $k(y, -x)$ ，其中 k 是任何非零常数。这意味着

$$(x_a, y_a) = k \left(\frac{\partial f}{\partial y}, -\frac{\partial f}{\partial x} \right). \quad (2.12)$$

组合方程 (2.11) 和 (2.12) 给出

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

where k' is any nonzero constant. By definition, "uphill" implies a positive change in f , so we would like $k' > 0$, and $k' = 1$ is a perfectly good convention.

As an example of the gradient, consider the implicit circle $x^2 + y^2 - 1 = 0$ with gradient vector $(2x, 2y)$, indicating that the outside of the circle is the positive region for the function $f(x, y) = x^2 + y^2 - 1$. Note that the length of the gradient vector can be different depending on the multiplier in the implicit equation. For example, the unit circle can be described by $Ax^2 + Ay^2 - A = 0$ for any nonzero A . The gradient for this curve is $(2Ax, 2Ay)$. This will be normal (perpendicular) to the circle, but will have a length determined by A . For $A > 0$, the normal will point outward from the circle, and for $A < 0$, it will point inward. This switch from outward to inward is as it should be, since the positive region switches inside the circle. In terms of the height-field view, $h = Ax^2 + Ay^2 - A$, and the circle is at zero altitude. For $A > 0$, the circle encloses a depression, and for $A < 0$, the circle encloses a bump. As A becomes more negative, the bump increases in height, but the $h = 0$ circle doesn't change. The direction of maximum uphill doesn't change, but the slope increases. The length of the gradient reflects this change in degree of the slope. So intuitively, you can think of the gradient's direction as pointing uphill and its magnitude as measuring how uphill the slope is.

Implicit 2D Lines

The familiar "slope-intercept" form of the line is

$$y = mx + b. \quad (2.13)$$

This can be converted easily to implicit form (Figure 2.28):

$$y - mx - b = 0. \quad (2.14)$$

Here m is the "slope" (ratio of rise to run) and b is the y value where the line crosses the y -axis, usually called the y -intercept. The line also partitions the 2D plane, but here "inside" and "outside" might be more intuitively called "over" and "under."

Because we can multiply an implicit equation by any constant without changing the points where it is zero, $kf(x, y) = 0$ is the same curve for any nonzero k . This allows several implicit forms for the same line, for example,

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

One reason the slope-intercept form is sometimes awkward is that it can't represent some lines such as $x = 0$ because m would have to be infinite. For this

其中 k' 是任何非零常数。根据定义, "上坡"意味着 f 的正变化, 所以我们希望 $k' > 0$, 并且 $k' = 1$ 是一个非常好的惯例。

作为梯度的一个例子, 考虑具有梯度向量 $(2x, 2y)$ 的隐式圆 $x^2 + y^2 - 1 = 0$, 表明圆的外部是函数 $f(x, y) = x^2 + y^2 - 1$ 的正区域。注意, 梯度向量的长度可以根据隐式方程中的乘数而不同。例如, 单位圆可以通过对于任何非零 A 的 $Ax^2 + Ay^2 - A = 0$ 来描述。该曲线的梯度为 $(2Ax, 2Ay)$ 。这将是正常的(垂直)圆, 但将具有由 A 确定的长度。对于 $A > 0$, 法线将从圆向外指向, 对于 $A < 0$, 它将向内指向。这个从外向内的开关是应该的, 因为正区域在圆圈内切换。在高度-视场视图方面, $h = Ax^2 + Ay^2 - A$, 并且圆处于零高度。对于 $A > 0$, 圆圈包围凹陷, 对于 $A < 0$, 圆圈包围凸。随着 A 变得更加消极,

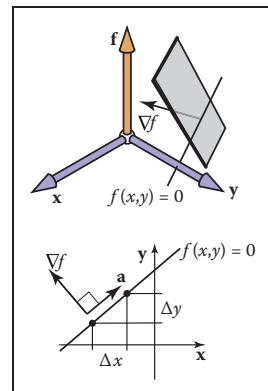
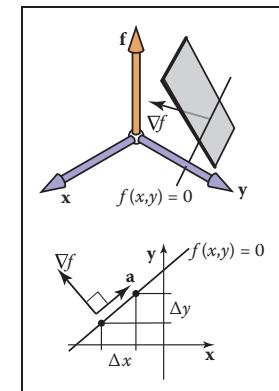


Figure 2.27. The vector a points in a direction where f has no change and is thus perpendicular to the gradient vector ∇f .

凹凸高度增加, 但 $h=0$ 圆不改变。最大上坡的方向不会改变, 但坡度增加。梯度的长度反映了斜率的这种程度变化。因此, 直观地, 您可以将梯度的方向视为指向山坡, 将其幅度视为测量坡度的上坡程度。



矢量 a 指向一个方向, 其中 f 没有变化, 因此每摆到梯度 vector f 。

隐式2D线

线的熟悉的"斜率-截距"形式是

$$y = mx + b. \quad (2.13)$$

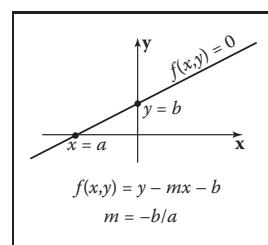


Figure 2.28. A 2D line can be described by the equation $y - mx - b = 0$.

这可以很容易地转换为隐式形式 (图2.28) :

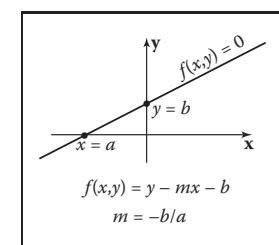
$$y - mx - b = 0. \quad (2.14)$$

这里 m 是"斜率" (上升与运行的比率), b 是线穿过 y 轴的 y 值, 通常称为 y 截距。该线还对 2D 平面进行了分区, 但这里的"内部"和"外部"可能更直观地称为"上方"和"下方"。"

因为我们可以将隐式方程乘以任何常数而不改变它为零的点, 所以对于任何非零 k , $kf(x, y) = 0$ 都是相同的曲线。这允许同一行的几种隐式形式, 例如

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

斜率截距形式有时很尴尬的一个原因是它不能表示某些线, 例如 $x = 0$, 因为 m 必须是无限的。为了这个



二维线可以用公式 $y - mx - b = 0$ 来描述。

reason, a more general form is often useful:

$$Ax + By + C = 0, \quad (2.15)$$

for real numbers A, B, C .

Suppose we know two points on the line, (x_0, y_0) and (x_1, y_1) . What A, B , and C describe the line through these two points? Because these points lie on the line, they must both satisfy Equation (2.15):

$$\begin{aligned} Ax_0 + By_0 + C &= 0, \\ Ax_1 + By_1 + C &= 0. \end{aligned}$$

Unfortunately we have two equations and *three* unknowns: A, B , and C . This problem arises because of the arbitrary multiplier we can have with an implicit equation. We could set $C = 1$ for convenience:

$$Ax + By + 1 = 0,$$

but we have a similar problem to the infinite slope case in slope-intercept form: lines through the origin would need to have $A(0) + B(0) + 1 = 0$, which is a contradiction. For example, the equation for a 45-degree line through the origin can be written $x - y = 0$, or equally well $y - x = 0$, or even $17y - 17x = 0$, but it cannot be written in the form $Ax + By + 1 = 0$.

Whenever we have such pesky algebraic problems, we try to solve the problems using geometric intuition as a guide. One tool we have, as discussed in Section 2.5.2, is the gradient. For the line $Ax + By + C = 0$, the gradient vector is (A, B) . This vector is perpendicular to the line (Figure 2.29), and points to the side of the line where $Ax + By + C$ is positive. Given two points on the line (x_0, y_0) and (x_1, y_1) , we know that the vector between them points in the same direction as the line. This vector is just $(x_1 - x_0, y_1 - y_0)$, and because it is parallel to the line, it must also be perpendicular to the gradient vector (A, B) . Recall that there are an infinite number of (A, B, C) that describe the line because of the arbitrary scaling property of implicits. We want any one of the valid (A, B, C) .

We can start with any (A, B) perpendicular to $(x_1 - x_0, y_1 - y_0)$. Such a vector is just $(A, B) = (y_0 - y_1, x_1 - x_0)$ by the same reasoning as in Section 2.5.2. This means that the equation of the line through (x_0, y_0) and (x_1, y_1) is

$$(y_0 - y_1)x + (x_1 - x_0)y + C = 0. \quad (2.16)$$

Now we just need to find C . Because (x_0, y_0) and (x_1, y_1) are on the line, they must satisfy Equation (2.16). We can plug either value in and solve for C . Doing this for (x_0, y_0) yields $C = x_0y_1 - x_1y_0$, and thus the full equation for the line is

$$(y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0. \quad (2.17)$$

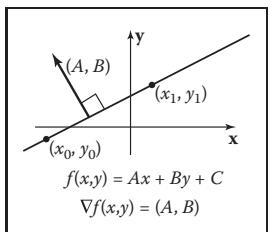


Figure 2.29. The gradient vector (A, B) is perpendicular to the implicit line $Ax + By + C = 0$.

原因, 更一般的形式往往是有用的:

$$Ax+By+C=0 \quad (2.15)$$

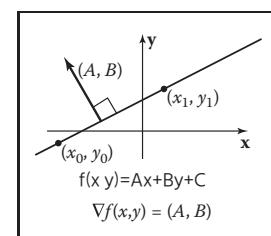
对于实数 A, B, C 。

假设我们知道线上的两个点, (x_0, y_0) 和 (x_1, y_1) 。什么 A, B 和 C 描述通过这两点的线? 因为这些点位于线上, 所以它们都必须满足方程 (2.15) :

$$\begin{aligned} Ax_0 + By_0 + C &= 0 \\ Ax_1 + By_1 + C &= 0. \end{aligned}$$

不幸的是, 我们有两个方程和三个未知数: A, B 和 C 。这个问题的出现是因为我们可以使用隐式方程的任意乘数。为了方便起见, 我们可以设置 $C = 1$:

$$Ax+By+1=0$$



梯度向量 (A, B) 垂直于隐含线 $Ax+By+C=0$ 。

但是我们有一个类似于斜率截距形式的无限斜率情况的问题: 通过原点的线需要有 $A(0) + B(0) + 1 = 0$, 这是一个矛盾。例如, 通过原点的45度线的方程可以写成 $x - y = 0$, 或者同样好的 $y - x = 0$, 甚至 $17y - 17x = 0$, 但不能写成 $Ax + By + 1 = 0$ 的形式。

每当我们有这样令人讨厌的代数问题时, 我们都会尝试使用几何直觉作为指导来解决 problems。正如第2.5.2节所讨论的我们有一个工具是梯度。对于线 $Ax+By+C=0$, 梯度向量为 (A, B) 。这个向量垂直于线 (图2.29), 并指向

$Ax+By+C$ 为正的线的一侧。给定线 (x_0, y_0) 和 (x_1, y_1) 上的两个点, 我们知道它们之间的矢量指向与线相同的方向。这个向量只是 $(x_1 - x_0, y_1 - y_0)$, 并且因为它与线平行, 所以它也必须垂直于梯度向量 (A, B) 。回想一下, 由于隐含的任意缩放属性, 有无限数量的 (A, B, C) 描述了该行。我们想要任何一个有效的 (A, B, C) 。我们可以从垂直于 $(x_1 - x_0, y_1 - y_0)$ 的任何 (A, B) 开始。通过与第2.5.2节相同的推理, 这样的向量只是 $(A, B) = (y_0 - y_1, x_1 - x_0)$ 。这意味着通过 (x_0, y_0) 和 (x_1, y_1) 的线的方程是

$$(y_0 - y_1)x + (x_1 - x_0)y + C = 0. \quad (2.16)$$

现在我们只需要找到 C 。因为 (x_0, y_0) 和 (x_1, y_1) 在线上, 它们必须满足方程 (2.16)。我们可以插入任何一个值并求解 C 。对 (x_0, y_0) 执行此操作会产生 $C = x_0y_1 - x_1y_0$, 因此该行的完整方程为

$$(y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0. \quad (2.17)$$

Again, this is one of infinitely many valid implicit equations for the line through two points, but this form has no division operation and thus no numerically degenerate cases for points with finite Cartesian coordinates. A nice thing about Equation (2.17) is that we can always convert to the slope-intercept form (when it exists) by moving the non- y terms to the right-hand side of the equation and dividing by the multiplier of the y term:

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

An interesting property of the implicit line equation is that it can be used to find the signed distance from a point to the line. The value of $Ax + By + C$ is proportional to the distance from the line (Figure 2.30). As shown in Figure 2.31, the distance from a point to the line is the length of the vector $k(A, B)$, which is

$$\text{distance} = k\sqrt{A^2 + B^2}. \quad (2.18)$$

For the point $(x, y) + k(A, B)$, the value of $f(x, y) = Ax + By + C$ is

$$\begin{aligned} f(x + kA, y + kB) &= Ax + kA^2 + By + kB^2 + C \\ &= k(A^2 + B^2). \end{aligned} \quad (2.19)$$

The simplification in that equation is a result of the fact that we know (x, y) is on the line, so $Ax + By + C = 0$. From Equations (2.18) and (2.19), we can see that the signed distance from line $Ax + By + C = 0$ to a point (a, b) is

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

Here “signed distance” means that its magnitude (absolute value) is the geometric distance, but on one side of the line, distances are positive and on the other they are negative. You can choose between the equally valid representations $f(x, y) = 0$ and $-f(x, y) = 0$ if your problem has some reason to prefer a particular side being positive. Note that if (A, B) is a unit vector, then $f(a, b)$ is the signed distance. We can multiply Equation (2.17) by a constant that ensures that (A, B) is a unit vector:

$$\begin{aligned} f(x, y) &= \frac{y_0 - y_1}{\sqrt{(x_1 - x_0)^2 + (y_0 - y_1)^2}}x + \frac{x_1 - x_0}{\sqrt{(x_1 - x_0)^2 + (y_0 - y_1)^2}}y \\ &\quad + \frac{x_0 y_1 - x_1 y_0}{\sqrt{(x_1 - x_0)^2 + (y_0 - y_1)^2}} = 0. \end{aligned} \quad (2.20)$$

Note that evaluating $f(x, y)$ in Equation (2.20) directly gives the signed distance, but it does require a square root to set up the equation. Implicit lines will turn out to be very useful for triangle rasterization (Section 8.1.2). Other forms for 2D lines are discussed in Chapter 14.

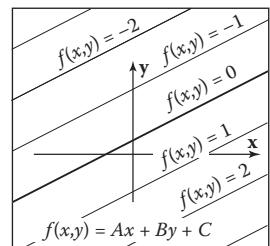


Figure 2.30. The value of the implicit function $f(x, y) = Ax + By + C$ is a constant times the signed distance from $Ax + By + C = 0$.

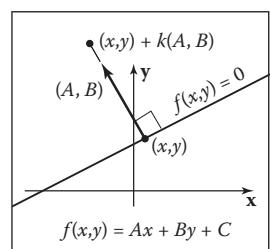


Figure 2.31. The vector $k(A, B)$ connects a point (x, y) on the line closest to a point not on the line. The distance is proportional to k .

同样，这是通过两个点的线的无限多有效隐式方程之一，但是这种形式没有除法运算，因此对于具有有限笛卡尔坐标的点没有数值退化情况。关于方程 (2.17) 的一个好处是，我们总是可以通过将nony项移动到方程的右侧并除以y项的乘数来转换为斜率截距形式（当它存在时）：

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

隐式线方程的一个有趣的属性是它可以用来找到从点到线的带符号距离。 $Ax+By+C$ 的值为

与线的距离成比例（图2.30）。如图2.31所示，点到线的距离是向量 $k(A, B)$ 的长度，即

$$\text{distance} = k\sqrt{A^2 + B^2}. \quad (2.18)$$

对于点 $(x, y) + k(A, B)$, $f(x, y) = Ax + By + C$ 的值为

$$\begin{aligned} f(x+kA, y+kB) &= Ax+kA^2+By+kB^2+C \\ &= k(A^2 + B^2). \end{aligned} \quad (2.19)$$

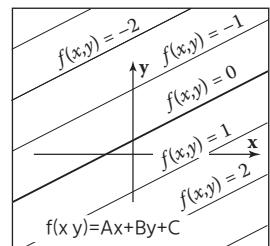
该方程中的简化是我们知道 (x, y) 在线上的事实的结果，因此 $Ax+By+C=0$ 。从方程 (2.18) 和 (2.19) 中，我们可以看到线 $Ax+By+C=0$ 到点 (a, b) 的有符号距离为

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

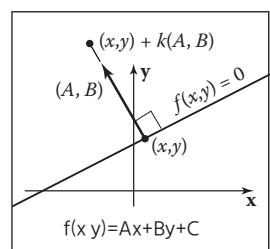
这里的“有符号距离”意味着它的大小（绝对值）是几何距离，但在线的一侧，距离是正的，而在另一侧则是负的。您可以在同样有效的表示 $f(x, y) = 0$ 和 $f(x, y) = 0$ 之间进行选择，如果您的问题有某种原因更喜欢特定的一面是积极的。请注意，如果 (A, B) 是单位向量，则 (a, b) 是带符号距离。我们可以将方程 (2.17) 乘以一个常数，该常数确保 (A, B) 是一个单位向量：

$$\begin{aligned} f(x, y) &= \frac{y_0 - y_1}{\sqrt{(x_1 - x_0)^2 + (y_0 - y_1)^2}}x + \frac{x_1 - x_0}{\sqrt{(x_1 - x_0)^2 + (y_0 - y_1)^2}}y \\ &\quad + \frac{x_0 y_1 - x_1 y_0}{\sqrt{(x_1 - x_0)^2 + (y_0 - y_1)^2}} = 0. \end{aligned} \quad (2.20)$$

请注意，在方程 (2.20) 中评估 $f(x, y)$ 直接给出了带符号的距离，但它确实需要一个平方根来建立方程。隐式线对于三角形栅格化非常有用（第 8.1.2 节）。2D 线的其他形式在第 14 章中讨论。



隐式函数 $f(x, y) = Ax + By + C$ 的值是距 $Ax + By + C = 0$ 的带符号距离的常数倍。



Vector $k(A, B)$ 连接最近线上的一个点 (x, y) 那条线。距离与 k 成正比。

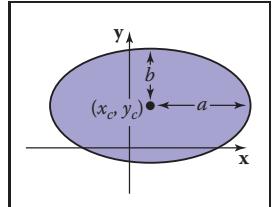


Figure 2.32. The ellipse with center (x_c, y_c) and semi-axes of length a and b .

Implicit Quadric Curves

In the previous section we saw that a linear function $f(x, y)$ gives rise to an implicit line $f(x, y) = 0$. If f is instead a quadratic function of x and y , with the general form

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

the resulting implicit curve is called a quadric. Two-dimensional quadric curves include ellipses and hyperbolas, as well as the special cases of parabolas, circles, and lines.

Examples of quadric curves include the circle with center (x_c, y_c) and radius r ,

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

and axis-aligned ellipses of the form

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

Try setting $a = b = r$ in the ellipse equation and compare to the circle equation.

where (x_c, y_c) is the center of the ellipse, and a and b are the minor and major semi-axes (Figure 2.32).

2.5.3 3D Implicit Surfaces

Just as implicit equations can be used to define curves in 2D, they can be used to define surfaces in 3D. As in 2D, implicit equations *implicitly* define a set of points that are on the surface:

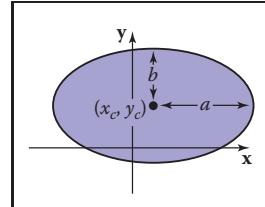
$$f(x, y, z) = 0.$$

Any point (x, y, z) that is on the surface results in zero when given as an argument to f . Any point not on the surface results in some number other than zero. You can check whether a point is on the surface by evaluating f , or you can check which side of the surface the point lies on by looking at the sign of f , but you cannot always explicitly construct points on the surface. Using vector notation, we will write such functions of $\mathbf{p} = (x, y, z)$ as

$$f(\mathbf{p}) = 0.$$

2.5.4 Surface Normal to an Implicit Surface

A surface normal (which is needed for lighting computations, among other things) is a vector perpendicular to the surface. Each point on the surface may have a



中心 (x_c, y_c) 和长度为 a 和 b 的半轴的椭圆。

隐式二次曲线

在上一节中，我们看到一个线性函数 $f(x, y)$ 产生一个im线 $f(x, y) = 0$ 。如果 f 替代 x 和 y 的二次函数，与一般形式

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

得到的隐式曲线称为二次曲线。二维二次曲线包括椭圆和双曲线，以及抛物线，圆和线的特殊情况。

二次曲线的例子包括中心圆 (x_c, y_c) 和 radiusr

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

和形式的轴对齐椭圆

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

其中 (x_c, y_c) 是椭圆的中心， a 和 b 是短轴和长半轴（图2.32）。

2.5.3 3d隐式表面

正如隐式方程可用于在2D中定义曲线一样，它们可用于在3d中定义曲面。：

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

表面上的任何点 (x, y, z) 在作为 f 的参数给出时都会导致零。任何不在表面上的点都会产生除零以外的一些数字。您可以通过计算 f 来检查点是否在表面上，也可以通过查看 f 的符号来检查点位于表面的哪一侧，但您不能始终在表面上显式构造点。使用向量表示法，我们将 $\mathbf{p} = (x, y, z)$ 的函数写为

$$f(\mathbf{p}) = 0.$$

2.5.4 表面法线到隐式表面

表面法线（光照计算等需要）是垂直于表面的矢量。表面上的每个点可以具有



different normal vector. In the same way that the gradient provides a normal to an implicit curve in 2D, the surface normal at a point \mathbf{p} on an implicit surface is given by the gradient of the implicit function

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

The reasoning is the same as for the 2D case: the gradient points in the direction of fastest increase in f , which is perpendicular to all directions tangent to the surface, in which f remains constant. The gradient vector points toward the side of the surface where $f(\mathbf{p}) > 0$, which we may think of as “into” the surface or “out from” the surface in a given context. If the particular form of f creates inward-facing gradients, and outward-facing gradients are desired, the surface $-f(\mathbf{p}) = 0$ is the same as surface $f(\mathbf{p}) = 0$ but has directionally reversed gradients, i.e., $-\nabla f(\mathbf{p}) = \nabla(-f(\mathbf{p}))$.

2.5.5 Implicit Planes

As an example, consider the infinite plane through point \mathbf{a} with surface normal \mathbf{n} . The implicit equation to describe this plane is given by

$$(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0. \quad (2.21)$$

Note that \mathbf{a} and \mathbf{n} are known quantities. The point \mathbf{p} is any unknown point that satisfies the equation. In geometric terms this equation says “the vector from \mathbf{a} to \mathbf{p} is perpendicular to the plane normal.” If \mathbf{p} were not in the plane, then $(\mathbf{p} - \mathbf{a})$ would not make a right angle with \mathbf{n} (Figure 2.33).

Sometimes we want the implicit equation for a plane through points \mathbf{a} , \mathbf{b} , and \mathbf{c} . The normal to this plane can be found by taking the cross product of any two vectors in the plane. One such cross product is

$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}).$$

This allows us to write the implicit plane equation:

$$(\mathbf{p} - \mathbf{a}) \cdot ((\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})) = 0. \quad (2.22)$$

A geometric way to read this equation is that the volume of the parallelepiped defined by $\mathbf{p} - \mathbf{a}$, $\mathbf{b} - \mathbf{a}$, and $\mathbf{c} - \mathbf{a}$ is zero, i.e., they are coplanar. This can only be true if \mathbf{p} is in the same plane as \mathbf{a} , \mathbf{b} , and \mathbf{c} . The full-blown Cartesian



不同的法向量。与梯度在2D中为隐式曲线提供法线相同，隐式表面上点 \mathbf{p} 处的表面法线由隐式函数的梯度给出

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

推理与2D情况相同：梯度指向 f 增加最快的方向，该方向垂直于与表面相切的所有方向，其中 f 保持恒定。梯度向量指向 $f(p) > 0$ 的表面一侧，我们可以将其视为在给定上下文中“进入”表面或“离开”表面。如果特定形式的 f 创建面向内的梯度，并且需要面向外的梯度，则表面 $-f(p)=0$ 与表面 $f(p)=0$ 相同，但具有方向相反的梯度，即 $\infty f(p)=\infty(f(p))$ 。

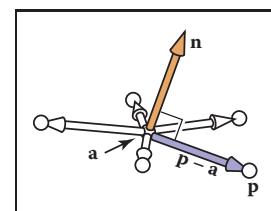


Figure 2.33. Any of the points \mathbf{p} shown are in the plane with normal vector \mathbf{n} that includes point \mathbf{a} if Equation (2.2) is satisfied.

2.5.5 隐式平面

作为一个例子，考虑通过具有表面法线 \mathbf{n} 的点 \mathbf{a} 的无限平面。
描述该平面的隐式方程由下式给出

$$(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0. \quad (2.21)$$

注意， \mathbf{a} 和 \mathbf{n} 是已知量。点 \mathbf{p} 是满足方程的任何未知点。在几何术语中，这个方程说：“从 \mathbf{a} 到 \mathbf{p} 的矢量垂直于平面法线。”如果 \mathbf{p} 不在平面内，那么 $(\mathbf{p} - \mathbf{a})$ 就不会与 \mathbf{n} 成直角（图2.33）。

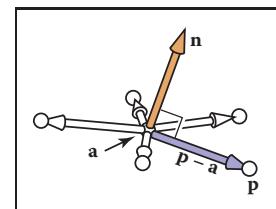
有时我们希望通过点 \mathbf{a} , \mathbf{b} 和 \mathbf{c} 的平面的隐式方程。通过取平面中任意两个矢量的叉积可以找到该平面的法线。一个这样的交叉产品是

$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}).$$

这使我们可以编写隐式平面方程：

$$(\mathbf{p} - \mathbf{a}) \cdot ((\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})) = 0. \quad (2.22)$$

读取该方程的几何方式是由 \mathbf{p} \mathbf{a} , \mathbf{b} \mathbf{a} 和 \mathbf{c} \mathbf{a} 定义的平行六面体的体积为零，即它们共面。只有当 \mathbf{p} 与 \mathbf{a} , \mathbf{b} 和 \mathbf{c} 处于同一平面时，这才可能成立。全面的笛卡尔



所示的任何点 \mathbf{p} 都在法向量为 \mathbf{n} 的平面中，如果满足Equation(2.2)，则包括点 \mathbf{a} 。

representation for this is given by the determinant (this is discussed in more detail in Section 5.3):

$$\begin{vmatrix} x - x_a & y - y_a & z - z_a \\ x_b - x_a & y_b - y_a & z_b - z_a \\ x_c - x_a & y_c - y_a & z_c - z_a \end{vmatrix} = 0. \quad (2.23)$$

The determinant can be expanded (see Section 5.3 for the mechanics of expanding determinants) to the bloated form with many terms.

Equations (2.22) and (2.23) are equivalent, and comparing them is instructive. Equation (2.22) is easy to interpret geometrically and will yield efficient code. In addition, it is relatively easy to avoid a typographic error that compiles into incorrect code if it takes advantage of debugged cross and dot product code. Equation (2.23) is also easy to interpret geometrically and will be efficient provided an efficient 3×3 determinant function is implemented. It is also easy to implement without a typo if a function `determinant(a, b, c)` is available. It will be especially easy for others to read your code if you rename the `determinant` function `volume`. So both Equations (2.22) and (2.23) map well into code. The full expansion of either equation into x -, y -, and z -components is likely to generate typos. Such typos are likely to compile and, thus, to be especially pesky. This is an excellent example of clean math generating clean code and bloated math generating bloated code.

3D Quadric Surfaces

Just as quadratic polynomials in two variables define quadric curves in 2D, quadratic polynomials in x , y , and z define *quadric surfaces* in 3D. For instance, a sphere can be written as

$$f(\mathbf{p}) = (\mathbf{p} - \mathbf{c})^2 - r^2 = 0,$$

and an axis-aligned ellipsoid may be written as

$$f(\mathbf{p}) = \frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2} + \frac{(z - z_c)^2}{c^2} - 1 = 0.$$

3D Curves from Implicit Surfaces

One might hope that an implicit 3D curve could be created with the form $f(\mathbf{p}) = 0$. However, all such curves are just degenerate surfaces and are rarely useful in practice. A 3D curve can be constructed from the intersection of two simultaneous implicit equations:

$$\begin{aligned} f(\mathbf{p}) &= 0, \\ g(\mathbf{p}) &= 0. \end{aligned}$$

对此的表示由行列式给出（这在第5.3节中更详细地讨论）：

$$\begin{vmatrix} x - x_a & y - y_a & z - z_a \\ x_b - x_a & y_b - y_a & z_b - z_a \\ x_c - x_a & y_c - y_a & z_c - z_a \end{vmatrix} = 0. \quad (2.23)$$

行列式可以扩展（有关扩展行列式的机制，请参阅第5.3节）到具有许多项的膨胀形式。

方程 (2.22) 和 (2.23) 是等价的，比较它们是指导性的。方程(2.22)很容易从几何上解释并将产生有效的代码。此外，如果利用调试的交叉和点积代码，则相对容易避免编译成不正确代码的印刷错误。方程 (2.23) 也很容易在几何上解释，并且在实现高效的 3×3 行列式函数之前将是高效的。如果函数行列式 `(a, b, c)` 可用，则在没有拼写错误的情况下也很容易实现。如果您重命名行列式function卷，其他人将特别容易阅读您的代码。所以两个方程 (2.22) 和 (2.23) 都很好地映射到代码中。将任一方程完全扩展为 x 、 y 和 z 分量都可能产生拼写错误。这样的拼写错误很可能会编译，因此特别令人讨厌。这是一个很好的例子，干净的数学生成干净的代码和臃肿的数学生成臃肿的代码。

3D Quadric Surfaces

正如两个变量中的二次多项式在2D中定义二次曲线一样， x ， y 和 z 中的二次多项式在3D中定义二次曲面。例如，球体可以写成

$$f(\mathbf{p}) = (\mathbf{p} - \mathbf{c})^2 - r^2 = 0,$$

并且轴对齐的椭球体可以写成

$$f(\mathbf{p}) = \frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2} + \frac{(z - z_c)^2}{c^2} - 1 = 0.$$

隐式曲面的3d曲线

人们可能希望可以使用 $f(\mathbf{p}) = 0$ 的形式创建隐式3d曲线。然而，所有这些曲线都只是退化曲面，在实践中很少有用。一条三维曲线可以由两个联立隐方程的交集构建：

$$\begin{aligned} f(\mathbf{p}) &= 0, \\ g(\mathbf{p}) &= 0. \end{aligned}$$



For example, a 3D line can be formed from the intersection of two implicit planes. Typically, it is more convenient to use parametric curves instead; they are discussed in the following sections.

2.5.6 2D Parametric Curves

A *parametric* curve is controlled by a single *parameter* that can be considered a sort of index that moves continuously along the curve. Such curves have the form

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

Here (x, y) is a point on the curve, and t is the parameter that influences the curve. For a given t , there will be some point determined by the functions g and h . For continuous g and h , a small change in t will yield a small change in x and y . Thus, as t continuously changes, points are swept out in a continuous curve. This is a nice feature because we can use the parameter t to explicitly construct points on the curve. Often we can write a parametric curve in vector form,

$$\mathbf{p} = f(t),$$

where f is a vector-valued function, $f : \mathbb{R} \mapsto \mathbb{R}^2$. Such vector functions can generate very clean code, so they should be used when possible.

We can think of the curve with a position as a function of time. The curve can go anywhere and could loop and cross itself. We can also think of the curve as having a velocity at any point. For example, the point $\mathbf{p}(t)$ is traveling slowly near $t = -2$ and quickly between $t = 2$ and $t = 3$. This type of “moving point” vocabulary is often used when discussing parametric curves even when the curve is not describing a moving point.

2D Parametric Lines

A parametric line in 2D that passes through points $\mathbf{p}_0 = (x_0, y_0)$ and $\mathbf{p}_1 = (x_1, y_1)$ can be written as

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

Because the formulas for x and y have such similar structure, we can use the vector form for $\mathbf{p} = (x, y)$ (Figure 2.34):

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

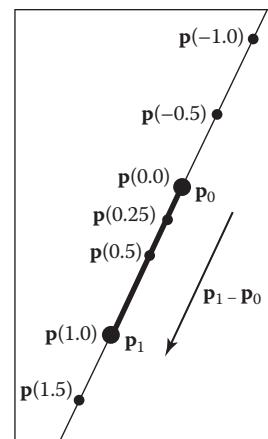


Figure 2.34. A 2D parametric line through \mathbf{p}_0 and \mathbf{p}_1 . The line segment defined by $t \in [0,1]$ is shown in bold.



例如，可以从两个隐式平面的交点形成3D线。通常，使用参数曲线更方便；它们将在以下各节中讨论。

2.5.6 二维参数曲线

参数曲线由一个参数控制，该参数可以被认为是一种沿着曲线连续移动的索引。这样的曲线具有形式

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

这里 (x, y) 是曲线上的一个点， t 是影响曲线的参数。对于给定的 t ，将存在由函数 g 和 h 确定的一些点。对于连续的 g 和 h ， t 的微小变化将产生 x 和 y 的微小变化。因此，随着 t 连续变化，点在连续曲线中被扫出。这是一个很好的功能，因为我们可以使用参数 t 在曲线上显式构造点。通常我们可以用矢量形式编写参数曲线

$$\mathbf{p} = f(t),$$

其中 f 是矢量值函数， $f : \mathbb{R} \mapsto \mathbb{R}^2$ 。这样的向量函数可以生成非常干净的代码，因此应该在可能的情况下使用它们。

我们可以把位置的曲线看作是时间的函数。曲线可以去任何地方，可以循环和交叉。我们也可以将曲线视为在任何点上具有速度。例如，点 $\mathbf{p}(t)$ 在 $t=2$ 附近缓慢行进，在 $t=2$ 和 $t=3$ 之间快速行进。这种类型的“移动点”词汇通常在讨论参数曲线时使用，即使曲线没有描述移动点。

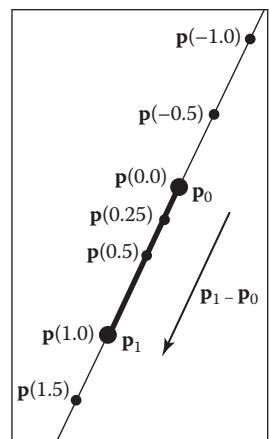
二维参数线

通过点 $\mathbf{p}_0 = (x_0, y_0)$ 和 $\mathbf{p}_1 = (x_1, y_1)$ 的2d中的参数线可以写为

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

由于 x 和 y 的公式具有类似的结构，我们可以使用 $\mathbf{p} = (x, y)$ 的向量形式（图2.34）：

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



通过 \mathbf{p}_0 和 \mathbf{p}_1 的2D参数化公制线。由 $t \in [0,1]$ 定义的线段以粗体示出。

You can read this in geometric form as: “start at point p_0 and go some distance toward p_1 determined by the parameter t .” A nice feature of this form is that $p(0) = p_0$ and $p(1) = p_1$. Since the point changes linearly with t , the value of t between p_0 and p_1 measures the fractional distance between the points. Points with $t < 0$ are to the “far” side of p_0 , and points with $t > 1$ are to the “far” side of p_1 .

Parametric lines can also be described as just a point \mathbf{o} and a vector \mathbf{d} :

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

When the vector \mathbf{d} has unit length, the line is *arc-length parameterized*. This means t is an exact measure of distance along the line. Any parametric curve can be arc-length parameterized, which is obviously a very convenient form, but not all can be converted analytically.

2D Parametric Circles

A circle with center (x_c, y_c) and radius r has a parametric form:

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

To ensure that there is a unique parameter ϕ for every point on the curve, we can restrict its domain: $\phi \in [0, 2\pi]$ or $\phi \in (-\pi, \pi]$ or any other half-open interval of length 2π .

An axis-aligned ellipse can be constructed by scaling the x and y parametric equations separately:

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

2.5.7 3D Parametric Curves

A 3D parametric curve operates much like a 2D parametric curve:

$$x = f(t),$$

$$y = g(t),$$

$$z = h(t).$$

For example, a spiral around the z -axis is written as:

$$x = \cos t,$$

$$y = \sin t,$$

$$z = t.$$

您可以将其以几何形式阅读为：“从点 p_0 开始，向由参数 t 确定的 p_1 走一段距离。”这种形式的一个很好的特点是 $p(0) = p_0$ 和 $p(1) = p_1$ 。由于点随 t 线性变化，因此 p_0 和 p_1 之间的 t 值测量点之间的分数距离。 $T < 0$ 的点到 p_0 的“远”侧， $t > 1$ 的点到 p_1 的“远”侧。

参数线也可以被描述为只是一个点 \mathbf{o} 和一个向量 \mathbf{d} :

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

当矢量 \mathbf{d} 具有单位长度时，线被弧长参数化。这意味着 t 是沿着线的距离的确切度量。任何参数曲线都可以进行弧长参数化，这显然是一种非常方便的形式，但不是所有的都可以解析地转换。

二维参数圆

具有中心 (x_c, y_c) 和半径 r 的圆具有参数形式:

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

为了确保曲线上的每个点都有一个唯一的参数 ϕ ，我们可以限制其域： $\phi \in [0, 2\pi]$ 或 $\phi \in (-\pi, \pi]$ 或任何其他长度为 2π 的半开区间。

可以通过分别缩放 x 和 y 参数方程来构造轴对齐的椭圆:

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

2.5.7 三维参数曲线

3D 参数曲线的操作方式与 2D 参数曲线非常相似:

$$x = f(t),$$

$$y = g(t),$$

$$z = h(t).$$

例如，围绕 z 轴的螺旋被写为:

$$x = \cos t,$$

$$y = \sin t,$$

$$z = t.$$



As with 2D curves, the functions f , g , and h are defined on a domain $D \subset \mathbb{R}$ if we want to control where the curve starts and ends. In vector form we can write

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{p}(t).$$

In this chapter we only discuss 3D parametric lines in detail. General 3D parametric curves are discussed more extensively in Chapter 15.

3D Parametric Lines

A 3D parametric line can be written as a straightforward extension of the 2D parametric line, e.g.,

$$\begin{aligned} x &= 2 + 7t, \\ y &= 1 + 2t, \\ z &= 3 - 5t. \end{aligned}$$

This is cumbersome and does not translate well to code variables, so we will write it in vector form:

$$\mathbf{p} = \mathbf{o} + t\mathbf{d},$$

where, for this example, \mathbf{o} and \mathbf{d} are given by

$$\begin{aligned} \mathbf{o} &= (2, 1, 3), \\ \mathbf{d} &= (7, 2, -5). \end{aligned}$$

Note that this is very similar to the 2D case. The way to visualize this is to imagine that the line passes through \mathbf{o} and is parallel to \mathbf{d} . Given any value of t , you get some point $\mathbf{p}(t)$ on the line. For example, at $t = 2$, $\mathbf{p}(t) = (2, 1, 3) + 2(7, 2, -5) = (16, 5, -7)$. This general concept is the same as for two dimensions (Figure 2.30).

As in 2D, a *line segment* can be described by a 3D parametric line and an interval $t \in [t_a, t_b]$. The line segment between two points \mathbf{a} and \mathbf{b} is given by $\mathbf{p}(t) = \mathbf{a} + t(\mathbf{b} - \mathbf{a})$ with $t \in [0, 1]$. Here $\mathbf{p}(0) = \mathbf{a}$, $\mathbf{p}(1) = \mathbf{b}$, and $\mathbf{p}(0.5) = (\mathbf{a} + \mathbf{b})/2$, the midpoint between \mathbf{a} and \mathbf{b} .

A *ray*, or *half-line*, is a 3D parametric line with a half-open interval, usually $[0, \infty)$. From now on we will refer to all lines, line segments, and rays as “rays.” This is sloppy, but corresponds to common usage and makes the discussion simpler.

与二维曲线一样，如果我们想控制曲线的开始和结束位置，函数f, g和h是在域D∈R上定义的。在矢量形式，我们可以写

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{p}(t).$$

在本章中，我们只详细讨论3D参数线。一般的3d参数曲线在第15章中有更广泛的讨论。

参数曲线是p: R→R3的范围。

三维参数线

三维参数线可以写成二维参数线的直接扩展，例如：

$$\begin{aligned} x &= 2 + 7t, \\ y &= 1 + 2t, \\ z &= 3 - 5t. \end{aligned}$$

这很麻烦，并且不能很好地转换为代码变量，因此我们将以矢量形式编写它：

$$\mathbf{p} = \mathbf{o} + t\mathbf{d},$$

其中，对于这个例子，o和d由

$$\begin{aligned} \mathbf{o} &= (2, 1, 3), \\ \mathbf{d} &= (7, 2, -5). \end{aligned}$$

请注意，这与2D情况非常相似。可视化这一点的方法是想象线穿过o并且平行于d。给定t的任何值，你会得到一些点p (t)就行了。例如，在t=2时， $p(t)=(2\ 1\ 3)+2(7\ 2\ -5)=(16\ 5\ -7)$ 。这个一般概念与二维相同（图2.30）。

如在2D中，线段可以用3D参数线和间隔 $t \in [ta tb]$ 来描述。两点a和b之间的线段由 $t \in [0 1]$ 的 $p(t)=a+t(b-a)$ 给出。这里 $p(0)=a$, $p(1)=b$, $p(0.5) = (a+b)/2$, a和b之间的中点。

射线，或半线，是具有半开间隔的3d参数线，usually $[0 \infty)$ 。从现在开始，我们将所有线，线段和射线称为“射线”。

这是草率的，但对应于常见的用法，并使讨论更简单。

2.5.8 3D Parametric Surfaces

The parametric approach can be used to define surfaces in 3D space in much the same way we define curves, except that there are two parameters to address the two-dimensional area of the surface. These surfaces have the form

$$\begin{aligned}x &= f(u, v), \\y &= g(u, v), \\z &= h(u, v).\end{aligned}$$

or, in vector form,

$$\begin{bmatrix}x \\ y \\ z\end{bmatrix} = \mathbf{p}(u, v).$$

The parametric surface is the range of the function \mathbf{p} : $\mathbb{R}^2 \rightarrow \mathbb{R}^3$.

Pretend for the sake of argument that the Earth is exactly spherical.

The θ and ϕ here may or may not seem reversed depending on your background; the use of these symbols varies across disciplines. In this book we will always assume the meaning of θ and ϕ used in Equation (2.24) and depicted in Figure 2.35.

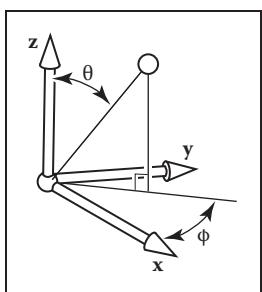


Figure 2.35. The geometry for spherical coordinates.

Ideally, we'd like to write this in vector form, but it isn't feasible for this particular parametric form.

We would also like to be able to find the (θ, ϕ) for a given (x, y, z) . If we assume that $\phi \in (-\pi, \pi]$ this is easy to do using the *atan2* function from Equation (2.2):

$$\begin{aligned}\theta &= \arccos(z / \sqrt{x^2 + y^2 + z^2}), \\ \phi &= \text{atan2}(y, x).\end{aligned}\quad (2.25)$$

With implicit surfaces, the derivative of the function f gave us the surface normal. With parametric surfaces, the derivatives of \mathbf{p} also give information about the surface geometry.

Consider the function $\mathbf{q}(t) = \mathbf{p}(t, v_0)$. This function defines a parametric curve obtained by varying u while holding v fixed at the value v_0 . This curve, called an *isoparametric curve* (or sometimes "isoparm" for short) lies in the surface. The derivative of \mathbf{q} gives a vector tangent to the curve, and since the curve

2.5.8 三维参数曲面

参数化方法可用于在3D空间中定义曲面，与定义曲线的方式大致相同，只是有两个参数来处理曲面的二维区域。这些表面具有形式

$$\begin{aligned}x &= f(u, v), \\y &= g(u, v), \\z &= h(u, v).\end{aligned}$$

或者，以矢量形式

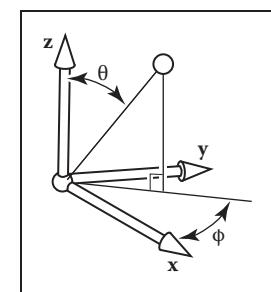
$$\begin{bmatrix}x \\ y \\ z\end{bmatrix} = \mathbf{p}(u, v).$$

参数曲面是函数p的范围
: R2→R3。

为了argument的缘故，假裝
地球是exactly球形的。

这里的θ和φ可能会或可
能不会根据你的背地而
颠倒;这些符号的使用在d
isciplines上有所不同。

在本书中，我们将始终
假设方程 (2.24) 中使用
的θ和φ的含义，并在图2
.35中使用depicted。



球坐标的几何。

$$\begin{aligned}x &= r \cos \phi \sin \theta \\y &= r \sin \phi \sin \theta \\z &= r \cos \theta.\end{aligned}\quad (2.24)$$

例子。例如，地球表面上的一个点可以通过
经度和纬度两个参数。如果我们将原点定义为地球的中心，并且让r是地球
的半径，那么以原点为中心的球面坐标系（图2.35）让我们推导出参数方
程

$$\begin{aligned}\theta &= \arccos(z / \sqrt{x^2 + y^2 + z^2}), \\ \phi &= \text{atan2}(y, x).\end{aligned}\quad (2.25)$$

理想情况下，我们希望以矢量形式编写它，但对于这种特定的参数形式来说是不可行的。
我们也希望能够找到给定 (x, y, z) 的 (θ, ϕ) 。如果我们假设 $\phi \in (-\pi, \pi]$ 这很容易使用来自Equation(2.2)的atan2函数来完成：

对于隐式曲面，函数f的导数给了我们曲面法线。对于参数曲面，p的导数也给出了有关曲面几何的信息。

考虑函数 $q(t)=p(t, v_0)$ 。该函数定义了通过改变u而将v固定在值 v_0 而获得的参数曲线。这条曲线，称为等参数曲线（或有时简称“等参数”）位于曲面中。Q的导数给出了与曲线相切的矢量，并且由于曲线



lies in the surface the vector \mathbf{q}' also lies in the surface. Since it was obtained by varying one argument of \mathbf{p} , the vector \mathbf{q}' is the partial derivative of \mathbf{p} with respect to u , which we'll denote \mathbf{p}_u . A similar argument shows that the partial derivative \mathbf{p}_v gives the tangent to the isoparametric curves for constant u , which is a second tangent vector to the surface.

The derivative of \mathbf{p} , then, gives two tangent vectors at any point on the surface. The normal to the surface may be found by taking the cross product of these vectors: since both are tangent to the surface, their cross product, which is perpendicular to both tangents, is normal to the surface. The right-hand rule for cross products provides a way to decide which side is the front, or outside, of the surface; we will use the convention that the vector

$$\mathbf{n} = \mathbf{p}_u \times \mathbf{p}_v$$

points toward the outside of the surface.

2.5.9 Summary of Curves and Surfaces

Implicit curves in 2D or surfaces in 3D are defined by scalar-valued functions of two or three variables, $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ or $f : \mathbb{R}^3 \rightarrow \mathbb{R}$, and the surface consists of all points where the function is zero:

$$S = \{\mathbf{p} | f(\mathbf{p}) = 0\}.$$

Parametric curves in 2D or 3D are defined by vector-valued functions of one variable, $\mathbf{p} : D \subset \mathbb{R} \rightarrow \mathbb{R}^2$ or $\mathbf{p} : D \subset \mathbb{R} \rightarrow \mathbb{R}^3$, and the curve is swept out as t varies over all of D :

$$S = \{\mathbf{p}(t) | t \in D\}.$$

Parametric surfaces in 3D are defined by vector-valued functions of two variables, $\mathbf{p} : D \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$, and the surface consists of the images of all points (u, v) in the domain:

$$S = \{\mathbf{p}(t) | (u, v) \in D\}.$$

For implicit curves and surfaces, the normal vector is given by the derivative of f (the gradient), and the tangent vector (for a curve) or vectors (for a surface) can be derived from the normal by constructing a basis.

For parametric curves and surfaces, the derivative of \mathbf{p} gives the tangent vector (for a curve) or vectors (for a surface), and the normal vector can be derived from the tangents by constructing a basis.



在表面矢量 \mathbf{q}' 也于表面。由于它是通过改变 \mathbf{p} 的一个参数获得的，向量 \mathbf{q}' 是 \mathbf{p} 相对于 u 的偏导数，我们将表示 \mathbf{p}_u 。类似的论点表明，偏导数 \mathbf{p}_v 给出了常数 u 的等参曲线的切线，这是曲面的第二个切线矢量。

然后， \mathbf{p} 的导数给出了 sur 上任意一点的两个切向量。通过取这些向量的叉积可以找到表面的法线：由于两者都与表面相切，因此垂直于两条切线的叉积与表面是法线。交叉产品的右手规则提供了一种方法来决定哪一侧是表面的正面或外部；我们将使用向量的约定

$$\mathbf{n} = \mathbf{p}_u \times \mathbf{p}_v$$

指向表面的外侧。

2.5.9 曲线和曲面摘要

2D 中的隐式曲线或 3D 中的曲面由两个或三个变量的标量值函数定义， $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ 或 $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ ，曲面由函数为零的所有点组成：

$$S = \{\mathbf{p} | f(\mathbf{p}) = 0\}.$$

2D 或 3D 中的参数曲线由一个变量的矢量值函数定义， $\mathbf{p} : D \subset \mathbb{R} \rightarrow \mathbb{R}^2$ 或 $\mathbf{p} : D \subset \mathbb{R} \rightarrow \mathbb{R}^3$ ，并且随着 t 在所有 D 上变化，曲线被扫出：

$$S = \{\mathbf{p}(t) | t \in D\}.$$

3D 中的参数曲面由两个变量的矢量值函数定义， $\mathbf{p} : D \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$ ，曲面由域中所有点 (u, v) 的图像组成：

$$S = \{\mathbf{p}(t) | (u, v) \in D\}.$$

对于隐式曲线和曲面，法向矢量由 f (梯度) 的导数给出，切向矢量（对于曲线）或矢量（对于曲面）可以通过构建基从法线导出。

对于参数曲线和曲面， \mathbf{p} 的导数给出切线矢量（对于曲线）或矢量（对于曲面），并且法向量可以通过构造基从切线导出。

2.6 Linear Interpolation

Perhaps the most common mathematical operation in graphics is *linear interpolation*. We have already seen an example of linear interpolation of position to form line segments in 2D and 3D, where two points a and b are associated with a parameter t to form the line $\mathbf{p} = (1 - t)\mathbf{a} + t\mathbf{b}$. This is *interpolation* because \mathbf{p} goes through a and b exactly at $t = 0$ and $t = 1$. It is *linear* interpolation because the weighting terms t and $1 - t$ are linear polynomials of t .

Another common linear interpolation is among a set of positions on the x -axis: x_0, x_1, \dots, x_n , and for each x_i we have an associated height, y_i . We want to create a continuous function $y = f(x)$ that interpolates these positions, so that f goes through every data point, i.e., $f(x_i) = y_i$. For linear interpolation, the points (x_i, y_i) are connected by straight line segments. It is natural to use parametric line equations for these segments. The parameter t is just the fractional distance between x_i and x_{i+1} :

$$f(x) = y_i + \frac{x - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i). \quad (2.26)$$

Because the weighting functions are linear polynomials of x , this is linear interpolation.

The two examples above have the common form of linear interpolation. We create a variable t that varies from 0 to 1 as we move from data item A to data item B . Intermediate values are just the function $(1 - t)A + tB$. Notice that Equation (2.26) has this form with

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

2.7 Triangles

Triangles in both 2D and 3D are the fundamental modeling primitive in many graphics programs. Often information such as color is tagged onto triangle vertices, and this information is interpolated across the triangle. The coordinate system that makes such interpolation straightforward is called *barycentric coordinates*; we will develop these from scratch. We will also discuss 2D triangles, which must be understood before we can draw their pictures on 2D screens.

2.6 线性插值

也许图形学中最常见的数学运算是线性间距。我们已经看到了位置线性插值以在2D和3D中形成线段的示例，其中两个点a和b与参数t相关联以形成线 $p= (1-t)a+t b$ 。这是插值，因为p恰好在t=0和t=1处经过a和b。它是线性插值，因为加权项t和 $1-t$ 是t的线性多项式。

另一种常见的线性插值是在x轴上的一组位置中： $x_0 x_1 \dots, x_n$ ，并且对于每个 x_i ，我们有一个相关的高度， y_i 。我们想创建一个连续函数 $y=f(x)$ ，对这些位置进行插值，使得 f 经过每个数据点，即 $f(x_i)=y_i$ 。对于线性插值，点 (x_i, y_i) 由直线段连接。对于这些线段使用参数线方程是很自然的。参数t只是 x_i 和 x_{i+1} 之间的分数距离：

$$f(x) = y_i + \frac{x - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i). \quad (2.26)$$

因为加权函数是x的线性多项式，所以这是线性间极化。

上面的两个示例具有线性插值的常见形式。当我们从数据项A移动到数据项B时，我们创建了一个从0到1的变量t。中间值只是函数 $(1-t)a+tB$ 。请注意，方程 (2.26) 具有这种形式与

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

2.7 Triangles

2D和3d中的三角形都是许多图形程序中的基本建模基元。通常，颜色等信息被标记到三角形顶点上，并且这些信息被插值到三角形上。使这种插值变得简单的坐标系称为重心坐标；我们将从头开始开发这些坐标。我们还将讨论2D三角形，在我们可以绘制它们的图片之前，必须了解这些三角形。



2.7.1 2D Triangles

If we have a 2D triangle defined by 2D points \mathbf{a} , \mathbf{b} , and \mathbf{c} , we can first find its area:

$$\begin{aligned} \text{area} &= \frac{1}{2} \begin{vmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{vmatrix} \\ &= \frac{1}{2} (x_a y_b + x_b y_c + x_c y_a - x_a y_c - x_b y_a - x_c y_b). \end{aligned} \quad (2.27)$$

The derivation of this formula can be found in Section 5.3. This area will have a positive sign if the points \mathbf{a} , \mathbf{b} , and \mathbf{c} are in counterclockwise order and a negative sign, otherwise.

Often in graphics, we wish to assign a property, such as color, at each triangle vertex and smoothly interpolate the value of that property across the triangle. There are a variety of ways to do this, but the simplest is to use *barycentric* coordinates. One way to think of barycentric coordinates is as a nonorthogonal coordinate system as was discussed briefly in Section 2.4.2. Such a coordinate system is shown in Figure 2.36, where the coordinate origin is \mathbf{a} and the vectors from \mathbf{a} to \mathbf{b} and \mathbf{c} are the basis vectors. With that origin and those basis vectors,

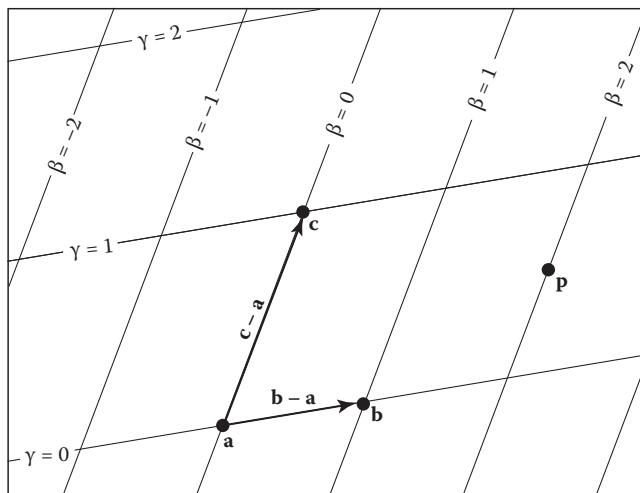


Figure 2.36. A 2D triangle with vertices \mathbf{a} , \mathbf{b} , \mathbf{c} can be used to set up a nonorthogonal coordinate system with origin \mathbf{a} and basis vectors $(\mathbf{b} - \mathbf{a})$ and $(\mathbf{c} - \mathbf{a})$. A point is then represented by an ordered pair (β, γ) . For example, the point $\mathbf{p} = (2.0, 0.5)$, i.e., $\mathbf{p} = \mathbf{a} + 2.0(\mathbf{b} - \mathbf{a}) + 0.5(\mathbf{c} - \mathbf{a})$.



2.7.1 2D Triangles

如果我们有一个由2D点 \mathbf{a} , \mathbf{b} 和 \mathbf{c} 定义的2d三角形，我们可以首先找到它的面积：

$$\begin{aligned} \text{area} &= \frac{1}{2} \begin{vmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{vmatrix} \\ &= \frac{1}{2} (x_a y_b + x_b y_c + x_c y_a - x_a y_c - x_b y_a - x_c y_b). \end{aligned} \quad (2.27)$$

这个公式的推导可以在第5.3节中找到。如果点 \mathbf{a} , \mathbf{b} 和 \mathbf{c} 按逆时针顺序，则此区域将具有正号，否则为负号。

通常在图形中，我们希望在每个三角形顶点分配一个属性，例如颜色，并在三角形上平滑地插入该属性的值。有多种方法可以做到这一点，但最简单的是使用重心坐标。考虑重心坐标的一种方法是作为一个非正心坐标系，如第2.4.2节中简要讨论的那样。这样的坐标系如图2.36所示，其中坐标原点是 \mathbf{a} ，从 \mathbf{a} 到 \mathbf{b} 和 \mathbf{c} 的矢量是基矢量。用那个原点和那些基向量

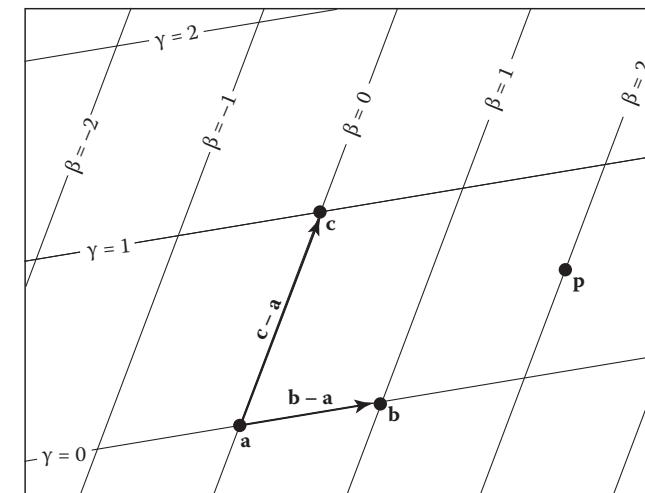


图2.36。具有顶点 \mathbf{a} , \mathbf{b} , \mathbf{c} 的2D三角形可用于设置具有原点 \mathbf{a} 和基向量 $(\mathbf{b} - \mathbf{a})$ 和 $(\mathbf{c} - \mathbf{a})$ 的非角坐标系。

由有序对 (β, γ) 表示。例如，点 $\mathbf{p}=(2.0, 0.5)$ ，即 $\mathbf{p}=\mathbf{a}+2.0(\mathbf{b}-\mathbf{a})+0.5(\mathbf{c}-\mathbf{a})$ 。

any point \mathbf{p} can be written as

$$\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}). \quad (2.28)$$

Note that we can reorder the terms in Equation (2.28) to get

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

Often people define a new variable α to improve the symmetry of the equations:

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

which yields the equation

$$\mathbf{p}(\alpha, \beta, \gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}, \quad (2.29)$$

with the constraint that

$$\alpha + \beta + \gamma = 1. \quad (2.30)$$

Barycentric coordinates seem like an abstract and unintuitive construct at first, but they turn out to be powerful and convenient. You may find it useful to think of how street addresses would work in a city where there are two sets of parallel streets, but where those sets are not at right angles. The natural system would essentially be barycentric coordinates, and you would quickly get used to them. Barycentric coordinates are defined for all points on the plane. A particularly nice feature of barycentric coordinates is that a point \mathbf{p} is inside the triangle formed by \mathbf{a} , \mathbf{b} , and \mathbf{c} if and only if

$$\begin{aligned} 0 < \alpha < 1, \\ 0 < \beta < 1, \\ 0 < \gamma < 1. \end{aligned}$$

If one of the coordinates is zero and the other two are between zero and one, then you are on an edge. If two of the coordinates are zero, then the other is one, and you are at a vertex. Another nice property of barycentric coordinates is that Equation (2.29) in effect mixes the coordinates of the three vertices in a smooth way. The same mixing coefficients (α, β, γ) can be used to mix other properties, such as color, as we will see in the next chapter.

Given a point \mathbf{p} , how do we compute its barycentric coordinates? One way is to write Equation (2.28) as a linear system with unknowns β and γ , solve, and set $\alpha = 1 - \beta - \gamma$. That linear system is

$$\begin{bmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x_p - x_a \\ y_p - y_a \end{bmatrix}. \quad (2.31)$$

任何点p都可以写成

$$\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}). \quad (2.28)$$

请注意，我们可以重新排序方程 (2.28) 中的项以获得

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

通常人们定义一个新的变量 α 来提高方程的对称性:

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

这就产生了方程

$$\mathbf{p}(\alpha, \beta, \gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}, \quad (2.29)$$

约束是

$$\alpha + \beta + \gamma = 1. \quad (2.30)$$

Barycentric坐标起初看起来像是一个抽象而不直观的构造，但结果却是强大而方便的。您可能会发现，考虑街道地址在有两组平行街道但这些街道不是直角的城市中如何工作是很有用的。自然系统基本上是中心坐标，你很快就会习惯它们。为平面上的所有点定义重心坐标。重心坐标的一个特别好的特征是，当且仅当a, b和c形成的三角形内有一个点p

$$\begin{aligned} 0 < \alpha < 1, \\ 0 < \beta < 1, \\ 0 < \gamma < 1. \end{aligned}$$

如果其中一个坐标为零，另外两个在零到一之间，那么你就在一条边上。如果其中两个坐标为零，则另一个是一个，并且您位于顶点。重心坐标的另一个很好的特性是方程 (2.29) 实际上以平滑的方式混合了三个顶点的坐标。相同的混合系数 (α, β, γ) 可用于混合其他属性，如颜色，我们将在下一章中看到。

给定一个点p，我们如何计算它的重心坐标？一种方法是将方程 (2.28) 写成具有未知数 β 和 γ 的线性系统，求解并设置 $\alpha=1-\beta-\gamma$ 。该线性系统是完整的xb-xa

Although it is straightforward to solve Equation (2.31) algebraically, it is often fruitful to compute a direct geometric solution.

One geometric property of barycentric coordinates is that they are the signed scaled distance from the lines through the triangle sides, as is shown for β in Figure 2.37. Recall from Section 2.5.2 that evaluating the equation $f(x, y)$ for the line $f(x, y) = 0$ returns the scaled signed distance from (x, y) to the line. Also recall that if $f(x, y) = 0$ is the equation for a particular line, so is $kf(x, y) = 0$ for any nonzero k . Changing k scales the distance and controls which side of the line has positive signed distance, and which negative. We would like to choose k such that, for example, $kf(x, y) = \beta$. Since k is only one unknown, we can force this with one constraint, namely that at point b we know $\beta = 1$. So if the line $f_{ac}(x, y) = 0$ goes through both a and c , then we can compute β for a point (x, y) as follows:

$$\beta = \frac{f_{ac}(x, y)}{f_{ac}(x_b, y_b)}, \quad (2.32)$$

and we can compute γ and α in a similar fashion. For efficiency, it is usually wise to compute only two of the barycentric coordinates directly and to compute the third using Equation (2.30).

To find this “ideal” form for the line through p_0 and p_1 , we can first use the technique of Section 2.5.2 to find *some* valid implicit lines through the vertices. Equation (2.17) gives us

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

Note that $f_{ab}(x_c, y_c)$ probably does not equal one, so it is probably not the ideal form we seek. By dividing through by $f_{ab}(x_c, y_c)$ we get

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

The presence of the division might worry us because it introduces the possibility of divide-by-zero, but this cannot occur for triangles with areas that are not near zero. There are analogous formulas for α and β , but typically only one is needed:

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

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

Another way to compute barycentric coordinates is to compute the areas A_a , A_b , and A_c , of subtriangles as shown in Figure 2.38. Barycentric coordinates obey

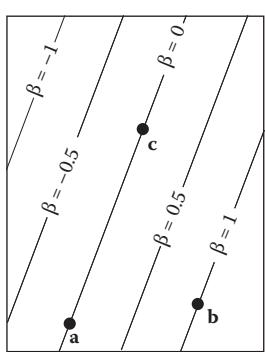


Figure 2.37. The barycentric coordinate β is the signed scaled distance from the line through a and c .

虽然用代数求解方程 (2.31) 很简单，但计算直接几何解通常是富有成效的。

重心坐标的一个几何属性是它们是通过三角形边的线的带符号缩放距离，如图2.37中的 β 所示。回顾第2.5.2节，计算 $f(x, y)=0$ 的方程 $f(x, y)$ 返回从 (x, y) 到该线的缩放符号距离。还记得，如果 $f(x, y)=0$ 是特定线的方程，那么对于任何非零 k ， $kf(x, y)=0$ 也是如此。改变 k 缩放距离并控制线的哪一侧有正符号距离，哪一侧有负符号距离。我们想选择 k ，例如， $kf(x, y)=\beta$ 。由于 k 只有一个未知数，我们可以用一个约束强制这个，即在 b 点我们知道 $\beta=1$ 。因此，如果线 $f_{ac}(x, y)=0$ 同时经过 a 和 c ，那么我们可以计算点 (x, y) 的 β ，如下所示：

$$\beta = \frac{f_{ac}(x, y)}{f_{ac}(x_b, y_b)}, \quad (2.32)$$

我们可以用类似的方式计算 γ 和 α 。为了提高效率，通常明智的做法是直接计算两个重心坐标，并使用公式 (2.30) 计算第三个坐标。

为了找到通过 p_0 和 p_1 的线的这种“理想”形式，我们可以首先使用第2.5.2节的技术找到通过顶点的一些有效的隐式线。方程 (2.17) 给出了我们

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

请注意， $f_{ab}(x_c, y_c)$ 可能不等于一，因此它可能不是我们寻求的理想形式。通过除以 $f_{ab}(x_c, y_c)$ 我们得到

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

除法的存在可能会让我们担心，因为它引入了被零除的可能性，但对于面积不接近零的三角形来说，这是不可能发生的。 α 和 β 有类似的公式，但通常只需要一个：

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

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

计算重心坐标的另一种方法是计算子矩形的区域 A_a ， ab 和 ac ，如图2.38所示。重心坐标服从

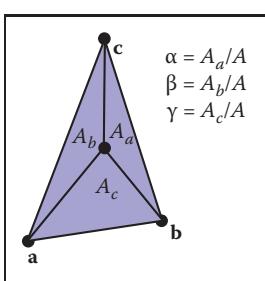
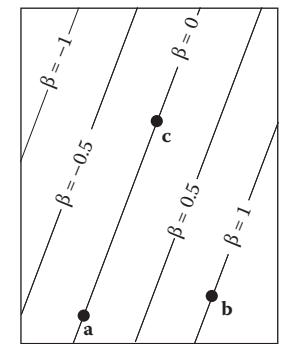
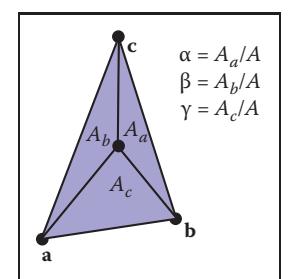


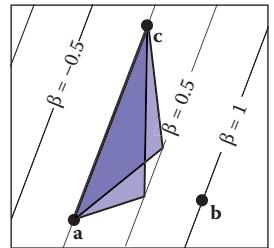
Figure 2.38. The barycentric coordinates are proportional to the areas of the three subtriangles shown.



Bary中心坐标 β 是通过 a 和 c 的线的带符号缩放距离。



以bary为中心的坐标对应于所示的三个子矩形的区域。



the rule

$$\begin{aligned}\alpha &= A_a/A, \\ \beta &= A_b/A, \\ \gamma &= A_c/A,\end{aligned}\quad (2.33)$$

where A is the area of the triangle. Note that $A = A_a + A_b + A_c$, so it can be computed with two additions rather than a full area formula. This rule still holds for points outside the triangle if the areas are allowed to be signed. The reason for this is shown in Figure 2.39. Note that these are signed areas and will be computed correctly as long as the same signed area computation is used for both A and the subtriangles A_a , A_b , and A_c .

Figure 2.39. The area of the two triangles shown is base times height and are thus the same, as is any triangle with a vertex on the $\beta = 0.5$ line. The height and thus the area is proportional to β .

2.7.2 3D Triangles

One wonderful thing about barycentric coordinates is that they extend almost transparently to 3D. If we assume the points a , b , and c are 3D, then we can still use the representation

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

Now, as we vary β and γ , we sweep out a plane.

The normal vector to a triangle can be found by taking the cross product of any two vectors in the plane of the triangle (Figure 2.40). It is easiest to use two of the three edges as these vectors, for example,

$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}). \quad (2.34)$$

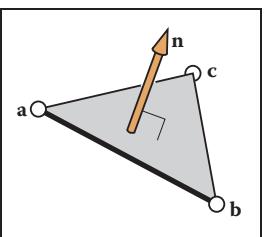


Figure 2.40. The normal vector of the triangle is perpendicular to all vectors in the plane of the triangle, and thus perpendicular to the edges of the triangle.

The area of the triangle can be found by taking the length of the cross product:

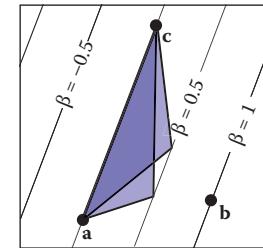
$$\text{area} = \frac{1}{2} \|(\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})\|. \quad (2.35)$$

Note that this is *not* a signed area, so it cannot be used directly to evaluate barycentric coordinates. However, we can observe that a triangle with a “clockwise” vertex order will have a normal vector that points in the opposite direction to the normal of a triangle in the same plane with a “counterclockwise” vertex order. Recall that

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \phi,$$

规则

$$\begin{aligned}\alpha &= A_a/A, \\ \beta &= A_b/A, \\ \gamma &= A_c/A,\end{aligned}\quad (2.33)$$



其中A是三角形的面积。请注意， $A=Aa+ab+ac$ ，因此可以通过两次加法而不是全面积公式来计算。如果允许签署区域，则此规则仍然适用于三角形以外的点。其原因如图2.39所示。请注意，这些是有符号区域，只要对a和子区域Aa、ab和ac使用相同的有符号区域计算，就会正确计算。

所示的两个三角形的面积是基乘高度，因此是相同的，任何三角形的顶点在 $\beta=0.5$ 线的高度并且因此面积与 β 成比例。

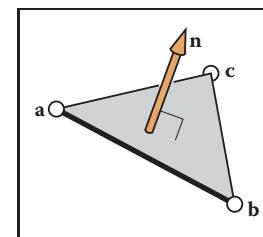
2.7.2 3D Triangles

关于重心坐标的一个奇妙之处在于它们几乎透明地延伸到3D，如果我们假设点a, b和c是3D，那么我们仍然可以使用表示

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

现在，当我们改变 β 和 γ 时，我们扫出一个平面。

三角形的法向量可以通过取三角形平面上任意两个向量的叉积来找到（图2.40）。使用三条边中的两条作为这些向量是最简单的，例如



$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}). \quad (2.34)$$

请注意，此法向量不一定是单位长度，并且它服从交叉乘积的右手规则。

三角形的面积可以通过取叉积的长度来找到：

$$\text{area} = \frac{1}{2} \|(\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})\|. \quad (2.35)$$

请注意，这不是一个有符号的区域，因此它不能直接用于评估barycen坐标。然而，我们可以观察到，具有“顺时针”顶点顺序的三角形将具有法向量，该法向量指向与具有“逆时针”顶点顺序的同一平面中的三角形的法线相反的方向。回想一下

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \phi,$$



where ϕ is the angle between the vectors. If \mathbf{a} and \mathbf{b} are parallel, then $\cos \phi = \pm 1$, and this gives a test of whether the vectors point in the same or opposite directions. This, along with Equations (2.33), (2.34), and (2.35), suggest the formulas

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

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

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

where \mathbf{n} is Equation (2.34) evaluated with vertices \mathbf{a} , \mathbf{b} , and \mathbf{c} ; \mathbf{n}_a is Equation (2.34) evaluated with vertices \mathbf{b} , \mathbf{c} , and \mathbf{p} , and so on, i.e.,

$$\begin{aligned}\mathbf{n}_a &= (\mathbf{c} - \mathbf{b}) \times (\mathbf{p} - \mathbf{b}), \\ \mathbf{n}_b &= (\mathbf{a} - \mathbf{c}) \times (\mathbf{p} - \mathbf{c}), \\ \mathbf{n}_c &= (\mathbf{b} - \mathbf{a}) \times (\mathbf{p} - \mathbf{a}).\end{aligned}\tag{2.36}$$

Frequently Asked Questions

- Why isn't there vector division?

It turns out that there is no “nice” analogy of division for vectors. However, it is possible to motivate the quaternions by examining this question in detail (see Hoffmann’s book referenced in the chapter notes).

- Is there something as clean as barycentric coordinates for polygons with more than three sides?

Unfortunately there is not. Even convex quadrilaterals are much more complicated. This is one reason triangles are such a common geometric primitive in graphics.

- Is there an implicit form for 3D lines?

No. However, the intersection of two 3D planes defines a 3D line, so a 3D line can be described by two simultaneous implicit 3D equations.



其中 ϕ 是矢量之间的角度。如果 \mathbf{a} 和 \mathbf{b} 是平行的，那么 $\cos\phi=\pm 1$ ，这给出了向量是否指向相同或相反方向的测试。这与方程 (2.33)，(2.34) 和 (2.35) 一起提出了公式

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

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

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

其中 \mathbf{n} 是用顶点 \mathbf{a} , \mathbf{b} 和 \mathbf{c} 评估的方程 (2.34); \mathbf{n}_a 是用顶点 \mathbf{b} , \mathbf{c} 和 \mathbf{p} 评估的方程 (2.34)，依此类推，即

$$\begin{aligned}\mathbf{n}_a &= (\mathbf{c} - \mathbf{b}) \times (\mathbf{p} - \mathbf{b}), \\ \mathbf{n}_b &= (\mathbf{a} - \mathbf{c}) \times (\mathbf{p} - \mathbf{c}), \\ \mathbf{n}_c &= (\mathbf{b} - \mathbf{a}) \times (\mathbf{p} - \mathbf{a}).\end{aligned}\tag{2.36}$$

常见问题

- 为什么没有向量划分？

事实证明，向量没有“好”的划分类比。但是，可以通过详细检查这个问题来激励四元数（参见章节注释中引用的霍夫曼的书）。

- 是否有像具有三个以上边的多边形的barycentric坐标一样干净的东西？

不幸的是没有。即使是凸四边形也要复杂得多。这是三角形是图形中常见的几何基元的一个原因。

- 3D线是否有隐式形式？

非也。然而，两个3d平面的交点定义了一条3d线，因此一条3d线可以用两个同时隐含的3d方程来描述。

Notes

The history of vector analysis is particularly interesting. It was largely invented by Grassman in the mid-1800s but was ignored and reinvented later (Crowe, 1994). Grassman now has a following in the graphics field of researchers who are developing *Geometric Algebra* based on some of his ideas (Doran & Lasenby, 2003). Readers interested in why the particular scalar and vector products are in some sense the right ones, and why we do not have a commonly used vector division, will find enlightenment in the concise *About Vectors* (Hoffmann, 1975). Another important geometric tool is the *quaternion* invented by Hamilton in the mid-1800s. Quaternions are useful in many situations, but especially where orientations are concerned (Hanson, 2005).

Exercises

1. The *cardinality* of a set is the number of elements it contains. Under IEEE floating point representation (Section 1.5), what is the cardinality of the *floats*?
2. Is it possible to implement a function that maps 32-bit integers to 64-bit integers that has a well-defined inverse? Do all functions from 32-bit integers to 64-bit integers have well-defined inverses?
3. Specify the unit cube (x -, y -, and z -coordinates all between 0 and 1 inclusive) in terms of the Cartesian product of three intervals.
4. If you have access to the natural log function $\ln(x)$, specify how you could use it to implement a $\log(b, x)$ function where b is the base of the log. What should the function do for negative b values? Assume an IEEE floating point implementation.
5. Solve the quadratic equation $2x^2 + 6x + 4 = 0$.
6. Implement a function that takes in coefficients A , B , and C for the quadratic equation $Ax^2 + Bx + C = 0$ and computes the two solutions. Have the function return the number of valid (not NaN) solutions and fill in the return arguments so the smaller of the two solutions is first.
7. Show that the two forms of the quadratic formula on page 17 are equivalent (assuming exact arithmetic) and explain how to choose one for each root in

Notes

矢量分析的历史特别有趣。它在很大程度上是由格拉斯曼在19世纪中期发明的，但后来被忽略和重新发明 (Crowe, 1994)。格拉斯曼现在在图形领域有一个追随者，他们正在根据他的一些想法开发几何代数 (Doran & Lasenby, 2003)。有兴趣了解为什么特定的标量和矢量产品在某种意义上是正确的，以及为什么我们没有常用的矢量划分的读者会在简明的关于矢量 (Hoffmann, 1975) 中找到启示。另一个重要的几何工具是汉密尔顿在19世纪中期发明的四元数，四元数在许多情况下都很有用，但特别是在取向方面 (Hanson, 2005)。

Exercises

- 1.集合的基数是它包含的元素数。在IEEE浮点表示（第1.5节）下，浮点数的基数是多少？
- 2.是否有可能实现一个将32位整数映射到具有明确定义的逆的64位整数的函数？从32位整数到64位整数的所有函数都有明确定义的反转吗？
- 3.根据三个区间的笛卡尔积指定单位立方体（ x 、 y 和 z 坐标均在0和1之间）。
- 4.如果您有权访问自然日志函数 $\ln(x)$ ，请指定如何使用它来实现日志(b
 x)函数，其中 b 是日志的基础。函数对负 b 值应该做什么？假设IEEE浮点实现。
- 5.求解二次方程 $2x^2+6x+4=0$ 。
- 6.实现一个函数，该函数接受二次方程 $Ax^2+Bx+C=0$ 的系数 a ， B 和 C ，并计算两个解。让函数返回有效（不是NaN）解决方案的数量，并填写返回参数，以便两个解决方案中较小的解决方案是第一个。
- 7.显示第17页二次公式的两种形式是等价的（假设精确算术），并解释如何为每个根选择一个



order to avoid subtracting nearly equal floating point numbers, which leads to loss of precision.

8. Show by counterexample that it is not always true that for 3D vectors \mathbf{a} , \mathbf{b} , and \mathbf{c} , $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \times \mathbf{b}) \times \mathbf{c}$.
9. Given the nonparallel 3D vectors \mathbf{a} and \mathbf{b} , compute a right-handed orthonormal basis such that \mathbf{u} is parallel to \mathbf{a} and \mathbf{v} is in the plane defined by \mathbf{a} and \mathbf{b} .
10. What is the gradient of $f(x, y, z) = x^2 + y - 3z^3$?
11. What is a parametric form for the axis-aligned 2D ellipse?
12. What is the implicit equation of the plane through 3D points $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$? What is the parametric equation? What is the normal vector to this plane?
13. Given four 2D points \mathbf{a}_0 , \mathbf{a}_1 , \mathbf{b}_0 , and \mathbf{b}_1 , design a robust procedure to determine whether the line segments $\mathbf{a}_0\mathbf{a}_1$ and $\mathbf{b}_0\mathbf{b}_1$ intersect.
14. Design a robust procedure to compute the barycentric coordinates of a 2D point with respect to three 2D non-collinear points.



为了避免减去几乎相等的浮点数，这导致精度损失。

8.通过反例表明，对于3D向量 \mathbf{a} , \mathbf{b} 和 \mathbf{c} , $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \times \mathbf{b}) \times \mathbf{c}$ 并不总是如此。

9.给定非平行的3d向量 \mathbf{a} 和 \mathbf{b} ，计算右旋正交基，使得 \mathbf{u} 平行于 \mathbf{a} 并且 \mathbf{v} 位于由 \mathbf{a} 和 \mathbf{b} 定义的平面中。

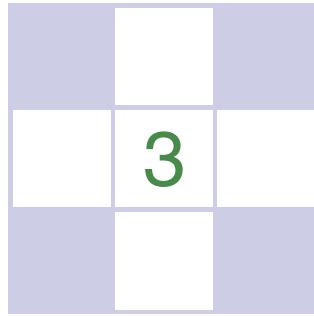
10.F (x, y, z) = x²+y-3z³的梯度是多少？

11.什么是轴对齐的2D椭圆的参数形式？

12.平面通过3D点的隐含方程是什么(1 0 0) (0 1 0) 和 (0, 0, 1) ? 什么是参数方程?这个平面的法向量是多少?

13.给定四个2D点 \mathbf{a}_0 , \mathbf{a}_1 , \mathbf{b}_0 和 \mathbf{b}_1 ，设计一个鲁棒程序来确定线段 $\mathbf{a}_0\mathbf{a}_1$ 和 $\mathbf{b}_0\mathbf{b}_1$ 是否相交。

14.设计一个稳健的过程来计算一个二维点相对于三个二维非共线点的重心坐标。



Raster Images

Most computer graphics images are presented to the user on some kind of *raster display*. Raster displays show images as rectangular arrays of *pixels*. A common example is a flat-panel computer display or television, which has a rectangular array of small light-emitting pixels that can individually be set to different colors to create any desired image. Different colors are achieved by mixing varying intensities of red, green, and blue light. Most printers, such as laser printers and ink-jet printers, are also raster devices. They are based on scanning: there is no physical grid of pixels, but the image is laid down sequentially by depositing ink at selected points on a grid.

Rasters are also prevalent in input devices for images. A digital camera contains an image sensor comprising a grid of light-sensitive pixels, each of which records the color and intensity of light falling on it. A desktop scanner contains a linear array of pixels that is swept across the page being scanned, making many measurements per second to produce a grid of pixels.

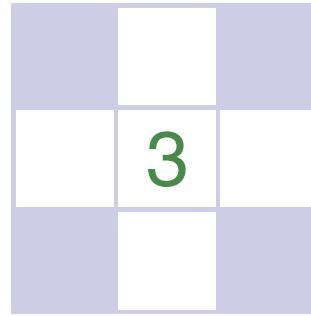
Because rasters are so prevalent in devices, *raster images* are the most common way to store and process images. A raster image is simply a 2D array that stores the *pixel value* for each pixel—usually a color stored as three numbers, for red, green, and blue. A raster image stored in memory can be displayed by using each pixel in the stored image to control the color of one pixel of the display.

But we don't always want to display an image this way. We might want to change the size or orientation of the image, correct the colors, or even show the image pasted on a moving three-dimensional surface. Even in televisions, the dis-

Pixel is short for “picture element.”

Color in printers is more complicated, involving mixtures of at least four pigments.

Or, maybe it's because raster images are so convenient that raster devices are prevalent.



Raster Images

大多数计算机图形图像在某种光栅显示器上呈现给用户。光栅显示器将图像显示为矩形像素阵列。一个共同的

示例是平板计算机显示器或电视，其具有小的发光像素的矩形阵列，这些像素可以单独地设置为不同的颜色以创建任何期望的图像。不同的颜色是通过混合不同强度的红色，绿色和蓝色光来实现的。大多数打印机，如激光打印机和喷墨打印机，也是光栅设备。它们基于扫描：没有像素的物理网格，但是通过在网格上的选定点上沉积墨水来按顺序放置图像。

Pixel是“图片元素”的简称。
。”

打印机中的颜色更复杂
涉及至少四种颜料的混合物。

栅格在用于图像的输入设备中也很普遍。一种数字照相机，包括光敏像素的网格的图像传感器，每个光敏像素记录在其上的光的颜色和强度。桌面扫描仪包含一个线性像素阵列，扫过被扫描的页面，每秒进行多次测量以产生像素网格。

由于栅格在设备中如此普遍，因此栅格图像是最常见的
的方式来存储和处理图像。光栅图像只是一个2D数组，用于存储每个像素的像素值—通常存储为红色，绿色和蓝色的三个数字的颜色。存储在存储器中的光栅图像可以通过使用存储的图像中的每个像素来控制显示器的一个像素的颜色来显示。

这是因为光栅图像是如此
convenient光栅设备
是普遍存在的。

但我们并不总是希望以这种方式显示图像。我们可能希望更改图像的大小或方向，校正颜色，甚至显示粘贴在移动的三维表面上的图像。即使在电视中，dis

play rarely has the same number of pixels as the image being displayed. Considerations like these break the direct link between image pixels and display pixels. It's best to think of a raster image as a *device-independent* description of the image to be displayed, and the display device as a way of approximating that ideal image.

There are other ways of describing images besides using arrays of pixels. A *vector image* is described by storing descriptions of shapes—areas of color bounded by lines or curves—with no reference to any particular pixel grid. In essence this amounts to storing the *instructions* for displaying the image rather than the pixels needed to display it. The main advantage of vector images is that they are *resolution independent* and can be displayed well on very high resolution devices. The corresponding disadvantage is that they must be *rasterized* before they can be displayed. Vector images are often used for text, diagrams, mechanical drawings, and other applications where crispness and precision are important and photographic images and complex shading aren't needed.

Or: you have to know what those numbers in your image actually mean.

In this chapter, we discuss the basics of raster images and displays, paying particular attention to the nonlinearities of standard displays. The details of how pixel values relate to light intensities are important to have in mind when we discuss computing images in later chapters.

3.1 Raster Devices

Before discussing raster images in the abstract, it is instructive to look at the basic operation of some specific devices that use these images. A few familiar raster devices can be categorized into a simple hierarchy:

- Output
 - Display
 - * Transmissive: liquid crystal display (LCD)
 - * Emissive: light-emitting diode (LED) display
 - Hardcopy
 - * Binary: ink-jet printer
 - * Continuous tone: dye sublimation printer
- Input
 - 2D array sensor: digital camera
 - 1D array sensor: flatbed scanner

play很少具有与正在显示的图像相同数量的像素。像这样的构造打破了图像像素和显示像素之间的直接联系。最好将光栅图像视为要显示的im年龄的与设备无关的描述，并将显示设备视为近似该理想图像的一种方式。

除了使用像素数组之外，还有其他描述图像的方法。矢量图像是通过存储形状的描述来描述的——以线条或曲线为界的颜色区域——而不参考任何特定的像素网格。从本质上讲，这相当于存储显示图像的指令，而不是显示图像所需的像素。矢量图像的主要优点是它们与分辨率无关，可以在非常高分辨率的设备上很好地显示。相应的缺点是它们必须被光栅化后才能被显示。矢量图像通常用于文本、图表、机械图和其他重要的清晰度和精度，不需要摄影图像和复杂的阴影的应用。

或者：你必须知道你的im年龄中的这些数字实际上意味着什么。

在本章中，我们讨论光栅图像和显示器的基础知识，特别注意标准显示器的非线性。如何做的细节
当我们在后面的章节中讨论计算图像时，与光强度相关的像素值是重要的。

3.1 Raster Devices

在抽象地讨论光栅图像之前，看看使用这些图像的一些特定设备的基本操作是有启发性的。一些熟悉的光栅设备可以分为一个简单的层次结构：

- Output
 - Display
 - * 透射式：液晶显示器（LCD）
 - * 发光：发光二极管（LED）显示
 - Hardcopy
 - * Binary: ink-jet printer
 - * 连续色调：染料升华打印机
- Input
 - 2D阵列传感器：数码相机
 - 1维阵列传感器：平板扫描仪



3.1.1 Displays

Current displays, including televisions and digital cinematic projectors as well as displays and projectors for computers, are nearly universally based on fixed arrays of pixels. They can be separated into emissive displays, which use pixels that directly emit controllable amounts of light, and transmissive displays, in which the pixels themselves don't emit light but instead vary the amount of light that they allow to pass through them. Transmissive displays require a light source to illuminate them: in a direct-viewed display this is a *backlight* behind the array; in a projector it is a lamp that emits light that is projected onto the screen after passing through the array. An emissive display is its own light source.

Light-emitting diode (LED) displays are an example of the emissive type. Each pixel is composed of one or more LEDs, which are semiconductor devices (based on inorganic or organic semiconductors) that emit light with intensity depending on the electrical current passing through them (see Figure 3.1).

The pixels in a color display are divided into three independently controlled *subpixels*—one red, one green, and one blue—each with its own LED made using different materials so that they emit light of different colors (Figure 3.2). When

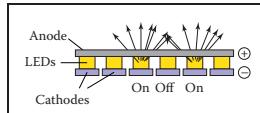


Figure 3.1. The operation of a light-emitting diode (LED) display.

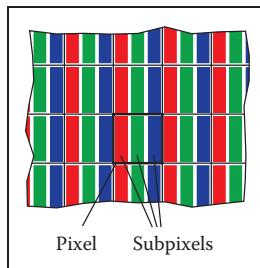


Figure 3.2. The red, green, and blue subpixels within a pixel of a flat-panel display.

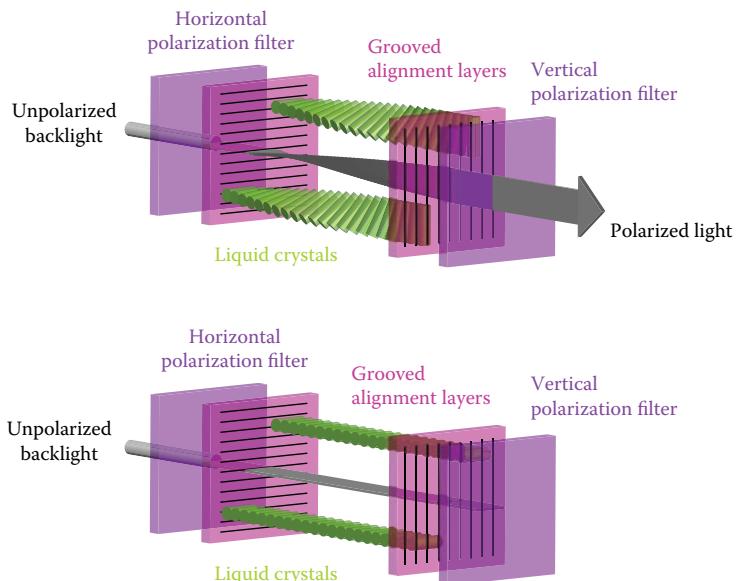


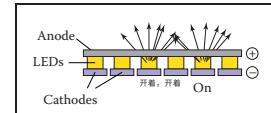
Figure 3.3. One pixel of an LCD display in the off state (bottom), in which the front polarizer blocks all the light that passes the back polarizer, and the on state (top), in which the liquid crystal cell rotates the polarization of the light so that it can pass through the front polarizer. *Figure courtesy Erik Reinhard (Reinhard, Khan, Akyuz, & Johnson, 2008).*



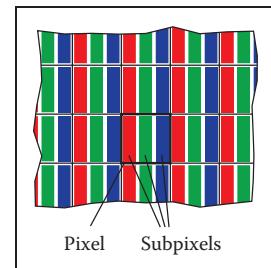
3.1.1 Displays

目前的显示器，包括电视和数字电影放映机以及计算机的显示器和放映机，几乎普遍基于固定的像素阵列。它们可以被分离成发射显示器，它们使用的像素是

直接发射可控量的光，和透射显示器，其中像素本身不发射光，而是改变它们允许通过它们的光量。透射式显示器需要光源来照亮它们：在直视显示器中，这是阵列后面的背光；在投影仪中，它是一盏灯，它发出的光线经过阵列后投射到屏幕上。自发光显示器是它自己的光源。



发光二极管(LED)显示器的歌剧。

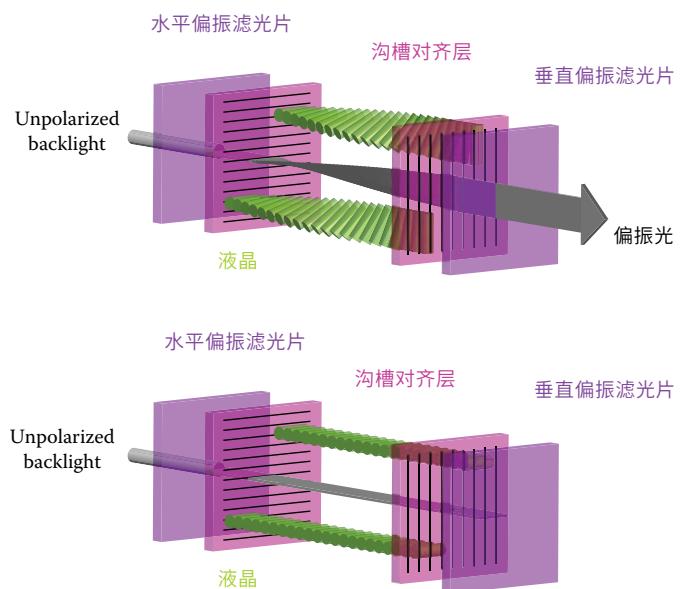


平板显示器的一个像素内的红色、绿色和蓝色子像素。

发光二极管(LED)显示器是发射型的一个例子。

每个像素由一个或多个Led组成，这些Led是半导体器件（基于无机或有机半导体），其根据通过它们的电流发出强度为de的光（见图3.1）。

彩色显示器中的像素分为三个独立控制的子像素——一个红色，一个绿色和一个蓝色——每个子像素都有自己的LED，使用不同的材料制成，使它们发出不同颜色的光（图3.2）。何时



LCD显示器的一个像素处于关闭状态（底部），其中前偏振器阻挡所有通过后偏振器的光，以及开启状态（顶部），其中液晶单元旋转光的偏振以使其可以通过前图由ErikReinhard提供（Reinhard, Khan, Akyuz, & Johnson, 2008）。

the display is viewed from a distance, the eye can't separate the individual subpixels, and the perceived color is a mixture of red, green, and blue.

Liquid crystal displays (LCDs) are an example of the transmissive type. A liquid crystal is a material whose molecular structure enables it to rotate the polarization of light that passes through it, and the degree of rotation can be adjusted by an applied voltage. An LCD pixel (Figure 3.3) has a layer of polarizing film behind it, so that it is illuminated by polarized light—let's assume it is polarized horizontally.

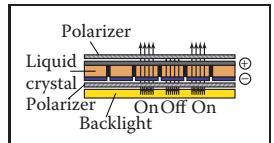


Figure 3.4. The operation of a liquid crystal display (LCD).

A second layer of polarizing film in front of the pixel is oriented to transmit only vertically polarized light. If the applied voltage is set so that the liquid crystal layer in between does not change the polarization, all light is blocked and the pixel is in the “off” (minimum intensity) state. If the voltage is set so that the liquid crystal rotates the polarization by 90 degrees, then all the light that entered through the back of the pixel will escape through the front, and the pixel is fully “on”—it has its maximum intensity. Intermediate voltages will partly rotate the polarization so that the front polarizer partly blocks the light, resulting in intensities between the minimum and maximum (Figure 3.4). Like color LED displays, color LCDs have red, green, and blue subpixels within each pixel, which are three independent pixels with red, green, and blue color filters over them.

Any type of display with a fixed pixel grid, including these and other technologies, has a fundamentally fixed *resolution* determined by the size of the grid. For displays and images, resolution simply means the dimensions of the pixel grid: if a desktop monitor has a resolution of 1920×1200 pixels, this means that it has 2,304,000 pixels arranged in 1920 columns and 1200 rows.

An image of a different resolution, to fill the screen, must be converted into a 1920×1200 image using the methods of Chapter 9.

The resolution of a display is sometimes called its “native resolution” since most displays can handle images of other resolutions, via built-in conversion.

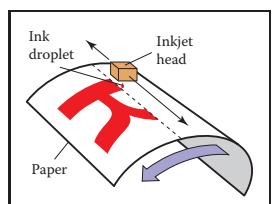


Figure 3.5. The operation of an ink-jet printer.

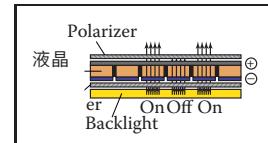
3.1.2 Hardcopy Devices

The process of recording images permanently on paper has very different constraints from showing images transiently on a display. In printing, pigments are distributed on paper or another medium so that when light reflects from the paper it forms the desired image. Printers are raster devices like displays, but many printers can only print *binary images*—pigment is either deposited or not at each grid position, with no intermediate amounts possible.

An ink-jet printer (Figure 3.5) is an example of a device that forms a raster image by scanning. An ink-jet print head contains liquid ink carrying pigment, which can be sprayed in very small drops under electronic control. The head

显示器从远处观看，眼睛不能分离单独的子像素，并且感知的颜色是红色，绿色和蓝色的混合物。

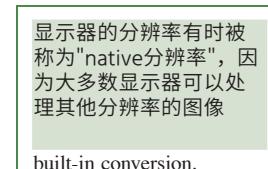
液晶显示器(Lcd)是透射型的一个例子。液晶是一种材料，其分子结构使其能够旋转穿过它的光的极化，并且旋转的程度可以通过施加的电压来调节。一个LCD像素（图3.3）背后有一层偏振膜，使其被偏振光照亮—让我们假设它是水平偏振的。



液晶显示器(LCD)的歌剧。

像素前面的第二层偏振膜被定向为仅透射垂直偏振光。如果所施加的电压被设置为使得介于其间的液晶层不改变偏振，则所有光被阻挡并且像素处于“关”（最小强度）状态。如果电压设置为使得

液晶将偏振旋转90度，然后通过像素背面的所有光线都将通过正面逸出，并且像素完全“打开”–它具有最大强度。中间电压将部分旋转偏振，使前偏振器部分阻挡光，导致最小和最大之间的强度（图3.4）。与彩色LED显示器一样，彩色LCD在每个像素内都有红色、绿色和蓝色子像素，它们是三个独立的像素，上面有红色、绿色和蓝色滤色器。



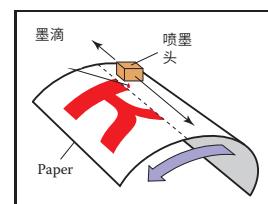
显示器的分辨率有时被称为“native分辨率”，因为大多数显示器可以处理其他分辨率的图像

built-in conversion.

具有固定像素网格的任何类型的显示器，包括这些和其他技术nologies，具有由网格大小确定的基本固定分辨率。对于显示器和图像，分辨率只是指像素的尺寸

网格：如果桌面显示器的分辨率为 1920×1200 像素，这意味着它有2 304 000像素排列在1920列和1200行中。

一个不同分辨率的图像，要填满屏幕，必须使用第9章的方法转换成 1920×1200 的图像。



喷墨打印机的操作。

将图像永久记录在纸上的过程与在显示器上暂时显示图像有很大的不同。在印刷中，颜料分布在纸或另一种介质上，以便当光从pa反射时，它形成所需的图像。打印机是像显示器一样的光栅设备，但许多打印机只能打印二进制图像—颜料在每个网格位置沉积或不沉积，没有中间量可能。

喷墨打印机（图3.5）是通过扫描形成光栅图像的设备的一个例子。一种喷墨打印机头，包含载有颜料的液体墨水其可以在电子控制下以非常小的液滴喷射。头

moves across the paper, and drops are emitted as it passes grid positions that should receive ink; no ink is emitted in areas intended to remain blank. After each sweep the paper is advanced slightly, and then the next row of the grid is laid down. Color prints are made by using several print heads, each spraying ink with a different pigment, so that each grid position can receive any combination of different colored drops. Because all drops are the same, an ink-jet printer prints binary images: at each grid point there is a drop or no drop; there are no intermediate shades.

An ink-jet printer has no physical array of pixels; the resolution is determined by how small the drops can be made and how far the paper is advanced after each sweep. Many ink-jet printers have multiple nozzles in the print head, enabling several sweeps to be made in one pass, but it is the paper advance, not the nozzle spacing, that ultimately determines the spacing of the rows.

The *thermal dye transfer* process is an example of a *continuous tone* printing process, meaning that varying amounts of dye can be deposited at each pixel—it is not all-or-nothing like an ink-jet printer (Figure 3.6). A *donor ribbon* containing colored dye is pressed between the paper, or *dye receiver*, and a *print head* containing a linear array of heating elements, one for each column of pixels in the image. As the paper and ribbon move past the head, the heating elements switch on and off to heat the ribbon in areas where dye is desired, causing the dye to diffuse from the ribbon to the paper. This process is repeated for each of several dye colors. Since higher temperatures cause more dye to be transferred, the amount of each dye deposited at each grid position can be controlled, allowing a continuous range of colors to be produced. The number of heating elements in the print head establishes a fixed resolution in the direction across the page, but the resolution along the page is determined by the rate of heating and cooling compared to the speed of the paper.

Unlike displays, the resolution of printers is described in terms of the *pixel density* instead of the total count of pixels. So a thermal dye transfer printer that has elements spaced 300 per inch across its print head has a resolution of 300 *pixels per inch* (ppi) across the page. If the resolution along the page is chosen to be the same, we can simply say the printer's resolution is 300 ppi. An ink-jet printer that places dots on a grid with 1200 grid points per inch is described as having a resolution of 1200 *dots per inch* (dpi). Because the ink-jet printer is a binary device, it requires a much finer grid for at least two reasons. Because edges are abrupt black/white boundaries, very high resolution is required to avoid stair-stepping, or aliasing, from appearing (see Section 8.3). When continuous-tone images are printed, the high resolution is required to simulate intermediate colors by printing varying-density dot patterns called *halftones*.

There are also continuous ink-jet printers that print in a continuous helical path on paper wrapped around a spinning drum, rather than moving the head back and forth.

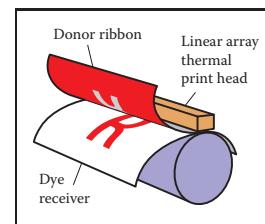


Figure 3.6. The operation of a thermal dye transfer printer.

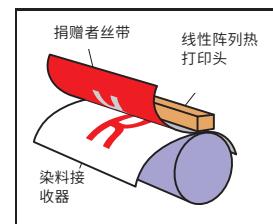
The term “dpi” is all too often used to mean “pixels per inch,” but dpi should be used in reference to binary devices and ppi in reference to continuous-tone devices.

在纸张上移动，当它通过应该接收墨水的网格位置时，就会喷出墨滴；在打算保持空白的区域不会喷出墨水。每次扫描后，纸张稍微前进，然后放下网格的下一行。彩色印刷品是通过使用几个打印头制作的，每个打印头都用一个

不同的颜料，从而每个网格位置可以接收不同颜色的滴的任何组合。因为所有的水滴都是相同的，所以喷墨打印机打印二进制图像：在每个网格点都有一滴或没有一滴；没有中间色调。

喷墨打印机没有像素的物理阵列；分辨率取决于滴落的大小和每次扫描后纸张前进的距离。许多喷墨打印机在打印头中有多个喷嘴，可以一次进行多次扫描，但最终决定行间距的是纸张前进，而不是喷嘴间距。

也有连续喷墨打印机在缠绕在旋转滚筒上的纸张上以连续螺旋路径打印，而不是来回移动头部。



热染料转移打印机的歌剧。

热染料转移过程是连续色调打印过程的一个例子，这意味着可以在每个像素上沉积不同数量的染料—它不像喷墨打印机那样全有或全无（图3.6）。一个含有有色染料的供体带被压在纸或染料接收器和一个含有线性阵列加热元件的打印头之间，每列像素在图像中一个。当纸张和色带经过头部时，加热元件会切换

打开和关闭以在需要染料的区域加热色带，导致染料从色带扩散到纸张。对于几种染料颜色中的每一种重复该过程。由于较高的温度导致更多的染料被转移，因此可以控制沉积在每个网格位置的每种染料的量，从而允许产生连续范围的颜色。打印头中加热元件的数量在横过页面的方向上建立固定的分辨率，但是沿页面的分辨率由加热和冷却的速率相比于纸张的速度来确定。

与显示器不同，打印机的分辨率是根据像素密度而不是像素总数来描述的。因此，在其打印头上具有每英寸间隔300个元素的热染料转印打印机在整个页面上具有每英寸300像素（ppi）的分辨率。如果选择页面上的分辨率相同，我们可以简单地说打印机的分辨率是300ppi。一种喷墨机

将点放置在每英寸1200个网格点的网格上的打印机被描述为具有每英寸1200点(dpi)的分辨率。因为喷墨打印机是二进制设备，所以至少出于两个原因需要细得多的网格。因为边缘是突然的黑色白色边界，所以需要非常高的分辨率来避免出现阶梯或混叠（见第8.3节）。当打印连续色调图像时，需要高分辨率来通过打印称为半色调的不同密度点图案来模拟中间颜色。

术语“dpi”通常用于表示“每英寸像素”，但dpi应用于参考二进制设备，ppi应用于参考连续色调设备。

3.1.3 Input Devices

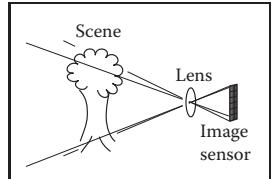


Figure 3.7. The operation of a digital camera.

G	B	G	B	G	B	G
R	G	R	G	R	G	R
G	B	G	B	G	B	G
R	G	R	G	R	G	R
G	B	G	B	G	B	G
R	G	R	G	R	G	R

Figure 3.8. Most color digital cameras use a color filter array similar to the *Bayer mosaic* shown here. Each pixel measures either red, green, or blue light.

People who are selling cameras use “mega” to mean 10^6 , not 2^{20} as with megabytes.

The resolution of a scanner is sometimes called its “optical resolution” since most scanners can produce images of other resolutions, via built-in conversion.

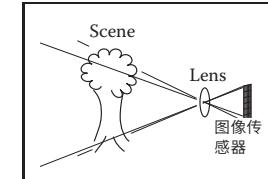
Raster images have to come from somewhere, and any image that wasn’t computed by some algorithm has to have been measured by some *raster input device*, most often a camera or scanner. Even in rendering images of 3D scenes, photographs are used constantly as texture maps (see Chapter 11). A raster input device has to make a light measurement for each pixel, and (like output devices) they are usually based on arrays of sensors.

A digital camera is an example of a 2D array input device. The image sensor in a camera is a semiconductor device with a grid of light-sensitive pixels. Two common types of arrays are known as CCDs (charge-coupled devices) and CMOS (complimentary metal–oxide–semiconductor) image sensors. The camera’s lens projects an image of the scene to be photographed onto the sensor, and then each pixel measures the light energy falling on it, ultimately resulting in a number that goes into the output image (Figure 3.7). In much the same way as color displays use red, green, and blue subpixels, most color cameras work by using a *color-filter array* or *mosaic* to allow each pixel to see only red, green, or blue light, leaving the image processing software to fill in the missing values in a process known as *demosaicking* (Figure 3.8).

Other cameras use three separate arrays, or three separate layers in the array, to measure independent red, green, and blue values at each pixel, producing a usable color image without further processing. The resolution of a camera is determined by the fixed number of pixels in the array and is usually quoted using the total count of pixels: a camera with an array of 3000 columns and 2000 rows produces an image of resolution 3000×2000 , which has 6 million pixels, and is called a 6 megapixel (MP) camera. It’s important to remember that a mosaic sensor does not measure a complete color image, so a camera that measures the same number of pixels but with independent red, green, and blue measurements records more information about the image than one with a mosaic sensor.

A flatbed scanner also measures red, green, and blue values for each of a grid of pixels, but like a thermal dye transfer printer it uses a 1D array that sweeps across the page being scanned, making many measurements per second. The resolution across the page is fixed by the size of the array, and the resolution along the page is determined by the frequency of measurements compared to the speed at which the scan head moves. A color scanner has a $3 \times n_x$ array, where n_x is the number of pixels across the page, with the three rows covered by red, green, and blue filters. With an appropriate delay between the times at which the three colors are measured, this allows three independent color measurements at each grid point. As with continuous-tone printers, the resolution of scanners is reported in pixels per inch (ppi).

3.1.3 输入装置



数码相机的操作。

G	B	G	B	G	B	G
R	G	R	G	R	G	R
G	B	G	B	G	B	G
R	G	R	G	R	G	R
G	B	G	B	G	B	G
R	G	R	G	R	G	R

大多数彩色digital相机使用类似于此处显示的Bayermo saic的滤色器阵列。每个像素测量红光、绿光或蓝光。

People who are selling "兆"是指106, 而不是220兆字节。

扫描仪的分辨率有时被称为“光学分辨率”，因为大多数扫描仪可以通过内置转换产生其他分辨率的图像。

光栅图像必须来自某个地方，任何没有被某种算法计算的图像都必须由一些光栅输入设备（通常是相机或扫描仪）测量。即使在3d场景的渲染图像中，photograph也经常被用作纹理贴图（参见第11章）。光栅输入设备必须对每个像素进行光测量，并且（与输出设备一样）它们通常基于传感器阵列。

数码相机是2D阵列输入设备的一个例子。图像传感器在照相机中是具有光敏像素网格的半导体器件。两种常见类型的阵列称为Ccd（电荷耦合器件）和CMOS（互补金属氧化物半导体）图像传感器。相机的镜头将要拍摄的场景的图像投射到传感器上，然后每个像素测量落在其上的光能，最终产生一个数字进入输出图像（图3.7）。与彩色显示器使用红色、绿色和蓝色子像素的方式大致相同，大多数彩色相机的工作方式是使用滤色器阵列或马赛克，允许每个像素只看到红色、绿色或蓝色光，让图像处理软件在一个称为demosaicking的过程中填充缺失值（图3.8）。

其他相机使用三个单独的阵列或阵列中的三个单独的层来测量每个像素的独立红色、绿色和蓝色值，从而产生可用的彩色图像，而无需进一步处理。相机的分辨率由阵列中的固定像素数决定，通常使用像素总数来引用：具有3000列和2000行阵列的相机产生分辨率为 3000×2000 的图像，其具有6百万像素，被称为重要的是要记住，mosiac传感器确实如此

不能测量完整的彩色图像，因此测量相同像素数但具有独立红色、绿色和蓝色测量值的相机比具有马赛克传感器的相机记录有关图像的更多信息。

平板扫描仪还可以测量每个像素网格的红色、绿色和蓝色值，但与热染料转移打印机一样，它使用一维阵列扫描被扫描的页面，每秒进行多次测量。整个页面的分辨率由数组的大小和分辨率固定

沿着页面由测量的频率与扫描头移动的速度相比来确定。彩色扫描仪有一个 $3 \times nx$ 阵列，其中nx是页面上的像素数，三行由红色、绿色和蓝色滤镜复盖。在测量三种颜色的时间之间具有适当的延迟，这允许在每个网格点进行三个独立的颜色测量。与连续色调打印机一样，扫描仪的分辨率以每英寸像素 (ppi) 报告。

With this concrete information about where our images come from and where they will go, we'll now discuss images more abstractly, in the way we'll use them in graphics algorithms.

3.2 Images, Pixels, and Geometry

We know that a raster image is a big array of pixels, each of which stores information about the color of the image at its grid point. We've seen what various output devices do with images we send to them and how input devices derive them from images formed by light in the physical world. But for computations in the computer, we need a convenient abstraction that is independent of the specifics of any device, that we can use to reason about how to produce or interpret the values stored in images.

When we measure or reproduce images, they take the form of two-dimensional distributions of light energy: the light emitted from the monitor as a function of position on the face of the display; the light falling on a camera's image sensor as a function of position across the sensor's plane; the *reflectance*, or fraction of light reflected (as opposed to absorbed) as a function of position on a piece of paper. So in the physical world, images are functions defined over two-dimensional areas—almost always rectangles. So we can abstract an image as a function

$$I(x, y) : R \rightarrow V,$$

where $R \subset \mathbb{R}^2$ is a rectangular area and V is the set of possible pixel values. The simplest case is an idealized grayscale image where each point in the rectangle has just a brightness (no color), and we can say $V = \mathbb{R}^+$ (the nonnegative reals). An idealized color image, with red, green, and blue values at each pixel, has $V = (\mathbb{R}^+)^3$. We'll discuss other possibilities for V in the next section.

How does a raster image relate to this abstract notion of a continuous image? Looking to the concrete examples, a pixel from a camera or scanner is a measurement of the average color of the image over some small area around the pixel. A display pixel, with its red, green, and blue subpixels, is designed so that the average color of the image over the face of the pixel is controlled by the corresponding pixel value in the raster image. In both cases, the pixel value is a local average of the color of the image, and it is called a *point sample* of the image. In other words, when we find the value x in a pixel, it means “the value of the image in the vicinity of this grid point is x .” The idea of images as sampled representations of functions is explored further in Chapter 9.

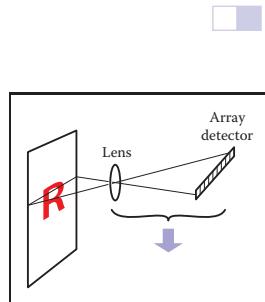


Figure 3.9. The operation of a flatbed scanner.

有了这些关于我们的图像来自哪里以及它们将去哪里的具体信息，我们现在将更抽象地讨论图像，就像我们将在图形算法中使用它们一样。

3.2 图像、像素和几何

我们知道光栅图像是一个大的像素阵列，每个像素阵列都存储着图像在网格点的颜色信息。我们已经看到了各种输出设备如何处理我们发送给他们的图像，以及输入设备如何从物理世界中的光形成的图像中获得它们。但是对于computer中的计算，我们需要一个方便的抽象，独立于任何设备的细节，我们可以用它来推理如何生成或解释存储在图像中的值。

当我们测量或再现图像时，它们采用光能的二维分布形式：从显示器发出的光作为显示器表面位置的函数；落在相机图像传感器上的光作为传感器平面位置的函数；反射率或

光反射（相对于吸收）作为一张纸上的位置的函数。因此，在物理世界中，图像是在二维区域上定义的函数—几乎总是矩形。所以我们可以将图像抽象为函数

$$I(x, y) : R \rightarrow V,$$

其中 $R \in \mathbb{R}^2$ 是矩形区域， V 是可能像素值的集合。该

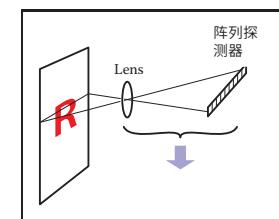
最简单的情况是一个理想化的灰度图像，其中矩形中的每个点都只有一个亮度（没有颜色），我们可以说 $V = \mathbb{R}^+$ （非负实数）。理想化的彩色图像，在每个像素处具有红色，绿色和蓝色值，具有 $V = (\mathbb{R}^+)^3$ 。我们将在下一节讨论 V 的其他可能性。

光栅图像如何与连续图像的抽象概念相关联？

看看具体的例子，来自相机或扫描仪的像素是在像素周围的一些小区域上的图像平均颜色的度量。一个具有其红色、绿色和蓝色子像素的显示像素被设计成使得图像在该像素的面部上的不同颜色由光栅图像中的相应像素值控制。在这两种情况下，像素值都是图像颜色的局部平均值，并且它被称为图像的点样本。换句话说，当我们在一个像素中找到值 x 时，它意味着“在这个网格点附近的图像的值是 x 。”图像作为函数的采样表示的想法在第9章中进一步探讨。

“A pixel is not a little square!”
—Alvy Ray Smith
(A. R. Smith, 1995)

Are there any raster devices that are not rectangular?



平板扫描仪的操作。

“一个像素不是一个小正方形！” —AlvyRaySmith
(A.R.Smith, 1995)

是否有任何不是矩形的光栅设备？

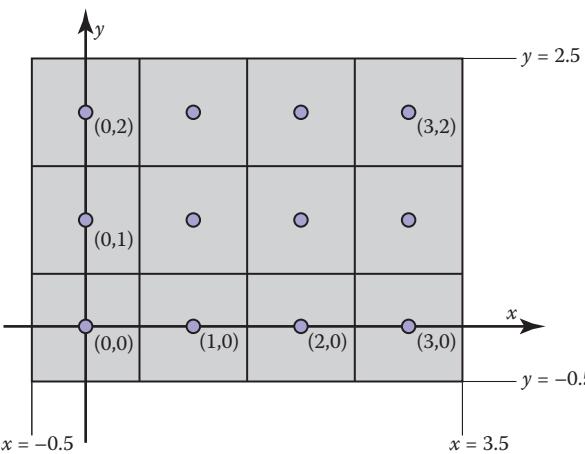


Figure 3.10. Coordinates of a four pixel \times three pixel screen. Note that in some APIs the y -axis will point downward.

A mundane but important question is where the pixels are located in 2D space. This is only a matter of convention, but establishing a consistent convention is important! In this book, a raster image is indexed by the pair (i, j) indicating the column (i) and row (j) of the pixel, counting from the bottom left. If an image has n_x columns and n_y rows of pixels, the bottom-left pixel is $(0, 0)$ and the top-right is pixel $(n_x - 1, n_y - 1)$. We need 2D real screen coordinates to specify pixel positions. We will place the pixels' sample points at integer coordinates, as shown by the 4×3 screen in Figure 3.10.

The rectangular domain of the image has width n_x and height n_y and is centered on this grid, meaning that it extends half a pixel beyond the last sample point on each side. So the rectangular domain of a $n_x \times n_y$ image is

$$R = [-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5].$$

Again, these coordinates are simply conventions, but they will be important to remember later when implementing cameras and viewing transformations.

In some APIs, and many file formats, the rows of an image are organized top-to-bottom, so that $(0, 0)$ is at the top left. This is for historical reasons: the rows in analog television transmission started from the top.

Some systems shift the coordinates by half a pixel to place the sample points halfway between the integers but place the edges of the image at integers.

3.2.1 Pixel Values

So far we have described the values of pixels in terms of real numbers, representing intensity (possibly separately for red, green, and blue) at a point in the image. This suggests that images should be arrays of floating-point numbers, with either one (for *grayscale*, or black and white, images) or three (for RGB color images)

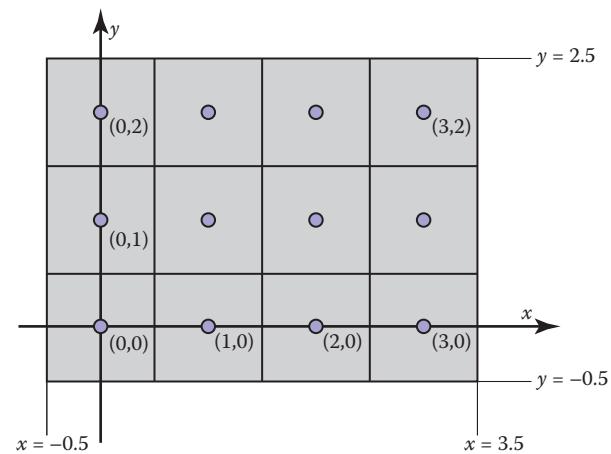


图3.10。 四像素×三像素屏幕的坐标。请注意，在某些API中，y轴将指向下方。

在一些API和许多文件格式中，图像的行被组织为top-to-bottom，因此 $(0, 0)$ 位于左上角。这是出于历史原因：模拟电视传输中的行从顶部开始。

一些系统将坐标移动半个像素，将样本点置于整数之间，但将图像的边缘置于整数处。

一个平凡但重要的问题是像素位于2D空间中的位置。

这只是一个惯例的问题，但建立一个一致的惯例是重要的！在这本书中，光栅图像由一对 (i, j) 索引，表示像素的列(i)和行(j)，从左下方开始计数。如果图像有 n_x 列和 n_y 行像素，则左下像素为 $(0, 0)$ ，右上像素为 $(n_x - 1, n_y - 1)$ 。我们需要2D真实屏幕坐标来指定像素位置。我们将像素的采样点放置在整数坐标处，如图3.10中的 4×3 屏幕所示。

图像的矩形域具有宽度 n_x 和高度 n_y ，并且在该网格上是centered，这意味着它在每侧的最后一个样本点之外延伸半个像素。所以 $n_x \times n_y$ 图像的矩形域是

$$R = [-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5].$$

同样，这些坐标只是约定，但稍后在实现摄像机和查看变换时要记住它们很重要。

3.2.1 像素值

到目前为止，我们已经用实数来描述像素的值，表示图像中某个点的强度（可能分别为红色、绿色和蓝色）。这表明图像应该是浮点数数组，具有一个（对于灰度或黑白图像）或三个（对于RGB彩色图像）



32-bit floating point numbers stored per pixel. This format is sometimes used, when its precision and range of values are needed, but images have a lot of pixels and memory and bandwidth for storing and transmitting images are invariably scarce. Just one ten-megapixel photograph would consume about 115 MB of RAM in this format.

Less range is required for images that are meant to be displayed directly. While the range of possible light intensities is unbounded in principle, any given device has a decidedly finite maximum intensity, so in many contexts it is perfectly sufficient for pixels to have a bounded range, usually taken to be $[0, 1]$ for simplicity. For instance, the possible values in an 8-bit image are $0, 1/255, 2/255, \dots, 254/255, 1$. Images stored with floating-point numbers, allowing a wide range of values, are often called *high dynamic range* (HDR) images to distinguish them from fixed-range, or *low dynamic range* (LDR) images that are stored with integers. See Chapter 21 for an in-depth discussion of techniques and applications for high dynamic range images.

Here are some pixel formats with typical applications:

- 1-bit grayscale—text and other images where intermediate grays are not desired (high resolution required);
- 8-bit RGB fixed-range color (24 bits total per pixel)—web and email applications, consumer photographs;
- 8- or 10-bit fixed-range RGB (24–30 bits/pixel)—digital interfaces to computer displays;
- 12- to 14-bit fixed-range RGB (36–42 bits/pixel)—raw camera images for professional photography;
- 16-bit fixed-range RGB (48 bits/pixel)—professional photography and printing; intermediate format for image processing of fixed-range images;
- 16-bit fixed-range grayscale (16 bits/pixel)—radiology and medical imaging;
- 16-bit “half-precision” floating-point RGB—HDR images; intermediate format for real-time rendering;
- 32-bit floating-point RGB—general-purpose intermediate format for software rendering and processing of HDR images.

Reducing the number of bits used to store each pixel leads to two distinctive types of *artifacts*, or artificially introduced flaws, in images. First, encoding

Why 115 MB and not 120 MB?

The denominator of 255, rather than 256, is awkward, but being able to represent 0 and 1 exactly is important.



每像素存储的32位浮点数。当需要其精度和值范围时，有时会使用这种格式，但图像具有大量像素，存储和传输图像的内存和带宽是不变的稀缺。只有一张1000万像素的照片将消耗大约115MB的RAM在这种格式。

对于直接显示的图像，需要更少的范围。虽然可能的光强度范围原则上是无界的，但任何给定的设备都有一个绝对有限的最大强度，因此在许多情况下，像素有一个有界范围是足够的，为了简单起见，通常取为 $[0, 1]$ 。例如，8位图像中的可能值为0、1255、2255、 $\dots, 254255$ 。用浮点数存储的图像，允许宽

值范围通常称为高动态范围(HDR)图像，以区别于用整数存储的固定范围或低动态范围(LDR)图像。有关高动态范围图像的技术和应用的深入讨论请参阅第21章。

以下是一些具有典型应用的像素格式：

- 1位灰度—不需要中间灰度的文本和其他图像（需要高分辨率）；
- 8位RGB固定范围颜色（每像素共24位）—web和电子邮件应用程序，消费者照片；
- 8或10位固定范围RGB（24–30位像素）—计算机显示器的数字接口；
- 12至14位固定范围RGB（36–42位像素）—专业摄影的原始相机图像；
- 16位固定距离像素（48位像素）—专业摄影和打印；固定距离图像图像处理的中间格式；
- 16位固定范围灰度（16位像素）—放射学和医学成像；
- 16位“半精度”浮点RGB—HDR图像；实时渲染的中间格式；
- 32位浮点RGB—用于软件渲染和处理HDR图像的通用中间格式。

减少用于存储每个像素的位数会导致图像中两种不同类型的伪像或人为引入的缺陷。一、编码

为什么115MB而不是120 MB?

255的分母，而不是256，是尴尬的，但能够准确地表示0和1是很重要的。

images with fixed-range values produces *clipping* when pixels that would otherwise be brighter than the maximum value are set, or clipped, to the maximum representable value. For instance, a photograph of a sunny scene may include reflections that are much brighter than white surfaces; these will be clipped (even if they were measured by the camera) when the image is converted to a fixed range to be displayed. Second, encoding images with limited precision leads to *quantization artifacts*, or *banding*, when the need to round pixel values to the nearest representable value introduces visible jumps in intensity or color. Banding can be particularly insidious in animation and video, where the bands may not be objectionable in still images, but become very visible when they move back and forth.

3.2.2 Monitor Intensities and Gamma

All modern monitors take digital input for the “value” of a pixel and convert this to an intensity level. Real monitors have some nonzero intensity when they are off because the screen reflects some light. For our purposes we can consider this “black” and the monitor fully on as “white.” We assume a numeric description of pixel color that ranges from zero to one. Black is zero, white is one, and a gray halfway between black and white is 0.5. Note that here “halfway” refers to the physical amount of light coming from the pixel, rather than the appearance. The human perception of intensity is nonlinear and will not be part of the present discussion; see Chapter 20 for more.

There are two key issues that must be understood to produce correct images on monitors. The first is that monitors are nonlinear with respect to input. For example, if you give a monitor 0, 0.5, and 1.0 as inputs for three pixels, the intensities displayed might be 0, 0.25, and 1.0 (off, one-quarter fully on, and fully on). As an approximate characterization of this nonlinearity, monitors are commonly characterized by a γ (“gamma”) value. This value is the degree of freedom in the formula

$$\text{displayed intensity} = (\text{maximum intensity})a^\gamma, \quad (3.1)$$

where a is the input pixel value between zero and one. For example, if a monitor has a gamma of 2.0, and we input a value of $a = 0.5$, the displayed intensity will be one fourth the maximum possible intensity because $0.5^2 = 0.25$. Note that $a = 0$ maps to zero intensity and $a = 1$ maps to the maximum intensity regardless of the value of γ . Describing a display’s nonlinearity using γ is only an approximation; we do not need a great deal of accuracy in estimating the γ of

当比最大值更亮的像素被设置或剪切到可表示的最大值时，具有固定范围值的图像会产生剪切。例如，阳光明媚的场景的照片可能包括比白色表面亮得多的反射；当图像转换到要显示的固定范围时，这些反射将被剪切（即使它们是由相机测量的）。其次，当需要将像素值舍入到最接近的可表示值时，以有限的精度编码图像会导致量化伪影或带状。条带在动画和视频中可能特别阴险，其中条带在静止图像中可能不会令人反感，但在它们来回移动时变得非常明显。

3.2.2 监测强度和伽马

所有现代显示器都采用像素“值”的数字输入，并将其转换为强度级别。真正的显示器在关闭时会有一些非零强度，因为屏幕会反射一些光线。为了我们的目的，我们可以把这个“黑色”和显示器完全打开为“白色”。“我们假设像素颜色的数字描述，范围从零到一。黑色为零，白色为一，介于黑色和白色之间的灰色为0.5。注意，这里“中途”指的是来自像素的光的物理量，而不是外观。人类对强度的感知是非线性的，不会成为本讨论的一部分；更多信息请参见第20章。

要在监视器上生成正确的图像，必须了解两个关键问题。第一个是监视器相对于输入是非线性的。例如，如果将监视器0、0.5和1.0作为三个像素的输入，则显示的强度可能为0、0.25和1.0（关闭、四分之一完全打开和完全打开）。作为这种非线性的近似表征，监视器通常由 γ （“gamma”）值表征。该值是公式中的自由度

$$\text{显示的强度} = (\text{最大强度}) a^\gamma \quad (3.1)$$

其中 a 是介于零和一之间的输入像素值。例如，如果监视器的gamma值为2.0，我们输入的值为 $a=0.5$ ，显示的强度将是最大可能强度的四分之一，因为 $0.5^2=0.25$ 。注意， $a=0$ 映射到零强度并且 $a=1$ 映射到最大强度而与 γ 的值无关。使用 γ 描述显示器的非线性只是一个近似值；我们不需要很高的精度来估计



a device. A nice visual way to gauge the nonlinearity is to find what value of a gives an intensity halfway between black and white. This a will be

$$0.5 = a^\gamma.$$

If we can find that a , we can deduce γ by taking logarithms on both sides:

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

We can find this a by a standard technique where we display a checkerboard pattern of black and white pixels next to a square of gray pixels with input a (Figure 3.11), then ask the user to adjust a (with a slider, for instance) until the two sides match in average brightness. When you look at this image from a distance (or without glasses if you are nearsighted), the two sides of the image will look about the same when a is producing an intensity halfway between black and white. This is because the blurred checkerboard is mixing even numbers of white and black pixels so the overall effect is a uniform color halfway between white and black.

Once we know γ , we can *gamma correct* our input so that a value of $a = 0.5$ is displayed with intensity halfway between black and white. This is done with the transformation

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

When this formula is plugged into Equation (3.1) we get

$$\begin{aligned} \text{displayed intensity} &= (a')^\gamma = \left(a^{\frac{1}{\gamma}}\right)^\gamma (\text{maximum intensity}) \\ &= a (\text{maximum intensity}). \end{aligned}$$

Another important characteristic of real displays is that they take quantized input values. So while we can manipulate intensities in the floating point range $[0, 1]$, the detailed input to a monitor is a fixed-size integer. The most common range for this integer is 0–255 which can be held in 8 bits of storage. This means that the possible values for a are not any number in $[0, 1]$ but instead

$$\text{possible values for } a = \left\{ \frac{0}{255}, \frac{1}{255}, \frac{2}{255}, \dots, \frac{254}{255}, \frac{255}{255} \right\}.$$

This means the possible displayed intensity values are approximately

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

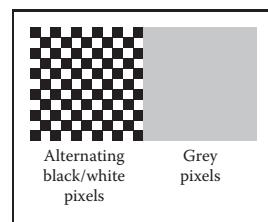


Figure 3.11. Alternating black and white pixels viewed from a distance are halfway between black and white. The gamma of a monitor can be inferred by finding a gray value that appears to have the same intensity as the black and white pattern.

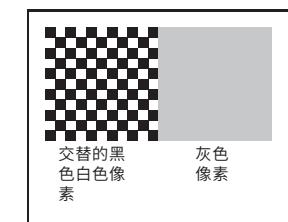


一个装置。测量非线性的一个很好的视觉方法是找到 a 的值给出了黑色和白色之间的强度。这个 a 将是

$$0.5 = a^\gamma.$$

如果我们能找到 a ，我们可以通过两边取对数来推断 γ :

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



Alternating black and white pixels from far away appear to be halfway between black and white. Monitors can infer the gamma by finding a gray value that looks like it has the same intensity as the black and white pattern.

我们可以用一个标准的技术来找到这个 a ，在输入 a 的灰色像素的正方形旁边显示一个黑白像素的棋盘图案（图3.11），然后要求用户调整 a （例如用滑块），直到双方的平均亮度匹配。当你从远处看这个图像时（如果你是近视的话，不戴眼镜），当 a 在黑白之间产生强度时，图像的两侧看起来大致相同。这是因为模糊的棋盘格混合了偶数个白色和黑色像素，所以整体效果是介于白色和黑色之间的均匀颜色。

一旦我们知道 γ ，我们可以gamma校正我们的输入，使 $a=0$ 的值。以介于黑色和白色之间的强度显示。这是通过转换完成的

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

当这个公式插入方程 (3.1) 时，我们得到

$$\begin{aligned} \text{显示强度} &= (a')^\gamma = \left(a^{\frac{1}{\gamma}}\right)^\gamma (\text{最大强度}) \\ &= a (\text{最大强度}) . \end{aligned}$$

For monitors with analog interfaces, which have difficulty changing intensity rapidly along the horizontal direction, horizontal black and white stripes work better than a checkerboard.

对于具有模拟接口的显示器，其难以沿水平方向快速改变强度，水平黑白条纹比棋盘更好地工作。

真实显示器的另一个重要特征是它们采用量化的输入值。因此，虽然我们可以在浮点范围 $[0, 1]$ 中操纵强度，但监视器的详细输入是一个固定大小的整数。这个整数最常见的范围是0–255，它可以保存在8位存储中。这意味着 a 的可能值不是 $[0, 1]$ 中的任何数字，而是

$$a=\text{的可能值} \quad \left\{ \frac{0}{255}, \frac{1}{255}, \frac{2}{255}, \dots, \frac{254}{255}, \frac{255}{255} \right\}.$$

这意味着可能显示的强度值大约是

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

where M is the maximum intensity. In applications where the exact intensities need to be controlled, we would have to actually measure the 256 possible intensities, and these intensities might be different at different points on the screen, especially for CRTs. They might also vary with viewing angle. Fortunately, few applications require such accurate calibration.

3.3 RGB Color

In grade school you probably learned that the primaries are red, yellow, and blue, and that, e.g., yellow + blue = green. This is *subtractive* color mixing, which is fundamentally different from the more familiar additive mixing that happens in displays.

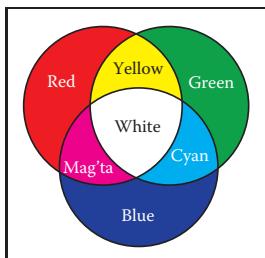


Figure 3.12. The additive mixing rules for colors red/green/blue.

Most computer graphics images are defined in terms of red-green-blue (RGB) color. RGB color is a simple space that allows straightforward conversion to the controls for most computer screens. In this section, RGB color is discussed from a user's perspective, and operational facility is the goal. A more thorough discussion of color is given in Chapter 19, but the mechanics of RGB color space will allow us to write most graphics programs. The basic idea of RGB color space is that the color is displayed by mixing three *primary* lights: one red, one green, and one blue. The lights mix in an *additive* manner.

In RGB additive color mixing we have (Figure 3.12)

$$\begin{aligned} \text{red} + \text{green} &= \text{yellow}, \\ \text{green} + \text{blue} &= \text{cyan}, \\ \text{blue} + \text{red} &= \text{magenta}, \\ \text{red} + \text{green} + \text{blue} &= \text{white}. \end{aligned}$$

The color "cyan" is a blue-green, and the color "magenta" is a purple.

If we are allowed to dim the primary lights from fully off (indicated by pixel value 0) to fully on (indicated by 1), we can create all the colors that can be displayed on an RGB monitor. The red, green, and blue pixel values create a three-dimensional *RGB color cube* that has a red, a green, and a blue axis. Allowable coordinates for the axes range from zero to one. The color cube is shown graphically in Figure 3.13.

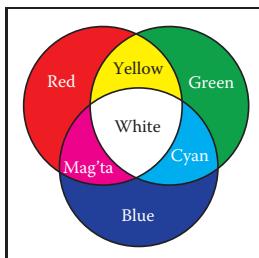
The colors at the corners of the cube are

$$\begin{aligned} \text{black} &= (0, 0, 0), \\ \text{red} &= (1, 0, 0), \\ \text{green} &= (0, 1, 0), \\ \text{blue} &= (0, 0, 1), \\ \text{yellow} &= (1, 1, 0), \\ \text{magenta} &= (1, 0, 1), \end{aligned}$$

其中M是最大强度。在需要控制精确强度的应用中，我们必须实际测量256个可能的强度，而这些强度在屏幕上的不同点可能不同，特别是对于CRT。它们也可能随视角而变化。幸运的是，很少有应用需要如此精确的校准。

3.3 RGB Color

在小学里，你可能了解到初选是红色，黄色和蓝色，例如，黄色+蓝色=绿色。这是减色混合，这与显示器中发生的更熟悉的添加剂混合有根本的不同。



颜色红绿蓝的添加剂混合规则。

在RGB加法混色我们有（图3.12）

$$\begin{aligned} \text{红色} + \text{绿色} &= \text{黄色} \\ \text{绿色} + \text{蓝色} &= \text{青色} \\ \text{蓝色} + \text{红色} &= \text{洋红色} \\ \text{红色} + \text{绿色} + \text{蓝色} &= \text{白色} \end{aligned}$$

颜色"青色"是蓝绿色，颜色"品红色"是紫色。

如果允许我们将主灯从完全关闭（由像素值0表示）调暗到完全打开（由1表示），我们可以创建可以在RGB显示器上显示的所有颜色。红色、绿色和蓝色像素值创建具有红色、绿色和蓝色轴的三维RGB颜色立方体。坐标轴的可低坐标范围从零到一。彩色立方体如图3.13所示。

立方体角落的颜色是

$$\begin{aligned} \text{black} &= (0, 0, 0), \\ \text{red} &= (1, 0, 0), \\ \text{green} &= (0, 1, 0), \\ \text{blue} &= (0, 0, 1), \\ \text{yellow} &= (1, 1, 0), \\ \text{magenta} &= (1, 0, 1), \end{aligned}$$

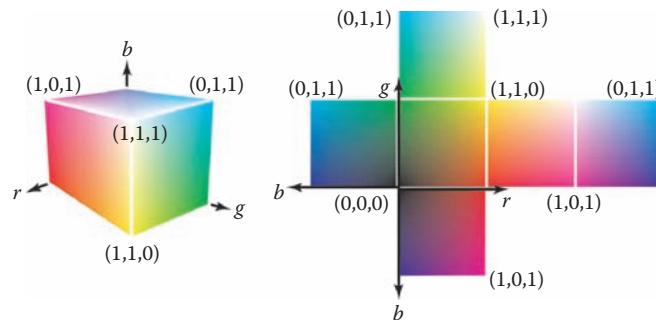


Figure 3.13. The RGB color cube in 3D and its faces unfolded. Any RGB color is a point in the cube.

$$\begin{aligned} \text{cyan} &= (0, 1, 1), \\ \text{white} &= (1, 1, 1). \end{aligned}$$

Actual RGB levels are often given in quantized form, just like the grayscales discussed in Section 3.2.2. Each component is specified with an integer. The most common size for these integers is one byte each, so each of the three RGB components is an integer between 0 and 255. The three integers together take up three bytes, which is 24 bits. Thus a system that has “24-bit color” has 256 possible levels for each of the three primary colors. Issues of gamma correction discussed in Section 3.2.2 also apply to each RGB component separately.

3.4 Alpha Compositing

Often we would like to only partially overwrite the contents of a pixel. A common example of this occurs in *compositing*, where we have a background and want to insert a foreground image over it. For opaque pixels in the foreground, we just replace the background pixel. For entirely transparent foreground pixels, we do not change the background pixel. For *partially* transparent pixels, some care must be taken. Partially transparent pixels can occur when the foreground object has partially transparent regions, such as glass. But, the most frequent case where foreground and background must be blended is when the foreground object only partly covers the pixel, either at the edge of the foreground object, or when there are sub-pixel holes such as between the leaves of a distant tree.

The most important piece of information needed to blend a foreground object over a background object is the *pixel coverage*, which tells the fraction of the pixel covered by the foreground layer. We can call this fraction α . If we want

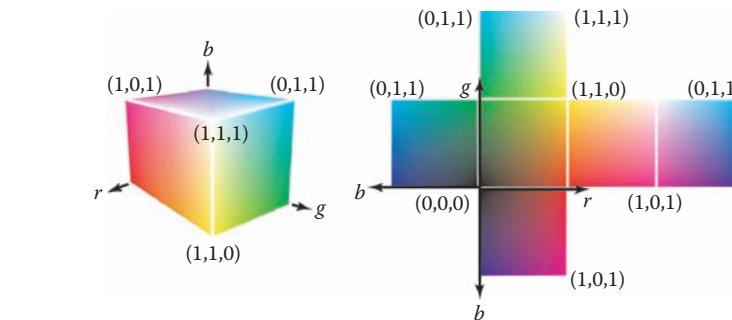


Figure 3.13. RGB颜色立方体在3D和它的脸展开。任何RGB颜色是一个点在cube.

$$\begin{aligned} \text{cyan} &= (0, 1, 1), \\ \text{white} &= (1, 1, 1). \end{aligned}$$

实际的RGB电平通常以量化形式给出，就像3.2.2节中讨论的grayscale一样。每个组件都用整数指定。这些整数最常见的大小是每个一个字节，因此三个RGB分量中的每一个都是0到255之间的整数。三个整数一起占用三个字节，即24位。因此，具有“24位颜色”的系统对于三原色中的每一种具有256个可能的级别。第3.2.2节中讨论的伽玛校正问题也分别适用于每个RGB组件。

3.4 阿尔法合成

通常我们只想部分复盖像素的内容。一个常见的例子发生在合成中，我们有一个背景，并希望在它上面插入一个前景图像。对于前景中的不透明像素，我们只是替换背景像素。对于完全透明的前景像素，我们不会更改背景像素。对于部分透明的像素，必须注意一些。部分透明的像素可以在前景对象具有部分透明的区域时发生，例如玻璃。但是，前景和背景必须混合的最常见情况是当前景对象仅部分复盖像素时，或者在前景对象的边缘，或者当存在诸如远处树的叶子之间的子像素孔时。

在背景对象上混合前景对象所需最重要的信息是像素复盖率，它告诉前景层复盖的像素的比例。我们可以称这个分数为 α 。如果我们想

to composite a foreground color \mathbf{c}_f over background color \mathbf{c}_b , and the fraction of the pixel covered by the foreground is α , then we can use the formula

$$\mathbf{c} = \alpha\mathbf{c}_f + (1 - \alpha)\mathbf{c}_b. \quad (3.2)$$

For an opaque foreground layer, the interpretation is that the foreground object covers area α within the pixel's rectangle and the background object covers the remaining area, which is $(1 - \alpha)$. For a transparent layer (think of an image painted on glass or on tracing paper, using translucent paint), the interpretation is that the foreground layer blocks the fraction $(1 - \alpha)$ of the light coming through from the background and contributes a fraction α of its own color to replace what was removed. An example of using Equation (3.2) is shown in Figure 3.14.

The α values for all the pixels in an image might be stored in a separate grayscale image, which is then known as an *alpha mask* or *transparency mask*. Or the information can be stored as a fourth channel in an RGB image, in which case it is called the *alpha channel*, and the image can be called an RGBA image. With 8-bit images, each pixel then takes up 32 bits, which is a conveniently sized chunk in many computer architectures.

Although Equation (3.2) is what is usually used, there are a variety of situations where α is used differently (Porter & Duff, 1984).

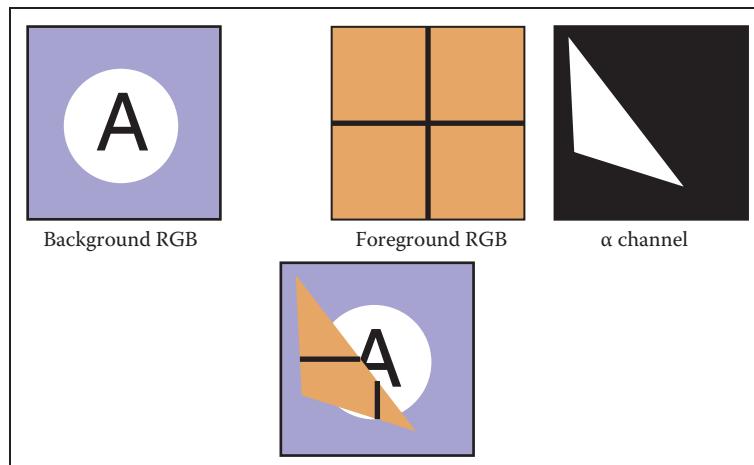


Figure 3.14. An example of compositing using Equation (3.2). The foreground image is in effect cropped by the α channel before being put on top of the background image. The resulting composite is shown on the bottom.

要将一个前景色 \mathbf{c}_f 复合在背景色 \mathbf{c}_b 之上，并且前景所复盖的像素的分数为 α ，那么我们可以使用公式

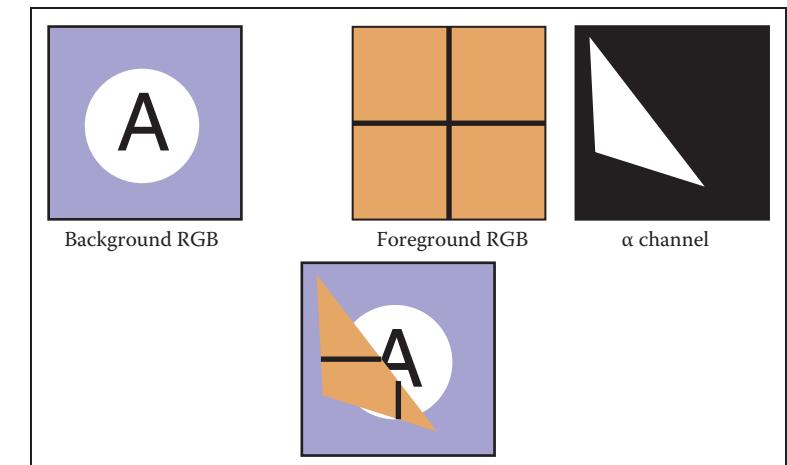
$$\mathbf{c} = \alpha\mathbf{c}_f + (1 - \alpha)\mathbf{c}_b. \quad (3.2)$$

对于不透明的前景层，解释是前景对象复盖像素矩形内的区域 α ，背景对象复盖剩余区域，即 $(1 - \alpha)$ 。对于透明层（想想图像

画在玻璃上或描图纸上，使用半透明油漆），解释是前景层阻挡了来自背景的光的分数 $(1 - \alpha)$ ，并贡献了它自己颜色的分数 α 来取代被移除的东西。使用公式 (3.2) 的示例如图3.14所示。

图像中所有像素的 α 值可能存储在单独的灰度图像中，然后称为alpha蒙版或透明度蒙版。或者信息可以作为第四通道存储在RGB图像中，在这种情况下它被称为alpha通道，并且图像可以被称为RGBA图像。对于8位图像，每个像素占用32位，这在许多计算机体系结构中是一个方便大小的块。

虽然方程 (3.2) 是通常使用的，但有多种情况下 α 的使用方式不同（波特和达夫，1984）。



使用等式(3.2)合成的示例。前景图像实际上是由 α 通道裁剪的，然后再放在背景图像的顶部。所得到的复合物显示在底部。



3.4.1 Image Storage

Most RGB image formats use eight bits for each of the red, green, and blue channels. This results in approximately three megabytes of raw information for a single million-pixel image. To reduce the storage requirement, most image formats allow for some kind of compression. At a high level, such compression is either *lossless* or *lossy*. No information is discarded in lossless compression, while some information is lost unrecoverably in a lossy system. Popular image storage formats include:

- **jpeg.** This lossy format compresses image blocks based on thresholds in the human visual system. This format works well for natural images.
- **tiff.** This format is most commonly used to hold binary images or losslessly compressed 8- or 16-bit RGB although many other options exist.
- **ppm.** This very simple lossless, uncompressed format is most often used for 8-bit RGB images although many options exist.
- **png.** This is a set of lossless formats with a good set of open source management tools.

Because of compression and variants, writing input/output routines for images can be involved. Fortunately one can usually rely on library routines to read and write standard file formats. For quick-and-dirty applications, where simplicity is valued above efficiency, a simple choice is to use raw ppm files, which can often be written simply by dumping the array that stores the image in memory to a file, prepending the appropriate header.

Frequently Asked Questions

- Why don't they just make monitors linear and avoid all this gamma business?

Ideally the 256 possible intensities of a monitor should *look* evenly spaced as opposed to being linearly spaced in energy. Because human perception of intensity is itself nonlinear, a gamma between 1.5 and 3 (depending on viewing conditions) will make the intensities approximately uniform in a subjective sense. In this way, gamma is a feature. Otherwise the manufacturers would make the monitors linear.



3.4.1 图像存储

大多数RGB图像格式对每个红色、绿色和蓝色通道使用8位。这导致一个百万像素的图像大约有三兆字节的原始信息。为了减少存储需求，大多数图像格式允许进行某种压缩。在较高的水平上，这种压缩要么是无损的，要么是有损的。在无损压缩中没有信息被丢弃，而一些信息在有损系统中不可恢复地丢失。流行的图像存储格式包括：

*jpeg。这种有损格式基于人类视觉系统中的阈值来压缩图像块。这种格式适用于自然图像。

*蒂夫。这种格式最常用于保存二进制图像或无损压缩的8或16位RGB，尽管存在许多其他选项。

*ppm。这种非常简单的无损，未压缩格式最常用于8位RGB图像，尽管存在许多选项。

*png。这是一组无损格式，具有一套很好的开源管理工具。

由于压缩和变体，可以涉及为图像编写输入输出例程。幸运的是，人们通常可以依靠库例程来读取和写入标准文件格式。对于简单性高于效率的快速和肮脏的应用程序，一个简单的选择是使用原始ppm文件，通常只需将存储在内存中的图像的数组转储到文件中，并在适当的头之前写入。

常见问题

- 为什么他们不只是使显示器线性，并避免所有这些伽玛业务？

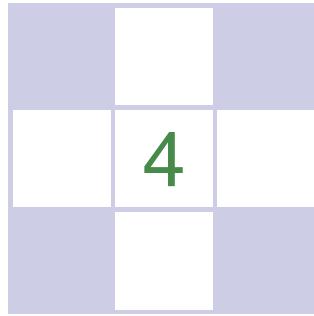
理想情况下，显示器的256个可能的强度应该看起来均匀间隔，而不是能量线性间隔。由于人类对强度的感知本身是非线性的，因此介于1.5和3之间的伽马（取决于观看条件）将使强度在主观意义上近似均匀。通过这种方式，gamma是一个特征。否则，制造商将使监视器线性。

Exercise

1. Simulate an image acquired from the Bayer mosaic by taking a natural image (preferably a scanned photo rather than a digital photo where the Bayer mosaic may already have been applied) and creating a grayscale image composed of interleaved red/green/blue channels. This simulates the raw output of a digital camera. Now create a true RGB image from that output and compare with the original.

Exercise

1. 通过拍摄自然图像（优选扫描照片而不是其中可能已经应用了拜耳马赛克的数码照片）并创建由交错的红绿蓝通道组成的灰度图像来模拟从拜耳马赛克这模拟了数码相机的原始输出。现在从该输出创建一个真正的RGB图像，并与原始图像进行比较。



Ray Tracing

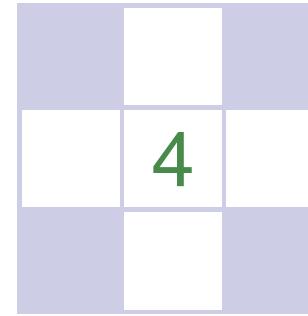
One of the basic tasks of computer graphics is *rendering* three-dimensional objects: taking a scene, or model, composed of many geometric objects arranged in 3D space and producing a 2D image that shows the objects as viewed from a particular viewpoint. It is the same operation that has been done for centuries by architects and engineers creating drawings to communicate their designs to others.

Fundamentally, rendering is a process that takes as its input a set of objects and produces as its output an array of pixels. One way or another, rendering involves considering how each object contributes to each pixel; it can be organized in two general ways. In *object-order rendering*, each object is considered in turn, and for each object all the pixels that it influences are found and updated. In *image-order rendering*, each pixel is considered in turn, and for each pixel all the objects that influence it are found and the pixel value is computed. You can think of the difference in terms of the nesting of loops: in image-order rendering the “for each pixel” loop is on the outside, whereas in object-order rendering the “for each object” loop is on the outside.

Image-order and object-order rendering approaches can compute exactly the same images, but they lend themselves to computing different kinds of effects and have quite different performance characteristics. We’ll explore the comparative strengths of the approaches in Chapter 8 after we have discussed them both, but, broadly speaking, image-order rendering is simpler to get working and more flexible in the effects that can be produced, and usually (though not always) takes much more execution time to produce a comparable image.

If the output is a vector image rather than a raster image, rendering doesn’t have to involve pixels, but we’ll assume raster images in this book.

In a ray tracer, it is easy to compute accurate shadows and reflections, which are awkward in the object-order framework.



光线追踪

计算机图形学的基本任务之一是渲染三维objects：拍摄一个场景或模型，由许多排列在3D空间中的几何对象组成，并生成一个2D图像，显示从特定视点看这是几个世纪以来由建筑师和工程师创建图纸以将他们的设计传达给其他人所做的相同操作。

从根本上说，渲染是一个过程，它将一组对象作为其输入，并产生一个像素数组作为其输出。不管怎样，渲染涉及到考虑每个对象对每个像素的贡献；它可以用两种一般方式组织。在对象顺序渲染中，依次考虑每个对象，并为每个对象找到并更新它影响的所有像素。在图像顺序渲染中，依次考虑每个像素，并为每个像素找到影响它的所有对象并计算像素值。您可以从循环嵌套的角度来考虑不同之处：在图像顺序渲染中，“每个像素”循环位于外部，而在对象顺序渲染中，“每个对象”循环位于外部。

图像顺序和对象顺序渲染方法可以计算完全相同的图像，但它们适用于计算不同类型的效果，并且具有完全不同的性能特征。我们将探索比较

在我们讨论了这两种方法之后，第8章中的方法的优势，但是，从广义上讲，图像顺序渲染更容易工作，并且在可以产生的效果上更灵活，并且通常（尽管不总是）需要更多的执行时间来生成可比的图像。

如果输出是矢量图像而不是光栅图像
渲染不必涉及像素，但我们将在本书中假设光栅图像。

在光线追踪器中，很容易计算出准确的阴影和反射，这在对象顺序框架中很尴尬。

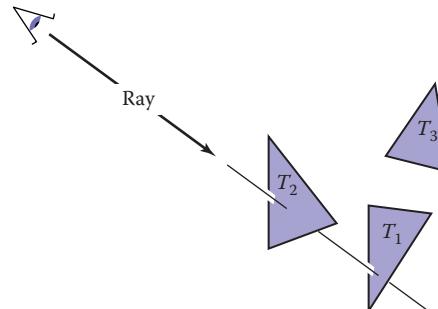


Figure 4.1. The ray is “traced” into the scene and the first object hit is the one seen through the pixel. In this case, the triangle T_2 is returned.

Ray tracing is an image-order algorithm for making renderings of 3D scenes, and we’ll consider it first because it’s possible to get a ray tracer working without developing any of the mathematical machinery that’s used for object-order rendering.

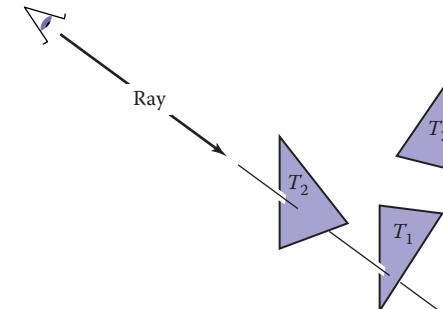
4.1 The Basic Ray-Tracing Algorithm

A ray tracer works by computing one pixel at a time, and for each pixel the basic task is to find the object that is seen at that pixel’s position in the image. Each pixel “looks” in a different direction, and any object that is seen by a pixel must intersect the *viewing ray*, a line that emanates from the viewpoint in the direction that pixel is looking. The particular object we want is the one that intersects the viewing ray nearest the camera, since it blocks the view of any other objects behind it. Once that object is found, a *shading* computation uses the intersection point, surface normal, and other information (depending on the desired type of rendering) to determine the color of the pixel. This is shown in Figure 4.1, where the ray intersects two triangles, but only the first triangle hit, T_2 , is shaded.

A basic ray tracer therefore has three parts:

1. *ray generation*, which computes the origin and direction of each pixel’s viewing ray based on the camera geometry;
2. *ray intersection*, which finds the closest object intersecting the viewing ray;
3. *shading*, which computes the pixel color based on the results of ray intersection.

The structure of the basic ray tracing program is:



光线被“追踪”到场景中，第一个命中的对象是通过像素看到的对象。在这种情况下，返回三角形T2。

光线追踪是一种用于渲染3D场景的图像顺序算法，我们将首先考虑它，因为它可以让光线追踪器工作，而无需开发用于对象顺序渲染的任何数学机器。

4.1 基本光线追踪算法

光线追踪器的工作原理是一次计算一个像素，对于每个像素，基本任务是找到在图像中该像素位置看到的对象。每个像素“看起来”在不同的方向上，并且任何被像素看到的物体都必须与观察射线相交，这条线从像素所看的方向上的视点发出。我们想要的特定对象是与距离相机最近的观察光线相交的对象，因为它会阻挡后面任何其他对象的视图。找到该对象后，着色计算将使用交点、表面法线和其他信息（取决于所需的渲染类型）来确定像素的颜色。这如图4.1所示，其中射线与两个三角形相交，但只有第一个三角形命中， T_2 ，是阴影。因此，一个基本的射线追踪器有三个部分：

- 1.射线生成，根据相机几何形状计算每个像素的观察射线的原点和方向；
- 2.射线相交，它找到与观察射线相交的最接近的对象；
- 3.阴影，它根据射线相交的结果计算像素颜色。

基本光线追踪程序的结构是：



```

for each pixel do
    compute viewing ray
    find first object hit by ray and its surface normal  $n$ 
    set pixel color to value computed from hit point, light, and  $n$ 

```

This chapter covers basic methods for ray generation, ray intersection, and shading, that are sufficient for implementing a simple demonstration ray tracer. For a really useful system, more efficient ray intersection techniques from Chapter 12 need to be added, and the real potential of a ray tracer will be seen with the more advanced shading methods from Chapter 10 and the additional rendering techniques from Chapter 13.

4.2 Perspective

The problem of representing a 3D object or scene with a 2D drawing or painting was studied by artists hundreds of years before computers. Photographs also represent 3D scenes with 2D images. While there are many unconventional ways to make images, from cubist painting to fisheye lenses (Figure 4.2) to peripheral cameras, the standard approach for both art and photography, as well as computer graphics, is *linear perspective*, in which 3D objects are projected onto an *image plane* in such a way that straight lines in the scene become straight lines in the image.

The simplest type of projection is *parallel projection*, in which 3D points are mapped to 2D by moving them along a *projection direction* until they hit the image plane (Figures 4.3–4.4). The view that is produced is determined by the choice of projection direction and image plane. If the image plane is perpendicular

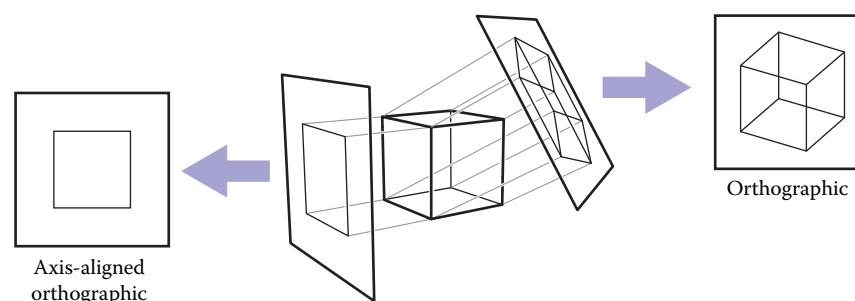


Figure 4.3. When projection lines are parallel and perpendicular to the image plane, the resulting views are called orthographic.



对于每个像素做
计算查看射线查找第一个被射线击中的物体及其表面法
线 n 设置像素颜色，以根据命中点、光和 n 计算值

本章介绍了射线生成、射线相交和着色的基本方法，这些方法足以实现一个简单的演示射线跟踪器。对于一个真正有用的系统，需要添加第12章中更有效的光线相交技术，并且光线跟踪器的真正潜力将通过第10章中更高级的着色方法和第13章中更多的渲染技术来看到。

4.2 Perspective

用2D绘图或绘画表示3D对象或场景的问题是在计算机之前数百年由艺术家研究的。照片也用2D图像表示3D场景。虽然有许多非常规的方式来制作图像，从立体派绘画到鱼眼镜头（图4.2）到外围相机，但艺术和摄影以及计算机图形学的标准方法是线性透视，其中3d物

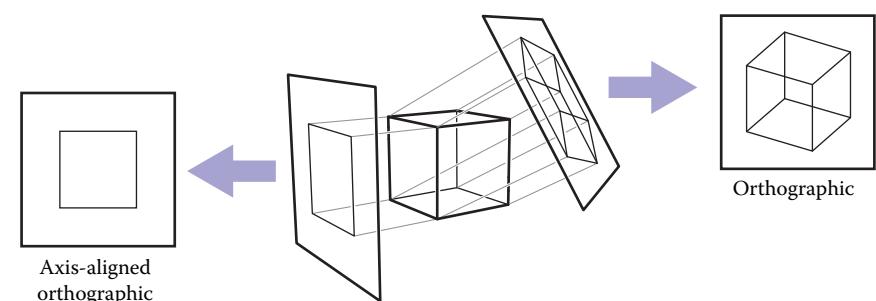


Figure 4.2. An image taken with a fisheye lens is not a linear perspective image. Photo courtesy Philip Green spun.



用鱼眼镜头拍摄的图像不是
林耳透视图像。照片由Philip
Green spun提供。

最简单的投影类型是平行投影，其中3D点通过沿投影方向移动它们直到它们击中图像平面而映射到2D（图4.3–4.4）。产生的视图由投影方向和图像平面的选择决定。如果图像平面垂直



当投影线平行且垂直于图像平面时，生成的视图称为正投影。

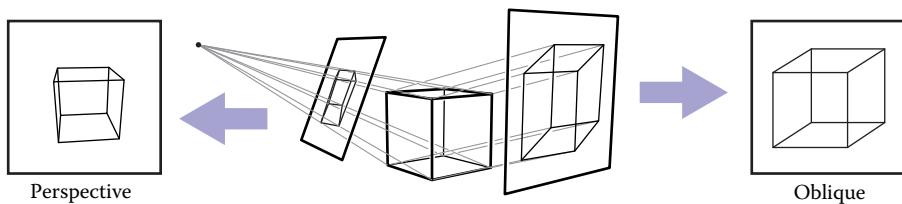


Figure 4.4. A parallel projection that has the image plane at an angle to the projection direction is called oblique (right). In perspective projection, the projection lines all pass through the viewpoint, rather than being parallel (left). The illustrated perspective view is non-oblique because a projection line drawn through the center of the image would be perpendicular to the image plane.

to the view direction, the projection is called *orthographic*; otherwise it is called *oblique*.

Some books reserve “orthographic” for projection directions that are parallel to the coordinate axes.

Parallel projections are often used for mechanical and architectural drawings because they keep parallel lines parallel and they preserve the size and shape of planar objects that are parallel to the image plane.

The advantages of parallel projection are also its limitations. In our everyday experience (and even more so in photographs) objects look smaller as they get farther away, and as a result parallel lines receding into the distance do not appear parallel. This is because eyes and cameras don’t collect light from a single viewing direction; they collect light that passes through a particular viewpoint. As has been recognized by artists since the Renaissance, we can produce natural-looking views using *perspective projection*: we simply project along lines that pass through a single point, the *viewpoint*, rather than along parallel lines (Fig-

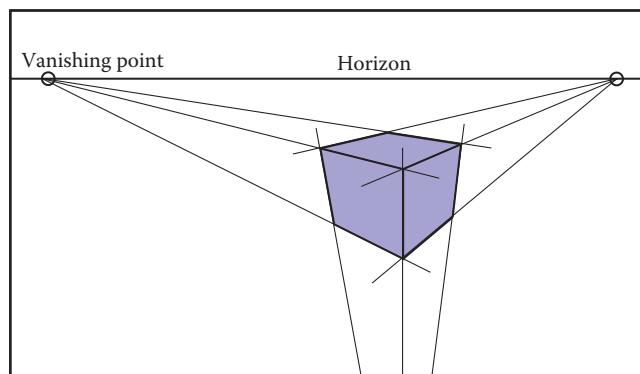
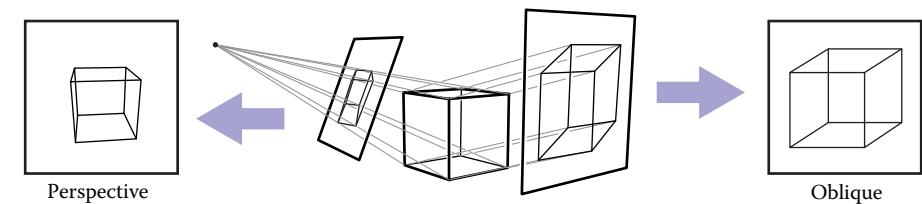


Figure 4.5. In three-point perspective, an artist picks “vanishing points” where parallel lines meet. Parallel horizontal lines will meet at a point on the horizon. Every set of parallel lines has its own vanishing points. These rules are followed automatically if we implement perspective based on the correct geometric principles.



像平面与投影方向成一定角度的平行投影称为斜（右）。在透视投影中，投影线都穿过视点，而不是平行（左）。图示透视图是非倾斜的，因为通过图像中心绘制的投影线将垂直于图像平面。

到视图方向，投影称为正投影；否则称为斜向。

有些书籍保留了与坐标轴平行的投影方向的“正投影”。

平行投影通常用于机械和建筑图纸，因为它们保持平行线平行，并且它们保留与图像平面平行的平面对象的大小和形状。

平行投影的优点也是其局限性。在我们的日常经验中（在照片中更是如此），物体看起来更小，因为它们越来越远，结果后退到远处的平行线并不平行。这是因为眼睛和相机不会从单个观察方向收集光线：他们收集穿过特定视点的光线。正如文艺复兴时期以来艺术家们所认识到的那样，我们可以使用透视投影来产生自然的视角：我们只是沿着穿过一个点的线投影，而不是沿着平行线（图）

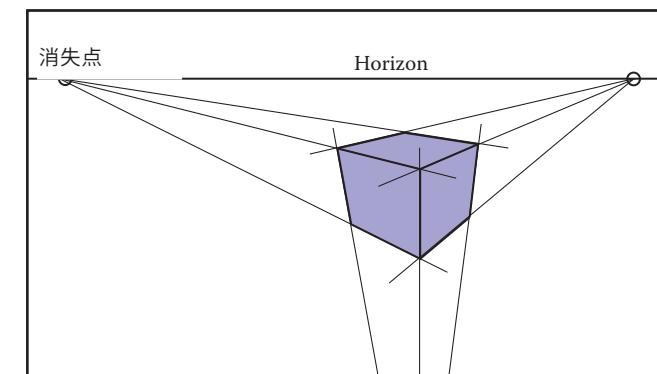


Figure 4.5. 在三点视角中，艺术家选择平行线相遇的“消失点”。平行的水平线将在地平线上的一点相遇。每组平行线都有自己的消失点。如果我们基于正确的几何原理实现透视，则会自动遵循这些规则。

ure 4.4). In this way, objects farther from the viewpoint naturally become smaller when they are projected. A perspective view is determined by the choice of viewpoint (rather than projection direction) and image plane. As with parallel views, there are oblique and non-oblique perspective views; the distinction is made based on the projection direction at the center of the image.

You may have learned about the artistic conventions of *three-point perspective*, a system for manually constructing perspective views (Figure 4.5). A surprising fact about perspective is that all the rules of perspective drawing will be followed automatically if we follow the simple mathematical rule underlying perspective: objects are projected directly toward the eye, and they are drawn where they meet a view plane in front of the eye.

4.3 Computing Viewing Rays

From the previous section, the basic tools of ray generation are the viewpoint (or view direction, for parallel views) and the image plane. There are many ways to work out the details of camera geometry; in this section we explain one based on orthonormal bases that supports normal and oblique parallel and orthographic views.

In order to generate rays, we first need a mathematical representation for a ray. A ray is really just an origin point and a propagation direction; a 3D parametric line is ideal for this. As discussed in Section 2.5.7, the 3D parametric line from the eye e to a point s on the image plane (Figure 4.6) is given by

$$\mathbf{p}(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e}).$$

This should be interpreted as, “we advance from e along the vector $(s - e)$ a fractional distance t to find the point p .” So given t , we can determine a point p . The point e is the ray’s *origin*, and $s - e$ is the ray’s *direction*.

Note that $\mathbf{p}(0) = \mathbf{e}$, and $\mathbf{p}(1) = \mathbf{s}$, and more generally, if $0 < t_1 < t_2$, then $\mathbf{p}(t_1)$ is closer to the eye than $\mathbf{p}(t_2)$. Also, if $t < 0$, then $\mathbf{p}(t)$ is “behind” the eye. These facts will be useful when we search for the closest object hit by the ray that is not behind the eye.

To compute a viewing ray, we need to know e (which is given) and s . Finding s may seem difficult, but it is actually straightforward if we look at the problem in the right coordinate system.

All of our ray-generation methods start from an orthonormal coordinate frame known as the *camera frame*, which we’ll denote by e , for the eye point, or viewpoint, and u , v , and w for the three basis vectors, organized with u pointing rightward (from the camera’s view), v pointing upward, and w pointing backward, so

ure4.4)。这样，离视点较远的物体在投影时自然变小。透视图由视点（而不是投影方向）和图像平面的选择决定。与平行视图一样，有斜向和非斜向透视视图；根据图像中心处的投影方向进行区分。

您可能已经了解了*three-point perspective*的艺术惯例，这是一个手动构建透视图的系统（图4.5）。关于透视的一个令人怀疑的事实是，如果我们遵循简单的数学规则，透视绘制的所有规则都将自动遵循：物体直接投射到眼睛，它们被绘制在它们与眼睛前面的视图平面相遇的地方。

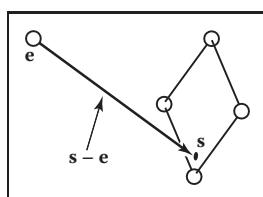


Figure 4.6. The ray from the eye to a point on the image plane.

Caution: we are overloading the variable t , which is the ray parameter and also the v -coordinate of the top edge of the image.

4.3 计算观察射线

从上一节开始，射线生成的基本工具是视点（或视图方向，用于平行视图）和图像平面。有很多方法可以计算出相机几何的细节；在本节中，我们将解释一个基于支持正常和斜向平行和正交视图的正交基的方法。

为了生成射线，我们首先需要一个射线的数学表示。射线实际上只是一个原点和一个传播方向；3D参数线是理想的。如第2.5.7节所讨论的，从眼睛 e 到图像平面上的点 s 的3D参数线（图4.6）由下式给出

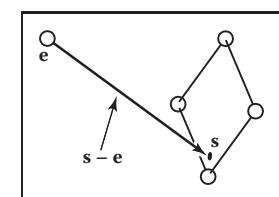
$$\mathbf{p}(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e}).$$

这应该被解释为，“我们从 e 沿着向量 $(s - e)$ 前进一个分数距离 t 以找到点 p 。”所以给定 t ，我们可以确定一个点 p 。点 e 是射线的原点， $s - e$ 是射线的方向。

请注意， $\mathbf{p}(0) = \mathbf{e}$ ，并且 $\mathbf{p}(1) = \mathbf{s}$ ，更一般地说，如果 $0 < t_1 < t_2$ ，则 $\mathbf{p}(t_1)$ 比 $\mathbf{p}(t_2)$ 更接近眼睛。此外，如果 $t < 0$ ，则 $\mathbf{p}(t)$ 在眼睛“后面”。当我们搜索被不在眼睛后面的光线击中的最接近的物体时，这些事实将是有用的。

要计算观察射线，我们需要知道 e （给定）和 s 。找到 s 可能看起来很困难，但如果我们在正确的坐标系中查看问题，实际上很简单。

我们所有的射线生成方法都是从一个称为相机框架的正交坐标系开始的，我们将用 e 表示眼睛点或视点， u 、 v 和 w 表示三个基矢量， u 指向右（从相机的角度看）， v 指向上， w 指向后，所以



从眼睛到图像平面上的点的射线。

注意：我们正在重载变量 t ，它是射线参数，也是图像顶部边缘的 v 坐标。

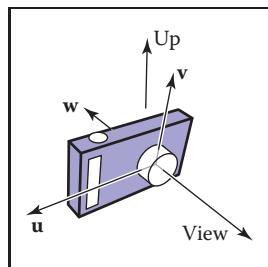


Figure 4.8. The vectors of the camera frame, together with the view direction and up direction. The w vector is opposite the view direction, and the v vector is coplanar with w and the up vector.

Since v and w have to be perpendicular, the up vector and v are not generally the same. But setting the up vector to point straight upward in the scene will orient the camera in the way we would think of as “up-right.”

It might seem logical that orthographic viewing rays should start from infinitely far away, but then it would not be possible to make orthographic views of an object inside a room, for instance.

Many systems assume that $l = -r$ and $b = -t$ so that a width and a height suffice.

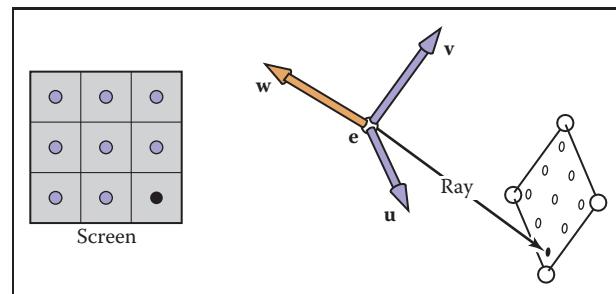


Figure 4.7. The sample points on the screen are mapped to a similar array on the 3D window. A viewing ray is sent to each of these locations.

that $\{u, v, w\}$ forms a right-handed coordinate system. The most common way to construct the camera frame is from the viewpoint, which becomes e , the *view direction*, which is $-w$, and the *up vector*, which is used to construct a basis that has v and w in the plane defined by the view direction and the up direction, using the process for constructing an orthonormal basis from two vectors described in Section 2.4.7.

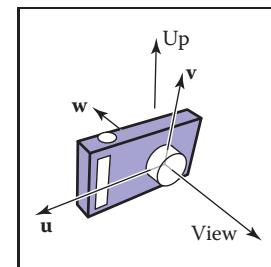
4.3.1 Orthographic Views

For an orthographic view, all the rays will have the direction $-w$. Even though a parallel view doesn't have a viewpoint per se, we can still use the origin of the camera frame to define the plane where the rays start, so that it's possible for objects to be behind the camera.

The viewing rays should start on the plane defined by the point e and the vectors u and v ; the only remaining information required is *where* on the plane the image is supposed to be. We'll define the image dimensions with four numbers, for the four sides of the image: l and r are the positions of the left and right edges of the image, as measured from e along the u direction; and b and t are the positions of the bottom and top edges of the image, as measured from e along the v direction. Usually $l < 0 < r$ and $b < 0 < t$. (See Figure 4.9.)

In Section 3.2 we discussed pixel coordinates in an image. To fit an image with $n_x \times n_y$ pixels into a rectangle of size $(r - l) \times (t - b)$, the pixels are spaced a distance $(r - l)/n_x$ apart horizontally and $(t - b)/n_y$ apart vertically, with a half-pixel space around the edge to center the pixel grid within the image rectangle. This means that the pixel at position (i, j) in the raster image has the position

$$\begin{aligned} u &= l + (r - l)(i + 0.5)/n_x, \\ v &= b + (t - b)(j + 0.5)/n_y, \end{aligned} \quad (4.1)$$

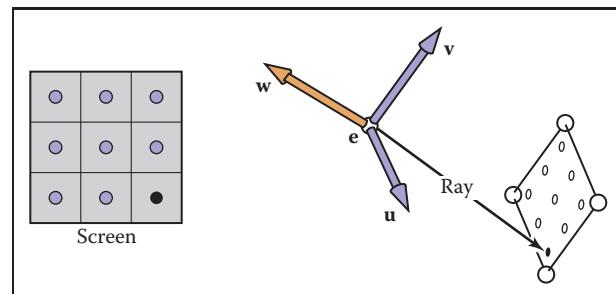


相机帧的矢量，连同视图方向和向上方向。W向量在视图方向上为op，v向量与w和up向量共面。

由于 v 和 w 必须垂直，所以向上矢量和 v 通常不相同。但是，将向上矢量设置为在场景中直向上指向将使摄像机以我们认为的“直立”的方式定向。

正射观察光线应该从无限远的地方开始，这似乎是合乎逻辑的，但是这样就不可能在一个房间内对一个物体进行正射观察。

许多系统假设 $l = -r$ 和 $b = -t$ ，因此宽度和高度就足够了。



屏幕上的样本点映射到3D窗口上的类似数组。一个观看射线被发送到这些位置中的每一个。

$\{u, v, w\}$ 形成了一个右手坐标系。最常见的方式构造相机帧是从视点，其变为 e ，视图方向，其为 $-w$ ，以及向上矢量，其用于构造在由视图方向和向上方向限定的平面中具有 v 和 w 的基础。

从2.4.7节中描述的两个向量构建正交基的过程。

4.3.1 正投影视图

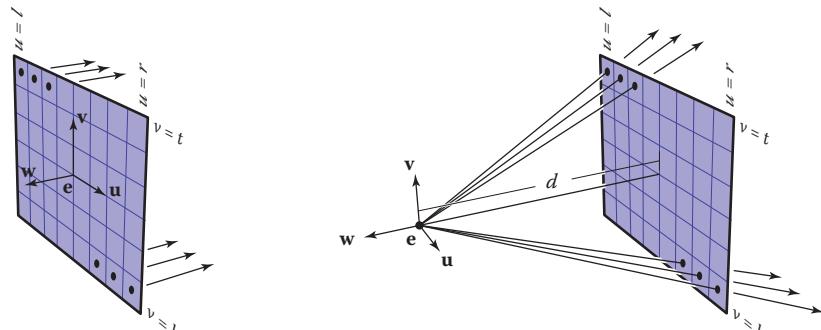
对于正投影视图，所有光线将具有方向 $-w$ 。即使平行视图本身没有视点，我们仍然可以使用相机框架用于定义光线开始的平面，以便物体可能位于相机后面。

观察射线应该从点 e 和矢量 u 和 v 定义的平面开始；唯一剩下的信息是图像应该在平面上的哪个位置。我们将用四个数字定义图像的尺寸，用于图像的四个边： l 和 r 是图像的左右边缘的位置，从 e 沿 u 方向测量； b 和 t 是

图像的底部和顶部边缘的位置，如从 e 沿 v 方向测量的。通常 $l < 0 < r$ 且 $b < 0 < t$ 。（见图4.9。）

在第3.2节中，我们讨论了图像中的像素坐标。为了将具有 $n_x \times n_y$ 像素的图像拟合到大小为 $(r - l) \times (t - b)$ 的矩形中，像素在水平方向间隔一段距离 $(r - l)/n_x$ ，在垂直方向间隔一段距离 $(t - b)/n_y$ ，边缘这意味着光栅图像中位置 (i, j) 处的像素具有该位置

$$\begin{aligned} u &= l + (r - l)(i + 0.5)/n_x, \\ v &= b + (t - b)(j + 0.5)/n_y, \end{aligned} \quad (4.1)$$



Parallel projection
same direction, different origins

Perspective projection
same origin, different directions

Figure 4.9. Ray generation using the camera frame. Left: In an orthographic view, the rays start at the pixels' locations on the image plane, and all share the same direction, which is equal to the view direction. Right: In a perspective view, the rays start at the viewpoint, and each ray's direction is defined by the line through the viewpoint, e , and the pixel's location on the image plane.

where (u, v) are the coordinates of the pixel's position on the image plane, measured with respect to the origin e and the basis $\{u, v\}$.

In an orthographic view, we can simply use the pixel's image-plane position as the ray's starting point, and we already know the ray's direction is the view direction. The procedure for generating orthographic viewing rays is then:

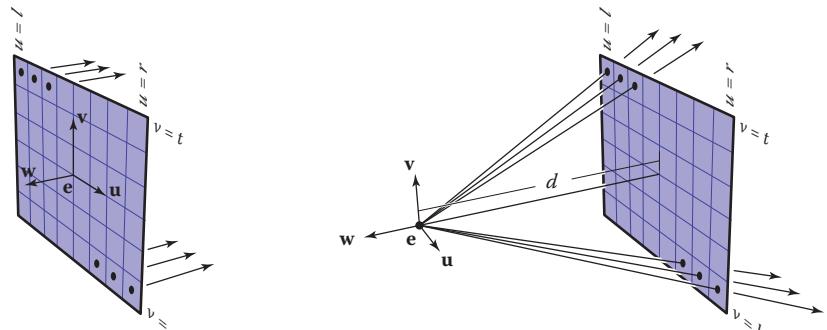
```
compute  $u$  and  $v$  using (4.1)
ray.direction  $\leftarrow -w$ 
ray.origin  $\leftarrow e + u \mathbf{u} + v \mathbf{v}$ 
```

It's very simple to make an oblique parallel view: just allow the image plane normal w to be specified separately from the view direction d . The procedure is then exactly the same, but with d substituted for $-w$. Of course w is still used to construct u and v .

4.3.2 Perspective Views

For a perspective view, all the rays have the same origin, at the viewpoint; it is the directions that are different for each pixel. The image plane is no longer positioned at e , but rather some distance d in front of e ; this distance is the *image plane distance*, often loosely called the *focal length*, because choosing d plays the same role as choosing focal length in a real camera. The direction of each ray is defined by the viewpoint and the position of the pixel on the image plane. This situation is illustrated in Figure 4.9, and the resulting procedure is similar to the

With l and r both specified, there is redundancy: moving the viewpoint a bit to the right and correspondingly decreasing l and r will not change the view (and similarly on the v -axis).



平行投影同向不同原点

透视投影同源, 不同方向

使用相机框架的射线生成。左图：在正交视图中，光线从图像平面上像素的位置开始，并且都共享相同的方向，该方向等于视图方向。右：在透视图中，光线从视点开始，每条光线的方向由穿过视点的线 e 和像素在图像平面上的位置定义。

其中 (u, v) 是像素在图像平面上的位置坐标，相对于原点 e 和基础 $\{u, v\}$ 测量。

在正交视图中，我们可以简单地使用像素的图像平面位置作为射线的起点，并且我们已经知道射线的方向是视图方向。生成正投影观察射线的程序是：

```
使用(4.1)ray计算u和v。
方向 $-w$ 射线。产地 $\leftarrow e +$ 
 $uu+vv$ 
```

制作斜平行视图非常简单：只需允许图像平面法线 w 与视图方向 d 分开指定。然后过程完全相同，但用 d 替代 w 。当然 w 仍然用于构造 u 和 v 。

当 l 和 r 都指定时，存在冗余：将视点向右移动一点并相应地减小 l 和 r 不会改变视图（并且类似地在 v 轴上）。

4.3.2 透视视图

对于透视视图，所有光线在视点处具有相同的原点；它是每个像素不同的方向。像平面不再定位在 e 处，而是在 e 前面的某个距离 d ；这个距离是像平面距离，通常松散地称为焦距，因为选择 d 与在真实相机中选择焦距起着相同的作用：每条射线的方向由视点和像素在图像平面上的位置定义。这种情况如图 4.9 所示，所得到的过程类似于

orthographic one:

```
compute  $u$  and  $v$  using (4.1)
ray.direction  $\leftarrow -d \mathbf{w} + u \mathbf{u} + v \mathbf{v}$ 
ray.origin  $\leftarrow \mathbf{e}$ 
```

As with parallel projection, oblique perspective views can be achieved by specifying the image plane normal separately from the projection direction, then replacing $-d \mathbf{w}$ with dd in the expression for the ray direction.

4.4 Ray-Object Intersection

Once we've generated a ray $\mathbf{e} + t\mathbf{d}$, we next need to find the first intersection with any object where $t > 0$. In practice, it turns out to be useful to solve a slightly more general problem: find the first intersection between the ray and a surface that occurs at a t in the interval $[t_0, t_1]$. The basic ray intersection is the case where $t_0 = 0$ and $t_1 = +\infty$. We solve this problem for both spheres and triangles. In the next section, multiple objects are discussed.

4.4.1 Ray-Sphere Intersection

Given a ray $\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$ and an implicit surface $f(\mathbf{p}) = 0$ (see Section 2.5.3), we'd like to know where they intersect. Intersection points occur when points on the ray satisfy the implicit equation, so the values of t we seek are those that solve the equation

$$f(\mathbf{p}(t)) = 0 \quad \text{or} \quad f(\mathbf{e} + t\mathbf{d}) = 0.$$

A sphere with center $\mathbf{c} = (x_c, y_c, z_c)$ and radius R can be represented by the implicit equation

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - R^2 = 0.$$

We can write this same equation in vector form:

$$(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0.$$

Any point \mathbf{p} that satisfies this equation is on the sphere. If we plug points on the ray $\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$ into this equation, we get an equation in terms of t that is satisfied by the values of t that yield points on the sphere:

$$(\mathbf{e} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{e} + t\mathbf{d} - \mathbf{c}) - R^2 = 0.$$

orthographic one:

```
使用(4.1)ray计算u和v。方向ζ-
dw+uu+vv射线。产地来源: e
```

与平行投影一样，斜视透视图可以通过指定与投影方向分开的图像平面法线，然后在射线方向的表达式中用 dd 替换 dw 来实现。

4.4 Ray-Object Intersection

一旦我们生成了射线 $\mathbf{e}+td$ ，我们接下来需要找到与 $t>0$ 的任何对象的第一个交点。在实践中，事实证明，解决一个稍微更一般的问题是有用的：找到射线和在区间 $[t_0 t_1]$ 中以 t 发生的表面之间的第一个交点。基本射线相交是 $t_0=0$ 和 $t_1=+\infty$ 的情况。我们为球体和三角形解决了这个问题。在下一节中，将讨论多个对象。

4.4.1 Ray-Sphere Intersection

给定一条射线 $\mathbf{p}(t)=\mathbf{e}+td$ 和一个隐式表面 $f(p)=0$ （见第2.5.3节），我们想知道它们在哪里相交。当射线上的点满足隐式方程时会出现交点，因此我们寻求的 t 值是求解方程的值

$$f(\mathbf{p}(t)) = 0 \quad \text{or} \quad f(\mathbf{e} + t\mathbf{d}) = 0.$$

中心 $c = (x_c, y_c, z_c)$ 和半径 R 的球体可以用隐式方程表示

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - R^2 = 0.$$

我们可以用矢量形式写出同样的方程：

$$(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0.$$

满足该方程的任何点 p 都在球体上。如果我们将射线 $p(t)=e+td$ 上的点插入这个方程中，我们得到一个以 t 为单位的方程，该方程由产生球体上点的 t 的值满足：

$$(\mathbf{e} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{e} + t\mathbf{d} - \mathbf{c}) - R^2 = 0.$$



Rearranging terms yields

$$(\mathbf{d} \cdot \mathbf{d})t^2 + 2\mathbf{d} \cdot (\mathbf{e} - \mathbf{c})t + (\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - R^2 = 0.$$

Here, everything is known except the parameter t , so this is a classic quadratic equation in t , meaning it has the form

$$At^2 + Bt + C = 0.$$

The solution to this equation is discussed in Section 2.2. The term under the square root sign in the quadratic solution, $B^2 - 4AC$, is called the *discriminant* and tells us how many real solutions there are. If the discriminant is negative, its square root is imaginary and the line and sphere do not intersect. If the discriminant is positive, there are two solutions: one solution where the ray enters the sphere and one where it leaves. If the discriminant is zero, the ray grazes the sphere, touching it at exactly one point. Plugging in the actual terms for the sphere and canceling a factor of two, we get

$$t = \frac{-\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}) \pm \sqrt{(\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}))^2 - (\mathbf{d} \cdot \mathbf{d})((\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - R^2)}}{(\mathbf{d} \cdot \mathbf{d})}.$$

In an actual implementation, you should first check the value of the discriminant before computing other terms. If the sphere is used only as a bounding object for more complex objects, then we need only determine whether we hit it; checking the discriminant suffices.

As discussed in Section 2.5.4, the normal vector at point \mathbf{p} is given by the gradient $\mathbf{n} = 2(\mathbf{p} - \mathbf{c})$. The unit normal is $(\mathbf{p} - \mathbf{c})/R$.

4.4.2 Ray-Triangle Intersection

There are many algorithms for computing ray-triangle intersections. We will present the form that uses barycentric coordinates for the parametric plane containing the triangle, because it requires no long-term storage other than the vertices of the triangle (Snyder & Barr, 1987).

To intersect a ray with a parametric surface, we set up a system of equations where the Cartesian coordinates all match:

$$\left. \begin{aligned} x_e + tx_d &= f(u, v) \\ y_e + ty_d &= g(u, v) \\ z_e + tz_d &= h(u, v) \end{aligned} \right\} \quad \text{or, } \mathbf{e} + t\mathbf{d} = \mathbf{f}(u, v).$$



重新排列项产生

$$(\mathbf{d} \cdot \mathbf{d})t^2 + 2\mathbf{d} \cdot (\mathbf{e} - \mathbf{c})t + (\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - R^2 = 0.$$

在这里，除了参数 t 之外，一切都是已知的，所以这是 t 中的经典二次方程，意味着它具有以下形式

$$At^2 + Bt + C = 0.$$

该方程的解在第2.2节中讨论。二次解中平方根符号下的项， B^2-4AC ，称为判别式，告诉我们有多少个真正的解决方案。如果判别式为负，则其平方根为虚数，线和球体不相交。如果判别式为正，则有两种解决方案：一种是射线进入球体的解决方案，另一种是射线离开球体的解决方案。如果判别式为零，则射线擦过球体，恰好在一个点上接触它。插入球体的实际项并取消2的因子，我们得到

$$t = \frac{-\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}) \pm \sqrt{(\mathbf{d} \cdot (\mathbf{e} - \mathbf{c}))^2 - (\mathbf{d} \cdot \mathbf{d})((\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - R^2)}}{(\mathbf{d} \cdot \mathbf{d})}.$$

在实际实现中，在计算其他项之前，应首先检查判别式的值。如果球体仅用作更复杂对象的边界对象，那么我们只需要确定是否击中它；检查判别式就足够了。

如第2.5.4节所讨论的，点 \mathbf{p} 处的法向量由梯度 $\mathbf{n}=2(\mathbf{p}-\mathbf{c})$ 给出。单位法线为 $(\mathbf{p}-\mathbf{c})/R$ 。

4.4.2 Ray-Triangle Intersection

有许多算法用于计算射线-三角形交叉点。我们将展示使用barycentric坐标为参数平面containing三角形的形式，因为它不需要长期存储，除了三角形的vertices (Snyder & Barr, 1987)。

为了使射线与参数曲面相交，我们建立了一个笛卡尔坐标全部匹配的方程组：

$$\left. \begin{aligned} x_e + tx_d &= f(u, v) \\ y_e + ty_d &= g(u, v) \\ z_e + tz_d &= h(u, v) \end{aligned} \right\} \quad \text{or, } \mathbf{e} + t\mathbf{d} = \mathbf{f}(u, v).$$

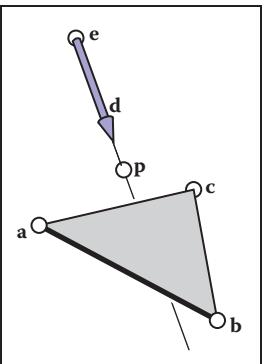


Figure 4.10. The ray hits the plane containing the triangle at point **p**.

Here, we have three equations and three unknowns (t , u , and v), so we can solve numerically for the unknowns. If we are lucky, we can solve for them analytically.

In the case where the parametric surface is a parametric plane, the parametric equation can be written in vector form as discussed in Section 2.7.2. If the vertices of the triangle are \mathbf{a} , \mathbf{b} , and \mathbf{c} , then the intersection will occur when

$$\mathbf{e} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}), \quad (4.2)$$

for some t , β , and γ . The intersection \mathbf{p} will be at $\mathbf{e} + t\mathbf{d}$ as shown in Figure 4.10. Again, from Section 2.7.2, we know the intersection is inside the triangle if and only if $\beta > 0$, $\gamma > 0$, and $\beta + \gamma < 1$. Otherwise, the ray has hit the plane outside the triangle, so it misses the triangle. If there are no solutions, either the triangle is degenerate or the ray is parallel to the plane containing the triangle.

To solve for t , β , and γ in Equation (4.2), we expand it from its vector form into the three equations for the three coordinates:

$$\begin{aligned} x_e + t x_d &= x_a + \beta(x_b - x_a) + \gamma(x_c - x_a), \\ y_e + t y_d &= y_a + \beta(y_b - y_a) + \gamma(y_c - y_a), \\ z_e + t z_d &= z_a + \beta(z_b - z_a) + \gamma(z_c - z_a). \end{aligned}$$

This can be rewritten as a standard linear system:

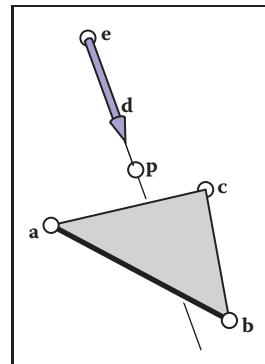
$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}.$$

The fastest classic method to solve this 3×3 linear system is *Cramer's rule*. This gives us the solutions

$$\beta = \frac{\begin{vmatrix} x_a - x_e & x_a - x_c & x_d \\ y_a - y_e & y_a - y_c & y_d \\ z_a - z_e & z_a - z_c & z_d \end{vmatrix}}{|\mathbf{A}|},$$

$$\gamma = \frac{\begin{vmatrix} x_a - x_b & x_a - x_e & x_d \\ y_a - y_b & y_a - y_e & y_d \\ z_a - z_b & z_a - z_e & z_d \end{vmatrix}}{|\mathbf{A}|},$$

$$t = \frac{\begin{vmatrix} x_a - x_b & x_a - x_c & x_a - x_e \\ y_a - y_b & y_a - y_c & y_a - y_e \\ z_a - z_b & z_a - z_c & z_a - z_e \end{vmatrix}}{|\mathbf{A}|},$$



射线在点p处撞击包含triangle的平面。

在这里，我们有三个方程和三个未知数（ t , u 和 v ），因此我们可以对未知数进行数值求解。如果我们幸运的话，我们可以分析地解决它们。在参数化表面是参数化平面的情况下，参数化方程可以如在第2.7.2节中讨论的那样以矢量形式编写。如果三角形的顶点是 \mathbf{a} , \mathbf{b} 和 \mathbf{c} ，那么当

$$\mathbf{e} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}), \quad (4.2)$$

对于一些 t , β 和 γ ，交点 \mathbf{p} 将在 $\mathbf{e}+t\mathbf{d}$ 处，如图4.10所示。再次，从第2.7.2节中，当且仅当 $\beta>0$, $\gamma>0$ ，并且 $\beta+\gamma<1$ 时，我们知道交点在三角形内部。否则，射线已经击中了外面的飞机。

三角形，所以它错过了三角形。如果没有解，则三角形是退化的，或者射线平行于包含三角形的平面。

为了求解方程 (4.2) 中的 t , β 和 γ ，我们将其从矢量形式扩展为三个坐标的三个方程：

$$\begin{aligned} x_e + t x_d &= x_a + \beta(x_b - x_a) + \gamma(x_c - x_a), \\ y_e + t y_d &= y_a + \beta(y_b - y_a) + \gamma(y_c - y_a), \\ z_e + t z_d &= z_a + \beta(z_b - z_a) + \gamma(z_c - z_a). \end{aligned}$$

这可以重写为标准线性系统：

$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_e \\ y_a - y_e \\ z_a - z_e \end{bmatrix}.$$

解决这个 3×3 线性系统的最快的经典方法是Cramer的规则。这给了我们解决方案

$$\beta = \frac{\begin{vmatrix} x_a - x_e & x_a - x_c & x_d \\ y_a - y_e & y_a - y_c & y_d \\ z_a - z_e & z_a - z_c & z_d \end{vmatrix}}{|\mathbf{A}|},$$

$$\gamma = \frac{\begin{vmatrix} x_a - x_b & x_a - x_e & x_d \\ y_a - y_b & y_a - y_e & y_d \\ z_a - z_b & z_a - z_e & z_d \end{vmatrix}}{|\mathbf{A}|},$$

$$t = \frac{\begin{vmatrix} x_a - x_b & x_a - x_c & x_a - x_e \\ y_a - y_b & y_a - y_c & y_a - y_e \\ z_a - z_b & z_a - z_c & z_a - z_e \end{vmatrix}}{|\mathbf{A}|},$$

where the matrix \mathbf{A} is

$$\mathbf{A} = \begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix},$$

and $|\mathbf{A}|$ denotes the determinant of \mathbf{A} . The 3×3 determinants have common sub-terms that can be exploited. Looking at the linear systems with dummy variables

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix},$$

Cramer's rule gives us

$$\beta = \frac{j(ei - hf) + k(gf - di) + l(dh - eg)}{M},$$

$$\gamma = \frac{i(ak - jb) + h(jc - al) + g(bl - kc)}{M},$$

$$t = -\frac{f(ak - jb) + e(jc - al) + d(bl - kc)}{M},$$

where

$$M = a(ei - hf) + b(gf - di) + c(dh - eg).$$

We can reduce the number of operations by reusing numbers such as “ei-minus-hf”

The algorithm for the ray-triangle intersection for which we need the linear solution can have some conditions for early termination. Thus, the function should look something like:

```
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
```

其中矩阵A为

$$\mathbf{A} = \begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix},$$

而A表示A的行列式。3×3行列式具有可以利用的公共子项。用虚拟变量观察线性系统

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix},$$

克莱默的规则给了我们

$$\beta=j(ei - hf)+k(gf - di)+l(dh - eg),$$

$$\gamma=i(ak - jb)+h(jc - al)+g(bl - kc),$$

$$t=-f(ak - jb)+e(jc - al)+d(bl - kc),$$

where

$$M=a(ei - hf)+b(gf - di)+c(dh - eg).$$

我们可以通过重复使用“ei-minus-hf”等数字来减少操作次数。”

射线-三角形相交的算法，我们需要线性的，所以lution可以有提前终止的一些条件。因此，函数应该看起来像：

```
布尔raytri (rayr, vector3a, vector3b, vector
3c, interval[t0, t1])
计算t如果(t<t0)或(t>t1)然
后返回false计算γ如果(γ<0)
或(γ>1)然后返回false计算β
如果(β<0)或(β>1 - γ)然后返
回false返回true
```

4.4.3 Ray-Polygon Intersection

Given a planar polygon with m vertices \mathbf{p}_1 through \mathbf{p}_m and surface normal \mathbf{n} , we first compute the intersection points between the ray $\mathbf{e} + t\mathbf{d}$ and the plane containing the polygon with implicit equation

$$(\mathbf{p} - \mathbf{p}_1) \cdot \mathbf{n} = 0.$$

We do this by setting $\mathbf{p} = \mathbf{e} + t\mathbf{d}$ and solving for t to get

$$t = \frac{(\mathbf{p}_1 - \mathbf{e}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}.$$

This allows us to compute \mathbf{p} . If \mathbf{p} is inside the polygon, then the ray hits it; otherwise, it does not.

We can answer the question of whether \mathbf{p} is inside the polygon by projecting the point and polygon vertices to the xy plane and answering it there. The easiest way to do this is to send any 2D ray out from \mathbf{p} and to count the number of intersections between that ray and the boundary of the polygon (Sutherland, Sproull, & Schumacker, 1974; Glassner, 1989). If the number of intersections is odd, then the point is inside the polygon; otherwise it is not. This is true because a ray that goes in must go out, thus creating a pair of intersections. Only a ray that starts inside will not create such a pair. To make computation simple, the 2D ray may as well propagate along the x -axis:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \end{bmatrix} + s \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

It is straightforward to compute the intersection of that ray with the edges such as (x_1, y_1, x_2, y_2) for $s \in (0, \infty)$.

A problem arises, however, for polygons whose projection into the xy plane is a line. To get around this, we can choose among the xy , yz , or zx planes for whichever is best. If we implement our points to allow an indexing operation, e.g., $\mathbf{p}(0) = x_p$ then this can be accomplished as follows:

```

if (abs(zn) > abs(xn)) and (abs(zn) > abs(yn)) then
    index0 = 0
    index1 = 1
else if (abs(yn) > abs(xn)) then
    index0 = 0
    index1 = 2
else
    index0 = 1
    index1 = 2

```

Now, all computations can use $\mathbf{p}(\text{index0})$ rather than x_p , and so on.

4.4.3 Ray-Polygon Intersection

给定一个具有 m 个顶点 p_1 到 p_m 和表面法线 n 的平面多边形，我们首先用隐式方程计算射线 $e+td$ 和包含多边形的平面之间的交点

$$(\mathbf{p} - \mathbf{p}_1) \cdot \mathbf{n} = 0.$$

我们通过设置 $p=e+td$ 并求解 t 来获得

$$t = \frac{(\mathbf{p}_1 - \mathbf{e}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}.$$

这使我们能够计算 p 。如果 p 在多边形内，那么射线击中它;否则，它不会。

我们可以通过将点和多边形顶点投影到 xy 平面并在那里回答 p 是否在多边形内部的问题。最简单的方法是从 p 发送任何2D射线，并计算该射线与多边形边界之间的截面数 (Sutherland, Sproull, & Schumacker, 1974; Glassner, 1989)。如果交叉点的数量是奇数，则该点在多边形内;否则不是。这是真的，因为进入的光线必须熄灭，从而创建一对交叉点。只有内部开始的光线不会产生这样的一对。为了简化计算，2D射线也可以沿 x 轴传播：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \end{bmatrix} + s \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

计算该射线与 $s \in (0, \infty)$ 的边 (x_1, y_1, x_2, y_2) 的交点很简单。

但是，对于投影到 xy 平面的多边形是一条线的多边形，则会出现问题。为了解决这个问题，我们可以在 xy , yz 或 zx 平面中选择最好的。如果我们实现我们的点以允许索引操作，例如， $p(0)=xp$ ，那么这可以如下完成：

```

if (abs(zn) > abs(xn)) and (abs(zn) > abs(yn)) then
    index0 = 0 index1 = 1
else if (abs(yn) > abs(xn)) then
    index0 = 0 index1 = 2
else
    index0 = 1 index1 = 2

```

index0 = 1
index1 = 2

现在，所有计算都可以使用 $p(\text{index0})$ 而不是 x_p ，依此类推。



Another approach to polygons, one that is often used in practice, is to replace them by several triangles.

4.4.4 Intersecting a Group of Objects

Of course, most interesting scenes consist of more than one object, and when we intersect a ray with the scene we must find only the closest intersection to the camera along the ray. A simple way to implement this is to think of a group of objects as itself being another type of object. To intersect a ray with a group, you simply intersect the ray with the objects in the group and return the intersection with the smallest t value. The following code tests for hits in the interval $t \in [t_0, t_1]$:

```
hit = false
for each object o in the group do
    if (o is hit at ray parameter t and t ∈ [t0, t1]) then
        hit = true
        hitobject = o
        t1 = t
    return hit
```

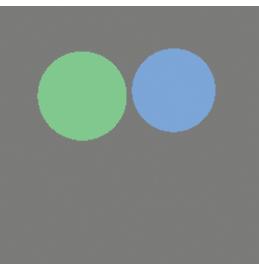


Figure 4.11. A simple scene rendered with only ray generation and surface intersection, but no shading; each pixel is just set to a fixed color depending on which object it hit.

4.5 Shading

Once the visible surface for a pixel is known, the pixel value is computed by evaluating a *shading model*. How this is done depends entirely on the application—methods range from very simple heuristics to elaborate numerical computations. In this chapter we describe the two most basic shading models; more advanced models are discussed in Chapter 10.

Most shading models, one way or another, are designed to capture the process of light reflection, whereby surfaces are illuminated by light sources and reflect part of the light to the camera. Simple shading models are defined in terms of illumination from a point light source. The important variables in light reflection are the light direction \mathbf{l} , which is a unit vector pointing toward the light source; the view direction \mathbf{v} , which is a unit vector pointing toward the eye or camera; the surface normal \mathbf{n} , which is a unit vector perpendicular to the surface at the point where reflection is taking place; and the characteristics of the surface—color, shininess, or other properties depending on the particular model.

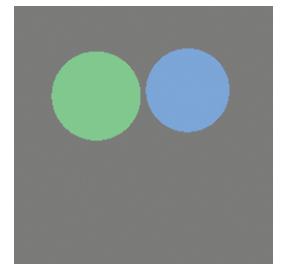


多边形的另一种方法，在实践中经常使用的方法是用几个三角形替换它们。

4.4.4 相交一组对象

当然，大多数有趣的场景由多个对象组成，当我们将光线与场景相交时，我们必须沿着光线找到距离相机最近的交点。实现这一点的一个简单方法是将一组对象视为另一种类型的对象。要将射线与组相交，您只需将射线与组中的对象相交，并返回具有最小 t 值的交点。以下代码测试区间 $t \in [t_0, t_1]$ 中的命中：

```
hit=false对于组中的每个对象
odo
如果 (o在射线参数t和t∈[t0 t1]处被击中) ,
则hit=truehitobject=ot1=t返回hit
```



一个简单的场景，只用光线生成和表面截面，但没有阴影；每个像素只是设置为一个固定的颜色取决于它击中的对象。

4.5 Shading

一旦已知像素的可见表面，就通过评估着色模型来计算像素值。如何做到这一点完全取决于应用程序—方法范围从非常简单的启发式到复杂的数值计算。在本章中，我们将介绍两种最基本的着色模型；更高级的模型将在第10章中讨论。

大多数着色模型，这种或那种方式，旨在捕捉光反射的过程，即表面被光源照亮并将部分光反射到相机。简单的阴影模型是根据点光源的照明来定义的。光反射中的重要变量是光方向 \mathbf{l} ，它是指向光源的单位向量；视图方向 \mathbf{v} ，它是指向眼睛或相机的单位向量；表面法线 \mathbf{n} ，它是垂直于反射发生点的表面的单

4.5.1 Lambertian Shading

Illumination from real point sources falls off as distance squared, but that is often more trouble than it's worth in a simple renderer.

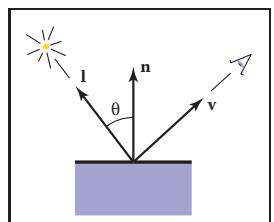


Figure 4.12. Geometry for Lambertian shading.

When in doubt, make light sources neutral in color, with equal red, green, and blue intensities.

The simplest shading model is based on an observation made by Lambert in the 18th century: the amount of energy from a light source that falls on an area of surface depends on the angle of the surface to the light. A surface facing directly toward the light receives maximum illumination; a surface tangent to the light direction (or facing away from the light) receives no illumination; and in between the illumination is proportional to the cosine of the angle θ between the surface normal and the light source (Figure 4.12). This leads to the *Lambertian shading model*:

$$L = k_d I \max(0, \mathbf{n} \cdot \mathbf{l})$$

where L is the pixel color; k_d is the *diffuse coefficient*, or the surface color; and I is the intensity of the light source. Because \mathbf{n} and \mathbf{l} are unit vectors, we can use $\mathbf{n} \cdot \mathbf{l}$ as a convenient shorthand (both on paper and in code) for $\cos \theta$. This equation (as with the other shading equations in this section) applies separately to the three color channels, so the red component of the pixel value is the product of the red diffuse component, the red light source intensity, and the dot product; the same holds for green and blue.

The vector \mathbf{l} is computed by subtracting the intersection point of the ray and surface from the light source position. Don't forget that \mathbf{v} , \mathbf{l} , and \mathbf{n} all must be unit vectors; failing to normalize these vectors is a very common error in shading computations.

4.5.2 Blinn-Phong Shading

Lambertian shading is *view independent*: the color of a surface does not depend on the direction from which you look. Many real surfaces show some degree of shininess, producing highlights, or *specular reflections*, that appear to move around as the viewpoint changes. Lambertian shading doesn't produce any highlights and leads to a very matte, chalky appearance, and many shading models add a *specular component* to Lambertian shading; the Lambertian part is then the *diffuse component*.

A very simple and widely used model for specular highlights was proposed by Phong (Phong, 1975) and later updated by Blinn (J. F. Blinn, 1976) to the form most commonly used today. The idea is to produce reflection that is at its brightest when \mathbf{v} and \mathbf{l} are symmetrically positioned across the surface normal, which is when mirror reflection would occur; the reflection then decreases smoothly as the vectors move away from a mirror configuration.

4.5.1 Lambertian Shading

从真实的点源脱落为距离平方，但这往往是更多的麻烦比它的价值在一个简单的渲染器。

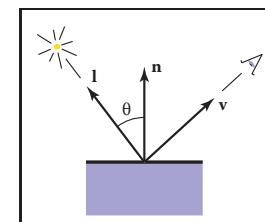


Figure 4.12. 几何形状 Lambertian shading.

如果有疑问，请使光源的颜色为中性，具有相同的红色，绿色和蓝色强度。

$$L = k_d I \max(0, \mathbf{n} \cdot \mathbf{l})$$

其中 L 是像素颜色； k_d 是漫射系数，或表面颜色；以及 I 是光源的强度。因为 n 和 l 是单位向量，我们可以使用 $n \cdot l$ 作为 $\cos \theta$ 的方便速记（无论是在纸上还是在代码中）。该方程（与本节中的其他着色方程一样）分别适用于三个颜色通道，因此像素值的红色分量是红色漫射分量，红色光源强度和点积的乘积；对于绿色和蓝色

通过从光源位置减去射线和表面的交点来计算矢量 l 。不要忘记， v ， l 和 n 都必须是单位向量；未能对这些向量进行归一化是阴影计算中非常常见的错误。

4.5.2 Blinn-Phong Shading

朗伯阴影是独立于视图的：表面的颜色不依赖于你看的方向。许多真实表面显示一定程度的光泽，产生高光或镜面反射，似乎随着视点的变化而移动。朗伯着色不会产生任何高光，并导致非常哑光，白垩外观，许多着色模型为朗伯着色添加了镜面组件；朗伯部分然后是漫射组件。

Phong (Phong, 1975) 提出了一个非常简单且广泛使用的镜面高光模型，后来由Blinn (J.F.Blinn, 1976) 更新为今天最常用的形式。这个想法是在 v 和 l 对称地放置在表面法线上时产生最亮的反射，这是镜面反射发生的时候；然后当矢量远离镜面配置时，反射平滑地减小。

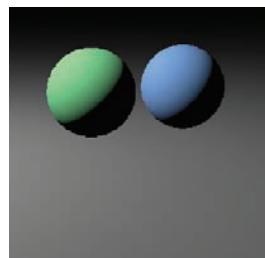


Figure 4.13. A simple scene rendered with diffuse shading from a single light source.

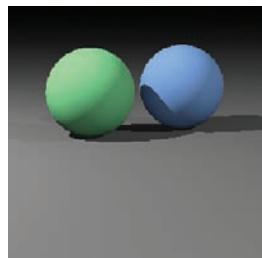


Figure 4.14. A simple scene rendered with diffuse shading and shadows (blue sphere) from three light sources.

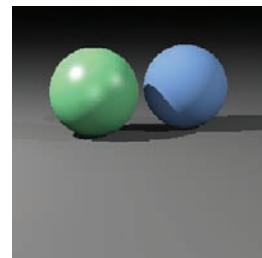


Figure 4.15. A simple scene rendered with diffuse shading and shadows (blue sphere), Blinn-Phong shading (green sphere), and shadows from three light sources.

We can tell how close we are to a mirror configuration by comparing the half vector \mathbf{h} (the bisector of the angle between \mathbf{v} and \mathbf{l}) to the surface normal (Figure 4.16). If the half vector is near the surface normal, the specular component should be bright; if it is far away it should be dim. This result is achieved by computing the dot product between \mathbf{h} and \mathbf{n} (remember they are unit vectors, so $\mathbf{n} \cdot \mathbf{h}$ reaches its maximum of 1 when the vectors are equal), then taking the result to a power $p > 1$ to make it decrease faster. The power, or *Phong exponent*, controls the apparent shininess of the surface. The half vector itself is easy to compute: since \mathbf{v} and \mathbf{l} are the same length, their sum is a vector that bisects the angle between them, which only needs to be normalized to produce \mathbf{h} .

Putting this all together, the Blinn-Phong shading model is as follows:

$$\mathbf{h} = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|},$$

$$L = k_d I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s I \max(0, \mathbf{n} \cdot \mathbf{h})^p,$$

where k_s is the *specular coefficient*, or the specular color, of the surface.

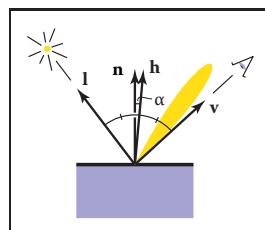


Figure 4.16. Geometry for Blinn-Phong shading.

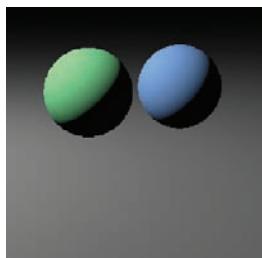
Typical values of p :
10—"eggshell";
100—"mildly shiny";
1000—"really glossy";
10,000—"nearly mirror-like".

When in doubt, make the specular color gray, with equal red, green, and blue values.

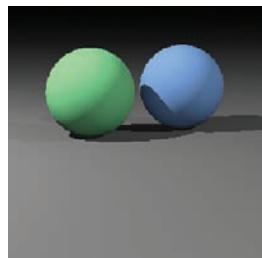
4.5.3 Ambient Shading

Surfaces that receive no illumination at all will be rendered as completely black, which is often not desirable. A crude but useful heuristic to avoid black shadows is to add a constant component to the shading model, one whose contribution to the pixel color depends only on the object hit, with no dependence on the surface geometry at all. This is known as ambient shading—it is as if surfaces were illuminated by "ambient" light that comes equally from everywhere. For

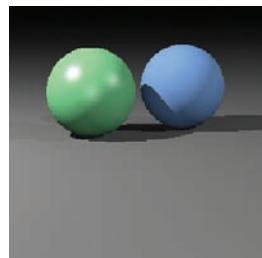
In the real world, surfaces that are not illuminated by light sources are illuminated by indirect reflections from other surfaces.



一个简单的场景渲染与漫射阴影从一个单一的光源。



一个用漫射阴影和阴影渲染的简单场景



使用漫反射阴影 (蓝色球体)、BlinnPhong阴影 (绿色球体) 和来自三个光源的阴影渲染的简单场景。

我们可以通过比较半矢量 \mathbf{h} (\mathbf{v} 和 \mathbf{l} 之间夹角的平分线) 与表面法线 (图4.16) 来判断我们与镜子配置的接近程度。如果半矢量靠近表面法线，则镜面分量

应该是明亮的;如果它很远，它应该是暗淡的。这个结果是通过计算 \mathbf{h} 和 \mathbf{n} 之间的点积来实现的 (记住它们是单位向量，所以当向量相等时， $\mathbf{n} \cdot \mathbf{h}$ 达到最大值1)，然后将结果取为 $p > 1$ 的幂，使其下降得更快。功率，或Phong指数，控制表面的表观光泽。半向量本身很容易计算：由于 \mathbf{v} 和 \mathbf{l} 是相同的长度，它们的总和是一个将它们之间的角度平分的向量，只需要归一化以产生 \mathbf{h} 。

把这一切放在一起，Blinn-Phong阴影模型如下所示：

$$\mathbf{h} = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|},$$

$$L = k_d I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s I \max(0, \mathbf{n} \cdot \mathbf{h})^p,$$

其中 k_s 是表面的镜面系数或镜面颜色。

4.5.3 环境阴影

完全没有照明的表面将呈现为完全黑色，这通常是不可取的。一个粗糙但有用的启发式，以避免黑色阴影

是在着色模型中添加一个常量组件，它对像素颜色的贡献仅取决于对象命中，而完全不依赖于表面几何。这就是所谓的环境阴影——就好像表面被同样来自世界各地的“环境”光照亮一样。为

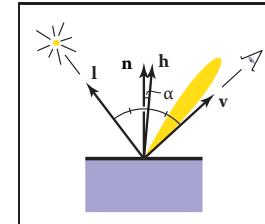


Figure 4.16. 几何形状 Blinn-Phong shading.

P的典型值：10—"蛋壳";
100—轻度有光泽;1000—真正有光泽;10 000—nearllymirror-like。

如果有疑问，请使镜面颜色为灰色，具有相等的红色、绿色和蓝色值。

在现实世界中，没有被光源照亮的表面被来自其他表面的间接反射照亮。

convenience in tuning the parameters, ambient shading is usually expressed as the product of a surface color with an ambient light color, so that ambient shading can be tuned for surfaces individually or for all surfaces together. Together with the rest of the Blinn-Phong model, ambient shading completes the full version of a simple and useful shading model:

$$L = k_a I_a + k_d I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s I \max(0, \mathbf{n} \cdot \mathbf{h})^n, \quad (4.3)$$

When in doubt set the ambient color to be the same as the diffuse color.

where k_a is the surface's ambient coefficient, or "ambient color," and I_a is the ambient light intensity.

4.5.4 Multiple Point Lights

A very useful property of light is *superposition*—the effect caused by more than one light source is simply the sum of the effects of the light sources individually. For this reason, our simple shading model can easily be extended to handle N light sources:

$$L = k_a I_a + \sum_{i=1}^N [k_d I_i \max(0, \mathbf{n} \cdot \mathbf{l}_i) + k_s I_i \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p], \quad (4.4)$$

where I_i , \mathbf{l}_i , and \mathbf{h}_i are the intensity, direction, and half vector of the i^{th} light source.

4.6 A Ray-Tracing Program

We now know how to generate a viewing ray for a given pixel, how to find the closest intersection with an object, and how to shade the resulting intersection. These are all the parts required for a program that produces shaded images with hidden surfaces removed.

```

for each pixel do
    compute viewing ray
    if (ray hits an object with  $t \in [0, \infty)$ ) then
        Compute  $\mathbf{n}$ 
        Evaluate shading model and set pixel to that color
    else
        set pixel color to background color
    
```

在调整参数时，环境阴影通常表示为表面颜色与环境光颜色的乘积，从而环境阴影可以单独地或一起针对所有表面进行调整。与Blinn-Phong模型的其余部分一起，环境着色完成了一个简单而有用的着色模型的完整版本：

$$L = k_a I_a + k_d I \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s I \max(0, \mathbf{n} \cdot \mathbf{h})^n, \quad (4.3)$$

如果有疑问，请将ambient
颜色设置为与漫射颜色相同
。

其中 k_a 是表面的环境系数，或"环境颜色"， I_a 是
境光强度。

4.5.4 多点灯

光的一个非常有用的属性是叠加-由多个光源引起的效果只是单独光源效果的总和。出于这个原因，我们的简单着色模型可以很容易地扩展到处理N个光源：

$$L = k_a I_a + \sum_{i=1}^N [k_d I_i \max(0, \mathbf{n} \cdot \mathbf{l}_i) + k_s I_i \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p], \quad (4.4)$$

其中 I_i 、 \mathbf{l}_i 和 \mathbf{h}_i 是第*i*个光源的强度、方向和半矢量。

4.6 A Ray-Tracing Program

我们现在知道如何为给定像素生成查看光线，如何找到与对象最近的交叉点，以及如何对结果交叉点进行阴影处理。这些是产生隐藏表面去除的阴影图像的程序所需的所有部分。

```

对于每个像素做
    计算查看射线if (射线以 $t \in [0, \infty)$  击中
    物体) 然后计算 $\mathbf{n}$ 
    评估着色模型并将像素设置为该颜色。
    将像素颜色设置为背景颜色
    
```



Here the statement “if ray hits an object . . .” can be implemented using the algorithm of Section 4.4.4.

In an actual implementation, the surface intersection routine needs to somehow return either a reference to the object that is hit, or at least its normal vector and shading-relevant material properties. This is often done by passing a record/structure with such information. In an object-oriented implementation, it is a good idea to have a class called something like *surface* with derived classes *triangle*, *sphere*, *group*, etc. Anything that a ray can intersect would be under that class. The ray-tracing program would then have one reference to a “surface” for the whole model, and new types of objects and efficiency structures can be added transparently.

4.6.1 Object-Oriented Design for a Ray-Tracing Program

As mentioned earlier, the key class hierarchy in a ray tracer are the geometric objects that make up the model. These should be subclasses of some geometric object class, and they should support a *hit* function (Kirk & Arvo, 1988). To avoid confusion from use of the word “object,” *surface* is the class name often used. With such a class, you can create a ray tracer that has a general interface that assumes little about modeling primitives and debug it using only spheres. An important point is that anything that can be “hit” by a ray should be part of this class hierarchy, e.g., even a collection of surfaces should be considered a subclass of the surface class. This includes efficiency structures, such as bounding volume hierarchies; they can be hit by a ray, so they are in the class.

For example, the “abstract” or “base” class would specify the *hit* function as well as a bounding box function that will prove useful later:

```
class surface
    virtual bool hit(ray e + td, real t0, real t1, hit-record rec)
    virtual box bounding-box()
```

Here (t_0, t_1) is the interval on the ray where hits will be returned, and *rec* is a record that is passed by reference; it contains data such as the *t* at the intersection when *hit* returns true. The type *box* is a 3D “bounding box,” that is two points that define an axis-aligned box that encloses the surface. For example, for a sphere, the function would be implemented by

```
box sphere::bounding-box()
    vector3 min = center - vector3(radius,radius,radius)
    vector3 max = center + vector3(radius,radius,radius)
    return box(min, max)
```



这里的语句“如果雷击中一个物体。.”可以使用第4.4.4节的算法来实现。

在实际实现中，曲面相交例程需要考虑如何返回对被击中对象的引用，或者至少返回其正常的vector和着色相关的材质属性。这通常是通过传递包含此类信息的记录结构来完成的。在面向对象的实现中，有一个名为*surface*的类和派生类*triangle*, *sphere*, *group*等是一个好主意。任何光线可以相交的东西都在该类之下。然后，光线追踪程序将对整个模型的“表面”有一个引用，并且可以透明地添加新类型的对象和效率结构。

4.6.1 光线追踪程序的面向对象设计

如前所述，光线追踪器中的关键类层次结构是构成模型的几何对象。这些应该是一些几何对象类的子类，它们应该支持一个命中函数（Kirk & Arvo, 1988）。为了避免混淆使用“对象”一词，表面是经常使用的类名。使用这样的类，您可以创建一个具有一般接口的光线追踪器，该接口对建模基元几乎没有任何假设，并仅使用球体对其进行调试。重要的一点是，任何可以被光线“击中”的东西都应该是这个类层次结构的一部分，例如，即使是曲面的集合也应该被视为*surface*类的子类。这包括效率结构，如边界体积层次结构；它们可以被射线击中，因此它们在类中。

例如，“abstract”或“base”类将指定*hit*函数以及稍后证明有用的边界框函数：

类表面
virtual bool hit (ray e+td, real t₀, real t₁, hit-record rec)
virtual box bounding-box ()

这里 (t_0, t_1) 是射线上返回命中的间隔，*rec*是一个通过引用传递的记录；它包含数据，如命中返回true时相交处的*t*。类型框是一个3D“边界框”，它是两个点，用于定义包围曲面的轴对齐框。例如，对于一个球体，该函数将通过以下方式实现

```
box sphere::bounding-box()
    vector3 min = center - vector3(radius, radius, radius)
    vector3 max = center + vector3(radius, radius, radius)
    return box(min, max)
```

Another class that is useful is material. This allows you to abstract the material behavior and later add materials transparently. A simple way to link objects and materials is to add a pointer to a material in the surface class, although more programmable behavior might be desirable. A big question is what to do with textures; are they part of the material class or do they live outside of the material class? This will be discussed more in Chapter 11.

4.7 Shadows

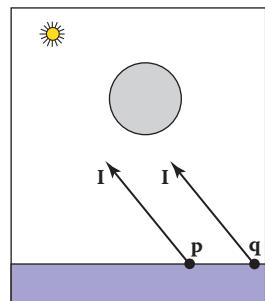


Figure 4.17. The point p is not in shadow, while the point q is in shadow.

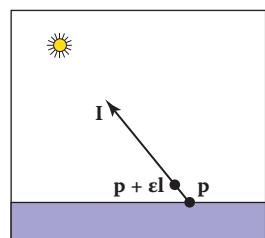


Figure 4.18. By testing in the interval starting at ϵ , we avoid numerical imprecision causing the ray to hit the surface p is on.

To get the algorithm for shading, we add an if statement to determine whether the point is in shadow. In a naive implementation, the shadow ray will check for $t \in [0, \infty)$, but because of numerical imprecision, this can result in an intersection with the surface on which p lies. Instead, the usual adjustment to avoid that problem is to test for $t \in [\epsilon, \infty)$ where ϵ is some small positive constant (Figure 4.18).

If we implement shadow rays for Phong lighting with Equation 4.3 then we have the following:

```
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.t) d
    color c = rec.ka Ia
    if (not scene→hit(p + sl, ε, ∞, srec)) then
        vector3 h = normalized(normalized(l) + normalized(-d))
        c = c + rec.kd I max (0, rec.n · l) + (rec.ks) I (rec.n · h)rec.p
    return c
else
    return background-color
```

另一个有用的类是material。这允许您抽象材质行为，然后透明地添加材质。链接对象和材质的一种简单方法是在surface类中添加指向材质的指针，尽管可能需要更多的可编程行为。一个大问题是处理纹理；它们是材质类的一部分还是位于材质类之外？这将在第11章中更多地讨论。

4.7 Shadows

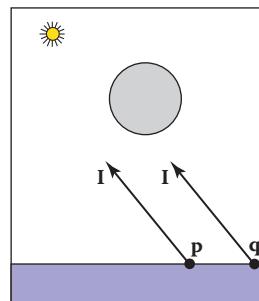
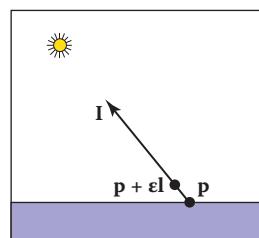


图4.17。 点p不处于阴影中，而点q处于阴影中。

一旦你有一个基本的光线跟踪程序，阴影可以很容易地添加。从第4.5节回想一下，光线来自某个方向l。如果我们想象自己在一个被阴影的表面上的一个点p，如果我们“看”方向l并看到一个物体，这个点是在阴影中。如果没有物体，那么光不会被阻挡。

如图4.17所示，其中光线p+tl没有击中任何物体，因此不处于阴影中。点q处于阴影中，因为光线q+tl确实击中了一个物体。矢量l对于两个点是相同的，因为光“远”。这个假设稍后将被放松。确定进入或离开阴影的射线称为阴影射线，以将它们与观察射线区分开来。

为了得到阴影的算法，我们添加了一个if语句来确定点是否在阴影中。在一个天真的实现中，阴影射线将检查 $t \in [0, \infty)$ ，但由于数值不精确，这可能导致与p所在的表面之间的截面。相反，为了避免这个问题，通常的调整是测试 $t \in [\epsilon, \infty)$ ，其中 ϵ 是一些小的正常数（图4.18）。



通过在开始的间隔中测试ing，我们避免了导致射线撞击表面p上的数值模糊。

如果我们用公式4.3实现Phong照明的阴影射线，那么我们有以下内容：

```
函数raycolor(ray e + td realt0 realt1)hit-rec
ordrec srecif(scene→hit(e + td t0 t1 rec))th
enp=e+(rec.t) d颜色c=rec. kaif(notscen
e→hit(p+sl ← srec))then
```

```
vector3h=归一化(归一化(l) + 归一化(-d)) c=c+r
ec. kdimax(0 rec.n*l)+(rec.ks)I(rec.n*h)rec.p
返回c其
他
return background-color
```

Note that the ambient color is added whether \mathbf{p} is in shadow or not. If there are multiple light sources, we can send a shadow ray before evaluating the shading model for each light. The code above assumes that \mathbf{d} and \mathbf{l} are not necessarily unit vectors. This is crucial for \mathbf{d} , in particular, if we wish to cleanly add *instancing* later (see Section 13.2).

4.8 Ideal Specular Reflection

It is straightforward to add *ideal specular reflection*, or *mirror reflection*, to a ray-tracing program. The key observation is shown in Figure 4.19 where a viewer looking from direction \mathbf{e} sees what is in direction \mathbf{r} as seen from the surface. The vector \mathbf{r} is found using a variant of the Phong lighting reflection Equation (10.6). There are sign changes because the vector \mathbf{d} points toward the surface in this case, so,

$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}. \quad (4.5)$$

In the real world, some energy is lost when the light reflects from the surface, and this loss can be different for different colors. For example, gold reflects yellow more efficiently than blue, so it shifts the colors of the objects it reflects. This can be implemented by adding a recursive call in *raycolor*:

$$\text{color } c = c + k_m \text{raycolor}(\mathbf{p} + s\mathbf{r}, \epsilon, \infty)$$

where k_m (for “mirror reflection”) is the specular RGB color. We need to make sure we test for $s \in [\epsilon, \infty)$ for the same reason as we did with shadow rays; we don’t want the reflection ray to hit the object that generates it.

The problem with the recursive call above is that it may never terminate. For example, if a ray starts inside a room, it will bounce forever. This can be fixed by adding a maximum recursion depth. The code will be more efficient if a reflection ray is generated only if k_m is not zero (black).

4.9 Historical Notes

Ray tracing was developed early in the history of computer graphics (Appel, 1968) but was not used much until sufficient compute power was available (Kay & Greenberg, 1979; Whitted, 1980).

Ray tracing has a lower asymptotic time complexity than basic object-order rendering (Snyder & Barr, 1987; Muuss, 1995; S. Parker et al., 1999; Wald, Slusallek, Benthin, & Wagner, 2001). Although it was traditionally thought of

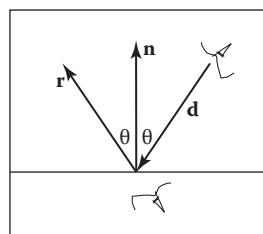


Figure 4.19. When looking into a perfect mirror, the viewer looking in direction \mathbf{d} will see whatever the viewer “below” the surface would see in direction \mathbf{r} .

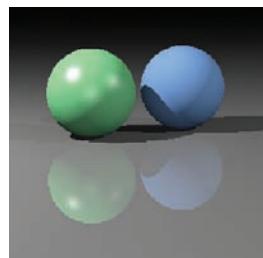


Figure 4.20. A simple scene rendered with diffuse and Blinn-Phong shading, shadows from three light sources, and specular reflection from the floor.

请注意，无论 \mathbf{p} 是否处于阴影中，都会添加环境颜色。如果有多个光源，我们可以在评估每个光源的阴影模型之前发送阴影射线。上面的代码假设 \mathbf{d} 和 \mathbf{l} 不一定是单位向量。这对于 \mathbf{d} 来说是至关重要的，特别是如果我们希望稍后干净地添加实例化（参见第13.2节）。

4.8 理想镜面反射

将理想的镜面反射或镜面反射添加到光线跟踪程序非常简单。关键观察如图4.19所示，从方向 \mathbf{e} 看的观察者看到的是从表面看到的方向 \mathbf{r} 。使用Phong照明反射方程 (10.6) 的变体找到矢量 \mathbf{r} 。有符号变化，因为矢量 \mathbf{d} 在这种情况下指向表面，所以

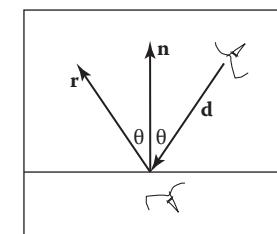
$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}. \quad (4.5)$$

在现实世界中，当光线从表面反射时会损失一些能量，并且这种损失对于不同的颜色可能是不同的。例如，黄金反射黄色比蓝色更有效，因此它会改变它反射的对象的颜色。这可以通过在*raycolor*中添加递归调用来实现：

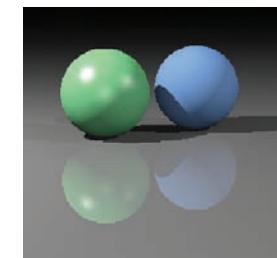
$$\text{color } c = c + k_m \text{raycolor}(\mathbf{p} + s\mathbf{r}, \epsilon, \infty)$$

其中 k_m （用于“镜面反射”）是镜面RGB颜色。我们需要确保我们测试 $s \in [\epsilon, \infty)$ 的原因与我们对阴影射线所做的相同；我们不希望反射射线击中生成它的对象。

上面的递归调用的问题是它可能永远不会终止。例如，如果光线在房间内开始，它将永远反弹。这可以通过添加最大递归深度来修复。如果仅在 k_m 不为零（黑色）时生成反射射线，则代码将更有效。



当看到一个完美的镜子，观看者看方向 \mathbf{d} 将看到任何观看者“下面”的表面将看到方向 \mathbf{r} 。



用diffuse和Blinn-Phong shading渲染的模拟场景，来自三个光源的阴影，以及来自地板的镜面反射。

4.9 历史笔记

光线追踪在计算机图形学的早期发展 (Appel, 1968年)，但直到有足够的计算能力可用 (Kay&Greenberg, 1979年; Whitted, 1980年) 才被大量使用。

光线追踪具有比基本对象阶渲染更低的渐近时间复杂度 (Snyder & Barr, 1987; Muuss, 1995; S.Parker等人。, 1999; Wald, Slusallek, Benthin, & Wagner, 2001)。虽然传统上认为

as an offline method, real-time ray tracing implementations are becoming more and more common.

Frequently Asked Questions

• Why is there no perspective matrix in ray tracing?

The perspective matrix in a z-buffer exists so that we can turn the perspective projection into a parallel projection. This is not needed in ray tracing, because it is easy to do the perspective projection implicitly by fanning the rays out from the eye.

• Can ray tracing be made interactive?

For sufficiently small models and images, any modern PC is sufficiently powerful for ray tracing to be interactive. In practice, multiple CPUs with a shared frame buffer are required for a full-screen implementation. Computer power is increasing much faster than screen resolution, and it is just a matter of time before conventional PCs can ray trace complex scenes at screen resolution.

• Is ray tracing useful in a hardware graphics program?

Ray tracing is frequently used for *picking*. When the user clicks the mouse on a pixel in a 3D graphics program, the program needs to determine which object is visible within that pixel. Ray tracing is an ideal way to determine that.

Exercises

- What are the ray parameters of the intersection points between ray $(1, 1, 1) + t(-1, -1, -1)$ and the sphere centered at the origin with radius 1? Note: this is a good debugging case.
- What are the barycentric coordinates and ray parameter where the ray $(1, 1, 1) + t(-1, -1, -1)$ hits the triangle with vertices $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$? Note: this is a good debugging case.
- Do a back of the envelope computation of the approximate time complexity of ray tracing on “nice” (non-adversarial) models. Split your analysis into the cases of preprocessing and computing the image, so that you can predict the behavior of ray tracing multiple frames for a static model.

作为一种离线方法，实时光线追踪的实现变得越来越普遍。

常见问题

*为什么光线追踪中没有透视矩阵？

存在z缓冲区中的透视矩阵，以便我们可以将透视投影转换为平行投影。这在光线追踪中是不需要的，因为通过从眼睛扇出光线来隐式地进行透视投影很容易。

*光线追踪可以进行交互吗？

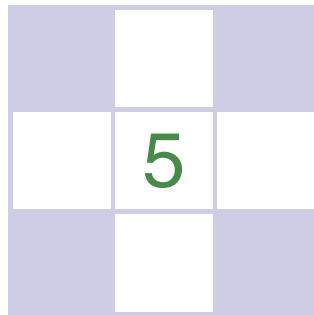
对于足够小的模型和图像，任何现代PC都足够强大，光线跟踪是交互式的。在实践中，全屏实现需要具有共享帧缓冲区的多个CPU。计算机功率的增长速度比屏幕分辨率快得多，传统PC能够以屏幕分辨率对复杂场景进行光线追踪只是时间问题。

*光线追踪在硬件图形程序中有用吗？

光线追踪经常用于拾取。当用户在3D图形程序中的像素上单击鼠标时，程序需要确定哪个对象在该像素内可见。光线追踪是确定这一点的理想方法。

Exercises

- Ray $(1 1 1) + t$ 之间的交点的射线参数是什么(1 1 1)以及以半径为1的原点为中心的球体？注意：这是一个很好的调试案例。
- 射线 $(1 1 1) + t$ 的重心坐标和射线参数是什么(1 1 1)用顶点打三角形 $(1 0 0)$ $(0 1 0)$ 和 $(0, 0, 1)$? 注意：这是一个很好的调试案例。
- 在“nice”（非对抗性）模型上对光线追踪的近似时间复杂度进行包络计算。将您的分析拆分为预处理和计算图像的案例，以便您可以预测静态模型的多帧光线追踪的行为。



Linear Algebra

Perhaps the most universal tools of graphics programs are the matrices that change or *transform* points and vectors. In the next chapter, we will see how a vector can be represented as a matrix with a single column, and how the vector can be represented in a different basis via multiplication with a square matrix. We will also describe how we can use such multiplications to accomplish changes in the vector such as scaling, rotation, and translation. In this chapter, we review basic linear algebra from a geometric perspective, focusing on intuition and algorithms that work well in the two- and three-dimensional case.

This chapter can be skipped by readers comfortable with linear algebra. However, there may be some enlightening tidbits even for such readers, such as the development of determinants and the discussion of singular and eigenvalue decomposition.

5.1 Determinants

We usually think of determinants as arising in the solution of linear equations. However, for our purposes, we will think of determinants as another way to multiply vectors. For 2D vectors a and b , the determinant $|ab|$ is the area of the parallelogram formed by a and b (Figure 5.1). This is a signed area, and the sign is positive if a and b are right-handed and negative if they are left-handed. This means $|ab| = -|ba|$. In 2D we can interpret "right-handed" as meaning we rotate the first vector counterclockwise to close the smallest angle to the second vector. In 3D, the determinant must be taken with three vectors at a time. For

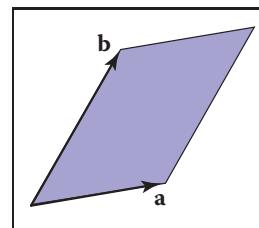


Figure 5.1. The signed area of the parallelogram is $|ab|$, and in this case the area is positive.

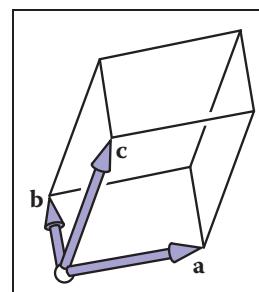
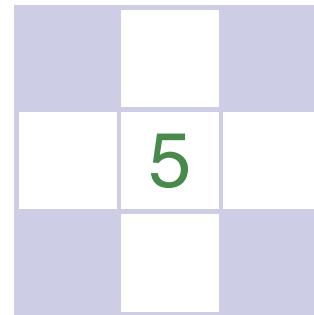


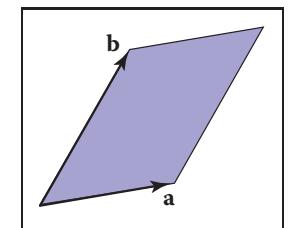
Figure 5.2. The signed volume of the parallelepiped shown is denoted by the determinant $|abc|$, and in this case the volume is positive because the vectors form a right-handed basis.



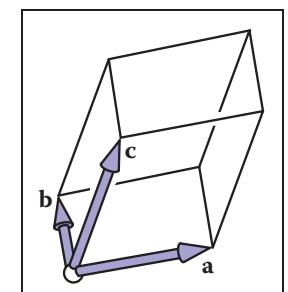
线性代数

也许图形程序最通用的工具是改变或变换点和矢量的矩阵。在下一章中，我们将看到如何将向量表示为具有单列的矩阵，以及如何通过与方阵的乘法在不同的基础上表示向量。我们还将描述如何使用这种乘法来完成矢量的变化，如缩放，旋转和平移。在这

章，我们从几何角度回顾基本的线性代数，专注于直觉和算法，在两个和三维情况下运作良好。熟悉线性代数的读者可以跳过本章。然而，即使对于这样的读者来说，也可能有一些启发性的花絮，例如决定因素的发展以及奇异和特征值分解的讨论。



平行四边形的带符号区域是 ab ，在这种情况下该区域是 positive。



所示的平行六面体的有符号体积由 de 终止 abc 表示，在这种情况下，体积是正的，因为矢量形成右手基础。

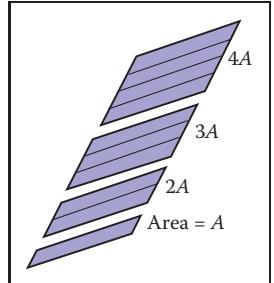


Figure 5.3. Scaling a parallelogram along one direction changes the area in the same proportion.

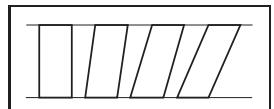


Figure 5.4. Shearing a parallelogram does not change its area. These four parallelograms have the same length base and thus the same area.

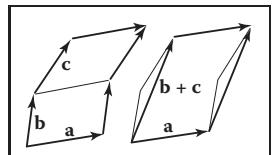


Figure 5.5. The geometry behind Equation 5.1. Both of the parallelograms on the left can be sheared to cover the single parallelogram on the right.

three 3D vectors, \mathbf{a} , \mathbf{b} , and \mathbf{c} , the determinant $|\mathbf{abc}|$ is the signed volume of the parallelepiped (3D parallelogram; a sheared 3D box) formed by the three vectors (Figure 5.2). To compute a 2D determinant, we first need to establish a few of its properties. We note that scaling one side of a parallelogram scales its area by the same fraction (Figure 5.3):

$$|(k\mathbf{a})\mathbf{b}| = |\mathbf{a}(k\mathbf{b})| = k|\mathbf{ab}|.$$

Also, we note that “shearing” a parallelogram does not change its area (Figure 5.4):

$$|(\mathbf{a} + k\mathbf{b})\mathbf{b}| = |\mathbf{a}(\mathbf{b} + k\mathbf{a})| = |\mathbf{ab}|.$$

Finally, we see that the determinant has the following property:

$$|\mathbf{a}(\mathbf{b} + \mathbf{c})| = |\mathbf{ab}| + |\mathbf{ac}|, \quad (5.1)$$

because as shown in Figure 5.5 we can “slide” the edge between the two parallelograms over to form a single parallelogram without changing the area of either of the two original parallelograms.

Now let’s assume a Cartesian representation for \mathbf{a} and \mathbf{b} :

$$\begin{aligned} |\mathbf{ab}| &= |(x_a\mathbf{x} + y_a\mathbf{y})(x_b\mathbf{x} + y_b\mathbf{y})| \\ &= x_a x_b |\mathbf{xx}| + x_a y_b |\mathbf{xy}| + y_a x_b |\mathbf{yx}| + y_a y_b |\mathbf{yy}| \\ &= x_a x_b (0) + x_a y_b (+1) + y_a x_b (-1) + y_a y_b (0) \\ &= x_a y_b - y_a x_b. \end{aligned}$$

This simplification uses the fact that $|\mathbf{vv}| = 0$ for any vector \mathbf{v} , because the parallelograms would all be collinear with \mathbf{v} and thus without area.

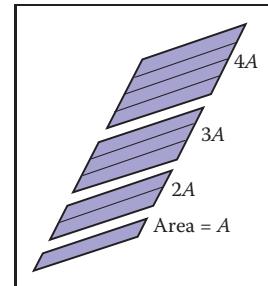
In three dimensions, the determinant of three 3D vectors \mathbf{a} , \mathbf{b} , and \mathbf{c} is denoted $|\mathbf{abc}|$. With Cartesian representations for the vectors, there are analogous rules for parallelepipeds as there are for parallelograms, and we can do an analogous expansion as we did for 2D:

$$\begin{aligned} |\mathbf{abc}| &= |(x_a\mathbf{x} + y_a\mathbf{y} + z_a\mathbf{z})(x_b\mathbf{x} + y_b\mathbf{y} + z_b\mathbf{z})(x_c\mathbf{x} + y_c\mathbf{y} + z_c\mathbf{z})| \\ &= x_a y_b z_c - x_a z_b y_c - y_a x_b z_c + y_a z_b x_c + z_a x_b y_c - z_a y_b x_c. \end{aligned}$$

As you can see, the computation of determinants in this fashion gets uglier as the dimension increases. We will discuss less error-prone ways to compute determinants in Section 5.3.

Example. Determinants arise naturally when computing the expression for one vector as a linear combination of two others—for example, if we wish to express a vector \mathbf{c} as a combination of vectors \mathbf{a} and \mathbf{b} :

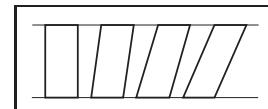
$$\mathbf{c} = a_c \mathbf{a} + b_c \mathbf{b}.$$



沿着一个方向缩放par等位基因图以相同的比例更改区域。

$$|(k\mathbf{a})\mathbf{b}| = |\mathbf{a}(k\mathbf{b})| = k|\mathbf{ab}|.$$

此外，我们注意到“剪切”平行四边形不会改变其面积（图5.4）：



剪切平行四边形不会改变其面积。这四个平行的ogram s具有相同的长度基部并且因此具有相同的面积。

$$\mathbf{a}(\mathbf{b}+\mathbf{c})=\mathbf{ab}+\mathbf{ac} \quad (5.1)$$

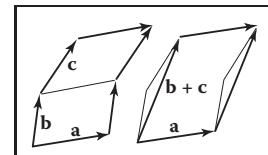
因为如图5.5所示，我们可以“滑动”两个平行四边形之间的边缘，以形成单个平行四边形，而不会改变两个原始平行四边形中的任何一个的面积。

现在让我们假设 \mathbf{a} 和 \mathbf{b} 的笛卡尔表示：

$$\begin{aligned} |\mathbf{ab}| &= |(x_a\mathbf{x} + y_a\mathbf{y})(x_b\mathbf{x} + y_b\mathbf{y})| \\ &= x_a x_b |\mathbf{xx}| + x_a y_b |\mathbf{xy}| + y_a x_b |\mathbf{yx}| + y_a y_b |\mathbf{yy}| \\ &= x_a x_b (0) + x_a y_b (+1) + y_a x_b (-1) + y_a y_b (0) \\ &= x_a y_b - y_a x_b. \end{aligned}$$

这种简化使用了任何向量 \mathbf{v} 的 $|\mathbf{vv}|=0$ 的事实，因为平行四边形都将与 \mathbf{v} 共线，因此没有面积。

在三维中，三个三维向量 \mathbf{a} , \mathbf{b} 和 \mathbf{c} 的行列式表示为 $|\mathbf{abc}|$ 。对于矢量的笛卡尔表示，对于平行四边形有类似的规则，我们可以做一个类似的扩展，就像我们对2D所做的那样：



Geome尝试在等式5.1后面。左侧的两个平行四边形都可以剪切以复盖右侧的单个平行四边形。

$$\begin{aligned} |\mathbf{abc}| &= |(x_a\mathbf{x} + y_a\mathbf{y} + z_a\mathbf{z})(x_b\mathbf{x} + y_b\mathbf{y} + z_b\mathbf{z})(x_c\mathbf{x} + y_c\mathbf{y} + z_c\mathbf{z})| \\ &= x_a y_b z_c - x_a z_b y_c - y_a x_b z_c + y_a z_b x_c + z_a x_b y_c - z_a y_b x_c. \end{aligned}$$

正如您所看到的，随着维度的增加，以这种方式计算行列式变得更加丑陋。我们将在第5.3节中讨论计算行列式的不太容易出错的方法。

例子。当将一个向量的表达式计算为另外两个向量的线性组合时，决定因素自然会出现—例如，如果我们希望将向量 \mathbf{c} 表示为向量 \mathbf{a} 和 \mathbf{b} 的组合：

$$\mathbf{c} = a_c \mathbf{a} + b_c \mathbf{b}.$$

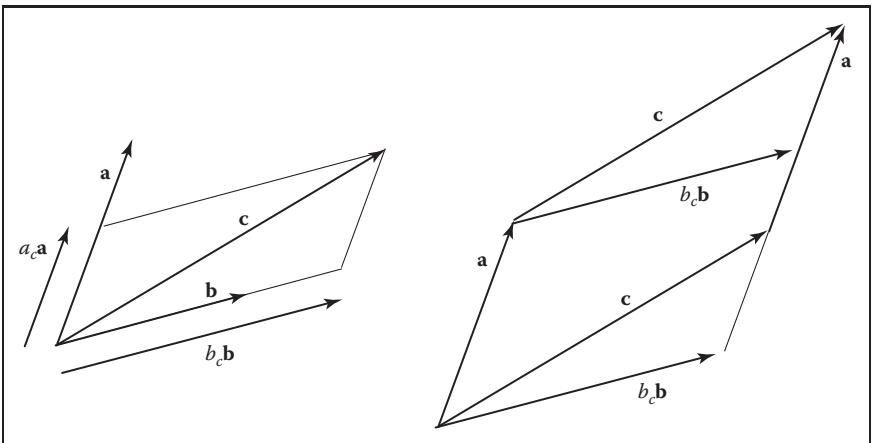


Figure 5.6. On the left, the vector \mathbf{c} can be represented using two basis vectors as $a_c \mathbf{a} + b_c \mathbf{b}$. On the right, we see that the parallelogram formed by \mathbf{a} and \mathbf{c} is a sheared version of the parallelogram formed by $b_c \mathbf{b}$ and \mathbf{a} .

We can see from Figure 5.6 that

$$|(b_c \mathbf{b})\mathbf{a}| = |\mathbf{c}\mathbf{a}|,$$

because these parallelograms are just sheared versions of each other. Solving for b_c yields

$$b_c = \frac{|\mathbf{c}\mathbf{a}|}{|\mathbf{b}\mathbf{a}|}.$$

An analogous argument yields

$$a_c = \frac{|\mathbf{b}\mathbf{c}|}{|\mathbf{b}\mathbf{a}|}.$$

This is the two-dimensional version of *Cramer's rule* which we will revisit in Section 5.3.2. □

5.2 Matrices

A matrix is an array of numeric elements that follow certain arithmetic rules. An example of a matrix with two rows and three columns is

$$\begin{bmatrix} 1.7 & -1.2 & 4.2 \\ 3.0 & 4.5 & -7.2 \end{bmatrix}.$$

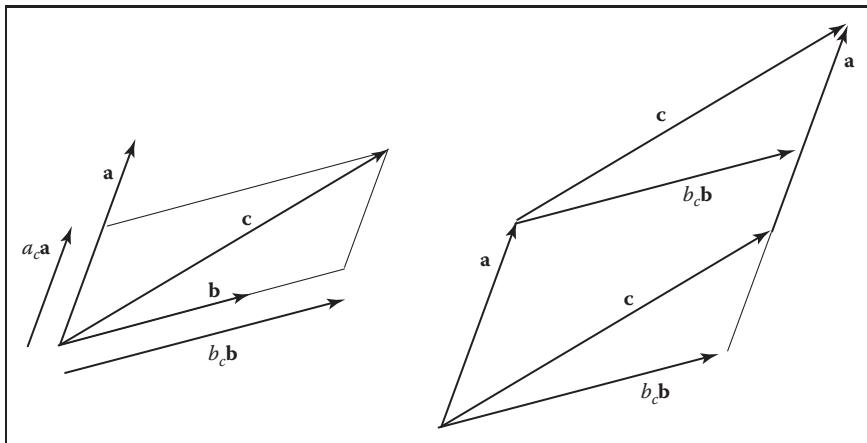


图5.6。在左边，向量 \mathbf{c} 可以使用两个基向量表示为 $a\mathbf{a}+b\mathbf{b}$ 。在右边，我们看到由 \mathbf{a} 和 \mathbf{c} 形成的平行四边形是由 $b\mathbf{b}$ 和 \mathbf{a} 形成的平行四边形的剪切版本。

从图5.6中我们可以看出

$$|(b_c \mathbf{b})\mathbf{a}| = |\mathbf{c}\mathbf{a}|,$$

因为这些平行四边形只是彼此的剪切版本。求解 b_c 产量

$$b_c = \frac{|\mathbf{c}\mathbf{a}|}{|\mathbf{b}\mathbf{a}|}.$$

类似的论点产生

$$a_c = \frac{|\mathbf{b}\mathbf{c}|}{|\mathbf{b}\mathbf{a}|}.$$

这是克拉默规则的二维版本，我们将在
Section 5.3.2. □

5.2 Matrices

矩阵是遵循某些算术规则的数字元素数组。具有两行和三列的矩阵的示例是

$$\begin{bmatrix} 1.7 & -1.2 & 4.2 \\ 3.0 & 4.5 & -7.2 \end{bmatrix}.$$

Matrices are frequently used in computer graphics for a variety of purposes including representation of spatial transforms. For our discussion, we assume the elements of a matrix are all real numbers. This chapter describes both the mechanics of matrix arithmetic and the *determinant* of “square” matrices, i.e., matrices with the same number of rows as columns.

5.2.1 Matrix Arithmetic

A matrix times a constant results in a matrix where each element has been multiplied by that constant, e.g.,

$$2 \begin{bmatrix} 1 & -4 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -8 \\ 6 & 4 \end{bmatrix}.$$

Matrices also add element by element, e.g.,

$$\begin{bmatrix} 1 & -4 \\ 3 & 2 \end{bmatrix} + \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 5 & 4 \end{bmatrix}.$$

For matrix multiplication, we “multiply” rows of the first matrix with columns of the second matrix:

$$\begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ \boxed{a_{i1}} & \dots & a_{im} \\ \vdots & & \vdots \\ a_{r1} & \dots & a_{rm} \end{bmatrix} \begin{bmatrix} b_{11} & \dots & \boxed{b_{1j}} & \dots & b_{1c} \\ \vdots & & \vdots & & \vdots \\ b_{m1} & \dots & b_{mj} & \dots & b_{mc} \\ \vdots & & \vdots & & \vdots \\ \end{bmatrix} = \begin{bmatrix} p_{11} & \dots & p_{1j} & \dots & p_{1c} \\ \vdots & & \vdots & & \vdots \\ p_{i1} & \dots & \boxed{p_{ij}} & \dots & p_{ic} \\ \vdots & & \vdots & & \vdots \\ p_{r1} & \dots & p_{rj} & \dots & p_{rc} \end{bmatrix}.$$

So the element p_{ij} of the resulting product is

$$p_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{im}b_{mj}. \quad (5.2)$$

Taking a product of two matrices is only possible if the number of columns of the left matrix is the same as the number of rows of the right matrix. For example,

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 6 & 7 & 8 & 9 \\ 0 & 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 12 & 17 & 22 & 27 \\ 24 & 33 & 42 & 51 \end{bmatrix}.$$

Matrix multiplication is *not* commutative in most instances:

$$\mathbf{AB} \neq \mathbf{BA}. \quad (5.3)$$

矩阵经常用于计算机图形学中，用于各种目的，包括空间变换的表示。对于我们的讨论，我们假设矩阵的元素都是实数。本章描述了矩阵算术的机制和“平方”矩阵的行列式，即行数与列数相同的矩阵。

5.2.1 矩阵算术

一个矩阵乘一个常数得到一个矩阵，其中每个元素都乘以该常数，例如。

$$2 \begin{bmatrix} 1 & -4 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -8 \\ 6 & 4 \end{bmatrix}.$$

矩阵也会逐元素添加元素，例如

$$\begin{bmatrix} 1 & -4 \\ 3 & 2 \end{bmatrix} + \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 5 & 4 \end{bmatrix}.$$

对于矩阵乘法，我们将第一个矩阵的行与第二个矩阵的列“相乘”：

$$\begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ \boxed{a_{i1}} & \dots & a_{im} \\ \vdots & & \vdots \\ a_{r1} & \dots & a_{rm} \end{bmatrix} \begin{bmatrix} b_{11} & \dots & \boxed{b_{1j}} & \dots & b_{1c} \\ \vdots & & \vdots & & \vdots \\ b_{m1} & \dots & b_{mj} & \dots & b_{mc} \\ \vdots & & \vdots & & \vdots \\ \end{bmatrix} = \begin{bmatrix} p_{11} & \dots & p_{1j} & \dots & p_{1c} \\ \vdots & & \vdots & & \vdots \\ p_{i1} & \dots & \boxed{p_{ij}} & \dots & p_{ic} \\ \vdots & & \vdots & & \vdots \\ p_{r1} & \dots & p_{rj} & \dots & p_{rc} \end{bmatrix}.$$

所以所得产物的元素 p_{ij} 为

$$p_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\dots+a_{im}b_{mj}. \quad (5.2)$$

只有当左矩阵的列数与右矩阵的行数相同时，才可能取两个矩阵的乘积。例如

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 6 & 7 & 8 & 9 \\ 0 & 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 12 & 17 & 22 & 27 \\ 24 & 33 & 42 & 51 \end{bmatrix}.$$

矩阵乘法在大多数情况下不是交换的：

$$\mathbf{AB}=\mathbf{BA}. \quad (5.3)$$



Also, if $\mathbf{AB} = \mathbf{AC}$, it does not necessarily follow that $\mathbf{B} = \mathbf{C}$. Fortunately, matrix multiplication is associative and distributive:

$$\begin{aligned}(\mathbf{AB})\mathbf{C} &= \mathbf{A}(\mathbf{BC}), \\ \mathbf{A}(\mathbf{B} + \mathbf{C}) &= \mathbf{AB} + \mathbf{AC}, \\ (\mathbf{A} + \mathbf{B})\mathbf{C} &= \mathbf{AC} + \mathbf{BC}.\end{aligned}$$

5.2.2 Operations on Matrices

We would like a matrix analog of the inverse of a real number. We know the inverse of a real number x is $1/x$ and that the product of x and its inverse is 1. We need a matrix \mathbf{I} that we can think of as a “matrix one.” This exists only for square matrices and is known as the *identity matrix*; it consists of ones down the *diagonal* and zeroes elsewhere. For example, the four by four identity matrix is

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The *inverse matrix* \mathbf{A}^{-1} of a matrix \mathbf{A} is the matrix that ensures $\mathbf{AA}^{-1} = \mathbf{I}$. For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1} = \begin{bmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{bmatrix} \quad \text{because} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Note that the inverse of \mathbf{A}^{-1} is \mathbf{A} . So $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. The inverse of a product of two matrices is the product of the inverses, but with the order reversed:

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}. \quad (5.4)$$

We will return to the question of computing inverses later in the chapter.

The *transpose* \mathbf{A}^T of a matrix \mathbf{A} has the same numbers but the rows are switched with the columns. If we label the entries of \mathbf{A}^T as a'_{ij} then

$$a_{ij} = a'_{ji}.$$

For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}.$$



此外, 如果 $\mathbf{AB}=\mathbf{AC}$, 则不一定遵循 $\mathbf{B}=\mathbf{C}$ 。幸运的是, 矩阵乘法是关联和分布的:

$$\begin{aligned}(\mathbf{AB})\mathbf{C} &= \mathbf{A}(\mathbf{BC}), \\ \mathbf{A}(\mathbf{B} + \mathbf{C}) &= \mathbf{AB} + \mathbf{AC}, \\ (\mathbf{A} + \mathbf{B})\mathbf{C} &= \mathbf{AC} + \mathbf{BC}.\end{aligned}$$

5.2.2 矩阵运算

我们想要一个实数倒数的矩阵模拟。我们知道实数 x 的倒数是 $1/x$, x 与其倒数的乘积是1。我们需要一个矩阵 \mathbf{I} , 我们可以认为是一个“矩阵一。”这存在于方阵, 被称为单位矩阵; 它由对角线下的一个和其他地方的零组成。例如, 四乘四单位矩阵是

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

矩阵 \mathbf{A} 的逆矩阵 \mathbf{A}^{-1} 是保证 $\mathbf{AA}^{-1}=\mathbf{I}$ 的矩阵。

例如

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1} = \begin{bmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{bmatrix} \quad \text{because} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

请注意, \mathbf{A}^{-1} 的倒数是 \mathbf{A} 。所以 $\mathbf{AA}^{-1}=\mathbf{A}^{-1}\mathbf{A}=\mathbf{I}$ 。两个矩阵的乘积的逆是逆的乘积, 但顺序相反:

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}. \quad (5.4)$$

矩阵 \mathbf{A} 的转置 \mathbf{A}^T 具有相同的数字, 但行与列切换。如果我们将 \mathbf{A}^T 的条目标记为 a'_{ij} 然后

$$a_{ij} = a'_{ji}.$$

例如

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}.$$

The transpose of a product of two matrices obeys a rule similar to Equation (5.4):

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T.$$

The determinant of a square matrix is simply the determinant of the columns of the matrix, considered as a set of vectors. The determinant has several nice relationships to the matrix operations just discussed, which we list here for reference:

$$|\mathbf{AB}| = |\mathbf{A}| |\mathbf{B}|, \quad (5.5)$$

$$|\mathbf{A}^{-1}| = \frac{1}{|\mathbf{A}|}, \quad (5.6)$$

$$|\mathbf{A}^T| = |\mathbf{A}|. \quad (5.7)$$

5.2.3 Vector Operations in Matrix Form

In graphics, we use a square matrix to transform a vector represented as a matrix. For example, if you have a 2D vector $\mathbf{a} = (x_a, y_a)$ and want to rotate it by 90 degrees about the origin to form vector $\mathbf{a}' = (-y_a, x_a)$, you can use a product of a 2×2 matrix and a 2×1 matrix, called a *column vector*. The operation in matrix form is

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_a \\ y_a \end{bmatrix} = \begin{bmatrix} -y_a \\ x_a \end{bmatrix}.$$

We can get the same result by using the transpose of this matrix and multiplying on the left ("premultiplying") with a row vector:

$$\begin{bmatrix} x_a & y_a \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -y_a & x_a \end{bmatrix}.$$

These days, postmultiplication using column vectors is fairly standard, but in many older books and systems you will run across row vectors and premultiplication. The only difference is that the transform matrix must be replaced with its transpose.

We also can use matrix formalism to encode operations on just vectors. If we consider the result of the dot product as a 1×1 matrix, it can be written

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}.$$

For example, if we take two 3D vectors we get

$$\begin{bmatrix} x_a & y_a & z_a \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = [x_a x_b + y_a y_b + z_a z_b].$$

两个矩阵的乘积的转置服从类似于公式 (5.4) 的规则:

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T.$$

方阵的行列式只是矩阵列的行列式，被视为一组向量。行列式与刚才讨论的矩阵运算有几个很好的关系，我们在此列出以供参考:

$$|\mathbf{AB}| = |\mathbf{A}| |\mathbf{B}|, \quad (5.5)$$

$$|\mathbf{A}^{-1}| = \frac{1}{|\mathbf{A}|}, \quad (5.6)$$

$$|\mathbf{A}^T| = |\mathbf{A}|. \quad (5.7)$$

5.2.3 矩阵形式的向量运算

在图形中，我们使用方阵来变换表示为矩阵的矢量。例如，如果您有一个2D向量 $\mathbf{a} = (x_a, y_a)$ 并希望将其绕原点旋转90度以形成向量 $\mathbf{a}' = (-y_a, x_a)$ ，则可以使用 2×2 矩阵和 2×1 矩阵的乘积，称为列向量。矩阵形式的运算为

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_a \\ y_a \end{bmatrix} = \begin{bmatrix} -y_a \\ x_a \end{bmatrix}.$$

我们可以通过使用此矩阵的转置并在左侧乘以行向量（“预乘”）来获得相同的结果:

$$\begin{bmatrix} x_a & y_a \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -y_a & x_a \end{bmatrix}.$$

这些天来，使用列向量的postmultiplication是相当标准的，但在许多较旧的书籍和系统中，您将跨行向量和premultiplication运行。唯一的区别是变换矩阵必须用它的转置替换。

我们还可以使用矩阵形式主义对向量进行编码操作。如果我们将点积的结果视为 1×1 矩阵，则可以写

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}.$$

例如，如果我们取两个3d向量，我们得到

$$\begin{bmatrix} x_a & y_a & z_a \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = [x_a x_b + y_a y_b + z_a z_b].$$

A related vector product is the *outer product* between two vectors, which can be expressed as a matrix multiplication with a column vector on the left and a row vector on the right: \mathbf{ab}^T . The result is a matrix consisting of products of all pairs of an entry of \mathbf{a} with an entry of \mathbf{b} . For 3D vectors, we have

$$\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \begin{bmatrix} x_b & y_b & z_b \end{bmatrix} = \begin{bmatrix} x_a x_b & x_a y_b & x_a z_b \\ y_a x_b & y_a y_b & y_a z_b \\ z_a x_b & z_a y_b & z_a z_b \end{bmatrix}.$$

It is often useful to think of matrix multiplication in terms of vector operations. To illustrate using the three-dimensional case, we can think of a 3×3 matrix as a collection of three 3D vectors in two ways: either it is made up of three column vectors side-by-side, or it is made up of three row vectors stacked up. For instance, the result of a matrix-vector multiplication $\mathbf{y} = \mathbf{Ax}$ can be interpreted as a vector whose entries are the dot products of \mathbf{x} with the rows of \mathbf{A} . Naming these row vectors \mathbf{r}_i , we have

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_1- \\ -\mathbf{r}_2- \\ -\mathbf{r}_3- \end{bmatrix} \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix};$$

$$y_i = \mathbf{r}_i \cdot \mathbf{x}.$$

Alternatively, we can think of the same product as a sum of the three columns \mathbf{c}_i of \mathbf{A} , weighted by the entries of \mathbf{x} :

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix};$$

$$\mathbf{y} = x_1 \mathbf{c}_1 + x_2 \mathbf{c}_2 + x_3 \mathbf{c}_3.$$

Using the same ideas, one can understand a matrix-matrix product \mathbf{AB} as an array containing the pairwise dot products of all rows of \mathbf{A} with all columns of \mathbf{B} (cf. (5.2)); as a collection of products of the matrix \mathbf{A} with all the column vectors of \mathbf{B} , arranged left to right; as a collection of products of all the row vectors of \mathbf{A} with the matrix \mathbf{B} , stacked top to bottom; or as the sum of the pairwise outer products of all columns of \mathbf{A} with all rows of \mathbf{B} . (See Exercise 8.)

These interpretations of matrix multiplication can often lead to valuable geometric interpretations of operations that may otherwise seem very abstract.

5.2.4 Special Types of Matrices

The identity matrix is an example of a *diagonal matrix*, where all nonzero elements occur along the diagonal. The diagonal consists of those elements whose column index equals the row index counting from the upper left.

一个相关的向量乘积是两个向量之间的外积，它可以表示为一个矩阵乘法，左边是一个列向量，右边是一个行向量： \mathbf{ab}^T 。结果是一个矩阵，由 a 条目与 b 条目的所有对的乘积组成。对于3D矢量，我们有

$$\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} \begin{bmatrix} x_b & y_b & z_b \end{bmatrix} = \begin{bmatrix} x_a x_b & x_a y_b & x_a z_b \\ y_a x_b & y_a y_b & y_a z_b \\ z_a x_b & z_a y_b & z_a z_b \end{bmatrix}.$$

在向量运算方面考虑矩阵乘法通常是有用的。

为了使用三维情况进行说明，我们可以通过两种方式将 3×3 矩阵视为三个3D向量的集合：要么由三个列向量并排组成，要么由三个行向量堆叠而成。例如，矩阵向量乘法 $\mathbf{y} = \mathbf{Ax}$ 的结果可以解释为一个向量，其条目是 \mathbf{x} 与 \mathbf{a} 行的点积。命名这些行向量 \mathbf{r}_i ，我们有

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} -\mathbf{r}_1- \\ -\mathbf{r}_2- \\ -\mathbf{r}_3- \end{bmatrix} \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix};$$

$$y_i = \mathbf{r}_i \cdot \mathbf{x}.$$

或者，我们可以将相同的产品视为 \mathbf{A} 的三列 \mathbf{c}_i 的总和，由 \mathbf{x} 的条目加权：

$$\begin{bmatrix} | \\ \mathbf{y} \\ | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix};$$

$$\mathbf{y} = x_1 \mathbf{c}_1 + x_2 \mathbf{c}_2 + x_3 \mathbf{c}_3.$$

使用相同的思想，人们可以将矩阵-矩阵乘积 \mathbf{AB} 理解为包含 \mathbf{A} 的所有行与 \mathbf{B} 的所有列的成对点积的数组 (cf.(5.2));作为矩阵 \mathbf{A} 与 \mathbf{b} 的所有列向量的乘积的集合，从左到右排列；作为 \mathbf{A} 与矩阵 \mathbf{B} 的所有行向量的乘积的集合，从上到下堆叠；或作为 \mathbf{A} 的所有列与 \mathbf{b} 的 (见练习8。)

矩阵乘法的这些解释通常会导致对操作的有价值的地理度量解释，否则这些解释可能看起来非常抽象。

5.2.4 特殊类型的矩阵

单位矩阵是对角矩阵的一个例子，其中所有非零元素都沿着对角线出现。对角线由列索引等于从左上角计数的行索引的那些元素组成。

The identity matrix also has the property that it is the same as its transpose. Such matrices are called *symmetric*.

The idea of an orthogonal matrix corresponds to the idea of an orthonormal basis, not just a set of orthogonal vectors—an unfortunate glitch in terminology.

The identity matrix is also an *orthogonal* matrix, because each of its columns considered as a vector has length 1 and the columns are orthogonal to one another. The same is true of the rows (see Exercise 2). The determinant of any orthogonal matrix is either +1 or -1.

A very useful property of orthogonal matrices is that they are nearly their own inverses. Multiplying an orthogonal matrix by its transpose results in the identity,

$$\mathbf{R}^T \mathbf{R} = I = \mathbf{R} \mathbf{R}^T \quad \text{for orthogonal } \mathbf{R}.$$

This is easy to see because the entries of $\mathbf{R}^T \mathbf{R}$ are dot products between the columns of \mathbf{R} . Off-diagonal entries are dot products between orthogonal vectors, and the diagonal entries are dot products of the (unit-length) columns with themselves.

Example. The matrix

$$\begin{bmatrix} 8 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

is diagonal, and therefore symmetric, but not orthogonal (the columns are orthogonal but they are not unit length).

The matrix

$$\begin{bmatrix} 1 & 1 & 2 \\ 1 & 9 & 7 \\ 2 & 7 & 1 \end{bmatrix}$$

is symmetric, but not diagonal or orthogonal.

The matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

is orthogonal, but neither diagonal nor symmetric.

5.3 Computing with Matrices and Determinants

Recall from Section 5.1 that the determinant takes n n -dimensional vectors and combines them to get a signed n -dimensional volume of the n -dimensional parallelepiped defined by the vectors. For example, the determinant in 2D is the area

单位矩阵还具有与其转置相同的属性。这样的矩阵被称为对称矩阵。

正交矩阵的想法对应于正交基的想法，而不仅仅是一组正交向量—术语中的一个不幸的小故障。
。

单位矩阵也是正交矩阵，因为它的每一列被视为向量的长度为1，列相互正交。行也是如此（见练习2）。任何正交矩阵的行列式是+1或-1。

正交矩阵的一个非常有用的属性是它们几乎是它们自己的逆。将正交矩阵乘以其转置结果为同一性

$$\mathbf{R}^T \mathbf{R} = I = \mathbf{R} \mathbf{R}^T \quad \text{为正交R。}$$

这很容易看到，因为 $\mathbf{R}^T \mathbf{R}$ 的条目是 \mathbf{R} 列之间的点积。非对角线条目是正交向量之间的点积，并且对角线条目是与自身的（单位长度）列的点积。

例子。矩阵

$$\begin{bmatrix} 8 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

是对角线的，因此是对称的，但不是正交的（列是正交的，但它们不是单位长度）。

矩阵

$$\begin{bmatrix} 1 & 1 & 2 \\ 1 & 9 & 7 \\ 2 & 7 & 1 \end{bmatrix}$$

是对称的，但不是对角线或正交的。

矩阵

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

是正交的，但既不是对角线也不是对称的。

of the parallelogram formed by the vectors. We can use matrices to handle the mechanics of computing determinants.

If we have 2D vectors \mathbf{r} and \mathbf{s} , we denote the determinant $|\mathbf{rs}|$; this value is the signed area of the parallelogram formed by the vectors. Suppose we have two 2D vectors with Cartesian coordinates (a, b) and (A, B) (Figure 5.7). The determinant can be written in terms of column vectors or as a shorthand:

$$\begin{vmatrix} a & A \\ b & B \end{vmatrix} \equiv \begin{vmatrix} a & A \\ b & B \end{vmatrix} = aB - Ab. \quad (5.8)$$

Note that the determinant of a matrix is the same as the determinant of its transpose:

$$\begin{vmatrix} a & A \\ b & B \end{vmatrix} = \begin{vmatrix} a & b \\ A & B \end{vmatrix} = aB - Ab.$$

This means that for any parallelogram in 2D there is a “sibling” parallelogram that has the same area but a different shape (Figure 5.8). For example, the parallelogram defined by vectors $(3, 1)$ and $(2, 4)$ has area 10, as does the parallelogram defined by vectors $(3, 2)$ and $(1, 4)$.

Example. The geometric meaning of the 3D determinant is helpful in seeing why certain formulas make sense. For example, the equation of the plane through the points (x_i, y_i, z_i) for $i = 0, 1, 2$ is

$$\begin{vmatrix} x - x_0 & x - x_1 & x - x_2 \\ y - y_0 & y - y_1 & y - y_2 \\ z - z_0 & z - z_1 & z - z_2 \end{vmatrix} = 0.$$

Each column is a vector from point (x_i, y_i, z_i) to point (x, y, z) . The volume of the parallelepiped with those vectors as sides is zero only if (x, y, z) is coplanar with the three other points. Almost all equations involving determinants have similarly simple underlying geometry.

As we saw earlier, we can compute determinants by a brute force expansion where most terms are zero, and there is a great deal of bookkeeping on plus and minus signs. The standard way to manage the algebra of computing determinants is to use a form of *Laplace's expansion*. The key part of computing the determinant this way is to find *cofactors* of various matrix elements. Each element of a square matrix has a cofactor which is the determinant of a matrix with one fewer row and column possibly multiplied by minus one. The smaller matrix is obtained by eliminating the row and column that the element in question is in. For example, for a 10×10 matrix, the cofactor of a_{82} is the determinant of the 9×9 matrix with the 8th row and 2nd column eliminated. The sign of a cofactor is positive if

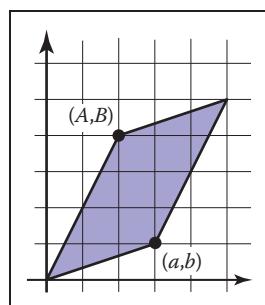


Figure 5.7. The 2D determinant in Equation 5.8 is the area of the parallelogram formed by the 2D vectors.

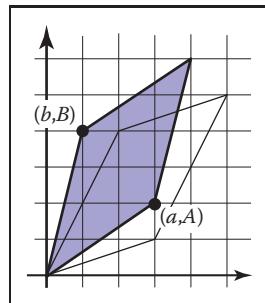


Figure 5.8. The sibling parallelogram has the same area as the parallelogram in Figure 5.7.

矢量形成的平行四边形。我们可以使用矩阵来处理计算行列式的机制。

如果我们有2D向量 \mathbf{r} 和 \mathbf{s} , 我们表示行列式 $|\mathbf{rs}|$;这个值是由向量形成的平行四边形的有符号区域。假设我们有两个具有笛卡尔坐标 (a, b) 和 (A, B) 的2D向量 (图5.7)。行列式可以用列向量来写, 也可以用简写来写:

$$\begin{vmatrix} a & A \\ b & B \end{vmatrix} \equiv \begin{vmatrix} a & A \\ b & B \end{vmatrix} = aB - Ab. \quad (5.8)$$

请注意, 矩阵的行列式与其反式的行列式相同:

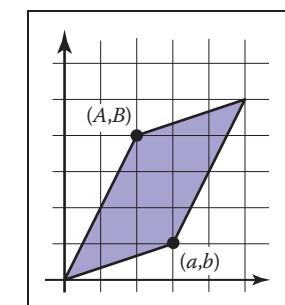
$$b \quad B = A \quad B = aB - Ab.$$

这意味着对于2D中的任何平行四边形, 都有一个具有相同面积但形状不同的“兄弟”平行四边形 (图5.8)。例如, 由向量 (3 1) 和 (2 4) 定义的平行四边形具有区域10, 由向量 (3 2) 和 (1 4) 定义的平行四边形也是如此。

例子。3D行列式的几何含义有助于了解为什么某些公式有意义。例如, 通过点 (x_i, y_i, z_i) 的平面的方程为 $i=0, 1, 2$

$$\begin{vmatrix} x - x_0 & x - x_1 & x - x_2 \\ y - y_0 & y - y_1 & y - y_2 \\ z - z_0 & z - z_1 & z - z_2 \end{vmatrix} = 0.$$

每列是从点 (x_i, y_i, z_i) 到点 (x, y, z) 的向量。只有当 (x, y, z) 与其他三个点共面时, 具有这些矢量作为边的平行六面体的体积才为零。几乎所有涉及行列式的方程都具有类似简单的基础几何。



公式5.8中的2Dde终止符是由2D矢量形成的平行四边形的面积。

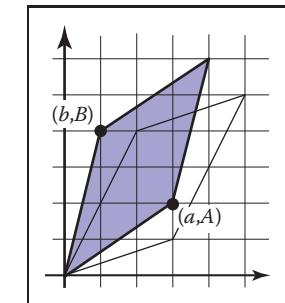


图5.8。同级平行四边形与图5.7中的平行四边形具有相同的面积。

正如我们前面所看到的, 我们可以通过蛮力扩展来计算行列式, 其中大多数项都是零, 并且有大量的正负符号簿记。管理计算行列式代数的标准方法是使用拉普拉斯扩展的一种形式。以这种方式计算行列式的关键部分是找到各种矩阵元素的辅因子。方阵的每个元素都有一个辅因子, 它是一个矩阵的行列式, 行和列可能乘以减一。较小的矩阵是通过消除所讨论的元素所在的行和列来获得的。例如, 对于 10×10 矩阵, 82的辅因子是消除了第8行和第2列的 9×9 矩阵的行列式。辅助因子的标志是积极的, 如果

the sum of the row and column indices is even and negative otherwise. This can be remembered by a checkerboard pattern:

$$\begin{bmatrix} + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

So, for a 4×4 matrix,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}.$$

The cofactors of the first row are

$$a_{11}^c = \begin{vmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{vmatrix}, \quad a_{12}^c = -\begin{vmatrix} a_{21} & a_{23} & a_{24} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{vmatrix},$$

$$a_{13}^c = \begin{vmatrix} a_{21} & a_{22} & a_{24} \\ a_{31} & a_{32} & a_{34} \\ a_{41} & a_{42} & a_{44} \end{vmatrix}, \quad a_{14}^c = -\begin{vmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{vmatrix}.$$

The determinant of a matrix is found by taking the sum of products of the elements of any row or column with their cofactors. For example, the determinant of the 4×4 matrix above taken about its second column is

$$|\mathbf{A}| = a_{12}a_{12}^c + a_{22}a_{22}^c + a_{32}a_{32}^c + a_{42}a_{42}^c.$$

We could do a similar expansion about any row or column and they would all yield the same result. Note the recursive nature of this expansion.

Example. A concrete example for the determinant of a particular 3×3 matrix by expanding the cofactors of the first row is

$$\begin{aligned} \begin{vmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{vmatrix} &= 0 \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} - 1 \begin{vmatrix} 3 & 5 \\ 6 & 8 \end{vmatrix} + 2 \begin{vmatrix} 3 & 4 \\ 6 & 7 \end{vmatrix} \\ &= 0(32 - 35) - 1(24 - 30) + 2(21 - 24) \\ &= 0. \end{aligned}$$

行和列索引的总和为偶数，否则为负值。这可以通过棋盘图案记住：

$$\begin{bmatrix} + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

所以，对于 4×4 矩阵

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}.$$

第一行的辅因子是

$$a_{11}^c = \begin{vmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{vmatrix}, \quad a_{12}^c = -\begin{vmatrix} a_{21} & a_{23} & a_{24} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{vmatrix},$$

$$a_{13}^c = \begin{vmatrix} a_{21} & a_{22} & a_{24} \\ a_{31} & a_{32} & a_{34} \\ a_{41} & a_{42} & a_{44} \end{vmatrix}, \quad a_{14}^c = -\begin{vmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{vmatrix}.$$

矩阵的行列式是通过取任何行或列的元素与其辅因子的乘积之和来找到的。例如，上面取的关于其第二列的 4×4 矩阵的行列式是

$$|\mathbf{A}| = a_{12}a_{12}^c + a_{22}a_{22}^c + a_{32}a_{32}^c + a_{42}a_{42}^c.$$

我们可以对任何行或列进行类似的扩展，它们都会产生相同的结果。请注意此扩展的递归性质。

例子。通过扩展第一行的辅因子，特定 3×3 矩阵的行列式的一个具体示例是

$$\begin{aligned} \begin{vmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{vmatrix} &= 0 \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} - 1 \begin{vmatrix} 3 & 5 \\ 6 & 8 \end{vmatrix} + 2 \begin{vmatrix} 3 & 4 \\ 6 & 7 \end{vmatrix} \\ &= 0(32 - 35) - 1(24 - 30) + 2(21 - 24) \\ &= 0. \end{aligned}$$

We can deduce that the volume of the parallelepiped formed by the vectors defined by the columns (or rows since the determinant of the transpose is the same) is zero. This is equivalent to saying that the columns (or rows) are not linearly independent. Note that the sum of the first and third rows is twice the second row, which implies linear dependence. □

5.3.1 Computing Inverses

Determinants give us a tool to compute the inverse of a matrix. It is a very inefficient method for large matrices, but often in graphics our matrices are small. A key to developing this method is that the determinant of a matrix with two identical rows is zero. This should be clear because the volume of the n -dimensional parallelepiped is zero if two of its sides are the same. Suppose we have a 4×4 \mathbf{A} and we wish to find its inverse \mathbf{A}^{-1} . The inverse is

$$\mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \begin{bmatrix} a_{11}^c & a_{21}^c & a_{31}^c & a_{41}^c \\ a_{12}^c & a_{22}^c & a_{32}^c & a_{42}^c \\ a_{13}^c & a_{23}^c & a_{33}^c & a_{43}^c \\ a_{14}^c & a_{24}^c & a_{34}^c & a_{44}^c \end{bmatrix}.$$

Note that this is just the transpose of the matrix where elements of \mathbf{A} are replaced by their respective cofactors multiplied by the leading constant (1 or -1). This matrix is called the *adjoint* of \mathbf{A} . The adjoint is the transpose of the *cofactor* matrix of \mathbf{A} . We can see why this is an inverse. Look at the product $\mathbf{A}\mathbf{A}^{-1}$ which we expect to be the identity. If we multiply the first row of \mathbf{A} by the first column of the adjoint matrix we need to get $|\mathbf{A}|$ (remember the leading constant above divides by $|\mathbf{A}|$):

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} a_{11}^c & \cdot & \cdot & \cdot \\ a_{12}^c & \cdot & \cdot & \cdot \\ a_{13}^c & \cdot & \cdot & \cdot \\ a_{14}^c & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} |\mathbf{A}| & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$

This is true because the elements in the first row of \mathbf{A} are multiplied exactly by their cofactors in the first column of the adjoint matrix which is exactly the determinant. The other values along the diagonal of the resulting matrix are $|\mathbf{A}|$ for analogous reasons. The zeros follow a similar logic:

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} a_{11}^c & \cdot & \cdot & \cdot \\ a_{12}^c & \cdot & \cdot & \cdot \\ a_{13}^c & \cdot & \cdot & \cdot \\ a_{14}^c & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$

我们可以推断出由列（或行，因为转置的行列式相同）定义的向量形成的平行六面体的体积为零。这相当于说列（或行）不是线性独立的。请注意，第一行和第三行的总和是第二行的两倍，这意味着线性依赖性。

5.3.1 Computing Inverses

行列式给了我们一个工具来计算矩阵的逆。对于大型矩阵来说，这是一种非常低效的方法，但在图形中，我们的矩阵通常很小。开发这种方法的一个关键是具有两个相同行的矩阵的行列式为零。这应该是清楚的，因为 n 维平行六面体的体积是零，如果它的两个侧面是相同的。假设我们有一个 4×4 \mathbf{A} ，我们希望找到它的逆 \mathbf{A}^{-1} 。相反的是

$$\mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \begin{bmatrix} a_{11}^c & a_{21}^c & a_{31}^c & a_{41}^c \\ a_{12}^c & a_{22}^c & a_{32}^c & a_{42}^c \\ a_{13}^c & a_{23}^c & a_{33}^c & a_{43}^c \\ a_{14}^c & a_{24}^c & a_{34}^c & a_{44}^c \end{bmatrix}.$$

请注意，这只是矩阵的转置，其中 a 的元素被其各自的辅因子乘以前导常数（1或-1）替换。这个矩阵被称为 a 的伴随。伴随是 a 的辅因子矩阵的转置。我们可以看到为什么这是相反的。看看产品 \mathbf{AA}^{-1}

我们期望的是身份。如果我们将 a 的第一行乘以伴随矩阵的第一列，我们需要得到 a （记住上面的前导常数除以 A ）：

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} a_{11}^c & \cdot & \cdot & \cdot \\ a_{12}^c & \cdot & \cdot & \cdot \\ a_{13}^c & \cdot & \cdot & \cdot \\ a_{14}^c & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} |\mathbf{A}| & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$

这是真的，因为 a 的第一行中的元素与它们的辅因子在邻接矩阵的第一列中完全相乘，这正是行列式。由于类似的原因，沿所得矩阵对角线的其他值是 A 。零遵循类似的逻辑：

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} a_{11}^c & \cdot & \cdot & \cdot \\ a_{12}^c & \cdot & \cdot & \cdot \\ a_{13}^c & \cdot & \cdot & \cdot \\ a_{14}^c & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$

Note that this product is a determinant of *some* matrix:

$$a_{21}a_{11}^c + a_{22}a_{12}^c + a_{23}a_{13}^c + a_{24}a_{14}^c.$$

The matrix in fact is

$$\begin{bmatrix} a_{21} & a_{22} & a_{23} & a_{24} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}.$$

Because the first two rows are identical, the matrix is singular, and thus, its determinant is zero.

The argument above does not apply just to four by four matrices; using that size just simplifies typography. For any matrix, the inverse is the adjoint matrix divided by the determinant of the matrix being inverted. The adjoint is the transpose of the cofactor matrix, which is just the matrix whose elements have been replaced by their cofactors.

Example. The inverse of one particular three by three matrix whose determinant is 6 is

$$\begin{aligned} \begin{bmatrix} 1 & 1 & 2 \\ 1 & 3 & 4 \\ 0 & 2 & 5 \end{bmatrix}^{-1} &= \frac{1}{6} \begin{bmatrix} \begin{vmatrix} 3 & 4 \\ 2 & 5 \end{vmatrix} - \begin{vmatrix} 1 & 2 \\ 2 & 5 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \\ -\begin{vmatrix} 1 & 4 \\ 0 & 5 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 0 & 5 \end{vmatrix} - \begin{vmatrix} 1 & 2 \\ 1 & 4 \end{vmatrix} \\ \begin{vmatrix} 1 & 3 \\ 0 & 2 \end{vmatrix} & -\begin{vmatrix} 1 & 1 \\ 0 & 2 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 3 \end{vmatrix} \end{bmatrix} \\ &= \frac{1}{6} \begin{bmatrix} 7 & -1 & -2 \\ -5 & 5 & -2 \\ 2 & -2 & 2 \end{bmatrix}. \end{aligned}$$

You can check this yourself by multiplying the matrices and making sure you get the identity.

5.3.2 Linear Systems

We often encounter linear systems in graphics with “ n equations and n unknowns,” usually for $n = 2$ or $n = 3$. For example,

$$\begin{aligned} 3x + 7y + 2z &= 4, \\ 2x - 4y - 3z &= -1, \\ 5x + 2y + z &= 1. \end{aligned}$$

请注意，此产品是一些矩阵的行列式：

$$a_{21}a_{11}^c + a_{22}a_{12}^c + a_{23}a_{13}^c + a_{24}a_{14}^c.$$

矩阵实际上是

$$\begin{bmatrix} a_{21} & a_{22} & a_{23} & a_{24} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}.$$

因为前两行是相同的，所以矩阵是奇异的，因此，它的最小值是零。

上面的参数不仅适用于四乘四矩阵；使用该大小只是简化了排版。对于任何矩阵，逆是伴随矩阵除以被反转的矩阵的行列式。伴随的是辅因子矩阵的反式姿势，它只是其元素已被其辅因子替换的矩阵。

例子。行列式为6的一个特定的三乘三矩阵的倒数是

$$\begin{aligned} \begin{bmatrix} 1 & 1 & 2 \\ 1 & 3 & 4 \\ 0 & 2 & 5 \end{bmatrix}^{-1} &= \frac{1}{6} \begin{bmatrix} \begin{vmatrix} 3 & 4 \\ 2 & 5 \end{vmatrix} - \begin{vmatrix} 1 & 2 \\ 2 & 5 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \\ -\begin{vmatrix} 1 & 4 \\ 0 & 5 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 0 & 5 \end{vmatrix} - \begin{vmatrix} 1 & 2 \\ 1 & 4 \end{vmatrix} \\ \begin{vmatrix} 1 & 3 \\ 0 & 2 \end{vmatrix} & -\begin{vmatrix} 1 & 1 \\ 0 & 2 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 3 \end{vmatrix} \end{bmatrix} \\ &= \frac{1}{6} \begin{bmatrix} 7 & -1 & -2 \\ -5 & 5 & -2 \\ 2 & -2 & 2 \end{bmatrix}. \end{aligned}$$

您可以通过乘以矩阵并确保获得身份来检查自己。

5.3.2 线性系统

我们经常在图形中遇到具有“ n 个方程和 n 个未知数”的线性系统，通常为 $n=2$ 或 $n=3$ 。例如

$$\begin{aligned} 3x + 7y + 2z &= 4, \\ 2x - 4y - 3z &= -1, \\ 5x + 2y + z &= 1. \end{aligned}$$

Here x , y , and z are the “unknowns” for which we wish to solve. We can write this in matrix form:

$$\begin{bmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ 1 \end{bmatrix}.$$

A common shorthand for such systems is $\mathbf{Ax} = \mathbf{b}$ where it is assumed that \mathbf{A} is a square matrix with known constants, \mathbf{x} is an unknown column vector (with elements x , y , and z in our example), and \mathbf{b} is a column matrix of known constants.

There are many ways to solve such systems, and the appropriate method depends on the properties and dimensions of the matrix \mathbf{A} . Because in graphics we so frequently work with systems of size $n \leq 4$, we’ll discuss here a method appropriate for these systems, known as *Cramer’s rule*, which we saw earlier, from a 2D geometric viewpoint, in the example on page 90. Here, we show this algebraically. The solution to the above equation is

$$x = \frac{\begin{vmatrix} 4 & 7 & 2 \\ -1 & -4 & -3 \\ 1 & 2 & 1 \end{vmatrix}}{\begin{vmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}, \quad y = \frac{\begin{vmatrix} 3 & 7 & 4 \\ 2 & -4 & -1 \\ 5 & 2 & 1 \end{vmatrix}}{\begin{vmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}, \quad z = \frac{\begin{vmatrix} 3 & 7 & 4 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}{\begin{vmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}.$$

The rule here is to take a ratio of determinants, where the denominator is $|\mathbf{A}|$ and the numerator is the determinant of a matrix created by replacing a column of \mathbf{A} with the column vector \mathbf{b} . The column replaced corresponds to the position of the unknown in vector \mathbf{x} . For example, y is the second unknown and the second column is replaced. Note that if $|\mathbf{A}| = 0$, the division is undefined and there is no solution. This is just another version of the rule that if \mathbf{A} is singular (zero determinant) then there is no unique solution to the equations.

5.4 Eigenvalues and Matrix Diagonalization

Square matrices have *eigenvalues* and *eigenvectors* associated with them. The eigenvectors are those *nonzero* vectors whose directions do not change when multiplied by the matrix. For example, suppose for a matrix \mathbf{A} and vector \mathbf{a} , we have

$$\mathbf{A}\mathbf{a} = \lambda\mathbf{a}. \quad (5.9)$$

This means we have stretched or compressed \mathbf{a} , but its direction has not changed. The scale factor λ is called the eigenvalue associated with eigenvector \mathbf{a} . Knowing

这里 x , y 和 z 是我们希望解决的“未知数”。我们可以用矩阵形式写这个：

$$\begin{bmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ 1 \end{bmatrix}.$$

此类系统的常见简写是 $\mathbf{Ax} = \mathbf{b}$, 其中假设 \mathbf{A} 是具有已知常数的方阵, \mathbf{x} 是未知列向量 (在我们的示例中具有元素 x , y 和 z) , \mathbf{b} 是已知常数的列矩阵。解决此类系统的方法很多, 适当的方法取决于矩阵 \mathbf{A} 的性质和维度。因为在图形中, 我们经常使用大小为 $n \leq 4$ 的系统, 我们将在这里讨论一个适合这些系统的方法, 称为 Cramer 规则, 我们前面从 2D 几何角度看到, 在第 9 页的示例中。在这里, 我们以代数方式展示这一点。上述方程的解是

$$x = \frac{\begin{vmatrix} 4 & 7 & 2 \\ -1 & -4 & -3 \\ 1 & 2 & 1 \end{vmatrix}}{\begin{vmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}, \quad y = \frac{\begin{vmatrix} 3 & 4 & 2 \\ 2 & -1 & -3 \\ 5 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}, \quad z = \frac{\begin{vmatrix} 3 & 7 & 4 \\ 2 & -4 & -1 \\ 5 & 2 & 1 \end{vmatrix}}{\begin{vmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{vmatrix}}.$$

这里的规则是取行列式的比率, 其中分母是 \mathbf{A} , 分子是通过用列向量 \mathbf{b} 替换 \mathbf{A} 的列而创建的矩阵的行列式。替换的列对应于向量 \mathbf{x} 中未知的位置。例如, y 是第二列, 第二列被替换。请注意, 如果 $\mathbf{A}=0$, 则除法未定义, 并且没有解决方案。这只是规则的另一个版本, 如果 \mathbf{A} 是奇异的 (零行列式) , 那么方程就没有唯一的解决方案。

5.4 特征值和矩阵对角化

方阵具有与之相关联的特征值和特征向量。特征向量是那些在乘以矩阵时方向不改变的非零向量。例如, 假设对于矩阵 \mathbf{A} 和向量 \mathbf{a} , 我们有

$$\mathbf{A}\mathbf{a} = \lambda\mathbf{a}. \quad (5.9)$$

这意味着我们已经拉伸或压缩了 \mathbf{a} , 但它的方向没有改变。比例因子 λ 被称为与特征向量 \mathbf{a} 相关联的特征值。知道

the eigenvalues and eigenvectors of matrices is helpful in a variety of practical applications. We will describe them to gain insight into geometric transformation matrices and as a step toward singular values and vectors described in the next section.

If we assume a matrix has at least one eigenvector, then we can do a standard manipulation to find it. First, we write both sides as the product of a square matrix with the vector \mathbf{a} :

$$\mathbf{A}\mathbf{a} = \lambda\mathbf{I}\mathbf{a}, \quad (5.10)$$

where \mathbf{I} is an identity matrix. This can be rewritten

$$\mathbf{A}\mathbf{a} - \lambda\mathbf{I}\mathbf{a} = 0. \quad (5.11)$$

Because matrix multiplication is distributive, we can group the matrices:

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{a} = 0. \quad (5.12)$$

This equation can only be true if the matrix $(\mathbf{A} - \lambda\mathbf{I})$ is singular, and thus its determinant is zero. The elements in this matrix are the numbers in \mathbf{A} except along the diagonal. For example, for a 2×2 matrix the eigenvalues obey

$$\begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{vmatrix} = \lambda^2 - (a_{11} + a_{22})\lambda + (a_{11}a_{22} - a_{12}a_{21}) = 0. \quad (5.13)$$

Because this is a quadratic equation, we know there are exactly two solutions for λ . These solutions may or may not be unique or real. A similar manipulation for an $n \times n$ matrix will yield an n th-degree polynomial in λ . Because it is not possible, in general, to find exact explicit solutions of polynomial equations of degree greater than four, we can only compute eigenvalues of matrices 4×4 or smaller by analytic methods. For larger matrices, numerical methods are the only option.

An important special case where eigenvalues and eigenvectors are particularly simple is symmetric matrices (where $\mathbf{A} = \mathbf{A}^T$). The eigenvalues of real symmetric matrices are always real numbers, and if they are also distinct, their eigenvectors are mutually orthogonal. Such matrices can be put into *diagonal form*:

$$\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T, \quad (5.14)$$

where \mathbf{Q} is an orthogonal matrix and \mathbf{D} is a diagonal matrix. The columns of \mathbf{Q} are the eigenvectors of \mathbf{A} and the diagonal elements of \mathbf{D} are the eigenvalues of \mathbf{A} . Putting \mathbf{A} in this form is also called the *eigenvalue decomposition*, because it decomposes \mathbf{A} into a product of simpler matrices that reveal its eigenvectors and eigenvalues.

Recall that an *orthogonal* matrix has *orthonormal* rows and *orthonormal* columns.

矩阵的特征值和特征向量在各种实际应用中都很有帮助。我们将描述它们以深入了解几何变换矩阵，并作为下一节中描述的奇异值和向量的一个步骤。

如果我们假设一个矩阵至少有一个特征向量，那么我们可以做一个标准的操作来找到它。首先，我们将双方写为与向量 \mathbf{a} 的方阵的乘积：

$$\mathbf{A}\mathbf{a} = \lambda\mathbf{I}\mathbf{a}, \quad (5.10)$$

其中 \mathbf{I} 是单位矩阵。这可以重写

$$\mathbf{A}\mathbf{a} - \lambda\mathbf{I}\mathbf{a} = 0. \quad (5.11)$$

因为矩阵乘法是分布的，所以我们可以对矩阵进行分组：

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{a} = 0. \quad (5.12)$$

只有当矩阵 $(\mathbf{A} - \lambda\mathbf{I})$ 是奇异的，因此它的行列式是零时，这个方程才能成立。该矩阵中的元素是沿着对角线的 \mathbf{A} 中的数字。例如，对于 2×2 矩阵，特征值服从

$$\begin{vmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{vmatrix} = \lambda^2 - (a_{11} + a_{22})\lambda + (a_{11}a_{22} - a_{12}a_{21}) = 0. \quad (5.13)$$

因为这是一个二次方程，我们知道 λ 正好有两个解。这些解决方案可能是也可能不是唯一的或真实的。对 $n \times n$ 矩阵进行类似的操作将产生 λ 中的 n 次多项式。由于一般情况下不可能找到大于4度的多项式方程的精确显式解，我们只能用解析方法计算矩阵 4×4 或更小的特征值。对于较大的矩阵，数值方法是唯一的选择。

特征值和特征向量非常简单的一个重要特例是对称矩阵（其中 $\mathbf{A}=\mathbf{A}^T$ ）。实对称矩阵的特征值总是实数，如果它们也是不同的，则它们的特征向量是相互正交的。这样的矩阵可以放入对角线形式：

$$\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^T, \quad (5.14)$$

其中 \mathbf{Q} 是正交矩阵， \mathbf{D} 是对角矩阵。 \mathbf{Q} 的列

是 \mathbf{A} 的特征向量， \mathbf{D} 的对角线元素是 \mathbf{A} 的特征值。将 \mathbf{A} 置于这种形式也称为特征值分解，因为它将 \mathbf{A} 分解为揭示其特征向量和特征值的更简单矩阵的乘积。



Example. Given the matrix

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix},$$

the eigenvalues of \mathbf{A} are the solutions to

$$\lambda^2 - 3\lambda + 1 = 0.$$

We approximate the exact values for compactness of notation:

$$\lambda = \frac{3 \pm \sqrt{5}}{2}, \approx \begin{bmatrix} 2.618 \\ 0.382 \end{bmatrix}.$$

Now we can find the associated eigenvector. The first is the nontrivial (not $x = y = 0$) solution to the homogeneous equation,

$$\begin{bmatrix} 2 - 2.618 & 1 \\ 1 & 1 - 2.618 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

This is approximately $(x, y) = (0.8507, 0.5257)$. Note that there are infinitely many solutions parallel to that 2D vector, and we just picked the one of unit length. Similarly the eigenvector associated with λ_2 is $(x, y) = (-0.5257, 0.8507)$. This means the diagonal form of \mathbf{A} is (within some precision due to our numeric approximation):

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 2.618 & 0 \\ 0 & 0.382 \end{bmatrix} \begin{bmatrix} 0.8507 & 0.5257 \\ -0.5257 & 0.8507 \end{bmatrix}.$$



例子。给定矩阵

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix},$$

\mathbf{a} 的特征值是解

$$\lambda^2 - 3\lambda + 1 = 0.$$

我们近似表示法紧凑性的的确切值:

$$\lambda = \frac{3 \pm \sqrt{5}}{2}, \approx \begin{bmatrix} 2.618 \\ 0.382 \end{bmatrix}.$$

现在我们可以找到相关的特征向量。第一个是齐次方程的非平凡（而不是 $x=y=0$ ）解， $\lambda_1 = 2.618$

这大约是 $(x, y) = (0.8507, 0.5257)$. 请注意，有无限多的解决方案平行于2D向量，我们只是选择了单位长度的一个。类似地，与 λ_2 相关的特征向量是 $(x, y) = (-0.5257, 0.8507)$. 这意味着 \mathbf{a} 的对角线形式是（由于我们的数字ap近似，在一定精度内）：

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 2.618 & 0 \\ 0 & 0.382 \end{bmatrix} \begin{bmatrix} 0.8507 & 0.5257 \\ -0.5257 & 0.8507 \end{bmatrix}.$$

我们将在下一章中重新审视这个矩阵的几何形状作为变换。

We will revisit the geometry of this matrix as a transform in the next chapter.

5.4.1 Singular Value Decomposition

We saw in the last section that any symmetric matrix can be diagonalized, or decomposed into a convenient product of orthogonal and diagonal matrices. However, most matrices we encounter in graphics are not symmetric, and the eigenvalue decomposition for nonsymmetric matrices is not nearly so convenient or illuminating, and in general involves complex-valued eigenvalues and eigenvectors even for real-valued inputs.

There is another generalization of the symmetric eigenvalue decomposition to nonsymmetric (and even non-square) matrices; it is the *singular value decomposition* (SVD). The main difference between the eigenvalue decomposition of a symmetric matrix and the SVD of a nonsymmetric matrix is that the orthogonal matrices on the left and right sides are not required to be the same in the SVD:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T.$$

We would recommend learning in this order: symmetric eigenvalues/vectors, singular values/vectors, and then nonsymmetric eigenvalues, which are much trickier.

还有另一种将对称特征值分解推广到非对称（甚至非正方形）矩阵；它是奇异值分解（SVD）。对称矩阵的特征值分解与非对称矩阵的SVD的主要区别在于，在SVD中不要求左右两侧的正交矩阵相同：

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T.$$

会建议按以下顺序学习：
sym度量特征值向量，单数

and then nonsymmetric eigenvalues, which are 要棘手得多。

Here \mathbf{U} and \mathbf{V} are two, potentially different, orthogonal matrices, whose columns are known as the left and right *singular vectors* of \mathbf{A} , and \mathbf{S} is a diagonal matrix whose entries are known as the *singular values* of \mathbf{A} . When \mathbf{A} is symmetric and has all nonnegative eigenvalues, the SVD and the eigenvalue decomposition are the same.

There is another relationship between singular values and eigenvalues that can be used to compute the SVD (though this is not the way an industrial-strength SVD implementation works). First we define $\mathbf{M} = \mathbf{AA}^T$. We assume that we can perform a SVD on \mathbf{M} :

$$\mathbf{M} = \mathbf{AA}^T = (\mathbf{USV}^T)(\mathbf{USV}^T)^T = \mathbf{US}(\mathbf{V}^T\mathbf{V})\mathbf{SU}^T = \mathbf{US}^2\mathbf{U}^T.$$

The substitution is based on the fact that $(\mathbf{BC})^T = \mathbf{C}^T\mathbf{B}^T$, that the transpose of an orthogonal matrix is its inverse, and the transpose of a diagonal matrix is the matrix itself. The beauty of this new form is that \mathbf{M} is symmetric and $\mathbf{US}^2\mathbf{U}^T$ is its eigenvalue decomposition, where \mathbf{S}^2 contains the (all nonnegative) eigenvalues. Thus, we find that the singular values of a matrix are the square roots of the eigenvalues of the product of the matrix with its transpose, and the left singular vectors are the eigenvectors of that product. A similar argument allows \mathbf{V} , the matrix of right singular vectors, to be computed from $\mathbf{A}^T\mathbf{A}$.

Example. We now make this concrete with an example:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}; \quad \mathbf{M} = \mathbf{AA}^T = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

We saw the eigenvalue decomposition for this matrix in the previous section. We observe immediately

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} \sqrt{2.618} & 0 \\ 0 & \sqrt{0.382} \end{bmatrix} \mathbf{V}^T.$$

We can solve for \mathbf{V} algebraically:

$$\mathbf{V} = (\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A})^T.$$

The inverse of \mathbf{S} is a diagonal matrix with the reciprocals of the diagonal elements of \mathbf{S} . This yields

$$\begin{aligned} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} &= \mathbf{U} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \mathbf{V}^T \\ &= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 1.618 & 0 \\ 0 & 0.618 \end{bmatrix} \begin{bmatrix} 0.5257 & 0.8507 \\ -0.8507 & 0.5257 \end{bmatrix}. \end{aligned}$$

这里 \mathbf{U} 和 \mathbf{V} 是两个可能不同的正交矩阵，其列被称为 \mathbf{A} 的左右奇异向量， \mathbf{S} 是对角矩阵，其条目被称为 \mathbf{a} 的奇异值。当 \mathbf{a} 是对称的并且具有所有非负特征值时，SVD 和 特征值分解是相同的。

奇异值和特征值之间还有另一种关系可用于计算 SVD（尽管这不是工业强度 SVD 实现的工作方式）。首先我们定义 $\mathbf{M} = \mathbf{A}\mathbf{A}^T$ 。我们假设我们可以在 \mathbf{M} 上执行 SVD：

$$\mathbf{M} = \mathbf{A}\mathbf{A}^T = (\mathbf{USV}^T)(\mathbf{USV}^T)^T = \mathbf{US}(\mathbf{V}^T\mathbf{V})\mathbf{SU}^T = \mathbf{US}^2\mathbf{U}^T.$$

替换是基于这样一个事实： $(\mathbf{BC})^T = \mathbf{C}^T\mathbf{B}^T$ ，即正交矩阵的转置是其逆，而对角矩阵的转置是矩阵本身。这种新形式的美妙之处在于 \mathbf{M} 是对称的， $\mathbf{US}^2\mathbf{U}^T$ 是其特征值分解，其中 \mathbf{S}^2 包含（所有非负）特征值。因此，我们发现矩阵的奇异值是矩阵与其转置的乘积的特征值的平方根，左奇异向量是该乘积的特征向量。类似的参数允许从 \mathbf{T} 计算右奇异向量矩阵 \mathbf{V} 。

例子。我们现在用一个例子来具体说明这一点：

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}; \quad \mathbf{M} = \mathbf{AA}^T = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

我们在前一节中看到了这个矩阵的特征值分解。我们立即观察

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} \sqrt{2.618} & 0 \\ 0 & \sqrt{0.382} \end{bmatrix} \mathbf{V}^T.$$

我们可以代数地求解 \mathbf{V} ：

$$\mathbf{V} = (\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A})^T.$$

\mathbf{S} 的逆是具有 s 的对角线元素的倒数的对角矩阵。这会产生

$$\begin{aligned} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} &= \mathbf{U} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \mathbf{V}^T \\ &= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 1.618 & 0 \\ 0 & 0.618 \end{bmatrix} \begin{bmatrix} 0.5257 & 0.8507 \\ -0.8507 & 0.5257 \end{bmatrix}. \end{aligned}$$



This form used the standard symbol σ_i for the i th singular value. Again, for a symmetric matrix, the eigenvalues and the singular values are the same ($\sigma_i = \lambda_i$). We will examine the geometry of SVD further in Section 6.1.6.

Frequently Asked Questions

- Why is matrix multiplication defined the way it is rather than just element by element?

Element by element multiplication is a perfectly good way to define matrix multiplication, and indeed it has nice properties. However, in practice it is not very useful. Ultimately, most matrices are used to transform column vectors, e.g., in 3D you might have

$$\mathbf{b} = \mathbf{M}\mathbf{a},$$

where \mathbf{a} and \mathbf{b} are vectors and \mathbf{M} is a 3×3 matrix. To allow geometric operations such as rotation, combinations of all three elements of \mathbf{a} must go into each element of \mathbf{b} . That requires us to either go row-by-row or column-by-column through \mathbf{M} . That choice is made based on composition of matrices having the desired property,

$$\mathbf{M}_2(\mathbf{M}_1\mathbf{a}) = (\mathbf{M}_2\mathbf{M}_1)\mathbf{a}$$

which allows us to use one composite matrix $\mathbf{C} = \mathbf{M}_2\mathbf{M}_1$ to transform our vector. This is valuable when many vectors will be transformed by the same composite matrix. So, in summary, the somewhat weird rule for matrix multiplication is engineered to have these desired properties.

- Sometimes I hear that eigenvalues and singular values are the same thing and sometimes that one is the square of the other. Which is right?

If a real matrix \mathbf{A} is symmetric, and its eigenvalues are nonnegative, then its eigenvalues and singular values are the same. If \mathbf{A} is not symmetric, the matrix $\mathbf{M} = \mathbf{A}\mathbf{A}^T$ is symmetric and has nonnegative real eigenvalues. The singular values of \mathbf{A} and \mathbf{A}^T are the same and are the square roots of the singular/eigenvalues of \mathbf{M} . Thus, when the square root statement is made, it is because two different matrices (with a very particular relationship) are being talked about: $\mathbf{M} = \mathbf{A}\mathbf{A}^T$.



这种形式使用标准符号 σ_i 表示第*i*个奇异值。同样，对于对称矩阵，特征值和奇异值是相同的 ($\sigma_i = \lambda_i$)。我们将在第6.1.6节中进一步研究SVD的几何。

常见问题

- 为什么矩阵乘法是按它的方式定义的，而不仅仅是逐个元素？

逐元素乘法是定义矩阵乘法的一个很好的方法，实际上它具有很好的属性。然而，在实践中它不是很有用。最终，大多数矩阵用于转换列向量，例如，在3D中您可能有

$$\mathbf{b} = \mathbf{M}\mathbf{a},$$

其中 \mathbf{a} 和 \mathbf{b} 是向量， \mathbf{M} 是 3×3 矩阵。为了允许旋转等几何操作， \mathbf{a} 的所有三个元素的组合必须进入 \mathbf{b} 的每个元素。这要求我们通过 \mathbf{M} 逐行或逐列进行。这种选择是基于具有所需属性的矩阵的组成

$$\mathbf{M}_2(\mathbf{M}_1\mathbf{a}) = (\mathbf{M}_2\mathbf{M}_1)\mathbf{a}$$

这允许我们使用一个复合矩阵 $\mathbf{C}=\mathbf{M}_2\mathbf{M}_1$ 来变换我们的向量。当许多向量将由相同的复合矩阵变换时，这是有价值的。因此，总之，矩阵乘法的有点奇怪的规则被设计为具有这些所需的属性。

- 有时我听到特征值和奇异值是同一件事，有时一个是另一个的平方。哪个是对的？

如果一个实矩阵 \mathbf{A} 是对称的，并且它的特征值是非负的，那么它的特征值和奇异值是相同的。如果 \mathbf{A} 不是对称的，则矩阵 $\mathbf{M}=\mathbf{A}\mathbf{A}^T$ 是对称的并且具有非负实特征值。 \mathbf{A} 和 \mathbf{A}^T 的奇异值相同，是 \mathbf{M} 的奇异特征值的平方根。因此，当平方根语句时，这是因为正在谈论两个不同的矩阵（具有非常特定的关系）： $\mathbf{M}=\mathbf{A}\mathbf{A}^T$ 。

Notes

The discussion of determinants as volumes is based on *A Vector Space Approach to Geometry* (Hausner, 1998). Hausner has an excellent discussion of vector analysis and the fundamentals of geometry as well. The geometric derivation of Cramer's rule in 2D is taken from *Practical Linear Algebra: A Geometry Toolbox* (Farin & Hansford, 2004). That book also has geometric interpretations of other linear algebra operations such as Gaussian elimination. The discussion of eigenvalues and singular values is based primarily on *Linear Algebra and Its Applications* (Strang, 1988). The example of SVD of the shear matrix is based on a discussion in *Computer Graphics and Geometric Modeling* (Salomon, 1999).

Exercises

1. Write an implicit equation for the 2D line through points (x_0, y_0) and (x_1, y_1) using a 2D determinant.
2. Show that if the columns of a matrix are orthonormal, then so are the rows.
3. Prove the properties of matrix determinants stated in Equations (5.5)–(5.7).
4. Show that the eigenvalues of a diagonal matrix are its diagonal elements.
5. Show that for a square matrix \mathbf{A} , $\mathbf{A}\mathbf{A}^T$ is a symmetric matrix.
6. Show that for three 3D vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$, the following identity holds: $|\mathbf{abc}| = (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$.
7. Explain why the volume of the tetrahedron with side vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$ (see Figure 5.2) is given by $|\mathbf{abc}|/6$.
8. Demonstrate the four interpretations of matrix-matrix multiplication by taking the following matrix-matrix multiplication code, rearranging the nested loops, and interpreting the resulting code in terms of matrix and vector operations.

```
function mat-mult(in a[m][p], in b[p][n], out c[m][n]) {
    // the array c is initialized to zero
    for i = 1 to m
        for j = 1 to n
            for k = 1 to p
                c[i][j] += a[i][k] * b[k][j]
}
```

Notes

将行列式作为体积的讨论基于几何的矢量空间方法 (Hausner, 1998)。豪斯纳对矢量分析和几何基础也有很好的讨论.克拉默规则在2D中的几何推导取自实用线性代数: 几何工具箱 (Farin & Hansford, 2004)。那本书也有其他线性代数运算的几何解释, 如高斯消元。特征值和奇异值的讨论主要基于线性代数及其应用 (Strang, 1988)。剪切矩阵的SVD示例基于计算机图形学和几何建模 (Salomon, 1999) 中的讨论。

Exercises

1. 使用2D行列式编写通过点 (x_0, y_0) 和 (x_1, y_1) 的2D线的隐式方程。
2. 显示如果矩阵的列是正交的, 那么行也是如此。
3. 证明方程 (5.5) (5.7) 中陈述的矩阵行列式的性质。
4. 示对角矩阵的特征值是其对角元素。
5. 示, 对于方阵A, $\mathbf{A}\mathbf{A}^T$ 是对称矩阵。
6. 表明, 对于三个3D向量a, b, c, 以下同一性成立: $\mathbf{abc} = (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$ 。
7. 解释为什么具有侧向量a, b, c的四面体的体积 (见图5.2) 由abc6给出。
8. 演示矩阵-矩阵乘法的四种解释, 方法是采用以下矩阵-矩阵乘法代码, 重新排列嵌套循环, 并根据矩阵和向量运算解释所得代码。

```
function mat-mult(in a[m][p] inb[p][n] outc[m][n]) {数组c初始化为零
    for i=1 to m
        for j=1 to n
            for k=1 to p
                c[i][j] += a[i][k] * b[k][j]
}
```



9. Prove that if \mathbf{A} , \mathbf{Q} , and \mathbf{D} satisfy Equation (5.14), \mathbf{v} is the i th row of \mathbf{Q} , and λ is the i th entry on the diagonal of \mathbf{D} , then \mathbf{v} is an eigenvector of \mathbf{A} with eigenvalue λ .
10. Prove that if \mathbf{A} , \mathbf{Q} , and \mathbf{D} satisfy Equation (5.14), the eigenvalues of \mathbf{A} are all distinct, and \mathbf{v} is an eigenvector of \mathbf{A} with eigenvalue λ , then for some i , \mathbf{v} is the i th row of \mathbf{Q} and λ is the i th entry on the diagonal of \mathbf{D} .
11. Given the (x, y) coordinates of the three vertices of a 2D triangle, explain why the area is given by

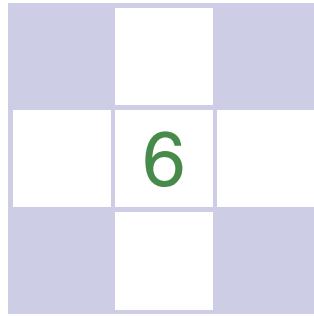
$$\frac{1}{2} \begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{vmatrix}.$$

9.证明如果A, Q和D满足方程 (5.14) , v是Q的第i行, λ是D对角线上的第i个条目, 那么v是具有特征值λ的a的特征向量。

10.证明如果A, Q和D满足方程 (5.14) , A的特征值都是不同的, 并且v是具有特征值λ的a的特征向量, 那么对于某些i, v是Q的第i行, λ是D对角线上的

11.给定2D三角形的三个顶点的 (x, y) 坐标, 解释为什么该区域由

$$\frac{1}{2} \begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{vmatrix}.$$



Transformation Matrices

The machinery of linear algebra can be used to express many of the operations required to arrange objects in a 3D scene, view them with cameras, and get them onto the screen. *Geometric transformations* like rotation, translation, scaling, and projection can be accomplished with matrix multiplication, and the *transformation matrices* used to do this are the subject of this chapter.

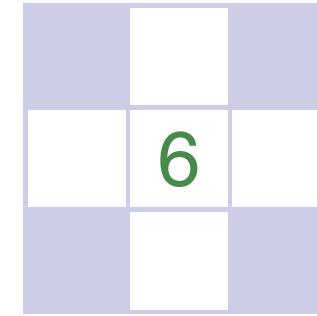
We will show how a set of points transforms if the points are represented as offset vectors from the origin, and we will use the clock shown in Figure 6.1 as an example of a point set. So think of the clock as a bunch of points that are the ends of vectors whose tails are at the origin. We also discuss how these transforms operate differently on locations (points), displacement vectors, and surface normal vectors.

6.1 2D Linear Transformations

We can use a 2×2 matrix to change, or transform, a 2D vector:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}.$$

This kind of operation, which takes in a 2-vector and produces another 2-vector by a simple matrix multiplication, is a *linear transformation*.



变换矩阵

线性代数的机器可以用来表达在3D场景中排列对象，用相机查看它们并将它们放到屏幕上所需的许多操作。像旋转、平移、缩放和投影这样的几何变换可以通过矩阵乘法来完成，用于此目的的变换矩阵是本章的主题。

我们将展示如果点被表示为原点的偏移向量，那么一组点是如何变换的，我们将使用图6.1所示的时钟作为点集的一个例子。因此，将时钟视为一堆点，它们是尾巴位于原点的向量的末端。我们还讨论了这些变换如何在位置（点），位移矢量和表面法线矢量上以不同的方式操作。

6.1 二维线性变换

我们可以使用 2×2 矩阵来改变或变换2D向量：

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}.$$

这种采用2向量并通过简单矩阵乘法产生另一个2向量的操作是线性变换。

By this simple formula we can achieve a variety of useful transformations, depending on what we put in the entries of the matrix, as will be discussed in the following sections. For our purposes, consider moving along the x -axis a horizontal move and along the y -axis, a vertical move.

6.1.1 Scaling

The most basic transform is a *scale* along the coordinate axes. This transform can change length and possibly direction:

$$\text{scale}(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}.$$

Note what this matrix does to a vector with Cartesian components (x, y) :

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}.$$

So, just by looking at the matrix of an axis-aligned scale, we can read off the two scale factors.

Example. The matrix that shrinks x and y uniformly by a factor of two is (Figure 6.1)

$$\text{scale}(0.5, 0.5) = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}.$$

A matrix which halves in the horizontal and increases by three-halves in the vertical is (see Figure 6.2)

$$\text{scale}(0.5, 1.5) = \begin{bmatrix} 0.5 & 0 \\ 0 & 1.5 \end{bmatrix}.$$

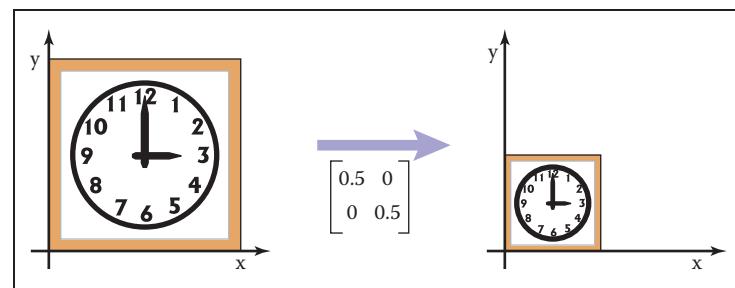


Figure 6.1. Scaling uniformly by half for each axis: The axis-aligned scale matrix has the proportion of change in each of the diagonal elements and zeroes in the off-diagonal elements.

通过这个简单的公式，我们可以实现各种有用的转换，具体取决于我们在矩阵条目中的内容，如下面的部分将讨论的那样。为了我们的目的，考虑沿 x 轴移动一个水平移动和沿 y 轴，一个垂直移动。

6.1.1 Scaling

最基本的变换是沿着坐标轴的尺度。这种变换可以改变长度和可能的方向：

$$\text{scale}(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}.$$

请注意此矩阵对具有笛卡尔分量 (x, y) 的向量做了什么：

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}.$$

因此，只需查看轴对齐尺度的矩阵，我们就可以读取两个比例因子。

例子。将 x 和 y 均匀收缩2倍的矩阵是（图6.1）

$$\text{scale}(0.5, 0.5) = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}.$$

一个矩阵，它在水平方向上减半，在垂直方向上增加三半（见图6.2）

$$\text{scale}(0.5, 1.5) = \begin{bmatrix} 0.5 & 0 \\ 0 & 1.5 \end{bmatrix}.$$

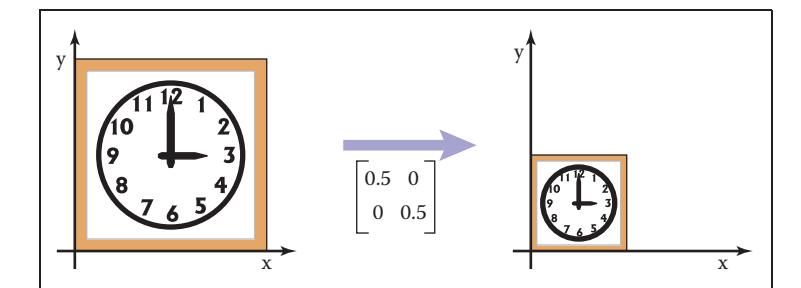


图6.1。 每个轴均匀地按一半缩放：轴对齐的缩放矩阵具有每个对角线元素和非对角线元素中的零的变化比例。

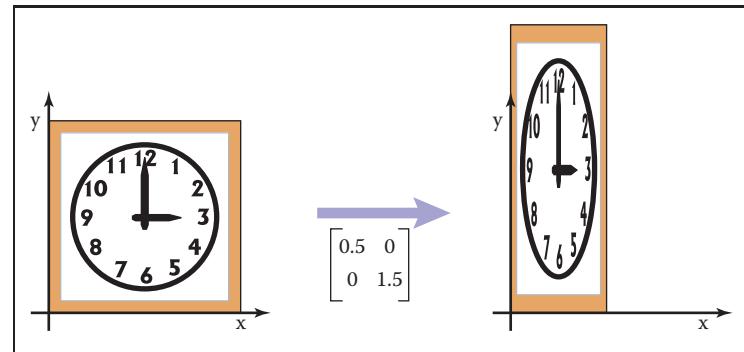


Figure 6.2. Scaling nonuniformly in x and y : The scaling matrix is diagonal with non-equal elements. Note that the square outline of the clock becomes a rectangle and the circular face becomes an ellipse.

6.1.2 Shearing

A shear is something that pushes things sideways, producing something like a deck of cards across which you push your hand; the bottom card stays put and cards move more the closer they are to the top of the deck. The horizontal and vertical shear matrices are

$$\text{shear-}x(s) = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}, \quad \text{shear-}y(s) = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}.$$

Example. The transform that shears horizontally so that vertical lines become 45° lines leaning toward the right is (see Figure 6.3)

$$\text{shear-}x(1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

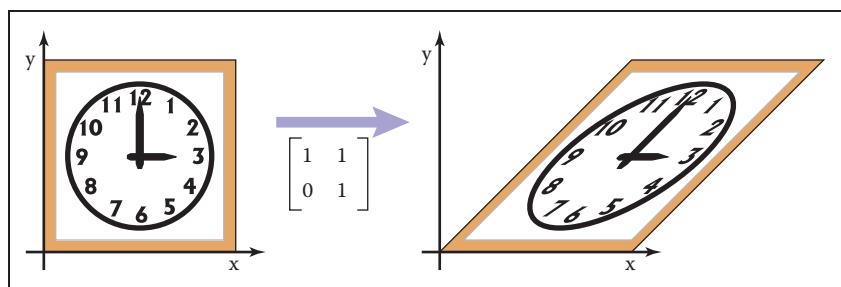


Figure 6.3. An x -shear matrix moves points to the right in proportion to their y -coordinate. Now the square outline of the clock becomes a parallelogram and, as with scaling, the circular face of the clock becomes an ellipse.

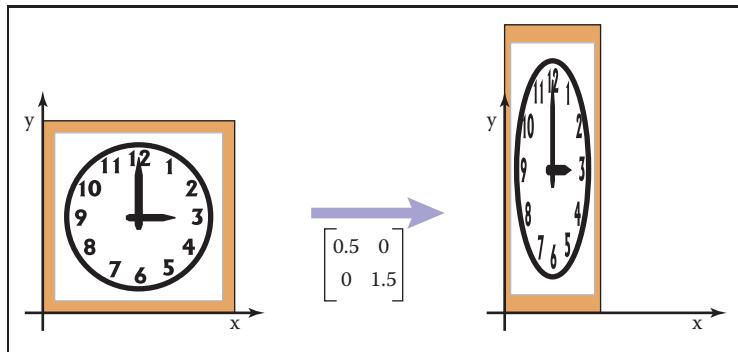


图6.2。在x和y中非均匀地缩放：缩放矩阵是非相等元素的对角线。请注意，时钟的方形轮廓变为矩形，圆形面变为椭圆。

6.1.2 剪切

剪切是一种将东西推向侧面的东西，产生像一副卡片一样的东西，你可以推你的手；底部的卡片保持放置，卡片移动得越靠近甲板的顶部。水平和垂直剪切矩阵为

$$\text{shear-}x(s) = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}, \quad \text{shear-}y(s) = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}.$$

例子。水平剪切使垂直线变为 45° 的变换
向右倾斜的线条是（见图6.3）

$$\text{shear-}x(1) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

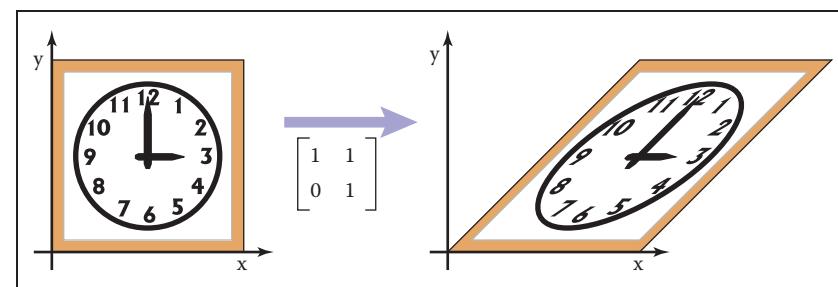


图6.3。 X剪切矩阵按其y坐标的比例向右移动点。现在，时钟的方形轮廓变成平行四边形，与缩放一样，时钟的圆形面变成椭圆。

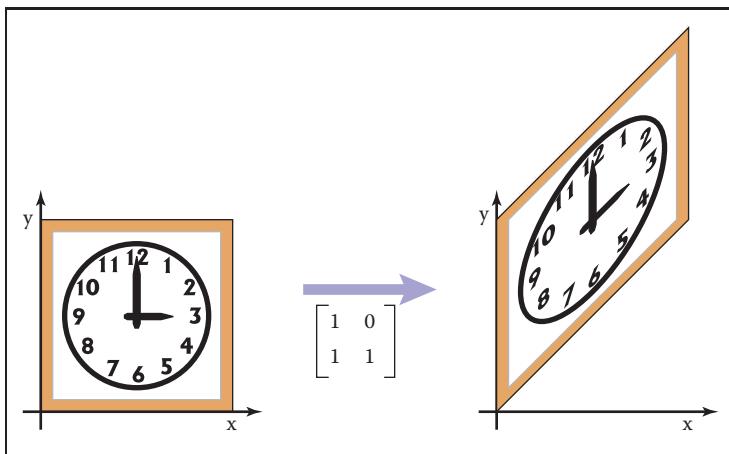


Figure 6.4. A y-shear matrix moves points up in proportion to their x-coordinate.

An analogous transform vertically is (see Figure 6.4)

$$\text{shear-y}(1) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

In both cases, the square outline of the sheared clock becomes a parallelogram, and the circular face of the sheared clock becomes an ellipse.

In fact, the image of a circle under any matrix transformation is an ellipse.

Another way to think of a shear is in terms of rotation of only the vertical (or horizontal) axis. The shear transform that takes a vertical axis and tilts it clockwise by an angle ϕ is

$$\begin{bmatrix} 1 & \tan \phi \\ 0 & 1 \end{bmatrix}.$$

Similarly, the shear matrix which rotates the horizontal axis counterclockwise by angle ϕ is

$$\begin{bmatrix} 1 & 0 \\ \tan \phi & 1 \end{bmatrix}.$$

6.1.3 Rotation

Suppose we want to rotate a vector \mathbf{a} by an angle ϕ counterclockwise to get vector \mathbf{b} (Figure 6.5). If \mathbf{a} makes an angle α with the x -axis, and its length is $r = \sqrt{x_a^2 + y_a^2}$, then we know that

$$\begin{aligned} x_a &= r \cos \alpha, \\ y_a &= r \sin \alpha. \end{aligned}$$

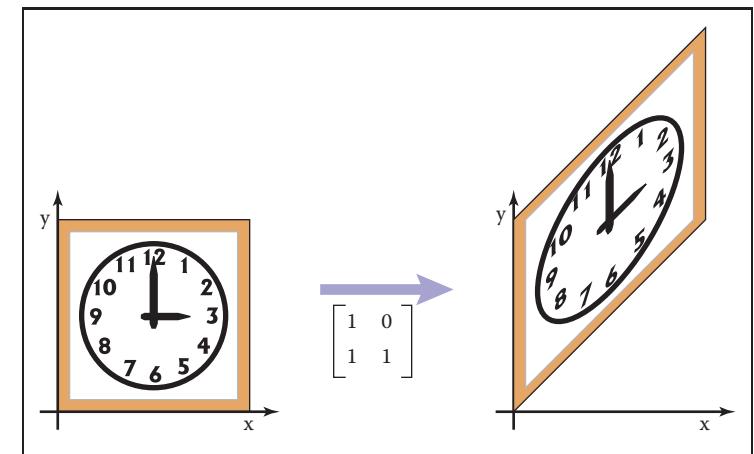


图6.4。Y剪切矩阵按其x坐标的比例向上移动点。

垂直方向的类似变换是（见图6.4）

$$\text{shear-y}(1) = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

在这两种情况下，剪切时钟的方形轮廓变成平行四边形
剪切时钟的圆面变成椭圆。

考虑剪切的另一种方式是仅垂直（或水平）轴的旋转。取垂直轴并将其顺时针倾斜角度 ϕ 的剪切变换为

$$\begin{bmatrix} 1 & \tan \phi \\ 0 & 1 \end{bmatrix}.$$

同样，将水平轴逆时针旋转角度 ϕ 的剪切矩阵为

$$\begin{bmatrix} 1 & 0 \\ \tan \phi & 1 \end{bmatrix}.$$

6.1.3 Rotation

假设我们要将一个矢量 a 逆时针旋转一个角度 ϕ 得到矢量 b （图6.5）。若 a 与 x 轴成夹角 α ，且其长度为 $r=x_2a+y_2a$ ，那么我们知道

$$\begin{aligned} x_a &= r \cos \alpha, \\ y_a &= r \sin \alpha. \end{aligned}$$

Because \mathbf{b} is a rotation of \mathbf{a} , it also has length r . Because it is rotated an angle ϕ from \mathbf{a} , \mathbf{b} makes an angle $(\alpha + \phi)$ with the x -axis. Using the trigonometric addition identities (Section 2.3.3):

$$\begin{aligned}x_b &= r \cos(\alpha + \phi) = r \cos \alpha \cos \phi - r \sin \alpha \sin \phi, \\y_b &= r \sin(\alpha + \phi) = r \sin \alpha \cos \phi + r \cos \alpha \sin \phi.\end{aligned}\quad (6.1)$$

Substituting $x_a = r \cos \alpha$ and $y_a = r \sin \alpha$ gives

$$\begin{aligned}x_b &= x_a \cos \phi - y_a \sin \phi, \\y_b &= y_a \cos \phi + x_a \sin \phi.\end{aligned}$$

In matrix form, the transformation that takes \mathbf{a} to \mathbf{b} is then

$$\text{rotate}(\phi) = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}.$$

Example. A matrix that rotates vectors by $\pi/4$ radians (45 degrees) is (see Figure 6.6)

$$\begin{bmatrix} \cos \frac{\pi}{4} & -\sin \frac{\pi}{4} \\ \sin \frac{\pi}{4} & \cos \frac{\pi}{4} \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix}.$$

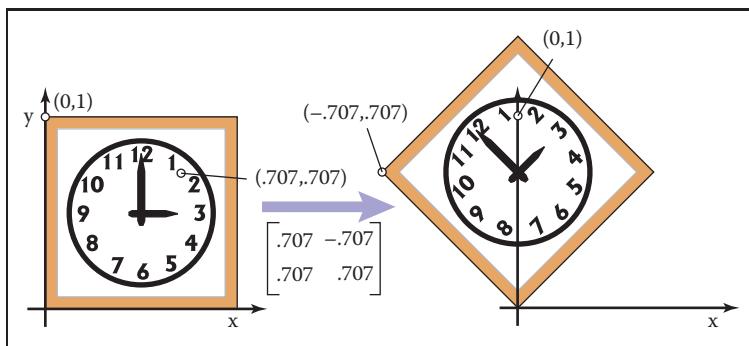


Figure 6.6. A rotation by 45° . Note that the rotation is counterclockwise and that $\cos(45^\circ) = \sin(45^\circ) \approx .707$.

A matrix that rotates by $\pi/6$ radians (30 degrees) in the *clockwise* direction is a rotation by $-\pi/6$ radians in our framework (see Figure 6.7):

$$\begin{bmatrix} \cos \frac{-\pi}{6} & -\sin \frac{-\pi}{6} \\ \sin \frac{-\pi}{6} & \cos \frac{-\pi}{6} \end{bmatrix} = \begin{bmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{bmatrix}.$$

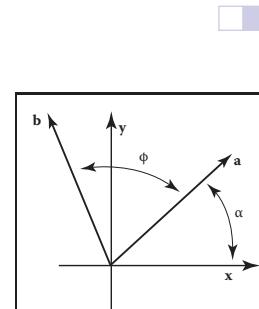


Figure 6.5. The geometry for Equation (6.1).

因为 b 是 a 的旋转，所以它也有长度 r 。因为它从 a 旋转一个角度 ϕ ，所以 b 与 x 轴成一个角度 $(\alpha+\phi)$ 。使用三角加法恒等式（第2.3.3节）：

$$\begin{aligned}xb &= r \cos(\alpha + \phi) = r \cos \alpha \cos \phi - r \sin \alpha \sin \phi \\yb &= r \sin(\alpha + \phi) = r \sin \alpha \cos \phi + r \cos \alpha \sin \phi.\end{aligned}\quad (6.1)$$

代入 $xa=r \cos \alpha$ 和 $ya=r \sin \alpha$ 给出

$$\begin{aligned}xb &= x_a \cos \phi - y_a \sin \phi \\yb &= y_a \cos \phi + x_a \sin \phi.\end{aligned}$$

在矩阵形式中， a 到 b 的变换是

$$\text{rotate}(\phi) = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}.$$

例子。将矢量旋转 $\pi/4$ 弧度 (45度) 的矩阵是 (见图6.6)

$$\begin{bmatrix} \cos \frac{\pi}{4} & -\sin \frac{\pi}{4} \\ \sin \frac{\pi}{4} & \cos \frac{\pi}{4} \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix}.$$

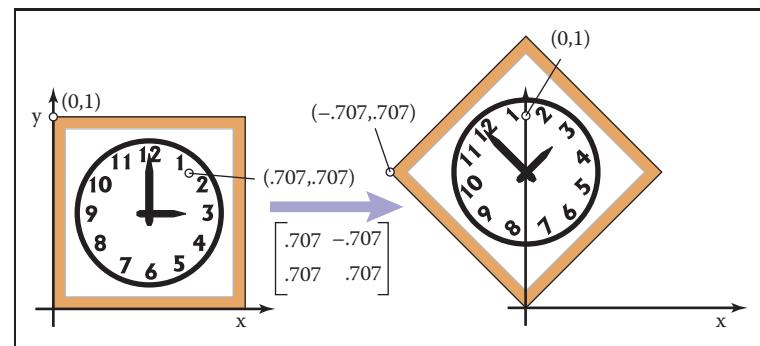


图6.6。旋转45度。请注意，旋转是逆时针的， $\cos(45^\circ)=\sin(45^\circ)\approx .707$ 。

顺时针方向旋转 $\pi/6$ 弧度 (30度) 的矩阵是我们框架中的旋转 $\pi/6$ 弧度 (见图6.7)：

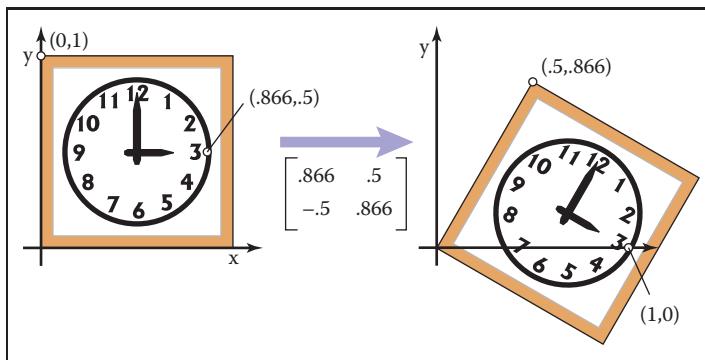


Figure 6.7. A rotation by -30° degrees. Note that the rotation is clockwise and that $\cos(-30^\circ) \approx .866$ and $\sin(-30^\circ) = -.5$.

Because the norm of each row of a rotation matrix is one ($\sin^2 \phi + \cos^2 \phi = 1$), and the rows are orthogonal ($\cos \phi (-\sin \phi) + \sin \phi \cos \phi = 0$), we see that rotation matrices are orthogonal matrices (Section 5.2.4). By looking at the matrix we can read off two pairs of orthonormal vectors: the two columns, which are the vectors to which the transformation sends the canonical basis vectors $(1, 0)$ and $(0, 1)$; and the rows, which are the vectors that the transformation sends to the canonical basis vectors.

Said briefly, $\mathbf{Re}_i = \mathbf{u}_i$ and $\mathbf{Rv}_i = \mathbf{u}_i$, for a rotation with columns \mathbf{u}_i and rows \mathbf{v}_i .

6.1.4 Reflection

We can reflect a vector across either of the coordinate axes by using a scale with one negative scale factor (see Figures 6.8 and 6.9):

$$\text{reflect-y} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \text{reflect-x} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

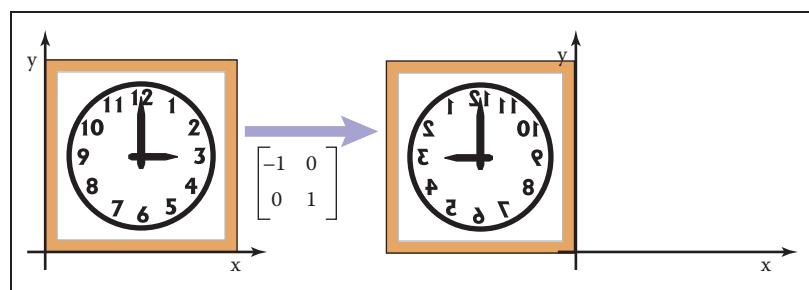


Figure 6.8. A reflection about the y -axis is achieved by multiplying all x -coordinates by -1 .

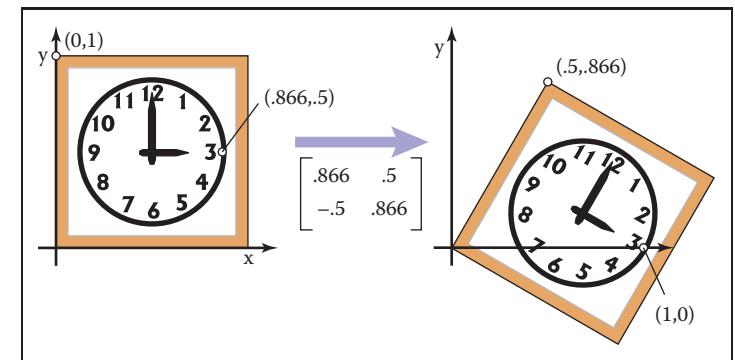


图6.7。旋转-30度。请注意，旋转是顺时针的， $\cos(-30^\circ) \approx .866$ 和 $\sin(-30^\circ) = -.5$ 。

因为旋转矩阵的每一行的范数是一 ($\sin 2\phi + \cos 2\phi = 1$)，并且行是正交的 ($\cos \phi (-\sin \phi) + \sin \phi \cos \phi = 0$)，我们看到旋转矩阵是正交矩阵（第5.2.4节）。通过查看矩阵，我们可以读出两对正交向量：两列，这是转换发送规范基向量 $(1, 0)$ 和 $(0, 1)$ 的向量；和行，这是转换发送到规范基向量的向量。

简单地说， $\mathbf{Re}_i = \mathbf{u}_i$ 和 $\mathbf{Rv}_i = \mathbf{u}_i$ ，用于具有列 \mathbf{u}_i 和 行 \mathbf{v}_i 的旋转。

6.1.4 Reflection

我们可以通过使用一个负比例因子的比例来反映一个跨坐标轴的矢量（见图6.8和图6.9）：

$$\text{reflect-y} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \text{reflect-x} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

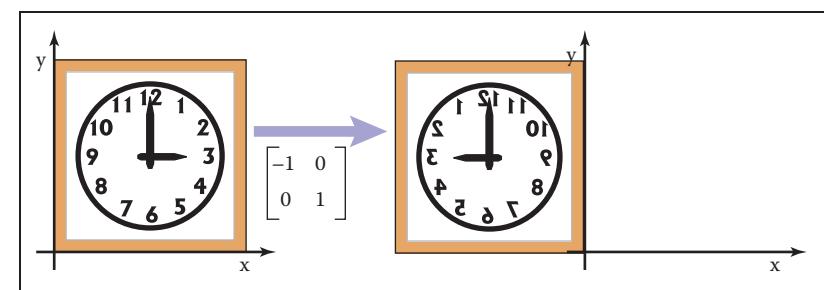


图6.8。通过将所有x坐标乘以-1来实现关于y轴的反射。

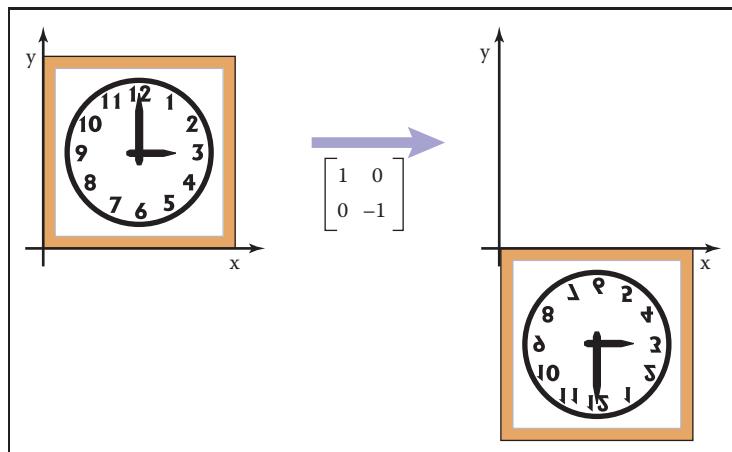


Figure 6.9. A reflection about the x -axis is achieved by multiplying all y -coordinates by -1 .

While one might expect that the matrix with -1 in both elements of the diagonal is also a reflection, in fact it is just a rotation by π radians.

This rotation can also be called a “reflection through the origin.”

6.1.5 Composition and Decomposition of Transformations

It is common for graphics programs to apply more than one transformation to an object. For example, we might want to first apply a scale S , and then a rotation R . This would be done in two steps on a 2D vector v_1 :

$$\text{first, } v_2 = Sv_1, \text{ then, } v_3 = Rv_2.$$

Another way to write this is

$$v_3 = R(Sv_1).$$

Because matrix multiplication is associative, we can also write

$$v_3 = (RS)v_1.$$

In other words, we can represent the effects of transforming a vector by two matrices in sequence using a single matrix of the same size, which we can compute by multiplying the two matrices: $M = RS$ (Figure 6.10).

It is *very important* to remember that these transforms are applied from the *right side first*. So the matrix $M = RS$ first applies S and then R .

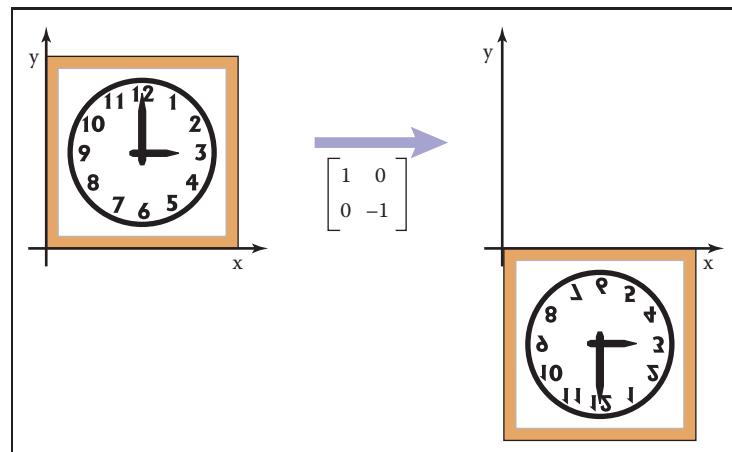


图6.9。通过将所有y坐标乘以 -1 来实现关于x轴的反射。

虽然人们可能期望在对角线的两个元素中具有 1 的矩阵也是反射，但实际上它只是 π 弧度的旋转。

这种旋转也可以称为“通过原点的反射”。

6.1.5 变换的组成和分解

图形程序对一个对象应用多个变换是很常见的。例如，我们可能希望首先应用缩放 S ，然后应用旋转 R 。这将在2D矢量 v_1 上分两步完成：

$$\text{首先, } v_2=Sv_1, \text{ 然后, } v_3=Rv_2.$$

另一种写法是

$$v_3 = R(Sv_1).$$

因为矩阵乘法是关联的，我们也可以写

$$v_3 = (RS)v_1.$$

换句话说，我们可以用一个相同大小的矩阵来表示将一个向量按顺序变换两个matr的效果，我们可以通过乘以两个矩阵来计算： $M=RS$ (图6.10)。

记住这些变换是从右侧首先应用的，这一点非常重要。所以矩阵 $M=RS$ 首先应用 S 然后应用 R 。

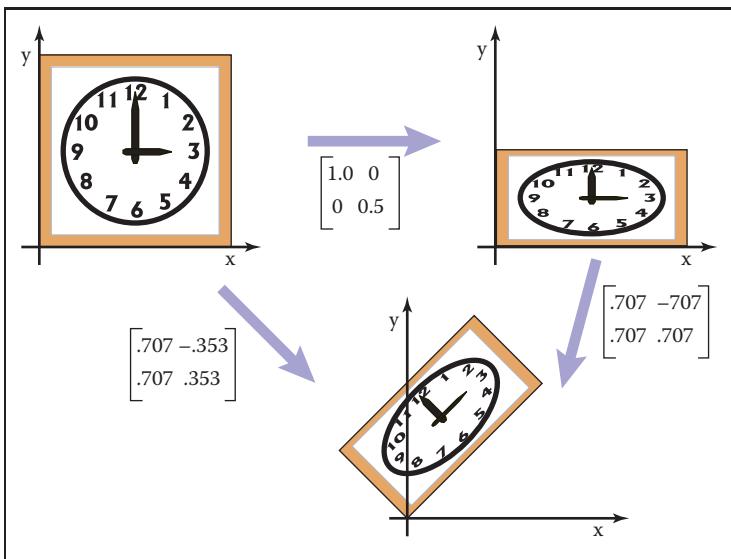


Figure 6.10. Applying the two transform matrices in sequence is the same as applying the product of those matrices once. This is a key concept that underlies most graphics hardware and software.

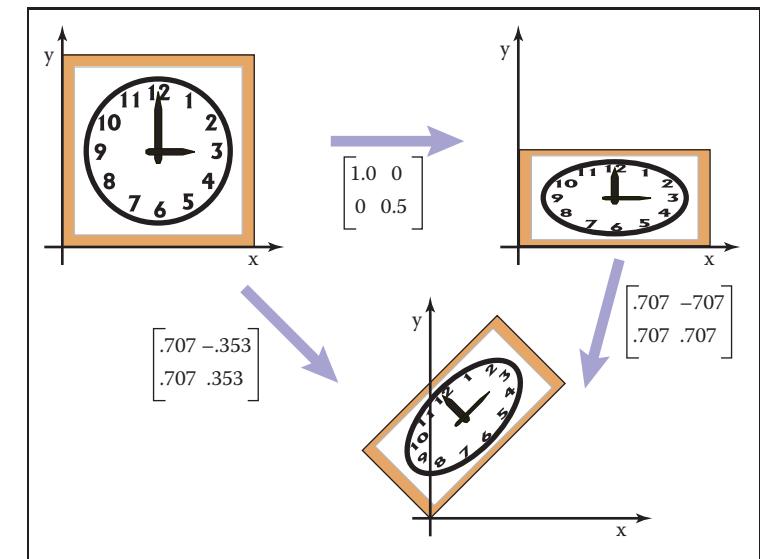
Example. Suppose we want to scale by one-half in the vertical direction and then rotate by $\pi/4$ radians (45 degrees). The resulting matrix is

$$\begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.353 \\ 0.707 & 0.353 \end{bmatrix}.$$

It is important to always remember that matrix multiplication is not commutative. So the order of transforms *does* matter. In this example, rotating first, and then scaling, results in a different matrix (see Figure 6.11):

$$\begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.353 & 0.353 \end{bmatrix}.$$

Example. Using the scale matrices we have presented, nonuniform scaling can only be done along the coordinate axes. If we wanted to stretch our clock by 50% along one of its diagonals, so that 8:00 through 1:00 move to the northwest and 2:00 through 7:00 move to the southeast, we can use rotation matrices in combination with an axis-aligned scaling matrix to get the result we want. The idea is to use a rotation to align the scaling axis with a coordinate axis, then scale along that axis, then rotate back. In our example, the scaling axis is the “backslash” diagonal of the square, and we can make it parallel to the x -axis with



按顺序应用两个变换矩阵与应用那些矩阵的乘积一次相同。这是大多数图形硬件和软件基础的关键概念。

例子。假设我们想在垂直方向上缩放二分之一，然后旋转 $\pi/4$ 弧度（45度）。所得矩阵为

$$\begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.353 \\ 0.707 & 0.353 \end{bmatrix}.$$

重要的是要始终记住矩阵乘法不是交换的。所以变换的顺序确实很重要。在这个例子中，首先旋转，然后缩放，产生不同的矩阵（见图6.11）：

$$\begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.353 & 0.353 \end{bmatrix}.$$

例子。使用我们提出的比例矩阵，只能沿着坐标轴进行非均匀缩放。如果我们想将时钟沿其中一条对角线拉伸50%，使8:00到1:00移动到西北，2:00到7:00移动到东南，我们可以使用旋转矩阵和轴对齐的缩放矩阵来获得我们想要的结果。这个想法是使用旋转将缩放轴与坐标轴对齐，然后沿着该轴缩放，然后向后旋转。在我们的例子中，缩放轴是正方形的“反斜杠”对角线，我们可以使它与x轴平行

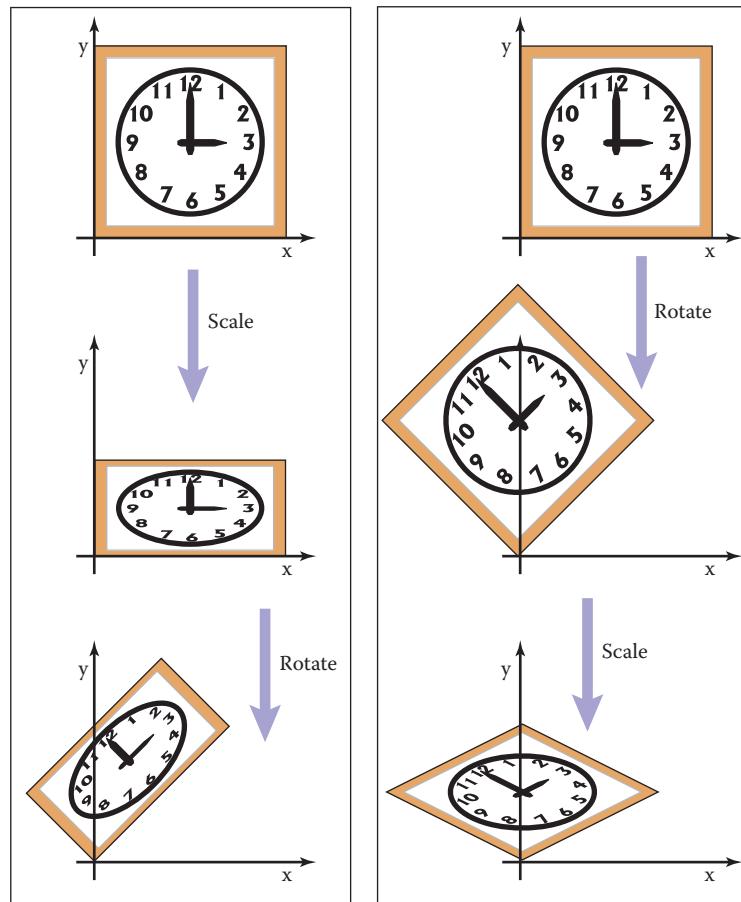


Figure 6.11. The order in which two transforms are applied is usually important. In this example, we do a scale by one-half in y and then rotate by 45° . Reversing the order in which these two transforms are applied yields a different result.

a rotation by $+45^\circ$. Putting these operations together, the full transformation is

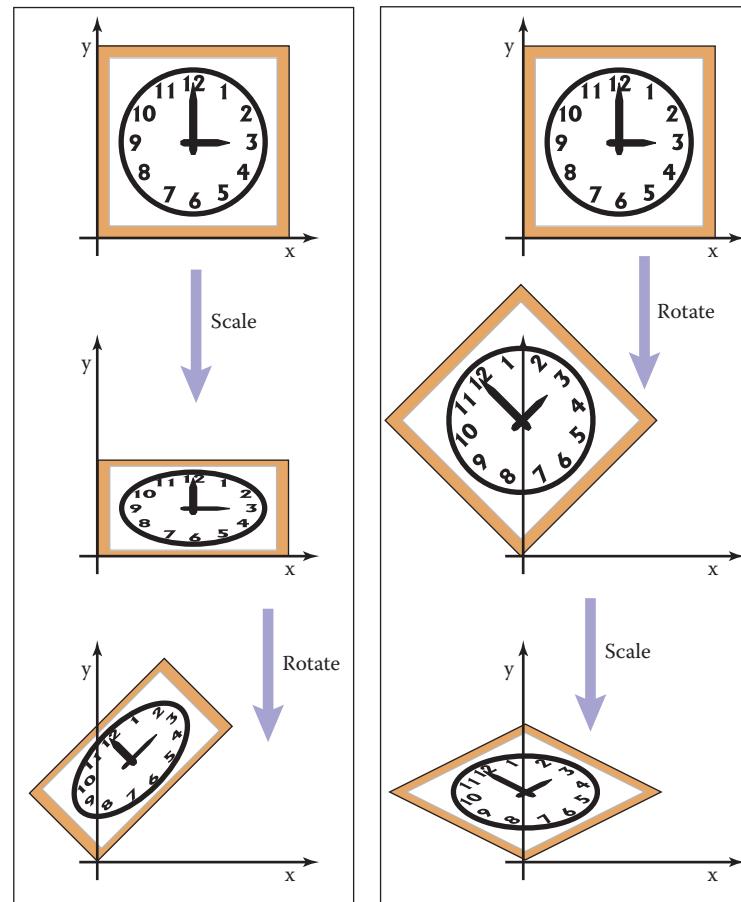
$$\text{rotate}(-45^\circ) \text{ scale}(1.5, 1) \text{ rotate}(45^\circ).$$

In mathematical notation, this can be written \mathbf{RSR}^T . The result of multiplying the three matrices together is

$$\begin{bmatrix} 1.25 & -0.25 \\ -0.25 & 1.25 \end{bmatrix}$$

Remember to read the transformations from right to left.

It is no coincidence that this matrix is symmetric—try applying the transpose-of-product rule to the formula \mathbf{RSR}^T .



应用两个变换的顺序通常很重要。在这次考试中，我们在 y 中做一个二分之一的刻度，然后旋转 45° 。反转应用这两个变换的顺序会产生不同的结果。

旋转+45：将这些操作放在一起，完整的转换是

$$\text{旋转 } (-45^\circ) \text{ 刻度 } (1.5, 1) \text{ 旋转 } (45^\circ)$$

阅读从右到左的转换。

在数学符号中，这可以写成 \mathbf{RSR}^T 。将三个矩阵相乘的结果是

$$\begin{bmatrix} 1.25 & -0.25 \\ -0.25 & 1.25 \end{bmatrix}$$

这个矩阵是对称的并非巧合—尝试将transpose-of-product规则应用于公式 \mathbf{RSR}^T 。

Building up a transformation from rotation and scaling transformations actually works for any linear transformation, and this fact leads to a powerful way of thinking about these transformations, as explored in the next section.

6.1.6 Decomposition of Transformations

Sometimes it's necessary to "undo" a composition of transformations, taking a transformation apart into simpler pieces. For instance, it's often useful to present a transformation to the user for manipulation in terms of separate rotations and scale factors, but a transformation might be represented internally simply as a

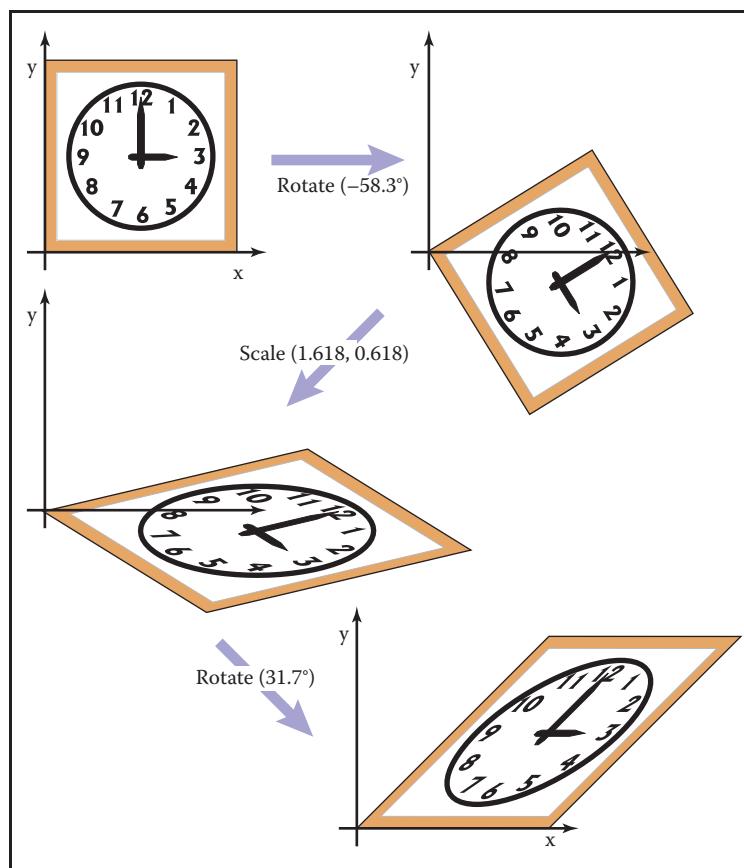
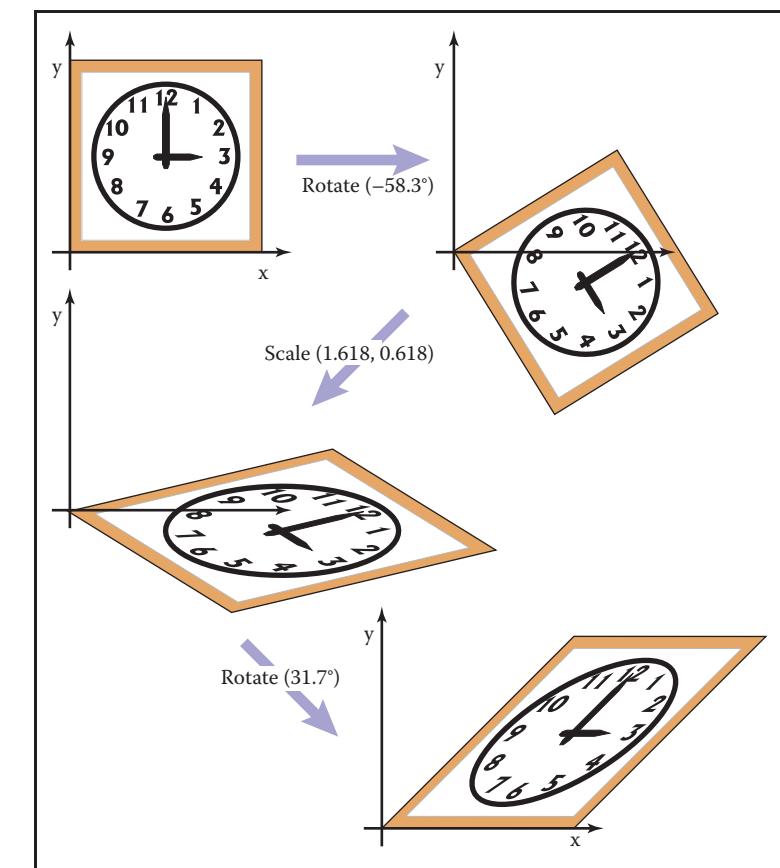


Figure 6.12. Singular Value Decomposition (SVD) for a shear matrix. Any 2D matrix can be decomposed into a product of rotation, scale, rotation. Note that the circular face of the clock must become an ellipse because it is just a rotated and scaled circle.

从旋转和缩放变换构建变换实际上适用于任何线性变换，这一事实导致了对这些变换的强大思考方式，如下一节所述。

6.1.6 变换的分解

有时需要“撤消”转换的组合，将转换拆分为更简单的部分。例如，通过单独的旋转和比例因子向用户呈现用于操作的转换通常很有用，但是转换可能在内部简单地表示为



剪切矩阵的奇异值分解(SVD)。任何2D矩阵都可以分解为旋转，缩放，旋转的乘积。请注意，时钟的圆面必须成为椭圆，因为它只是一个旋转和缩放的圆。

matrix, with the rotations and scales already mixed together. This kind of manipulation can be achieved if the matrix can be computationally disassembled into the desired pieces, the pieces adjusted, and the matrix reassembled by multiplying the pieces together again.

It turns out that this decomposition, or factorization, is possible, regardless of the entries in the matrix—and this fact provides a fruitful way of thinking about transformations and what they do to geometry that is transformed by them.

Symmetric Eigenvalue Decomposition

Let's start with symmetric matrices. Recall from Section 5.4 that a symmetric matrix can always be taken apart using the eigenvalue decomposition into a product of the form

$$\mathbf{A} = \mathbf{R}\mathbf{S}\mathbf{R}^T$$

where \mathbf{R} is an orthogonal matrix and \mathbf{S} is a diagonal matrix; we will call the columns of \mathbf{R} (the eigenvectors) by the names \mathbf{v}_1 and \mathbf{v}_2 , and we'll call the diagonal entries of \mathbf{S} (the eigenvalues) by the names λ_1 and λ_2 .

In geometric terms we can now recognize \mathbf{R} as a rotation and \mathbf{S} as a scale, so this is just a multi-step geometric transformation (Figure 6.13):

1. Rotate \mathbf{v}_1 and \mathbf{v}_2 to the x - and y -axes (the transform by \mathbf{R}^T).
2. Scale in x and y by (λ_1, λ_2) (the transform by \mathbf{S}).
3. Rotate the x - and y -axes back to \mathbf{v}_1 and \mathbf{v}_2 (the transform by \mathbf{R}).

Looking at the effect of these three transforms together, we can see that they have the effect of a nonuniform scale along a pair of axes. As with an axis-aligned scale, the axes are perpendicular, but they aren't the coordinate axes; instead they

矩阵，旋转和尺度已经混合在一起。如果矩阵可计算地分解成所需的片，调整片，并且通过将片再次乘在一起重新组装矩阵，则可实现这种操作。

事实证明，无论矩阵中的条目如何，这种分解或因式分解都是可能的—这一事实提供了一种富有效果的思考变换的方式，以及它们对由它们变换的几何体

对称特征值分解

让我们从对称矩阵开始。从第5.4节中回想一下，对称matrix总是可以使用特征值分解分解为形式的乘积

$$\mathbf{A} = \mathbf{R}\mathbf{S}\mathbf{R}^T$$

其中 \mathbf{R} 是正交矩阵， \mathbf{S} 是对角矩阵;我们将通过名称 \mathbf{v}_1 和 \mathbf{v}_2 调用 \mathbf{R} (特征向量) 的列，我们将通过名称 λ_1 和 λ_2 调用 \mathbf{S} (特征值) 的diagonal条目。

在几何术语中，我们现在可以将 \mathbf{R} 识别为旋转，将 \mathbf{S} 识别为刻度，因此这只是一个步骤的几何变换 (图6.13)：

1. 将 \mathbf{v}_1 和 \mathbf{v}_2 旋转到 x 和 y 轴 (由 \mathbf{R}^T 转换)。
2. 在 x 和 y 中按 (λ_1, λ_2) 缩放 (由 \mathbf{S} 转换)。
3. 将 x 和 y 轴旋转回 \mathbf{v}_1 和 \mathbf{v}_2 (由 \mathbf{R} 转换)。

看看这三个变换的效果，我们可以看到它们具有沿着一对轴的非均匀尺度的效果。与轴对齐的刻度一样，轴是垂直的，但它们不是坐标轴；相反，它们是

如果你喜欢计算dimensions: 一个对称的 2×2 矩阵有3degrees of freedom，并且特征值分解将它们重写为旋转角度和两个比例因子。

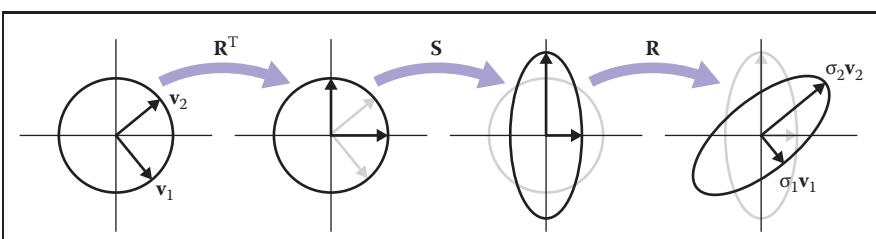


Figure 6.13. What happens when the unit circle is transformed by an arbitrary symmetric matrix \mathbf{A} , also known as a non-axis-aligned, nonuniform scale. The two perpendicular vectors \mathbf{v}_1 and \mathbf{v}_2 , which are the eigenvectors of \mathbf{A} , remain fixed in direction but get scaled. In terms of elementary transformations, this can be seen as first rotating the eigenvectors to the canonical basis, doing an axis-aligned scale, and then rotating the canonical basis back to the eigenvectors.

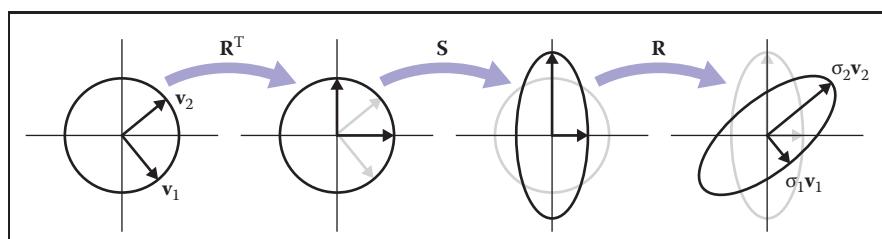


Figure 6.13. 当单位圆被任意对称矩阵变换时会发生什么 \mathbf{A} ，也称为非轴对齐、非均匀刻度。作为a的特征向量的两个垂直向量 \mathbf{v}_1 和 \mathbf{v}_2 在方向上保持固定，但得到缩放。就基本变换而言，这可以看作是首先将特征向量旋转到规范基础，做一个轴对齐的尺度，然后将规范基础旋转回特征向量。

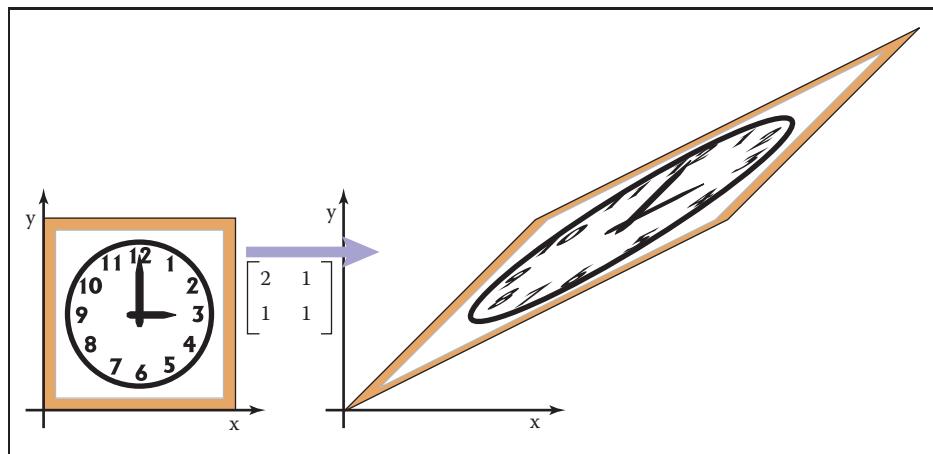


Figure 6.14. A symmetric matrix is always a scale along some axis. In this case it is along the $\phi = 31.7^\circ$ direction which means the real eigenvector for this matrix is in that direction.

are the eigenvectors of \mathbf{A} . This tells us something about what it means to be a symmetric matrix: symmetric matrices are just scaling operations—albeit potentially nonuniform and non-axis-aligned ones.

Example. Recall the example from Section 5.4:

$$\begin{aligned} \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} &= \mathbf{R} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \mathbf{R}^T \\ &= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 2.618 & 0 \\ 0 & 0.382 \end{bmatrix} \begin{bmatrix} 0.8507 & 0.5257 \\ -0.5257 & 0.8507 \end{bmatrix} \\ &= \text{rotate } (31.7^\circ) \text{ scale } (2.618, 0.382) \text{ rotate } (-31.7^\circ). \end{aligned}$$

The matrix above, then, according to its eigenvalue decomposition, scales in a direction 31.7° counterclockwise from three o'clock (the x -axis). This is a touch before 2 p.m. on the clockface as is confirmed by Figure 6.14.

We can also reverse the diagonalization process; to scale by (λ_1, λ_2) with the first scaling direction an angle ϕ clockwise from the x -axis, we have

$$\begin{aligned} \begin{bmatrix} -\cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} &= \\ \begin{bmatrix} \lambda_1 \cos^2 \phi + \lambda_2 \sin^2 \phi & (\lambda_2 - \lambda_1) \cos \phi \sin \phi \\ (\lambda_2 - \lambda_1) \cos \phi \sin \phi & \lambda_2 \cos^2 \phi + \lambda_1 \sin^2 \phi \end{bmatrix}. \end{aligned}$$

We should take heart that this is a symmetric matrix as we know must be true since we constructed it from a symmetric eigenvalue decomposition. □

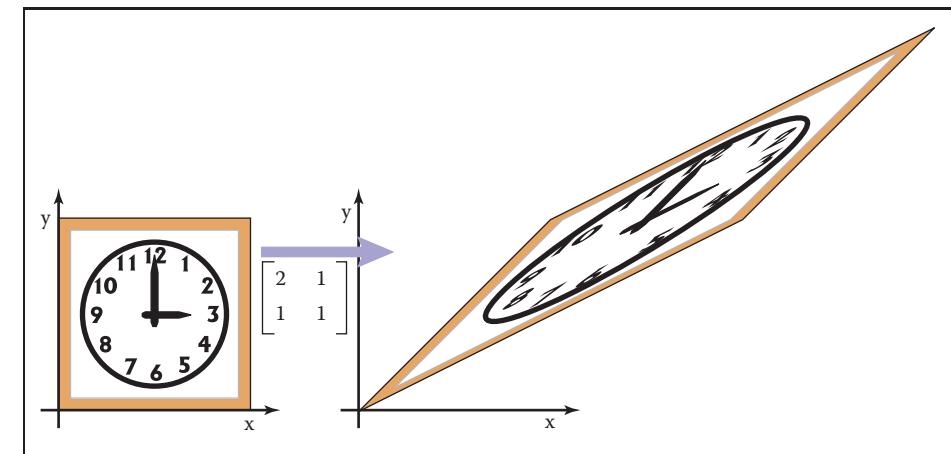


图6.14。 对称矩阵总是沿着某个轴的尺度。在这种情况下，它是沿着 $\phi=31.7^\circ$ 方向，这意味着该矩阵的实特征向量是在该方向上。

是 a 的特征向量。这告诉我们一些关于对称矩阵的含义：对称矩阵只是缩放操作—尽管可能是非均匀和非轴对齐的操作。

例子。回想一下第5.4节的例子：

$$\begin{aligned} \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} &= \mathbf{R} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \mathbf{R}^T \\ &= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 2.618 & 0 \\ 0 & 0.382 \end{bmatrix} \begin{bmatrix} 0.8507 & 0.5257 \\ -0.5257 & 0.8507 \end{bmatrix} \\ &= \text{旋转}(31.7^\circ) \text{比例尺}(2.618, 0.382) \text{旋转}(-31.7^\circ). \end{aligned}$$

上面的矩阵，然后，根据其特征值分解，在一个方向上缩放。从三点钟开始逆时针方向（ x 轴）。如图6.14所示，这是在下午2点之前在时钟面上的触摸。

我们也可以反转对角化过程；要用第一个缩放方向从 x 轴顺时针角度 ϕ 按 (λ_1, λ_2) 缩放，我们有

$$\begin{aligned} \begin{bmatrix} -\cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} &= \\ \begin{bmatrix} \lambda_1 \cos^2 \phi + \lambda_2 \sin^2 \phi & (\lambda_2 - \lambda_1) \cos \phi \sin \phi \\ (\lambda_2 - \lambda_1) \cos \phi \sin \phi & \lambda_2 \cos^2 \phi + \lambda_1 \sin^2 \phi \end{bmatrix}. \end{aligned}$$

我们应该记住这是一个对称矩阵，因为我们知道它必须是真实的，因为我们从对称特征值分解中构造它。

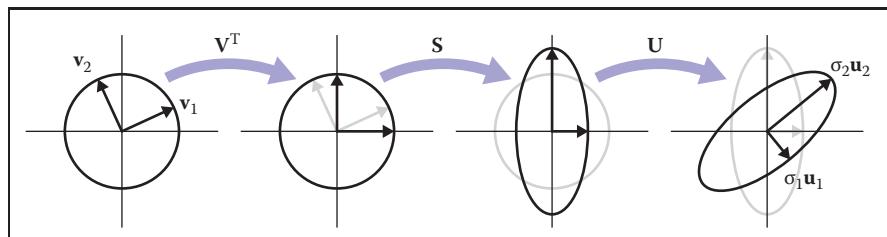


Figure 6.15. What happens when the unit circle is transformed by an arbitrary matrix A . The two perpendicular vectors v_1 and v_2 , which are the right singular vectors of A , get scaled and changed in direction to match the left singular vectors, u_1 and u_2 . In terms of elementary transformations, this can be seen as first rotating the right singular vectors to the canonical basis, doing an axis-aligned scale, and then rotating the canonical basis to the left singular vectors.

Singular Value Decomposition

A very similar kind of decomposition can be done with nonsymmetric matrices as well: it's the Singular Value Decomposition (SVD), also discussed in Section 5.4.1. The difference is that the matrices on either side of the diagonal matrix are no longer the same:

$$A = USV^T$$

The two orthogonal matrices that replace the single rotation R are called U and V , and their columns are called u_i (the *left singular vectors*) and v_i (the *right singular vectors*), respectively. In this context, the diagonal entries of S are called *singular values* rather than eigenvalues. The geometric interpretation is very similar to that of the symmetric eigenvalue decomposition (Figure 6.15):

1. Rotate v_1 and v_2 to the x - and y -axes (the transform by V^T).
2. Scale in x and y by (σ_1, σ_2) (the transform by S).
3. Rotate the x - and y -axes to u_1 and u_2 (the transform by U).

The principal difference is between a single rotation and two different orthogonal matrices. This difference causes another, less important, difference. Because the SVD has different singular vectors on the two sides, there is no need for negative singular values: we can always flip the sign of a singular value, reverse the direction of one of the associated singular vectors, and end up with the same transformation again. For this reason, the SVD always produces a diagonal matrix with all positive entries, but the matrices U and V are not guaranteed to be rotations—they could include reflection as well. In geometric applications like graphics this is an inconvenience, but a minor one: it is easy to differentiate rotations from reflections by checking the determinant, which is $+1$ for rotations

For dimension counters: a general 2×2 matrix has 4 degrees of freedom, and the SVD rewrites them as two rotation angles and two scale factors. One more bit is needed to keep track of reflections, but that doesn't add a dimension.

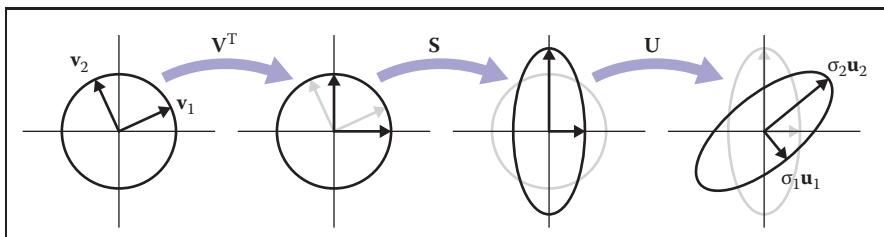


图6.15。当单位圆由任意矩阵a变换时会发生什么。作为a的右奇异向量的两个垂直向量v1和v2在方向上缩放和改变以匹配左奇异向量u1和u2。就基本变换而言，这可以看作是首先将右奇异向量旋转到规范基础，做一个轴对齐的尺度，然后将规范基础旋转到左奇异向量。

奇异值分解

一个非常类似的分解也可以用非对称矩阵来完成：它是奇异值分解（SVD），也在第5.4.1节中讨论。不同之处在于对角矩阵两侧的矩阵不再相同：

$$A = USV^T$$

取代单旋转R的两个正交矩阵称为U和V，它们的列分别称为 u_i （左奇异向量）和 v_i （右奇异向量）。在此上下文中， S 的对角线条目被称为奇异值而不是特征值。几何解释与对称特征值分解的解释非常相似（图6.15）：

1. 将 v_1 和 v_2 旋转到 x 和 y 轴（由 V^T 转换）。
2. 在 x 和 y 中按 $(\sigma_1 \sigma_2)$ 缩放(S 的变换)。
3. 将 x 和 y 轴旋转到 u_1 和 u_2 （由 U 转换）。

主要区别在于单个旋转和两个不同的正交矩阵之间。这种差异导致另一个不那么重要的差异。因为SVD在两侧具有不同的奇异向量，所以不需要负奇异值：我们总是可以翻转一个奇异值的符号，反转其中一个相关奇异向量的方向，并再次以相同的变由于这个原因，SVD总是产生一个包含所有正条目的对角矩阵，但是矩阵U和V不能保证是旋转—它们也可以包括反射。在像图形这样的几何应用中，这是一个不便，但也是一个小问题：通过检查行列式（旋转为+1）可以很容易地将旋转与反射区分开来

对于维度计数器：一般的 2×2 矩阵有4个自由度，SVD将它们重写为两个旋转角度和两个比例因子。需要多一点来跟踪反射，但这不会增加维度。

and -1 for reflections, and if rotations are desired, one of the singular values can be negated, resulting in a rotation–scale–rotation sequence where the reflection is rolled in with the scale, rather than with one of the rotations.

Example. The example used in Section 5.4.1 is in fact a shear matrix (Figure 6.12):

$$\begin{aligned} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} &= \mathbf{R}_2 \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \mathbf{R}_1 \\ &= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 1.618 & 0 \\ 0 & 0.618 \end{bmatrix} \begin{bmatrix} 0.5257 & 0.8507 \\ -0.8507 & 0.5257 \end{bmatrix} \\ &= \text{rotate } (31.7^\circ) \text{ scale } (1.618, 0.618) \text{ rotate } (-58.3^\circ). \end{aligned}$$

An immediate consequence of the existence of SVD is that all the 2D transformation matrices we have seen can be made from rotation matrices and scale matrices. Shear matrices are a convenience, but they are not required for expressing transformations.

In summary, every matrix can be decomposed via SVD into a rotation times a scale times another rotation. Only symmetric matrices can be decomposed via eigenvalue diagonalization into a rotation times a scale times the inverse-rotation, and such matrices are a simple scale in an arbitrary direction. The SVD of a symmetric matrix will yield the same triple product as eigenvalue decomposition via a slightly more complex algebraic manipulation.

Paeth Decomposition of Rotations

Another decomposition uses shears to represent nonzero rotations (Paeth, 1990). The following identity allows this:

$$\begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} = \begin{bmatrix} 1 & \frac{\cos \phi - 1}{\sin \phi} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin \phi & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{\cos \phi - 1}{\sin \phi} \\ 0 & 1 \end{bmatrix}.$$

For example, a rotation by $\pi/4$ (45 degrees) is (see Figure 6.16)

$$\text{rotate}\left(\frac{\pi}{4}\right) = \begin{bmatrix} 1 & 1 - \sqrt{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{\sqrt{2}}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 - \sqrt{2} \\ 0 & 1 \end{bmatrix}. \quad (6.2)$$

This particular transform is useful for raster rotation because shearing is a very efficient raster operation for images; it introduces some jagginess, but will

和 -1 用于反射，如果需要旋转，则可以否定其中一个奇异值，从而产生一个旋转 尺度 旋转序列，其中反射随尺度滚动，而不是随旋转之一滚动。

例子。在5.4.1节中使用的例子实际上是一个剪切矩阵（图6.12）：

$$\begin{aligned} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} &= \mathbf{R}_2 \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \mathbf{R}_1 \\ &= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 1.618 & 0 \\ 0 & 0.618 \end{bmatrix} \begin{bmatrix} 0.5257 & 0.8507 \\ -0.8507 & 0.5257 \end{bmatrix} \\ &= \text{旋转}(31.7^\circ) \text{比例尺}(1.618, 0.618) \text{旋转}(-58.3^\circ). \end{aligned}$$

SVD存在的一个直接后果是，我们已经看到的所有2D变换矩阵都可以由旋转矩阵和尺度矩阵制成。剪切矩阵是一种方便，但它们不是表达变换所必需的。

总之，每个矩阵都可以通过SVD分解为一个旋转倍一个尺度倍另一个旋转。只有对称矩阵可以通过特征值对角化分解为旋转倍一个尺度倍逆旋转，并且这样的矩阵是一个任意方向的简单尺度。对称矩阵的SVD将通过稍微复杂的代数操作产生与特征值分解相同的三乘积。

旋转的Paeth分解

另一种分解使用剪切来表示非零旋转（Paeth, 1990）。以下身份允许这样做：

$$\begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} = \begin{bmatrix} 1 & \frac{\cos \phi - 1}{\sin \phi} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin \phi & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{\cos \phi - 1}{\sin \phi} \\ 0 & 1 \end{bmatrix}.$$

例如，旋转 $\pi/4$ （45度）是（见图6.16）

$$\text{rotate}\left(\frac{\pi}{4}\right) = \begin{bmatrix} 1 & 1 - \sqrt{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{\sqrt{2}}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 - \sqrt{2} \\ 0 & 1 \end{bmatrix}. \quad (6.2)$$

这种特殊的变换对于栅格旋转很有用，因为剪切是图像的一种非常有效的栅格操作；它引入了一些jagginess，但会

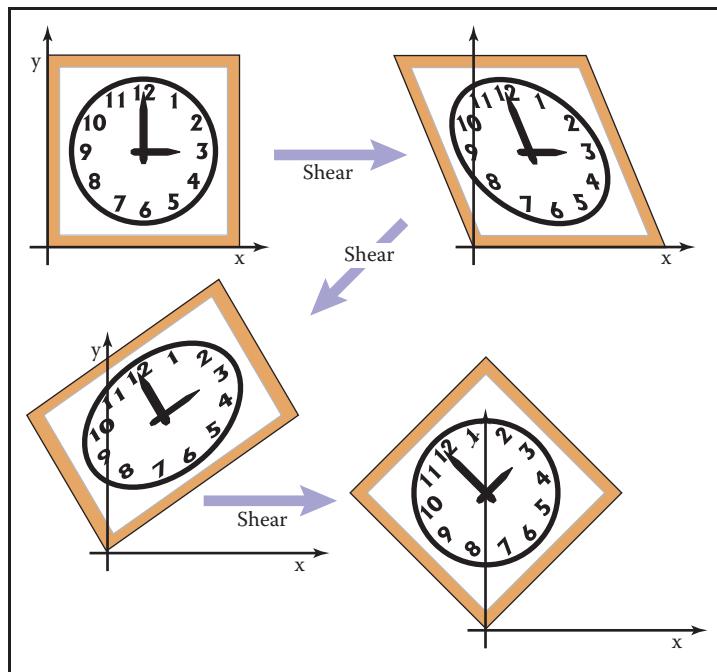


Figure 6.16. Any 2D rotation can be accomplished by three shears in sequence. In this case a rotation by 45° is decomposed as shown in Equation 6.2.

leave no holes. The key observation is that if we take a raster position (i, j) and apply a horizontal shear to it, we get

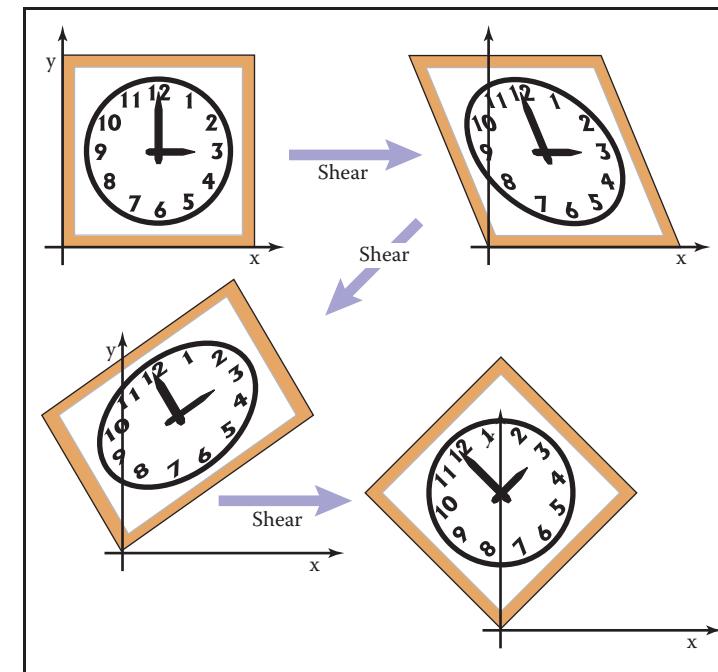
$$\begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i + sj \\ j \end{bmatrix}.$$

If we round sj to the nearest integer, this amounts to taking each row in the image and moving it sideways by some amount—a different amount for each row. Because it is the same displacement within a row, this allows us to rotate with no gaps in the resulting image. A similar action works for a vertical shear. Thus, we can implement a simple raster rotation easily.

6.2 3D Linear Transformations

The linear 3D transforms are an extension of the 2D transforms. For example, a scale along Cartesian axes is

$$\text{scale}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}. \quad (6.3)$$



任何二维旋转都可以通过三个剪切顺序完成。在这种情况下，旋转 45° 被分解，如公式6.2所示。

不要留下漏洞。关键的观察是，如果我们采取栅格位置 (i, j) 并对其施加水平剪切，我们得到

$$\begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i + sj \\ j \end{bmatrix}.$$

如果我们将 sj 舍入到最接近的整数，这相当于取图像中的每一行并将其横向移动一些量—每行的量不同。因为它是一行内的相同位移，这允许我们在生成的图像中没有间隙地旋转。类似的动作适用于垂直剪切。因此，我们可以很容易地实现一个简单的光栅旋转。

6.2 三维线性变换

线性3D变换是2D变换的扩展。例如，沿笛卡尔轴的尺度是

$$\text{scale}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}. \quad (6.3)$$

Rotation is considerably more complicated in 3D than in 2D, because there are more possible axes of rotation. However, if we simply want to rotate about the z -axis, which will only change x - and y -coordinates, we can use the 2D rotation matrix with no operation on z :

$$\text{rotate-}z(\phi) = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Similarly we can construct matrices to rotate about the x -axis and the y -axis:

$$\text{rotate-}x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix},$$

$$\text{rotate-}y(\phi) = \begin{bmatrix} \cos \phi & 0 & \sin \phi \\ 0 & 1 & 0 \\ -\sin \phi & 0 & \cos \phi \end{bmatrix}.$$

To understand why the minus sign is in the lower left for the y -axis rotation, think of the three axes in a circular sequence: y after x ; z after y ; x after z .

旋转在3D中比在2D中复杂得多，因为有更多可能的旋转轴。但是，如果我们只是想绕 z 轴旋转，这只会改变 x 和 y 坐标，我们可以使用2D旋转矩阵，而对 z 没有操作：

$$\text{rotate-}z(\phi) = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

同样，我们可以构造矩阵以绕 x 轴和 y 轴旋转：

$$\text{rotate-}x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix},$$

$$\text{rotate-}y(\phi) = \begin{bmatrix} \cos \phi & 0 & \sin \phi \\ 0 & 1 & 0 \\ -\sin \phi & 0 & \cos \phi \end{bmatrix}.$$

要了解为什么minus符号位于 y 轴旋转的左下方，请考虑圆形序列中的三个轴： x 之后的 y ; y 之后的 z ; z 之后的 x 。

We will discuss rotations about arbitrary axes in the next section.

As in two dimensions, we can shear along a particular axis, for example,

$$\text{shear-}x(d_y, d_z) = \begin{bmatrix} 1 & d_y & d_z \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

As with 2D transforms, any 3D transformation matrix can be decomposed using SVD into a rotation, scale, and another rotation. Any symmetric 3D matrix has an eigenvalue decomposition into rotation, scale, and inverse-rotation. Finally, a 3D rotation can be decomposed into a product of 3D shear matrices.

6.2.1 Arbitrary 3D Rotations

As in 2D, 3D rotations are *orthogonal* matrices. Geometrically, this means that the three rows of the matrix are the Cartesian coordinates of three mutually orthogonal unit vectors as discussed in Section 2.4.5. The columns are three, potentially different, mutually orthogonal unit vectors. There are an infinite number of such rotation matrices. Let's write down such a matrix:

$$\mathbf{R}_{uvw} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}.$$

我们将在下一节讨论关于任意轴的旋转。

在二维中，我们可以沿着特定的轴剪切，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2D变换一样，任何3d变换矩阵都可以使用SVD分解为旋转、缩放和另一个旋转。任何对称三维矩阵都有一个特征值分解为旋转、缩放和逆旋转。最后，三维旋转可以分解为三维剪切矩阵的乘积。

6.2.1 任意3D旋转

与2D一样，3D旋转是正交矩阵。在几何上，这意味着矩阵的三行是如第2.4.5节所讨论的三个相互正交的单位向量的笛卡尔坐标。列是三个可能不同的相互正交的单位向量。存在无限数量的这种旋转矩阵。让我们写下这样的矩阵：

$$\mathbf{R}_{uvw} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}.$$

Here, $\mathbf{u} = x_u \mathbf{x} + y_u \mathbf{y} + z_u \mathbf{z}$ and so on for \mathbf{v} and \mathbf{w} . Since the three vectors are orthonormal we know that

$$\begin{aligned}\mathbf{u} \cdot \mathbf{u} &= \mathbf{v} \cdot \mathbf{v} = \mathbf{w} \cdot \mathbf{w} = 1, \\ \mathbf{u} \cdot \mathbf{v} &= \mathbf{v} \cdot \mathbf{w} = \mathbf{w} \cdot \mathbf{u} = 0.\end{aligned}$$

We can infer some of the behavior of the rotation matrix by applying it to the vectors \mathbf{u} , \mathbf{v} and \mathbf{w} . For example,

$$\mathbf{R}_{uvw} \mathbf{u} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix} \begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix} = \begin{bmatrix} x_u x_u + y_u y_u + z_u z_u \\ x_v x_u + y_v y_u + z_v z_u \\ x_w x_u + y_w y_u + z_w z_u \end{bmatrix}.$$

Note that those three rows of $\mathbf{R}_{uvw} \mathbf{u}$ are all dot products:

$$\mathbf{R}_{uvw} \mathbf{u} = \begin{bmatrix} \mathbf{u} \cdot \mathbf{u} \\ \mathbf{v} \cdot \mathbf{u} \\ \mathbf{w} \cdot \mathbf{u} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{x}.$$

Similarly, $\mathbf{R}_{uvw} \mathbf{v} = \mathbf{y}$, and $\mathbf{R}_{uvw} \mathbf{w} = \mathbf{z}$. So \mathbf{R}_{uvw} takes the basis uvw to the corresponding Cartesian axes via rotation.

If \mathbf{R}_{uvw} is a rotation matrix with orthonormal rows, then \mathbf{R}_{uvw}^T is also a rotation matrix with orthonormal columns, and in fact is the inverse of \mathbf{R}_{uvw} (the inverse of an orthogonal matrix is always its transpose). An important point is that for transformation matrices, the algebraic inverse is also the geometric inverse. So if \mathbf{R}_{uvw} takes \mathbf{u} to \mathbf{x} , then \mathbf{R}_{uvw}^T takes \mathbf{x} to \mathbf{u} . The same should be true of \mathbf{v} and \mathbf{y} as we can confirm:

$$\mathbf{R}_{uvw}^T \mathbf{y} = \begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} = \mathbf{v}.$$

So we can always create rotation matrices from orthonormal bases.

If we wish to rotate about an arbitrary vector \mathbf{a} , we can form an orthonormal basis with $\mathbf{w} = \mathbf{a}$, rotate that basis to the canonical basis xyz , rotate about the z -axis, and then rotate the canonical basis back to the uvw basis. In matrix form, to rotate about the w -axis by an angle ϕ :

$$\begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}.$$

Here we have \mathbf{w} a unit vector in the direction of \mathbf{a} (i.e., \mathbf{a} divided by its own length). But what are \mathbf{u} and \mathbf{v} ? A method to find reasonable \mathbf{u} and \mathbf{v} is given in Section 2.4.6.

这里, 对于 \mathbf{v} 和 \mathbf{w} , $\mathbf{u}=\mathbf{x}\mathbf{u}+\mathbf{y}\mathbf{u}+\mathbf{z}\mathbf{u}$ 依此类推。由于三个向量是正交的, 我们知道

$$\begin{aligned}\mathbf{u} \cdot \mathbf{u} &= \mathbf{v} \cdot \mathbf{v} = \mathbf{w} \cdot \mathbf{w} = 1, \\ \mathbf{u} \cdot \mathbf{v} &= \mathbf{v} \cdot \mathbf{w} = \mathbf{w} \cdot \mathbf{u} = 0.\end{aligned}$$

我们可以通过将旋转矩阵应用于向量 \mathbf{u} , \mathbf{v} 和 \mathbf{w} 来推断旋转矩阵的一些行为。例如

$$\mathbf{R}_{uvw} \mathbf{u} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix} \begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix} = \begin{bmatrix} x_u x_u + y_u y_u + z_u z_u \\ x_v x_u + y_v y_u + z_v z_u \\ x_w x_u + y_w y_u + z_w z_u \end{bmatrix}.$$

注意 $\mathbf{R}_{uvw} \mathbf{u}$ 那三行都是点积:

$$\mathbf{R}_{uvw} \mathbf{u} = \begin{bmatrix} \mathbf{u} \cdot \mathbf{u} \\ \mathbf{v} \cdot \mathbf{u} \\ \mathbf{w} \cdot \mathbf{u} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{x}.$$

类似地, $\mathbf{R}_{uvw} \mathbf{v} = \mathbf{y}$, 并且 $\mathbf{R}_{uvw} \mathbf{w} = \mathbf{z}$ 。所以 \mathbf{R}_{uvw} 通过旋转将基础 uvw 带到相应的笛卡尔轴。

\mathbf{uvw} 也是具有正交列的rtation矩阵, 并且实际上是 \mathbf{R}_{uvw} 的逆 (正交矩阵的逆总是其转置)。重要的一点是, 对于变换矩阵, 代数逆也是几何逆。所以如果 \mathbf{R}_{uvw} 将 \mathbf{u} 带到 \mathbf{x} , 那么 \mathbf{R}^T

正如我们可以确认的那样:
 \mathbf{uvw} 将 \mathbf{x} 带到 \mathbf{u} 。V和也应该如此

$$\mathbf{R}_{uvw}^T \mathbf{y} = \begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} = \mathbf{v}.$$

所以我们总是可以从正交基创建旋转矩阵。

如果我们希望围绕任意矢量 \mathbf{a} 旋转, 我们可以用 $\mathbf{w}=\mathbf{a}$ 形成正交基, 将该基旋转到规范基 xyz , 绕 z 轴旋转, 然后将规范基旋转回 uvw 基。以矩阵形式, 绕 \mathbf{w} 轴旋转一个角度 ϕ :

$$\begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}.$$

这里我们有一个 \mathbf{wa} 方向的单位向量 (即 \mathbf{a} 除以它自己的长度)。但 \mathbf{u} 和 \mathbf{v} 是什么? 找到合理的 \mathbf{u} 和 \mathbf{v} 的方法在第2.4.6节中给出。

If we have a rotation matrix and we wish to have the rotation in axis-angle form, we can compute the one real eigenvalue (which will be $\lambda = 1$), and the corresponding eigenvector is the axis of rotation. This is the one axis that is not changed by the rotation.

See Chapter 16 for a comparison of the few most-used ways to represent rotations, besides rotation matrices.

6.2.2 Transforming Normal Vectors

While most 3D vectors we use represent positions (offset vectors from the origin) or directions, such as where light comes from, some vectors represent *surface normals*. Surface normal vectors are perpendicular to the tangent plane of a surface. These normals do not transform the way we would like when the underlying surface is transformed. For example, if the points of a surface are transformed by a matrix M , a vector t that is tangent to the surface and is multiplied by M will be tangent to the transformed surface. However, a surface normal vector n that is transformed by M may not be normal to the transformed surface (Figure 6.17).

We can derive a transform matrix N which does take n to a vector perpendicular to the transformed surface. One way to attack this issue is to note that a surface normal vector and a tangent vector are perpendicular, so their dot product is zero, which is expressed in matrix form as

$$n^T t = 0. \quad (6.4)$$

If we denote the desired transformed vectors as $t_M = Mt$ and $n_N = Nn$, our goal is to find N such that $n_N^T t_M = 0$. We can find N by some algebraic

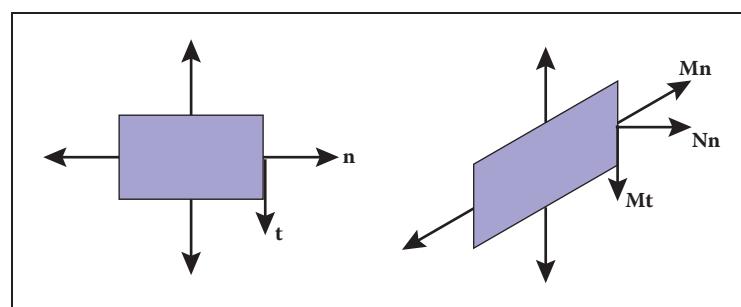


Figure 6.17. When a normal vector is transformed using the same matrix that transforms the points on an object, the resulting vector may not be perpendicular to the surface as is shown here for the sheared rectangle. The tangent vector, however, does transform to a vector tangent to the transformed surface.

如果我们有一个旋转矩阵，并且我们希望以轴角形式旋转，我们可以计算一个实特征值（将是 $\lambda=1$ ），相应的特征向量是旋转轴。这是一个不被旋转改变的轴。

参见第16章，比较了几种最常用的表示旋转矩阵的方法，除了旋转矩阵。

6.2.2 变换法向量

虽然我们使用的大多数3d矢量表示位置（从原点偏移矢量）或方向，例如光线来自哪里，但有些矢量表示表面法线。曲面法向量垂直于曲面的切平面。这些法线不会以我们希望的方式转换基础表面。例如，如果曲面的点由矩阵M变换，则与曲面相切并乘以M的矢量t将与变换后的曲面相切。但是，被M变换的表面法向量n可能对变换后的表面不是法线（图6.17）。我们可以推导出一个变换矩阵N，它将n带到垂直于变换表面的矢量。攻击这个问题的一种方法是注意一个表面法向量和一个切向量是垂直的，所以它们的点积是零，它以矩阵形式表示为

$$n^T t = 0. \quad (6.4)$$

如果我们将所需的变换矢量表示为 $t_M=Mt$ 和 $n_N=Nn$ ，我们的目标是找到N使得 $n^T t_M = 0$ 。NtM=0。我们可以通过一些代数找到N

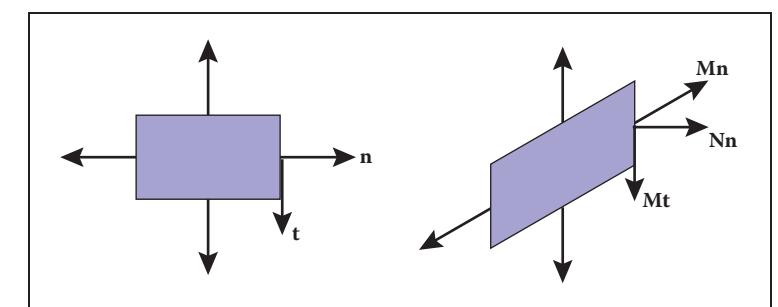


图6.17。当使用变换对象上的点的相同矩阵变换法向量时，生成的向量可能不会垂直于曲面，如剪切矩形所示。但是，切向矢量确实会转换为与已转换曲面相切的矢量。



tricks. First, we can sneak an identity matrix into the dot product, and then take advantage of $M^{-1}M = I$:

$$\mathbf{n}^T \mathbf{t} = \mathbf{n}^T \mathbf{I} \mathbf{t} = \mathbf{n}^T M^{-1} M \mathbf{t} = \mathbf{0}.$$

Although the manipulations above don't obviously get us anywhere, note that we can add parentheses that make the above expression more obviously a dot product:

$$(\mathbf{n}^T M^{-1}) (M \mathbf{t}) = (\mathbf{n}^T M^{-1}) \mathbf{t}_M = \mathbf{0}.$$

This means that the row vector that is perpendicular to \mathbf{t}_M is the left part of the expression above. This expression holds for any of the tangent vectors in the tangent plane. Since there is only one direction in 3D (and its opposite) that is perpendicular to all such tangent vectors, we know that the left part of the expression above must be the row vector expression for \mathbf{n}_N , i.e., it is \mathbf{n}_N^T , so this allows us to infer \mathbf{N} :

$$\mathbf{n}_N^T = \mathbf{n}^T M^{-1},$$

so we can take the transpose of that to get

$$\mathbf{n}_N = (\mathbf{n}^T M^{-1})^T = (M^{-1})^T \mathbf{n}. \quad (6.5)$$

Therefore, we can see that the matrix which correctly transforms normal vectors so they remain normal is $\mathbf{N} = (M^{-1})^T$, i.e., the transpose of the inverse matrix. Since this matrix may change the length of \mathbf{n} , we can multiply it by an arbitrary scalar and it will still produce \mathbf{n}_N with the right direction. Recall from Section 5.3 that the inverse of a matrix is the transpose of the cofactor matrix divided by the determinant. Because we don't care about the length of a normal vector, we can skip the division and find that for a 3×3 matrix,

$$\mathbf{N} = \begin{bmatrix} m_{11}^c & m_{12}^c & m_{13}^c \\ m_{21}^c & m_{22}^c & m_{23}^c \\ m_{31}^c & m_{32}^c & m_{33}^c \end{bmatrix}.$$

This assumes the element of \mathbf{M} in row i and column j is m_{ij} . So the full expression for \mathbf{N} is

$$\mathbf{N} = \begin{bmatrix} m_{22}m_{33} - m_{23}m_{32} & m_{23}m_{31} - m_{21}m_{33} & m_{21}m_{32} - m_{22}m_{31} \\ m_{13}m_{32} - m_{12}m_{33} & m_{11}m_{33} - m_{13}m_{31} & m_{12}m_{31} - m_{11}m_{32} \\ m_{12}m_{23} - m_{13}m_{22} & m_{13}m_{21} - m_{11}m_{23} & m_{11}m_{22} - m_{12}m_{21} \end{bmatrix}.$$



诡计。首先，我们可以将单位矩阵潜入点积中，然后利用 $M^{-1}M=I$ ：

$$\mathbf{n}^T \mathbf{t} = \mathbf{n}^T \mathbf{I} \mathbf{t} = \mathbf{n}^T M^{-1} M \mathbf{t} = \mathbf{0}.$$

虽然上面的操作显然没有让我们在任何地方，但请注意，我们可以添加括号，使上面的表达式更明显地成为点积：

$$\mathbf{n}^T M^{-1} (M \mathbf{t}) = \mathbf{n}^T M^{-1} \mathbf{t}_M = \mathbf{0}.$$

这意味着垂直于 \mathbf{t}_M 的行向量是上面表达式的左侧部分。此表达式适用于切平面中的任何切向量。由于3D中只有一个方向（及其相反）垂直于所有这样的切向量，我们知道上面表达式的左侧部分必须是 nN 的行向量表达式，即它是 n_N^T

N, 所以这个

允许我们推断N:

$$\mathbf{n}_N^T = \mathbf{n}^T M^{-1},$$

所以我们可以把它转置到

$$\mathbf{n}_N = \mathbf{n}^T M^{-1 T} = M^{-1 T} \mathbf{n}. \quad (6.5)$$

因此，我们可以看到正确变换法向量以使它们保持正常的矩阵是 $N = (M^{-1})^T$ ，即逆矩阵的转置。由于该矩阵可能会改变n的长度，我们可以将其乘以任意标量，并且它仍然会产生具有正确方向的 nN 。从第5.3节回想一下，矩阵的逆是辅因子矩阵除以行列式的转置。因为我们不关心法向量的长度，所以我们可以跳过除法，发现对于 3×3 矩阵

$$\mathbf{N} = \begin{bmatrix} m_{11}^c & m_{12}^c & m_{13}^c \\ m_{21}^c & m_{22}^c & m_{23}^c \\ m_{31}^c & m_{32}^c & m_{33}^c \end{bmatrix}.$$

这假设行i和列j中的m的元素是 m_{ij} 。所以N的完整表达式是

$$\mathbf{N} = \begin{bmatrix} m_{22}m_{33} - m_{23}m_{32} & m_{23}m_{31} - m_{21}m_{33} & m_{21}m_{32} - m_{22}m_{31} \\ m_{13}m_{32} - m_{12}m_{33} & m_{11}m_{33} - m_{13}m_{31} & m_{12}m_{31} - m_{11}m_{32} \\ m_{12}m_{23} - m_{13}m_{22} & m_{13}m_{21} - m_{11}m_{23} & m_{11}m_{22} - m_{12}m_{21} \end{bmatrix}.$$

6.3 Translation and Affine Transformations

We have been looking at methods to change vectors using a matrix M . In two dimensions, these transforms have the form,

$$\begin{aligned}x' &= m_{11}x + m_{12}y, \\y' &= m_{21}x + m_{22}y.\end{aligned}$$

We cannot use such transforms to *move* objects, only to scale and rotate them. In particular, the origin $(0, 0)$ always remains fixed under a linear transformation. To move, or *translate*, an object by shifting all its points the same amount, we need a transform of the form,

$$\begin{aligned}x' &= x + x_t, \\y' &= y + y_t.\end{aligned}$$

There is just no way to do that by multiplying (x, y) by a 2×2 matrix. One possibility for adding translation to our system of linear transformations is to simply associate a separate translation vector with each transformation matrix, letting the matrix take care of scaling and rotation and the vector take care of translation. This is perfectly feasible, but the bookkeeping is awkward and the rule for composing two transformations is not as simple and clean as with linear transformations.

Instead, we can use a clever trick to get a single matrix multiplication to do both operations together. The idea is simple: represent the point (x, y) by a 3D vector $[x \ y \ 1]^T$, and use 3×3 matrices of the form

$$\begin{bmatrix}m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1\end{bmatrix}.$$

The fixed third row serves to copy the 1 into the transformed vector, so that all vectors have a 1 in the last place, and the first two rows compute x' and y' as linear combinations of x , y , and 1:

$$\begin{bmatrix}x' \\ y' \\ 1\end{bmatrix} = \begin{bmatrix}m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1\end{bmatrix} \begin{bmatrix}x \\ y \\ 1\end{bmatrix} = \begin{bmatrix}m_{11}x + m_{12}y + x_t \\ m_{21}x + m_{22}y + y_t \\ 1\end{bmatrix}.$$

The single matrix implements a linear transformation followed by a translation! This kind of transformation is called an *affine transformation*, and this way of implementing affine transformations by adding an extra dimension is called *homogeneous coordinates* (Roberts, 1965; Riesenfeld, 1981; Penna & Patterson, 1986). Homogeneous coordinates not only clean up the code for transformations,

6.3 翻译和仿射变换

我们一直在研究使用矩阵M改变向量的方法。在二维中，这些变换具有以下形式

$$\begin{aligned}x' &= m_{11}x + m_{12}y, \\y' &= m_{21}x + m_{22}y.\end{aligned}$$

我们不能使用这种变换来移动对象，只能缩放和旋转它们。特别是，原点(0, 0)在线性变换下始终保持固定。要移动或翻译一个对象，通过移动它的所有点相同的数量，我们需要一个形式的转换

$$\begin{aligned}x' &= x + x_t, \\y' &= y + y_t.\end{aligned}$$

只是没有办法通过将 (x, y) 乘以 2×2 矩阵来做到这一点。将平移添加到我们的线性变换系统的一种可能性是简单地将单独的平移向量与每个变换矩阵相关联，让矩阵负责缩放和旋转，矢量负责平移。这是完全可行的，但簿记是尴尬的，组成两个变换的规则并不像线性变换那样简单和干净。

相反，我们可以使用一个巧妙的技巧来获得单个矩阵乘法来一起完成这两个操作。这个想法很简单：用3D向量 $[xy1]^T$ 表示点 (x, y) ，并使用形式的 3×3 矩阵

$$\begin{bmatrix}m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1\end{bmatrix}.$$

固定的第三行用于将1复制到变换后的向量中，以便所有向量在最后位置都有一个1，并且前两行将 x' 和 y' 计算为 x ， y 和1的线性组合：

$$\begin{bmatrix}x' \\ y' \\ 1\end{bmatrix} = \begin{bmatrix}m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1\end{bmatrix} \begin{bmatrix}x \\ y \\ 1\end{bmatrix} = \begin{bmatrix}m_{11}x + m_{12}y + x_t \\ m_{21}x + m_{22}y + y_t \\ 1\end{bmatrix}.$$

单个矩阵实现线性变换，然后是平移！这种变换称为仿射变换，这种通过添加额外维度来实现仿射变换的方式称为齐次坐标 (Roberts, 1965; Riesenfeld, 1981; Penna & Patterson, 1986)。齐次坐标不仅清理转换的代码



but this scheme also makes it obvious how to compose two affine transformations: simply multiply the matrices.

A problem with this new formalism arises when we need to transform vectors that are not supposed to be positions—they represent directions, or offsets between positions. Vectors that represent directions or offsets should not change when we translate an object. Fortunately, we can arrange for this by setting the third coordinate to zero:

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}.$$

If there is a scaling/rotation transformation in the upper-left 2×2 entries of the matrix, it will apply to the vector, but the translation still multiplies with the zero and is ignored. Furthermore, the zero is copied into the transformed vector, so direction vectors remain direction vectors after they are transformed.

This is exactly the behavior we want for vectors, so they fit smoothly into the system: the extra (third) coordinate will be either 1 or 0 depending on whether we are encoding a position or a direction. We actually do need to store the homogeneous coordinate so we can distinguish between locations and other vectors. For example,

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \text{ is a location and } \begin{bmatrix} 3 \\ 2 \\ 0 \end{bmatrix} \text{ is a displacement or direction.}$$

Later, when we do perspective viewing, we will see that it is useful to allow the homogeneous coordinate to take on values other than one or zero.

Homogeneous coordinates are used nearly universally to represent transformations in graphics systems. In particular, homogeneous coordinates underlie the design and operation of renderers implemented in graphics hardware. We will see in Chapter 7 that homogeneous coordinates also make it easy to draw scenes in perspective, another reason for their popularity.

Homogeneous coordinates can be considered just a clever way to handle the bookkeeping for translation, but there is also a different, geometric interpretation. The key observation is that when we do a 3D shear based on the z -coordinate we get this transform:

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + x_t z \\ y + y_t z \\ z \end{bmatrix}.$$

Note that this almost has the form we want in x and y for a 2D translation, but has a z hanging around that doesn't have a meaning in 2D. Now comes the key

This gives an explanation for the name “homogeneous:” translation, rotation, and scaling of positions and directions all fit into a single system.

Homogeneous coordinates are also ubiquitous in computer vision.



但是这个方案也使得如何组成两个仿射变换变得显而易见：简单地将矩阵相乘。

当我们需要转换不应该是位置的vec时，这种新形式主义就会出现问题—它们代表方向或位置之间的偏移。当我们翻译一个对象时，表示方向或偏移的向量不应该改变。幸运的是，我们可以通过将第三个坐标设置为零来安排这一点：

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}.$$

如果在矩阵的左上 2×2 条目中存在缩放旋转变换，则它将应用于向量，但平移仍然与零相乘并被忽略。此外，零被复制到变换后的矢量中，因此方向矢量在它们被变换后仍然是方向矢量。

这正是我们想要的向量行为，因此它们平滑地适合系统：额外的（第三个）坐标将是1或0，具体取决于我们是编码位置还是方向。我们确实需要储存homogeneous坐标，所以我们可以区分位置和其他向量。例如

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \text{ 是一个位置 and } \begin{bmatrix} 3 \\ 2 \\ 0 \end{bmatrix} \text{ 是位移或方向。}$$

稍后，当我们进行透视查看时，我们会看到允许齐次坐标接受除1或0以外的值是有用的。

均匀坐标几乎普遍用于表示图形系统中的变形映射。特别是，齐次坐标是

图形硬件中实现的渲染器的设计和操作。我们将在第7章中看到，齐次坐标也可以很容易地在透视中绘制场景，这是它们受欢迎的另一个原因。

齐次坐标可以被认为只是一个聪明的方式来处理簿记翻译，但也有一个不同的，几何解释。关键的观察是，当我们基于 z 坐标进行3d剪切时，我们得到了这个变换：

$$\begin{array}{cccc|cc} 1 & 0 & x_t & x & x + x_t z \\ 0 & 1 & y_t & y & y + y_t z \\ 0 & 0 & 1 & z & z \end{array}.$$

请注意，这几乎具有我们想要在 x 和 y 中进行2D翻译的形式，但是有一个 z 在2D中没有任何意义。

这给出了一个解释
名称
neous：“平移、旋转以及位置和方向的缩放都适合于一个系统。”

齐次坐标在计算机视觉中也无处不在。

decision: we will add a coordinate $z = 1$ to all 2D locations. This gives us

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_t \\ y + y_t \\ 1 \end{bmatrix}.$$

By associating a ($z = 1$)-coordinate with all 2D points, we now can encode translations into matrix form. For example, to first translate in 2D by (x_t, y_t) and then rotate by angle ϕ we would use the matrix

$$M = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix}.$$

Note that the 2D rotation matrix is now 3×3 with zeros in the “translation slots.” With this type of formalism, which uses shears along $z = 1$ to encode translations, we can represent any number of 2D shears, 2D rotations, and 2D translations as one composite 3D matrix. The bottom row of that matrix will always be $(0, 0, 1)$, so we don’t really have to store it. We just need to remember it is there when we multiply two matrices together.

In 3D, the same technique works: we can add a fourth coordinate, a homogeneous coordinate, and then we have translations:

$$\begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_t \\ y + y_t \\ z + z_t \\ 1 \end{bmatrix}.$$

Again, for a direction vector, the fourth coordinate is zero and the vector is thus unaffected by translations.

Example (Windowing transformations). Often in graphics we need to create a transform matrix that takes points in the rectangle $[x_l, x_h] \times [y_l, y_h]$ to the rectangle $[x'_l, x'_h] \times [y'_l, y'_h]$. This can be accomplished with a single scale and translate in sequence. However, it is more intuitive to create the transform from a sequence of three operations (Figure 6.18):

1. Move the point (x_l, y_l) to the origin.
2. Scale the rectangle to be the same size as the target rectangle.
3. Move the origin to point (x'_l, y'_l) .

决定：我们将为所有2D位置添加坐标 $z=1$ 。这给了我们

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_t \\ y + y_t \\ 1 \end{bmatrix}.$$

通过将一个($z=1$)坐标与所有2D点相关联，我们现在可以将平移编码为矩阵形式。例如，首先通过 (x_t, y_t) 在2D中平移，然后通过角度 ϕ 旋转，我们将使用矩阵

$$M = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix}.$$

请注意，2D旋转矩阵现在是 3×3 ，“平移槽”中有零。“使用这种使用沿 $z=1$ 的剪切来编码翻译的形式主义，我们可以将任意数量的2D剪切、2D旋转和2D翻译表示为一个复合3D矩阵。该矩阵的底行将始终为 $(0 0 1)$ ，因此我们不必真正存储它。我们只需要记住，当我们两个矩阵相乘时，它就在那里。

在3D中，同样的技术起作用：我们可以添加第四个坐标，一个homogeneous坐标，然后我们有翻译：

$$\begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_t \\ y + y_t \\ z + z_t \\ 1 \end{bmatrix}.$$

同样，对于方向矢量，第四坐标为零，因此矢量不受平移的影响。

示例（窗口转换）。通常在图形中，我们需要创建一个反式矩阵，它将矩形 $[x_l x_h] \times [y_l y_h]$ 中的点带到矩形 $[x' l x_h] \times [y' l y_h]$ 。这可以用一个单一的规模来完成并按顺序翻译。但是，从三个操作序列创建转换更直观（图6.18）：

1. 将点 (x_l, y_l) 移动到原点。
2. 将矩形缩放为与目标矩形大小相同。
3. 将原点移动到点 (x'_l, y'_l) 。

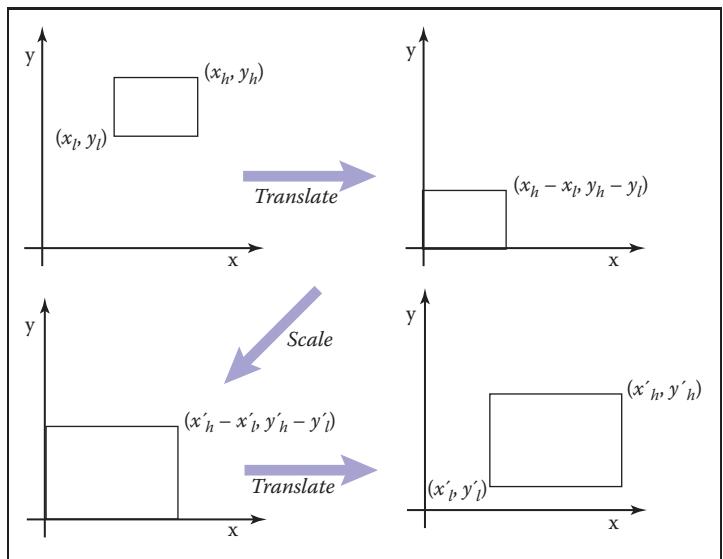


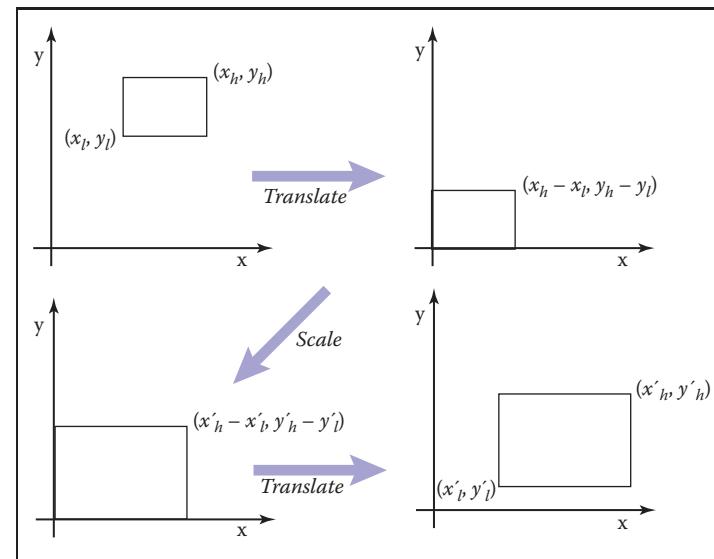
Figure 6.18. To take one rectangle (window) to the other, we first shift the lower-left corner to the origin, then scale it to the new size, and then move the origin to the lower-left corner of the target rectangle.

Remembering that the right-hand matrix is applied first, we can write

$$\begin{aligned} \text{window} &= \text{translate } (x'_l, y'_l) \text{ scale } \left(\frac{x'_h - x'_l}{x_h - x_l}, \frac{y'_h - y'_l}{y_h - y_l} \right) \text{ translate } (-x_l, -y_l) \\ &= \begin{bmatrix} 1 & 0 & x'_l \\ 0 & 1 & y'_l \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix}. \quad (6.6) \end{aligned}$$

It is perhaps not surprising to some readers that the resulting matrix has the form it does, but the constructive process with the three matrices leaves no doubt as to the correctness of the result.

An exactly analogous construction can be used to define a 3D windowing transformation, which maps the box $[x_l, x_h] \times [y_l, y_h] \times [z_l, z_h]$ to the box



要将一个矩形（窗口）移到另一个矩形（窗口），我们首先将左下角移动到原点，然后将其缩放到新的大小，然后将原点移动到目标矩形的左下角。

记住首先应用右手矩阵，我们可以写

$$\begin{aligned} \text{window} &= \text{translate } (x'_l, y'_l) \text{ scale } \left(\frac{x'_h - x'_l}{x_h - x_l}, \frac{y'_h - y'_l}{y_h - y_l} \right) \text{ translate } (-x_l, -y_l) \\ &= \begin{bmatrix} 1 & 0 & x'_l \\ 0 & 1 & y'_l \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix}. \quad (6.6) \end{aligned}$$

对于一些读者来说，所得到的矩阵具有它所做的形式可能并不奇怪，但是三个矩阵的建设性过程对结果的正确性毫不怀疑。

一个完全类似的结构可以用来定义一个3D窗口转换，它将盒子 $[x_l, x_h] \times [y_l, y_h] \times [z_l, z_h]$ 映射到盒子

$[x'_l, x'_h] \times [y'_l, y'_h] \times [z'_l, z'_h]$:

$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & \frac{z'_h - z'_l}{z_h - z_l} & \frac{z'_l z_h - z'_h z_l}{z_h - z_l} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (6.7)$$

It is interesting to note that if we multiply an arbitrary matrix composed of scales, shears, and rotations with a simple translation (translation comes second), we get

$$\begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & x_t \\ a_{21} & a_{22} & a_{23} & y_t \\ a_{31} & a_{32} & a_{33} & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Thus, we can look at any matrix and think of it as a scaling/rotation part and a translation part because the components are nicely separated from each other.

An important class of transforms are *rigid-body* transforms. These are composed only of translations and rotations, so they have no stretching or shrinking of the objects. Such transforms will have a pure rotation for the a_{ij} above.

6.4 Inverses of Transformation Matrices

While we can always invert a matrix algebraically, we can use geometry if we know what the transform does. For example, the inverse of $\text{scale}(s_x, s_y, s_z)$ is $\text{scale}(1/s_x, 1/s_y, 1/s_z)$. The inverse of a rotation is the same rotation with the opposite sign on the angle. The inverse of a translation is a translation in the opposite direction. If we have a series of matrices $M = M_1 M_2 \cdots M_n$ then $M^{-1} = M_n^{-1} \cdots M_2^{-1} M_1^{-1}$.

Also, certain types of transformation matrices are easy to invert. We've already mentioned scales, which are diagonal matrices; the second important example is rotations, which are orthogonal matrices. Recall (Section 5.2.4) that the inverse of an orthogonal matrix is its transpose. This makes it easy to invert rotations and rigid body transformations (see Exercise 6). Also, it's useful to know that a matrix with $[0 \ 0 \ 0 \ 1]$ in the bottom row has an inverse that also has $[0 \ 0 \ 0 \ 1]$ in the bottom row (see Exercise 7).

Interestingly, we can use SVD to invert a matrix as well. Since we know that any matrix can be decomposed into a rotation times a scale times a rotation,

$[x'_l, x'_h] \times [y'_l, y'_h] \times [z'_l, z'_h]$:

$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & \frac{z'_h - z'_l}{z_h - z_l} & \frac{z'_l z_h - z'_h z_l}{z_h - z_l} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (6.7)$$

有趣的是，如果我们将由尺度，剪切和旋转组成的任意矩阵乘以简单的平移（平移排在第二位），我们得到

$$\begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & x_t \\ a_{21} & a_{22} & a_{23} & y_t \\ a_{31} & a_{32} & a_{33} & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

因此，我们可以查看任何矩阵，并将其视为缩放旋转部分和平移部分，因为组件彼此很好地分离。

一类重要的变换是刚体变换。这些仅由平移和旋转构成，因此它们没有对象的拉伸或收缩。这样的变换对于上面的 a_{ij} 将具有纯旋转。

6.4 变换矩阵的反转

虽然我们总是可以代数反转矩阵，但如果我们知道变换的作用，我们可以使用几何。例如， $\text{scale}(sx \ sy \ sz)$ 的倒数是 $\text{scale}(1/sx \ 1/sy \ 1/sz)$ 。旋转的倒数是与角度上的相反符号相同的旋转。平移的倒数是相反方向的平移。如果我们有一系列矩阵 $M = M_1 M_2 \cdots M_n$ 则 $M^{-1} = M_n^{-1} \cdots M_2^{-1} M_1^{-1}$

此外，某些类型的变换矩阵易于反转。我们已经提到了尺度，这是对角矩阵；第二个重要的例子是旋转，这是正交矩阵。回想一下（第5.2.4节）正交矩阵的逆是它的转置。这使得反演和刚体变换变得容易（见练习6）。此外，知道在底行中具有 $[0 \ 0 \ 0 \ 1]$ 的矩阵具有在底行中也具有 $[0 \ 0 \ 0 \ 1]$ 的逆是有用的（见练习7）。

有趣的是，我们也可以使用SVD反转矩阵。因为我们知道任何矩阵都可以分解成一个旋转乘以一个刻度乘以一个旋转

inversion is straightforward. For example, in 3D we have

$$\mathbf{M} = \mathbf{R}_1 \text{scale}(\sigma_1, \sigma_2, \sigma_3) \mathbf{R}_2,$$

and from the rules above it follows easily that

$$\mathbf{M}^{-1} = \mathbf{R}_2^T \text{scale}(1/\sigma_1, 1/\sigma_2, 1/\sigma_3) \mathbf{R}_1^T.$$

6.5 Coordinate Transformations

All of the previous discussion has been in terms of using transformation matrices to move points around. We can also think of them as simply changing the coordinate system in which the point is represented. For example, in Figure 6.19, we see two ways to visualize a movement. In different contexts, either interpretation may be more suitable.

For example, a driving game may have a model of a city and a model of a car. If the player is presented with a view out the windshield, objects inside the car are always drawn in the same place on the screen, while the streets and buildings appear to move backward as the player drives. On each frame, we apply a transformation to these objects that moves them farther back than on the previous frame. One way to think of this operation is simply that it moves the buildings backward; another way to think of it is that the buildings are staying put but the coordinate system in which we want to draw them—which is attached to the car—is moving. In the second interpretation, the transformation is changing

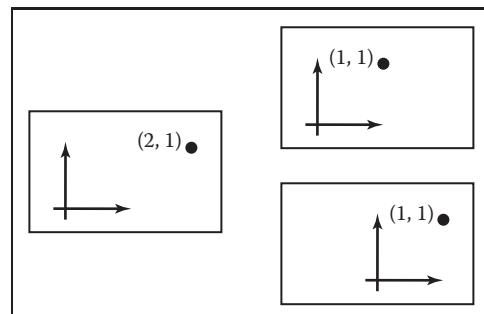


Figure 6.19. The point $(2,1)$ has a transform “translate by $(-1,0)$ ” applied to it. On the top right is our mental image if we view this transformation as a physical movement, and on the bottom right is our mental image if we view it as a change of coordinates (a movement of the origin in this case). The artificial boundary is just an artifice, and the relative position of the axes and the point are the same in either case.

反转是直截了当的。例如，在3D中，我们有

$$\mathbf{M} = \mathbf{R}_1 \text{scale}(\sigma_1, \sigma_2, \sigma_3) \mathbf{R}_2,$$

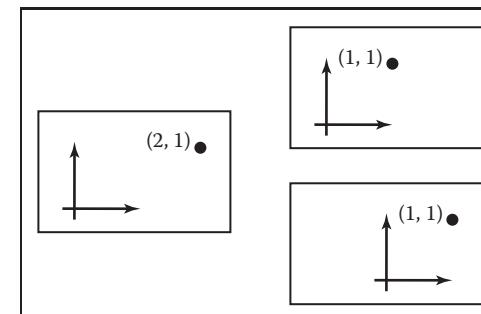
从上面的规则可以很容易地看出

$$\mathbf{M}^{-1} = \mathbf{R}_2^T \text{scale}(1/\sigma_1, 1/\sigma_2, 1/\sigma_3) \mathbf{R}_1^T.$$

6.5 坐标变换

前面所有的讨论都是关于使用变换矩阵来移动点。我们也可以把它们看作是简单地改变表示点的coordinate系统。例如，在图6.19中，我们看到了两种可视化运动的方法。在不同的上下文中，任一解释可能更合适。

例如，驾驶游戏可以具有城市的模型和汽车的模型。如果玩家看到挡风玻璃外的视图，汽车内的物体总是被画在屏幕上的同一个地方，而街道和建筑物似乎在玩家驾驶时向后移动。在每个帧上，我们对这些对象应用变换，使它们比前一帧向后移动得更远。考虑这种操作的一种方法是简单地将建筑物向后移动；另一种考虑它的方法是建筑物保持不变，但我们想要绘制它们的坐标系—连接到汽车—正在移动。在第二种解释中，转变正在发生变化



点 $(2,1)$ 应用了一个变换“translateby $(-1,0)$ ”。右上角是我们的心理形象，如果我们把这个转变看作是一个物理运动，右下角是我们的心理形象，如果我们把它看作一个坐标的变化（在这种情况下是原点的运动）。人工边界只是一种手段，轴和点的相对位置在任何一种情况下都是相同的。

the coordinates of the city geometry, expressing them as coordinates in the car's coordinate system. Both ways will lead to exactly the same matrix that is applied to the geometry outside the car.

If the game also supports an overhead view to show where the car is in the city, the buildings and streets need to be drawn in fixed positions while the car needs to move from frame to frame. The same two interpretations apply: we can think of the changing transformation as moving the car from its canonical position to its current location in the world; or we can think of the transformation as simply changing the coordinates of the car's geometry, which is originally expressed in terms of a coordinate system attached to the car, to express them instead in a coordinate system fixed relative to the city. The change-of-coordinates interpretation makes it clear that the matrices used in these two modes (city-to-car coordinate change vs. car-to-city coordinate change) are inverses of one another.

The idea of changing coordinate systems is much like the idea of type conversions in programming. Before we can add a floating-point number to an integer, we need to convert the integer to floating point or the floating-point number to an integer, depending on our needs, so that the types match. And before we can draw the city and the car together, we need to convert the city to car coordinates or the car to city coordinates, depending on our needs, so that the coordinates match.

When managing multiple coordinate systems, it's easy to get confused and wind up with objects in the wrong coordinates, causing them to show up in unexpected places. But with systematic thinking about transformations between coordinate systems, you can reliably get the transformations right.

Geometrically, a coordinate system, or coordinate *frame*, consists of an origin and a basis—a set of three vectors. Orthonormal bases are so convenient that we'll normally assume frames are orthonormal unless otherwise specified. In a frame with origin \mathbf{p} and basis $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$, the coordinates (u, v, w) describe the point

$$\mathbf{p} + uu + vv + ww.$$

When we store these vectors in the computer, they need to be represented in terms of some coordinate system. To get things started, we have to designate some canonical coordinate system, often called “global” or “world” coordinates, which is used to describe all other systems. In the city example, we might adopt the street grid and use the convention that the x -axis points along Main Street, the y -axis points up, and the z -axis points along Central Avenue. Then when we write the origin and basis of the car frame in terms of these coordinates it is clear what we mean.

In 2D our convention is to use the point \mathbf{o} for the origin, and \mathbf{x} and \mathbf{y} for the right-handed orthonormal basis vectors \mathbf{x} and \mathbf{y} (Figure 6.20).

In 2D, of course, there are two basis vectors.

In 2D, right-handed means \mathbf{y} is counterclockwise from \mathbf{x} .

城市几何的坐标，将其表示为汽车坐标系中的坐标。这两种方式都会导致应用于车外几何形状的完全相同的矩阵。

如果游戏还支持架空视图以显示汽车在城市中的位置，则需要在固定位置绘制建筑物和街道，而汽车需要从一个框架移动到另一个框架。同样的两种解释也适用：我们可以把变化的变换看作是将汽车从它的规范位置移动到它在世界上的当前位置；或者我们可以把变换看作是简单地改变汽车几何的坐标，它最初是用附加在汽车上的坐标系来表达的，而不是用相对于城市固定的坐标系来表达它们。坐标变化的解释清楚地表明，在这两种模式中使用的矩阵（城市到汽车坐标变化与汽车到城市坐标变化）是相互颠倒的。改变坐标系的想法很像编程中类型conversions的想法。在将浮点数添加到整数之前，我们需要根据需要将整数转换为浮点数或浮点数转换为整数，以便类型匹配。在我们绘制城市和汽车之前，我们需要根据我们的需要将城市转换为汽车坐标或汽车转换为城市坐标，以便坐标匹配。

当管理多个坐标系时，很容易混淆，并在错误的坐标中结束对象，导致它们显示在不预期的地方。但是，通过系统地思考坐标系之间的转换，您可以可靠地正确地获得转换。

几何上，坐标系或坐标系由一个原点和一个基础组成—一组三个矢量。Orthonormal 基地是如此方便，除非另有规定，我们通常会假设帧是正交的。在原点 \mathbf{p} 和基 $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ 的帧中，坐标 (u, v, w) 描述了点

$$\mathbf{p} + uu + vv + ww.$$

当我们把这些向量存储在计算机中时，它们需要用某种坐标系来表示。为了开始工作，我们必须指定一些规范坐标系，通常称为“全局”或“世界”坐标，用于描述所有其他系统。在城市示例中，我们可以采用街道网格，并使用 x 轴沿主街指向， y 轴向上指向， z 轴沿中央大道指向的约定。然后，当我们根据这些坐标编写汽车框架的原点和基础时，很明显我们的意思。

在2D中，右手意味着 \mathbf{y} 从 \mathbf{x} 逆时针。

在二维中，我们的约定是使用点 \mathbf{o} 作为原点， \mathbf{x} 和 \mathbf{y} 作为原点。右旋正交基向量 \mathbf{x} 和 \mathbf{y} （图6.20）。

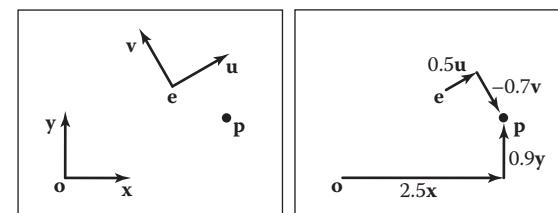


Figure 6.20. The point \mathbf{p} can be represented in terms of either coordinate system.

Another coordinate system might have an origin \mathbf{e} and right-handed orthonormal basis vectors \mathbf{u} and \mathbf{v} . Note that typically the canonical data \mathbf{o} , \mathbf{x} , and \mathbf{y} are never stored explicitly. They are the frame-of-reference for all other coordinate systems. In that coordinate system, we often write down the location of \mathbf{p} as an ordered pair, which is shorthand for a full vector expression:

$$\mathbf{p} = (x_p, y_p) \equiv \mathbf{o} + x_p \mathbf{x} + y_p \mathbf{y}.$$

For example, in Figure 6.20, $(x_p, y_p) = (2.5, 0.9)$. Note that the pair (x_p, y_p) implicitly assumes the origin \mathbf{o} . Similarly, we can express \mathbf{p} in terms of another equation:

$$\mathbf{p} = (u_p, v_p) \equiv \mathbf{e} + u_p \mathbf{u} + v_p \mathbf{v}.$$

In Figure 6.20, this has $(u_p, v_p) = (0.5, -0.7)$. Again, the origin \mathbf{e} is left as an implicit part of the coordinate system associated with \mathbf{u} and \mathbf{v} .

We can express this same relationship using matrix machinery, like this:

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_e \\ 0 & 1 & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & 0 \\ y_u & y_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & x_v & x_e \\ y_u & y_v & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix}.$$

Note that this assumes we have the point \mathbf{e} and vectors \mathbf{u} and \mathbf{v} stored in canonical coordinates; the (x, y) -coordinate system is the first among equals. In terms of the basic types of transformations we've discussed in this chapter, this is a rotation (involving \mathbf{u} and \mathbf{v}) followed by a translation (involving \mathbf{e}). Looking at the matrix for the rotation and translation together, you can see it's very easy to write down: we just put \mathbf{u} , \mathbf{v} , and \mathbf{e} into the columns of a matrix, with the usual $[0 \ 1]$ in the third row. To make this even clearer we can write the matrix like this:

$$\mathbf{p}_{xy} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{uv}.$$

We call this matrix the *frame-to-canonical* matrix for the (u, v) frame. It takes points expressed in the (u, v) frame and converts them to the same points expressed in the canonical frame.

The name “frame-to-canonical” is based on thinking about changing the coordinates of a vector from one system to another. Thinking in terms of moving vectors around, the frame-to-canonical matrix maps the canonical frame to the (u, v) frame.

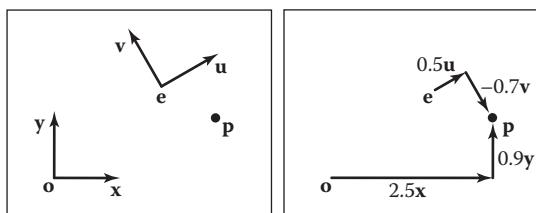


图6.20。点p可以用任一坐标系表示。

另一个坐标系可能具有原点 \mathbf{e} 和右旋正交基向量 \mathbf{u} 和 \mathbf{v} 。请注意，通常规范数据 \mathbf{o} 、 \mathbf{x} 和 \mathbf{y} 永远不会显式存储。它们是所有其他坐标系的参照系。在该坐标系中，我们经常将 \mathbf{p} 的位置记作有序对，这是全向量表达式的简写：

$$\mathbf{p} = (x_p, y_p) \equiv \mathbf{o} + x_p \mathbf{x} + y_p \mathbf{y}.$$

例如，在图6.20中， $(x_p, y_p) = (2.5, 0.9)$ 。请注意，对 (x_p, y_p) 隐式假设原点 \mathbf{o} 。同样，我们可以用另一个方程来表达 \mathbf{p} :

$$\mathbf{p} = (u_p, v_p) \equiv \mathbf{e} + u_p \mathbf{u} + v_p \mathbf{v}.$$

在图6.20中，这具有 $(u_p, v_p) = (0.5, -0.7)$ 。再次，原点 \mathbf{e} 被留下作为与 \mathbf{u} 和 \mathbf{v} 相关的坐标系的隐含部分。

我们可以用矩阵机械来表达同样的关系，就像这样：

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_e \\ 0 & 1 & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & 0 \\ y_u & y_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & x_v & x_e \\ y_u & y_v & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix}.$$

请注意，这假设我们有点 \mathbf{e} 和向量 \mathbf{u} 和 \mathbf{v} 存储在规范坐标中； (x, y) -坐标系是equals中的第一个。就我们在本章中讨论的转换的基本类型而言，这是一个旋转（涉及 \mathbf{u} 和 \mathbf{v} ），然后是一个平移（涉及 \mathbf{e} ）。看看矩阵的旋转和平移在一起，你可以看到它很容易写下来：我们只是把 \mathbf{u} ， \mathbf{v} 和 \mathbf{e} 放入矩阵的列中，在第三行中使用通常的[0 0 1]。为了更清楚，我们可以这样写矩阵：

$$\mathbf{p}_{xy} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{uv}.$$

我们将此矩阵称为 (u, v) 帧的帧到规范矩阵。它采用在 (u, v) 帧中表示的点，并将它们转换为在规范帧中表示的相同点。

The name “frame-to-canonical” is based on thinking about changing from one system to another system's vectors. From the perspective of moving vectors, the frame-to-canonical matrix maps the canonical frame to the (u, v) frame.

帧到规范矩阵将规范帧映射到 (u, v) 帧。

To go in the other direction we have

$$\begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & y_u & 0 \\ x_v & y_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_e \\ 0 & 1 & -y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix}.$$

This is a translation followed by a rotation; they are the inverses of the rotation and translation we used to build the frame-to-canonical matrix, and when multiplied together they produce the inverse of the frame-to-canonical matrix, which is (not surprisingly) called the canonical-to-frame matrix:

$$\mathbf{p}_{uv} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{p}_{xy}.$$

The canonical-to-frame matrix takes points expressed in the canonical frame and converts them to the same points expressed in the (u,v) frame. We have written this matrix as the inverse of the frame-to-canonical matrix because it can't immediately be written down using the canonical coordinates of \mathbf{e} , \mathbf{u} , and \mathbf{v} . But remember that all coordinate systems are equivalent; it's only our convention of storing vectors in terms of x - and y -coordinates that creates this seeming asymmetry. The canonical-to-frame matrix *can* be expressed simply in terms of the (u, v) coordinates of \mathbf{o} , \mathbf{x} , and \mathbf{y} :

$$\mathbf{p}_{uv} = \begin{bmatrix} \mathbf{x}_{uv} & \mathbf{y}_{uv} & \mathbf{o}_{uv} \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{xy}.$$

All these ideas work strictly analogously in 3D, where we have

$$\begin{aligned} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & x_e \\ 0 & 1 & 0 & y_e \\ 0 & 0 & 1 & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & x_w & 0 \\ y_u & y_v & y_w & 0 \\ z_u & z_v & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ w_p \\ 1 \end{bmatrix} \\ \mathbf{p}_{xyz} &= \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{uvw}, \end{aligned} \quad (6.8)$$

and

$$\begin{aligned} \begin{bmatrix} u_p \\ v_p \\ w_p \\ 1 \end{bmatrix} &= \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \\ \mathbf{p}_{uvw} &= \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{p}_{xyz}. \end{aligned} \quad (6.9)$$

去我们的另一个方向

$$\begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & y_u & 0 \\ x_v & y_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_e \\ 0 & 1 & -y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix}.$$

这是一个翻译，然后是一个旋转；它们是我们用来构建框架到规范矩阵的旋转和平移的反转，当它们相乘时，它们产生了框架到规范矩阵的反转，这（毫不奇怪）被称为规范到框架矩阵：

$$\mathbf{p}_{uv} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{p}_{xy}.$$

规范到帧矩阵采用在规范帧中表示的点，并将它们转换为在 (u, v) 帧中表示的相同点。我们已经将这个矩阵写成帧到规范矩阵的倒数，因为它不能立即使用 \mathbf{e} , \mathbf{u} 和 \mathbf{v} 的规范坐标写下来。但请记住，所有坐标系都是等价的；只有我们按照 x 和 y 坐标存储矢量的约定才会产生这种看起来不对称的情况。规范到帧矩阵可以简单地用 \mathbf{o} , \mathbf{x} 和 \mathbf{y} 的 (u, v) 坐标表示：

$$\mathbf{p}_{uv} = \begin{bmatrix} \mathbf{x}_{uv} & \mathbf{y}_{uv} & \mathbf{o}_{uv} \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{xy}.$$

所有这些想法在3D中严格类似地工作，我们有

$$\begin{aligned} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & x_e \\ 0 & 1 & 0 & y_e \\ 0 & 0 & 1 & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & x_w & 0 \\ y_u & y_v & y_w & 0 \\ z_u & z_v & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ w_p \\ 1 \end{bmatrix} \\ \mathbf{p}_{xyz} &= \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{uvw}, \end{aligned} \quad (6.8)$$

and

$$\begin{aligned} \begin{bmatrix} u_p \\ v_p \\ w_p \\ 1 \end{bmatrix} &= \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \\ \mathbf{p}_{uvw} &= \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{p}_{xyz}. \end{aligned} \quad (6.9)$$



Frequently Asked Questions

- Can't I just hardcode transforms rather than use the matrix formalisms?

Yes, but in practice it is harder to derive, harder to debug, and not any more efficient. Also, all current graphics APIs use this matrix formalism so it must be understood even to use graphics libraries.

- The bottom row of the matrix is always $(0,0,0,1)$. Do I have to store it?

You do not have to store it unless you include perspective transforms (Chapter 7).

Notes

The derivation of the transformation properties of normals is based on *Properties of Surface Normal Transformations* (Turkowski, 1990). In many treatments through the mid-1990s, vectors were represented as row vectors and premultiplied, e.g., $\mathbf{b} = \mathbf{aM}$. In our notation this would be $\mathbf{b}^T = \mathbf{a}^T \mathbf{M}^T$. If you want to find a rotation matrix \mathbf{R} that takes one vector \mathbf{a} to a vector \mathbf{b} of the same length: $\mathbf{b} = \mathbf{Ra}$ you could use two rotations constructed from orthonormal bases. A more efficient method is given in *Efficiently Building a Matrix to Rotate One Vector to Another* (Akenine-Möller, Haines, & Hoffman, 2008).

Exercises

1. Write down the 4×4 3D matrix to move by (x_m, y_m, z_m) .
2. Write down the 4×4 3D matrix to rotate by an angle θ about the y -axis.
3. Write down the 4×4 3D matrix to scale an object by 50% in all directions.
4. Write the 2D rotation matrix that rotates by 90 degrees clockwise.
5. Write the matrix from Exercise 4 as a product of three shear matrices.
6. Find the inverse of the rigid body transformation:

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where \mathbf{R} is a 3×3 rotation matrix and \mathbf{t} is a 3-vector.



常见问题

- 我不能只是硬编码变换而不是使用矩阵形式化吗？

是的，但在实践中，它更难派生，更难调试，而且效率更高。此外，所有当前的图形API都使用这种矩阵形式主义，因此即使使用图形库也必须理解。

- *矩阵的底行始终为 $(0\ 0\ 0\ 1)$ 。我必须存储它吗？

除非包含透视变换（第7章），否则不必存储它。

Notes

法线的变换属性的推导是基于表面法线变换的属性 (Turkowski, 1990)。在20世纪90年代中期的许多治疗中，向量表示为行向量并预乘，例如 $\mathbf{b} = \mathbf{aM}$ 。在我们的符号中，这将是 $\mathbf{b}^T = \mathbf{a}^T \mathbf{M}^T$ 。如果你想要找到一个旋转矩阵 \mathbf{R} ，它将一个向量 \mathbf{a} 带到一个相同长度的向量 \mathbf{b} : $\mathbf{b} = \mathbf{Ra}$ ，你可以使用两个由正交基构造的旋转。在高效构建矩阵以将一个向量旋转到另一个向量中给出了更有效的方法 (Akenine-Möller, Haines, & Hoffman, 2008)。

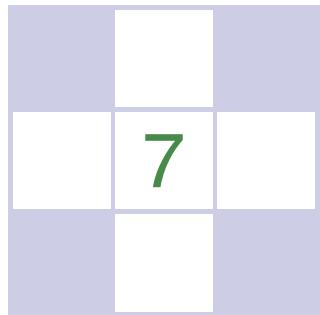
Exercises

1. 记下要移动的 4×4 3d矩阵 (xm, ym, zm) 。
2. 记下 4×4 3D矩阵以绕 y 轴旋转一个角度 θ 。
3. 记下 4×4 的3d矩阵，以在所有方向上将对象缩放50%。
4. 写入顺时针旋转90度的2D旋转矩阵。
5. 将练习4中的矩阵写为三个剪切矩阵的乘积。
6. 求刚体变换的逆： τR

其中 R 是 3×3 旋转矩阵， t 是3矢量。

7. Show that the inverse of the matrix for an affine transformation (one that has all zeros in the bottom row except for a one in the lower right entry) also has the same form.
8. Describe in words what this 2D transform matrix does:
$$\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$
9. Write down the 3×3 matrix that rotates a 2D point by angle θ about a point $\mathbf{p} = (x_p, y_p)$.
10. Write down the 4×4 rotation matrix that takes the orthonormal 3D vectors $\mathbf{u} = (x_u, y_u, z_u)$, $\mathbf{v} = (x_v, y_v, z_v)$, and $\mathbf{w} = (x_w, y_w, z_w)$, to orthonormal 3D vectors $\mathbf{a} = (x_a, y_a, z_a)$, $\mathbf{b} = (x_b, y_b, z_b)$, and $\mathbf{c} = (x_c, y_c, z_c)$. So $M\mathbf{u} = \mathbf{a}$, $M\mathbf{v} = \mathbf{b}$, and $M\mathbf{w} = \mathbf{c}$.
11. What is the inverse matrix for the answer to the previous problem?

- 7.显示仿射变换的矩阵的逆（除了右下条目中的一个之外，底部行中具有所有零的矩阵）也具有相同的形式。
- 8.用文字描述这个2D变换矩阵的作用:
$$\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$
- 9.记下关于一个点 $\mathbf{p}= (x_p, y_p)$ 按角度 θ 旋转一个2D点的 3×3 矩阵。
- 10.记下取正交3d向量 $\mathbf{u}= (x_u, y_u, z_u)$, $\mathbf{v}= (x_v, y_v, z_v)$ 和 $\mathbf{w}= (x_w, y_w, z_w)$ 的 4×4 旋转矩阵，以正交3d向量 $\mathbf{a}= (x_a, y_a, z_a)$, $\mathbf{b}= (x_b, y_b, z_b)$ 和 $\mathbf{c}= (x_c, y_c,$
- 11.前一个问题的答案的逆矩阵是什么？



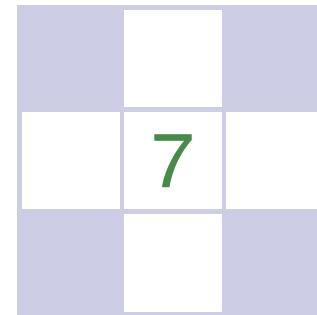
Viewing

In the previous chapter, we saw how to use matrix transformations as a tool for arranging geometric objects in 2D or 3D space. A second important use of geometric transformations is in moving objects between their 3D locations and their positions in a 2D view of the 3D world. This 3D to 2D mapping is called a *viewing transformation*, and it plays an important role in object-order rendering, in which we need to rapidly find the image-space location of each object in the scene.

When we studied ray tracing in Chapter 4, we covered the different types of perspective and orthographic views and how to generate viewing rays according to any given view. This chapter is about the inverse of that process. Here we explain how to use matrix transformations to express any parallel or perspective view. The transformations in this chapter project 3D points in the scene (world space) to 2D points in the image (image space), and they will project any point on a given pixel's viewing ray back to that pixel's position in image space.

If you have not looked at it recently, it is advisable to review the discussion of perspective and ray generation in Chapter 4 before reading this chapter.

By itself, the ability to project points from the world to the image is only good for producing *wireframe* renderings—renderings in which only the edges of objects are drawn, and closer surfaces do not occlude more distant surfaces (Figure 7.1). Just as a ray tracer needs to find the closest surface intersection along each viewing ray, an object-order renderer displaying solid-looking objects has to work out which of the (possibly many) surfaces drawn at any given point on the screen is closest and display only that one. In this chapter, we assume we



Viewing

在上一章中，我们看到了如何使用矩阵变换作为在2D或3D空间中排列几何对象的工具。地理度量转换的第二个重要用途是在3d世界的2D视图中的3d位置和位置之间移动对象。这种3D到2D映射称为观看变换，它在对象顺序渲染中起着重要作用，在这种渲染中我们需要快速找到场景中每个对象的图像空间位置。

当我们在第4章中研究光线追踪时，我们介绍了不同类型的透视和正交视图，以及如何根据任何给定视图生成观察光线。这一章是关于这个过程的反面。在这里，我们解释如何使用矩阵变换来表达任何并行或透视图。本章中的变换将场景（世界空间）中的3D点投影到图像（图像空间）中的2D点，它们将给定像素的查看射线上的任何点投影回该像素在图像空间中的位

如果您最近没有看过，在阅读本章之前，最好先回顾第4章中透视和射线生成的讨论。

就其本身而言，将点从世界投影到图像的能力仅适用于生成线框渲染-渲染，其中仅绘制对象的边缘，而更近的表面不会遮挡更远的表面（图7.1）。就像光线追踪器需要沿着每个观察光线找到最近的表面交叉点一样，显示实体对象的对象顺序渲染器必须计算出在屏幕上任何给定点绘制的（可能是在本章中，我们假设我们

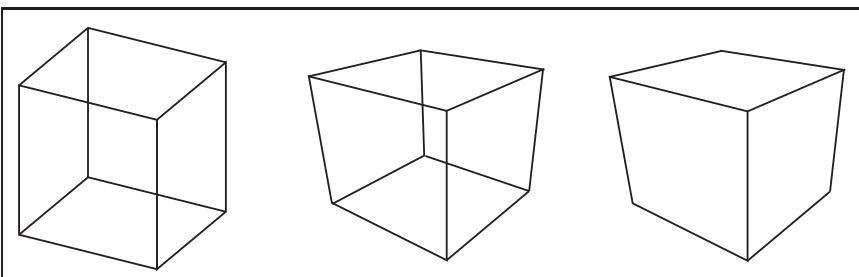


Figure 7.1. Left: wireframe cube in orthographic projection. Middle: wireframe cube in perspective projection. Right: perspective projection with hidden lines removed.

are drawing a model consisting only of 3D line segments that are specified by the (x, y, z) coordinates of their two endpoints. Later chapters will discuss the machinery needed to produce renderings of solid surfaces.

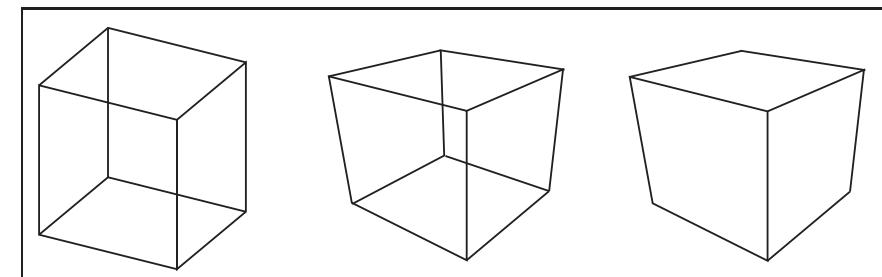
7.1 Viewing Transformations

Some APIs use “viewing transformation” for just the piece of our viewing transformation that we call the camera transformation.

The viewing transformation has the job of mapping 3D locations, represented as (x, y, z) coordinates in the canonical coordinate system, to coordinates in the image, expressed in units of pixels. It is a complicated beast that depends on many different things, including the camera position and orientation, the type of projection, the field of view, and the resolution of the image. As with all complicated transformations it is best approached by breaking it up into a product of several simpler transformations. Most graphics systems do this by using a sequence of three transformations:

- A *camera transformation* or *eye transformation*, which is a rigid body transformation that places the camera at the origin in a convenient orientation. It depends only on the position and orientation, or *pose*, of the camera.
- A *projection transformation*, which projects points from camera space so that all visible points fall in the range -1 to 1 in x and y . It depends only on the type of projection desired.
- A *viewport transformation* or *windowing transformation*, which maps this unit image rectangle to the desired rectangle in pixel coordinates. It depends only on the size and position of the output image.

To make it easy to describe the stages of the process (Figure 7.2), we give names to the coordinate systems that are the inputs and output of these transformations.



左图：正投影中的线框立方体。中间：透视投影中的线框立方体。右：删除隐藏线的透视投影。

绘制仅由3D线段组成的模型，这些线段由其两个端点的 (x, y, z) 坐标指定。后面的章节将讨论制作实体表面渲染所需的机械。

7.1 查看变换

一些API只使用“查看变换”
“作为我们称之为相机变换的查看变换的一部分”。

查看变换的工作是将3d位置（表示为规范坐标系中的 (x, y, z) 坐标）映射到图像中的坐标（以像素为单位表示）。这是一个复杂的野兽，取决于

许多不同的东西，包括相机的位置和方向，投影的类型，视野和图像的分辨率。与所有复杂的转换一样，最好将其分解为几个更简单的转换的产物。大多数图形系统通过使用三个转换序列来执行此操作：

*相机变换或眼睛变换，这是一个刚体变换，将相机放置在一个方便的方向的原点。它仅取决于摄像机的位置和方向或姿势。

*投影变换，它从摄像机空间投影点，使所有可见点都落在x和y中的1到1范围内。它仅取决于所需的投影类型。

*视口变换或窗口化变换，它以像素坐标将此单位图像矩形映射到所需的矩形。它仅取决于输出图像的大小和位置。

为了便于描述过程的各个阶段（图7.2），我们给作为这些转换的输入和输出的坐标系命名。

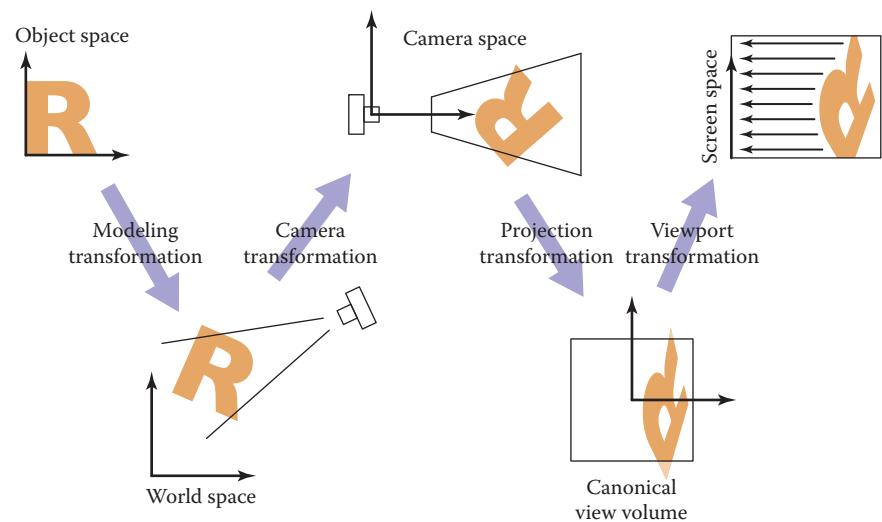


Figure 7.2. The sequence of spaces and transformations that gets objects from their original coordinates into screen space.

The camera transformation converts points in canonical coordinates (or world space) to *camera coordinates* or places them in *camera space*. The projection transformation moves points from camera space to the *canonical view volume*. Finally, the viewport transformation maps the canonical view volume to *screen space*.

Each of these transformations is individually quite simple. We'll discuss them in detail for the orthographic case beginning with the viewport transformation, then cover the changes required to support perspective projection.

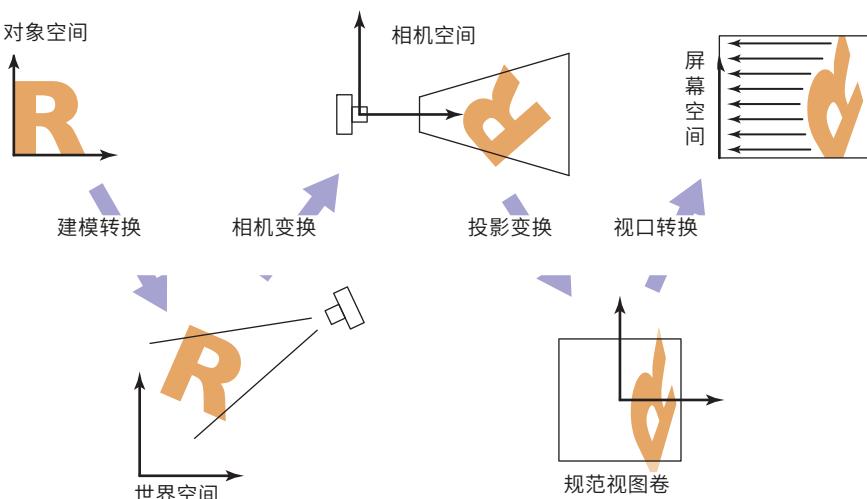
Other names: camera space is also “eye space” and the camera transformation is sometimes the “viewing transformation;” the canonical view volume is also “clip space” or “normalized device coordinates;” screen space is also “pixel coordinates.”

7.1.1 The Viewport Transformation

We begin with a problem whose solution will be reused for any viewing condition. We assume that the geometry we want to view is in the *canonical view volume*, and we wish to view it with an orthographic camera looking in the $-z$ direction. The canonical view volume is the cube containing all 3D points whose Cartesian coordinates are between -1 and $+1$ —that is, $(x, y, z) \in [-1, 1]^3$ (Figure 7.3). We project $x = -1$ to the left side of the screen, $x = +1$ to the right side of the screen, $y = -1$ to the bottom of the screen, and $y = +1$ to the top of the screen.

Recall the conventions for pixel coordinates from Chapter 3: each pixel “owns” a unit square centered at integer coordinates; the image boundaries have a half-

The word “canonical” crops up again—it means something arbitrarily chosen for convenience. For instance, the unit circle could be called the “canonical circle.”



将对象从其原始坐标转换为屏幕空间的空间和转换序列。

摄像机变换将规范坐标（或世界空间）中的点转换为摄像机坐标或将它们放置在摄像机空间中。投影变换将点从摄像机空间移动到规范视图体积。

最后，视口转换将规范视图体积映射到屏幕空间。

这些转换中的每一个都非常简单。我们将详细讨论从视口转换开始的正投影案例，然后介绍支持透视投影所需的更改。

名称：相机空间也是“眼睛空间”，相机转换有时是“观看转换”，规范的视图体积是

空间“或“归一化设备坐标”；“屏幕空间也是“像素坐标”。 ”

7.1.1 视口转换

我们从一个问题开始，其解决方案将被重用于任何查看条件。我们假设我们要查看的几何体在规范视图体积中

我们希望用一个正射相机看到它在 z 方向。规范视图体积是包含笛卡尔坐标介于 -1 和 $+1$ 之间的所有 3D 点的立方体—即 $(x, y, z) \in [-1, 1]^3$ (图 7.3)。我们将 $x = -1$ 投影到屏幕的左侧， $x = +1$ 投影到屏幕的右侧， $y = -1$ 投影到屏幕的底部， $y = +1$ 投影到屏幕的顶部。回想一下第 3 章中像素坐标的惯例：每个像素“拥有”一个以整数坐标为中心的单位正方形；图像边界有一半—

“规范”一词再次出现—它意味着什么

chosen for convenience. 单位圆可以称为“规范圆”。 ”

unit overshoot from the pixel centers; and the smallest pixel center coordinates are $(0, 0)$. If we are drawing into an image (or window on the screen) that has n_x by n_y pixels, we need to map the square $[-1, 1]^2$ to the rectangle $[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$.

Mapping a square to a potentially non-square rectangle is not a problem; x and y just end up with different scale factors going from canonical to pixel coordinates.

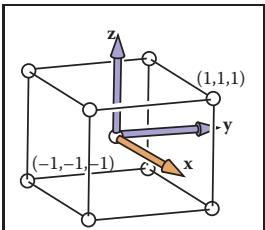


Figure 7.3. The canonical view volume is a cube with side of length two centered at the origin.

$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}. \quad (7.1)$$

Note that this matrix ignores the z -coordinate of the points in the canonical view volume, because a point's distance along the projection direction doesn't affect where that point projects in the image. But before we officially call this the *viewport matrix*, we add a row and column to carry along the z -coordinate without changing it. We don't need it in this chapter, but eventually we will need the z values because they can be used to make closer surfaces hide more distant surfaces (see Section 8.2.3).

$$M_{\text{vp}} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (7.2)$$

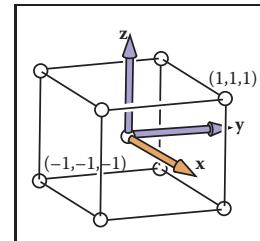
7.1.2 The Orthographic Projection Transformation

Of course, we usually want to render geometry in some region of space other than the canonical view volume. Our first step in generalizing the view will keep the view direction and orientation fixed looking along $-z$ with $+y$ up, but will allow arbitrary rectangles to be viewed. Rather than replacing the viewport matrix, we'll augment it by multiplying it with another matrix on the right.

Under these constraints, the view volume is an axis-aligned box, and we'll name the coordinates of its sides so that the view volume is $[l, r] \times [b, t] \times [f, n]$ shown in Figure 7.4. We call this box the *orthographic view volume* and refer to

单位从像素中心过冲;和最小的像素中心坐标是 $(0, 0)$ 。如果我们正在绘制具有 $n_x \times n_y$ 像素的图像（或屏幕上的窗口），我们需要将正方形 $[0, 1]^2$ 映射到矩形 $[0, n_x - 0.5] \times [0, n_y - 0.5]$ 。

将正方形映射到潜在的非正方形矩形不是问题; x 和 y 最终会得到从规范到像素坐标的不同比例因子。



卡诺尼卡尔视图体积是一个立方体，长度为2的边以原点为中心。

现在，我们假设所有要绘制的线段都完全在规范视图体积内。稍后我们将在讨论削波时放松这一假设。

由于视口变换将一个轴对齐的矩形映射到另一个，因此它是公式 (6.6) 给出的窗口变换的情况：

$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}. \quad (7.1)$$

请注意，此矩阵忽略规范视图中点的 z 坐标
体积，因为点沿投影方向的距离不会影响该点在图像中的投影位置。但是在我们正式称之为视口矩阵之前，我们添加了一个行和列来沿着 z 坐标进行，而不改变它。在本章中我们不需要它，但最终我们将需要 z 值，因为它们可以用来使更近的表面隐藏更远的表面（参见第8.2.3节）。

$$M_{\text{vp}} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (7.2)$$

7.1.2 正投影变换

当然，我们通常希望在规范视图体积之外的某些空间区域中渲染几何体。我们泛化视图的第一步将保持视图方向和方向固定沿 z 与 $+y$ 向上看，但将允许查看任意矩形。我们将通过将其与右侧的另一个矩阵相乘来增强视口矩阵，而不是替换视口矩阵。

在这些约束下，视图体积是一个轴对齐的框，我们将命名其边的坐标，以便视图体积为 $[l, r] \times [b, t] \times [f, n]$ ，如图7.4所示。我们将此框称为正投影视图体积，并参考

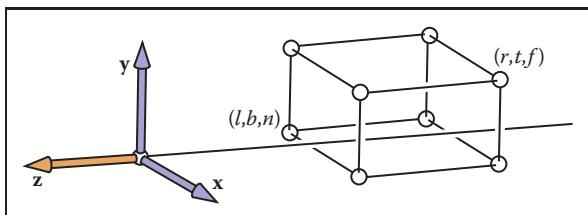


Figure 7.5. The orthographic view volume is along the negative z -axis, so f is a more negative number than n , thus $n > f$.

the bounding planes as follows:

$$\begin{aligned}x &= l \equiv \text{left plane}, \\x &= r \equiv \text{right plane}, \\y &= b \equiv \text{bottom plane}, \\y &= t \equiv \text{top plane}, \\z &= n \equiv \text{near plane}, \\z &= f \equiv \text{far plane}.\end{aligned}$$

That vocabulary assumes a viewer who is looking along the *minus* z -axis with his head pointing in the y -direction.¹ This implies that $n > f$, which may be unintuitive, but if you assume the entire orthographic view volume has negative z values then the $z = n$ “near” plane is closer to the viewer if and only if $n > f$; here f is a smaller number than n , i.e., a negative number of larger absolute value than n .

This concept is shown in Figure 7.5. The transform from orthographic view volume to the canonical view volume is another windowing transform, so we can simply substitute the bounds of the orthographic and canonical view volumes into Equation (6.7) to obtain the matrix for this transformation:

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (7.3)$$

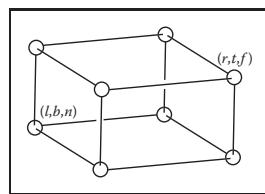


Figure 7.4. The orthographic view volume.

This matrix is very close to the one used traditionally in OpenGL, except that n , f , and $z_{\text{canonical}}$ all have the opposite sign.

¹Most programmers find it intuitive to have the x -axis pointing right and the y -axis pointing up. In a right-handed coordinate system, this implies that we are looking in the $-z$ direction. Some systems use a left-handed coordinate system for viewing so that the gaze direction is along $+z$. Which is best is a matter of taste, and this text assumes a right-handed coordinate system. A reference that argues for the left-handed system instead is given in the notes at the end of the chapter.

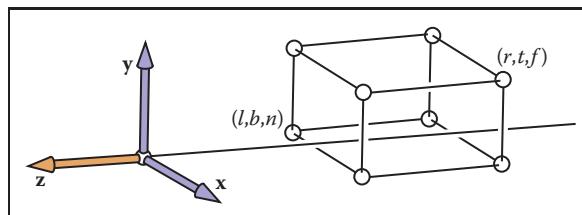
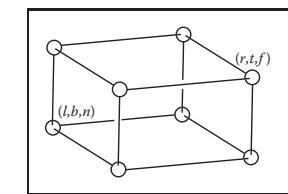


图7.5。 正投影视图体积沿着负z轴，因此f比n更负数，因此n>f。

边界平面如下：

$$\begin{aligned}x &= l \neq \text{左平面} \\x &= r \neq \text{右平面} \\y &= b \neq \text{底平面} \\y &= t \neq \text{顶平面} \\z &= n \infty \text{近平面} \\z &= f \neq \text{远平面}.\end{aligned}$$

这个词汇假设一个观众是沿着负z轴看的。他的头指向y方向。这意味着n>f，这可能是不直观的，但如果假设整个正投影视图体积具有负z值，那么当且仅当n>f时，z=n“近”平面更接近观看者；这里f是比n更小的数。



的正射图形视图体积。

这个概念如图7.5所示。从正射视图体积到规范视图体积的变换是另一个窗口变换，因此我们可以简单地将正射视图体积和规范视图体积的边界代入公式 (6.7) 以获得此变换的：

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (7.3)$$

这个矩阵非常接近OpenGL中传统使用的矩阵，除了n, f和z规范都有相反的符号。

大多数程序员发现x轴指向右，y轴指向上是很直观的。在右手坐标系中，这意味着我们正在寻找 $-z$ 方向。一些系统使用左手坐标系进行观看，使得注视方向沿 $+z$ 。哪个最好是品味问题，并且本文假设右手坐标系。本章末尾的注释中给出了一个主张左手系统的参考。

To draw 3D line segments in the orthographic view volume, we project them into screen x - and y -coordinates and ignore z -coordinates. We do this by combining Equations (7.2) and (7.3). Note that in a program we multiply the matrices together to form one matrix and then manipulate points as follows:

$$\begin{bmatrix} x_{\text{pixel}} \\ y_{\text{pixel}} \\ z_{\text{canonical}} \\ 1 \end{bmatrix} = (\mathbf{M}_{\text{vp}} \mathbf{M}_{\text{orth}}) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

The z -coordinate will now be in $[-1, 1]$. We don't take advantage of this now, but it will be useful when we examine z-buffer algorithms.

The code to draw many 3D lines with endpoints \mathbf{a}_i and \mathbf{b}_i thus becomes both simple and efficient:

```
construct Mvp
construct Morth
M = MvpMorth
for each line segment (ai, bi) do
    p = Mai
    q = Mbi
    drawline(xp, yp, xq, yq)
```

This is a first example of how matrix transformation machinery makes graphics programs clean and efficient.

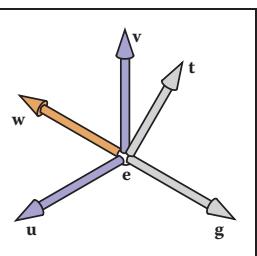


Figure 7.6. The user specifies viewing as an eye position e , a gaze direction g , and an up vector t . We construct a right-handed basis with w pointing opposite to the gaze and v being in the same plane as g and t .

7.1.3 The Camera Transformation

We'd like to be able to change the viewpoint in 3D and look in any direction. There are a multitude of conventions for specifying viewer position and orientation. We will use the following one (see Figure 7.6):

- the eye position e ,
- the gaze direction g ,
- the view-up vector t .

The eye position is a location that the eye "sees from." If you think of graphics as a photographic process, it is the center of the lens. The gaze direction is any vector in the direction that the viewer is looking. The view-up vector is any vector in the plane that both bisects the viewer's head into right and left halves and points "to the sky" for a person standing on the ground. These vectors provide us with enough information to set up a coordinate system with origin e and a uvw basis,

要在正投影视图体中绘制3D线段，我们将它们投影到屏幕x和y坐标中，并忽略z坐标。我们通过combining方程 (7.2) 和 (7.3) 来做到这一点。请注意，在程序中，我们将矩阵相乘以形成一个矩阵，然后按如下方式操作点：

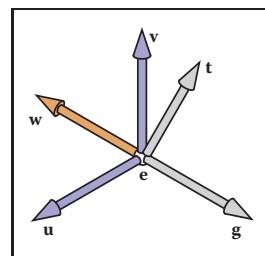
$$\begin{bmatrix} x_{\text{像素}} \\ y_{\text{像素}} \\ z_{\text{规范}} \\ 1 \end{bmatrix} = (\mathbf{M}_{\text{vp}} \mathbf{M}_{\text{orth}}) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Z坐标现在将位于[1]中。我们现在没有利用这一点，但是当我们检查z-buffer算法时，它会很有用。

使用端点ai和bi绘制许多3D线的代码因此变得既简单又高效：

```
constructMvpconstructmorth
m=mvporthforeachlinesegment(ai bi)dop=Maiq=MbIdrawline(xp yp xq yq)
```

这是矩阵变换机械如何使图形程序干净高效的第一个例子。



用户将观察指定为眼睛位置e、注视方向g和向上向量t。我们构建了一个右手基础，w指向与凝视相反，v与g和t处于同一平面。

7.1.3 相机变换

我们希望能够在3D中更改视点并朝任何方向看。有许多约定用于指定查看器的位置和方向。我们将使用以下一个（见图7.6）：

- *眼睛位置e
- *注视方向g
- *向上视图向量t。

眼睛位置是眼睛"看到的位置。"如果你把图形看作是一个摄影过程，那就是镜头的中心。注视方向是观看者正在看的方向上的任何矢量。视图向上矢量是平面上的任何矢量，它将观看者的头部分成左右两半，并指向站在地上的"天空"。这些向量为我们提供了足够的信息来建立一个原点e和uvw基础的坐标系

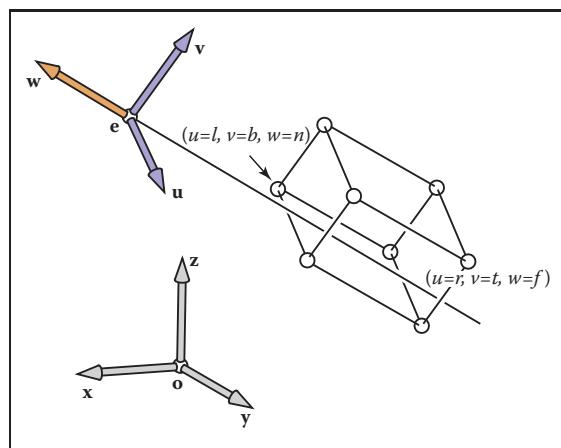


Figure 7.7. For arbitrary viewing, we need to change the points to be stored in the “appropriate” coordinate system. In this case it has origin e and offset coordinates in terms of uvw .

using the construction of Section 2.4.7:

$$\mathbf{w} = -\frac{\mathbf{g}}{\|\mathbf{g}\|},$$

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

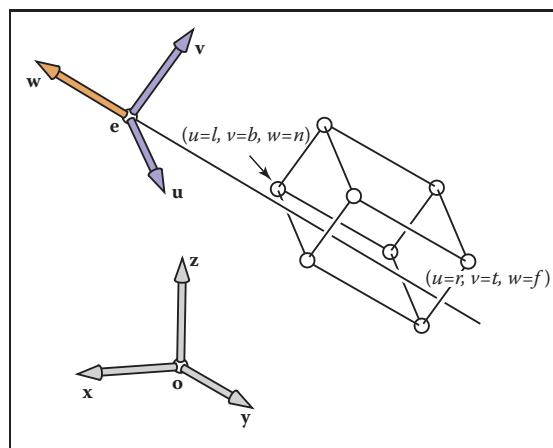
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}.$$

Our job would be done if all points we wished to transform were stored in coordinates with origin e and basis vectors \mathbf{u} , \mathbf{v} , and \mathbf{w} . But as shown in Figure 7.7, the coordinates of the model are stored in terms of the canonical (or world) origin \mathbf{o} and the x -, y -, and z -axes. To use the machinery we have already developed, we just need to convert the coordinates of the line segment endpoints we wish to draw from xyz -coordinates into uvw -coordinates. This kind of transformation was discussed in Section 6.5, and the matrix that enacts this transformation is the canonical-to-basis matrix of the camera’s coordinate frame:

$$\mathbf{M}_{\text{cam}} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (7.4)$$

Alternatively, we can think of this same transformation as first moving e to the origin, then aligning \mathbf{u} , \mathbf{v} , \mathbf{w} to x , y , z .

To make our previously z -axis-only viewing algorithm work for cameras with any location and orientation, we just need to add this camera transformation to



对于任意查看，我们需要更改要存储在“适当”坐标系中的点。在这种情况下它具有原点 e 和以 uvw 计的偏移坐标。

使用第2.4.7节的构造：

$$\mathbf{w} = -\frac{\mathbf{g}}{\|\mathbf{g}\|},$$

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

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}.$$

如果我们希望转换的所有点都存储在原点 e 和基向量 \mathbf{u} , \mathbf{v} 和 \mathbf{w} 的坐标中，我们的工作就会完成。但如图7.7所示，模型的坐标存储在规范（或世界）原点 \mathbf{o} 和 x 轴、 y 轴和 z 轴上。要使用我们已经开发的机器，我们只需要将我们希望绘制的线段端点的坐标从 xyz -坐标转换为 uvw -坐标。这种变换在第6.5节中讨论过，产生这种变换的矩阵是相机坐标系的规范到基矩阵：

$$\mathbf{M}_{\text{cam}} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (7.4)$$

或者，我们可以将此转换视为首先将 e 移动到原点，然后将 \mathbf{u} , \mathbf{v} , \mathbf{w} 对齐到 x , y , z 。

为了使我们以前的仅 z 轴查看算法适用于任何位置和方向的相机，我们只需要将此相机转换添加到

the product of the viewport and projection transformations, so that it converts the incoming points from world to camera coordinates before they are projected:

```

construct Mvp
construct Morth
construct Mcam
M = MvpMorthMcam
for each line segment (ai, bi) do
    p = Mai
    q = Mbi
    drawline(xp, yp, xq, yq)

```

Again, almost no code is needed once the matrix infrastructure is in place.

7.2 Projective Transformations

We have left perspective for last because it takes a little bit of cleverness to make it fit into the system of vectors and matrix transformations that has served us so well up to now. To see what we need to do, let's look at what the perspective projection transformation needs to do with points in camera space. Recall that the viewpoint is positioned at the origin and the camera is looking along the z-axis.

The key property of perspective is that the size of an object on the screen is proportional to $1/z$ for an eye at the origin looking up the negative z-axis. This can be expressed more precisely in an equation for the geometry in Figure 7.8:

$$y_s = \frac{d}{z}y, \quad (7.5)$$

For the moment we will ignore the sign of z to keep the equations simpler, but it will return on page 150.

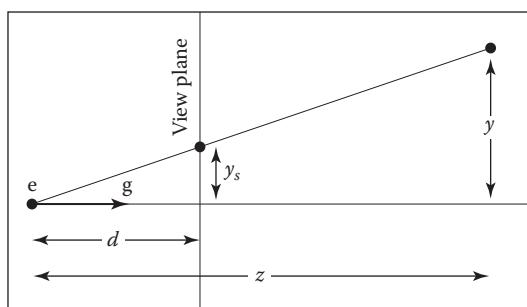


Figure 7.8. The geometry for Equation (7.5). The viewer's eye is at e and the gaze direction is g (the minus z -axis). The view plane is a distance d from the eye. A point is projected toward e and where it intersects the view plane is where it is drawn.

视口和投影变换的乘积，以便在投影之前将传入的点从世界转换为相机坐标：

```

constructMvpconstructmorth
constructmcamm=mvpMorth
mcamforeachlinesegment(ai
bi)dop=Maiq=McIdrawline(xp
yp xq yq)

```

同样，一旦矩阵基础设施到位，几乎不需要代码。

7.2 射影变换

我们把视角留到最后，因为它需要一点点聪明才能使它适应向量和矩阵变换系统，到目前为止，这对我们很有帮助。为了了解我们需要做什么，让我们来看看透视投影变换需要对摄像机空间中的点做什么。回想一下，

视点位于原点，相机沿z轴观察。透视的关键属性是屏幕上对象的大小与一只眼睛在原点向上看负z轴的 $1/z$ 成正比。这可以在图7.8中的几何方程中更精确地表达：

$$y_s = \frac{d}{z}y, \quad (7.5)$$

目前我们将忽略 z 符号以保持方程更简单，但它将返回第150页。

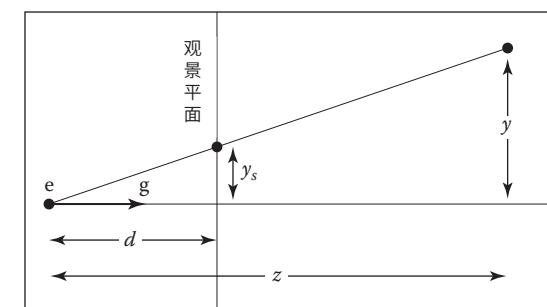


图7.8。 方程 (7.5) 的几何形状。观看者的眼睛在e处并且注视方向是g(减z轴)。视平面是距眼睛的距离d。一个点向e投影，它与视图平面相交的地方就是它被绘制的地方。

where y is the distance of the point along the y -axis, and y_s is where the point should be drawn on the screen.

We would really like to use the matrix machinery we developed for orthographic projection to draw perspective images; we could then just multiply another matrix into our composite matrix and use the algorithm we already have. However, this type of transformation, in which one of the coordinates of the input vector appears in the denominator, can't be achieved using affine transformations.

We can allow for division with a simple generalization of the mechanism of homogeneous coordinates that we have been using for affine transformations. We have agreed to represent the point (x, y, z) using the homogeneous vector $[x \ y \ z \ 1]^T$; the extra coordinate, w , is always equal to 1, and this is ensured by always using $[0 \ 0 \ 0 \ 1]^T$ as the fourth row of an affine transformation matrix.

Rather than just thinking of the 1 as an extra piece bolted on to coerce matrix multiplication to implement translation, we now define it to be the denominator of the x -, y -, and z -coordinates: the homogeneous vector $[x \ y \ z \ w]^T$ represents the point $(x/w, y/w, z/w)$. This makes no difference when $w = 1$, but it allows a broader range of transformations to be implemented if we allow any values in the bottom row of a transformation matrix, causing w to take on values other than 1.

Concretely, linear transformations allow us to compute expressions like

$$x' = ax + by + cz$$

and affine transformations extend this to

$$x' = ax + by + cz + d.$$

Treating w as the denominator further expands the possibilities, allowing us to compute functions like

$$x' = \frac{ax + by + cz + d}{ex + fy + gz + h};$$

this could be called a “linear rational function” of x , y , and z . But there is an extra constraint—the denominators are the same for all coordinates of the transformed point:

$$x' = \frac{a_1x + b_1y + c_1z + d_1}{ex + fy + gz + h},$$

$$y' = \frac{a_2x + b_2y + c_2z + d_2}{ex + fy + gz + h},$$

$$z' = \frac{a_3x + b_3y + c_3z + d_3}{ex + fy + gz + h}.$$

其中y是点沿y轴的距离，ys是点应该在屏幕上绘制的位置。

我们真的很想使用我们为正射图形投影开发的矩阵机器来绘制透视图像；然后我们可以将另一个矩阵乘以我们的复合矩阵，并使用我们已经拥有的算法。然而，这种类型的变换，其中输入向量的坐标之一出现在分母中，不能使用仿射变换来实现。我们可以通过简单的泛化我们一直用于仿射变换的齐次坐标机制来允许除法。我们已经同意使用齐次向量 $[xyz1]^T$ 表示点 (x, y, z) ；额外的坐标 w 总是等于1，并且通过始终使用 $[0001]^T$ 作为仿射变换矩阵的第四行来确保这

我们现在将其定义为 x , y 和 z 坐标的分母：齐次向量 $[xyzw]^T$ 表示点 (xw, yw, zw) ，而不是仅仅将1视为强制矩阵乘法以实现平移的额外部分。当 $w=1$ 时，这没有什么区别，但是如果我们将变换矩阵的底行中的任何值，则允许实现更广泛的变换，从而导致 w 接受1以外的值。具体地说，线性变换允许我们计算表达式，如

$$x' = ax + by + cz$$

仿射变换将其扩展到

$$x' = ax + by + cz + d.$$

将 w 视为分母进一步扩展了可能性，使我们能够计算如下函数

$$x' = \frac{ax}{ex + fy + gz + h};$$

这可以称为 x , y 和 z 的“线性有理函数”。但是有一个额外的约束—对于变换点的所有坐标，分母都是相同的：

$$x' = \frac{a_1x + b_1y + c_1z + d_1}{ex + fy + gz + h},$$

$$y' = \frac{a_2x + b_2y + c_2z + d_2}{ex + fy + gz + h},$$

$$z' = \frac{a_3x + b_3y + c_3z + d_3}{ex + fy + gz + h}.$$

Expressed as a matrix transformation,

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ e & f & g & h \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

and

$$(x', y', z') = (\tilde{x}/\tilde{w}, \tilde{y}/\tilde{w}, \tilde{z}/\tilde{w}).$$

A transformation like this is known as a *projective transformation* or a *homography*.

Example. The matrix

$$M = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 3 & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$

represents a 2D projective transformation that transforms the unit square $([0, 1] \times [0, 1])$ to the quadrilateral shown in Figure 7.9.

For instance, the lower-right corner of the square at $(1, 0)$ is represented by the homogeneous vector $[1 \ 0 \ 1]^T$ and transforms as follows:

$$\begin{bmatrix} 2 & 0 & -1 \\ 0 & 3 & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \frac{1}{3} \end{bmatrix},$$

which represents the point $(1/\frac{1}{3}, 0/\frac{1}{3})$, or $(3, 0)$. Note that if we use the matrix

$$3M = \begin{bmatrix} 6 & 0 & -3 \\ 0 & 9 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

instead, the result is $[3 \ 0 \ 1]^T$, which also represents $(3, 0)$. In fact, any scalar multiple cM is equivalent: the numerator and denominator are both scaled by c , which does not change the result. □

There is a more elegant way of expressing the same idea, which avoids treating the w -coordinate specially. In this view a 3D projective transformation is simply a 4D linear transformation, with the extra stipulation that all scalar multiples of a vector refer to the same point:

$$\mathbf{x} \sim \alpha \mathbf{x} \quad \text{for all } \alpha \neq 0.$$

The symbol \sim is read as "is equivalent to" and means that the two homogeneous vectors both describe the same point in space.

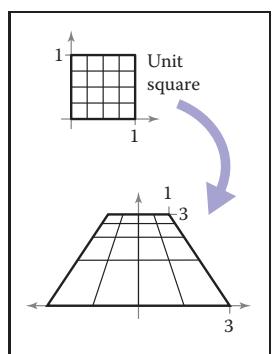


Figure 7.9. A projective transformation maps a square to a quadrilateral, preserving straight lines but not parallel lines.

表示为矩阵变换

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ e & f & g & h \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

and

$$(x', y', z') = (\tilde{x}/\tilde{w}, \tilde{y}/\tilde{w}, \tilde{z}/\tilde{w}).$$

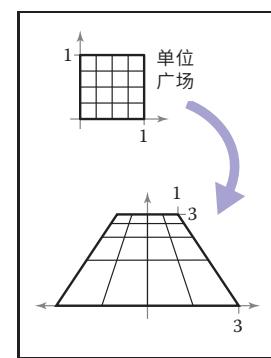
像这样的变换被称为射影变换或单应性。

例子。矩阵

$$M = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 3 & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$

表示变换单位平方的2D射影变换($[0 \ 1] \times [0 \ 1]$)到图7.9所示的四边形。

例如, $(1 \ 0)$ 处正方形的右下角由齐次向量 $[1 \ 0 \ 1]^T$ 表示, 并按如下方式变换:



射影变换将正方形映射到四边形, 保留直线但不保留平行线。

其中表示点 $(1 \ 0 \ 1)^T$

$$\begin{bmatrix} 2 & 0 & -1 \\ 0 & 3 & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \frac{1}{3} \end{bmatrix},$$

, 或 $(3, 0)$ 。请注意, 如果我们使用矩阵

$$3M = \begin{bmatrix} 6 & 0 & -3 \\ 0 & 9 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

相反, 结果是 $[3 \ 0 \ 1]^T$, 它也表示 $(3, 0)$ 。事实上, 任何标量倍数 cM 都是等价的: 分子和分母都被 c 缩放, 这不会改变结果。

有一种更优雅的方式来表达相同的想法, 这避免了专门处理 w 坐标。在这个视图中, 3D射影变换只是一个4d线性变换, 额外的规定是矢量的所有标量倍数都指向同一个点:

$$\mathbf{x} \sim \alpha \mathbf{x} \quad \text{对于所有 } \alpha=0.$$

符号 被读作"等价于", 并且意味着两个齐次向量都描述了空间中的同一点。

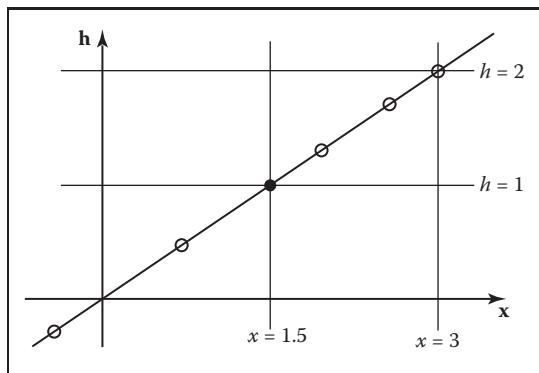


Figure 7.10. The point $x = 1.5$ is represented by any point on the line $x = 1.5h$, such as points at the hollow circles. However, before we interpret x as a conventional Cartesian coordinate, we first divide by h to get $(x, h) = (1.5, 1)$ as shown by the black point.

Example. In 1D homogeneous coordinates, in which we use 2-vectors to represent points on the real line, we could represent the point (1.5) using the homogeneous vector $[1.5 \ 1]^T$, or any other point on the line $x = 1.5h$ in homogeneous space. (See Figure 7.10.)

In 2D homogeneous coordinates, in which we use 3-vectors to represent points in the plane, we could represent the point $(-1, -0.5)$ using the homogeneous vector $[-2; -1; 2]^T$, or any other point on the line $\mathbf{x} = \alpha[-1 \ -0.5 \ 1]^T$. Any homogeneous vector on the line can be mapped to the line's intersection with the plane $w = 1$ to obtain its Cartesian coordinates. (See Figure 7.11.)

It's fine to transform homogeneous vectors as many times as needed, without worrying about the value of the w -coordinate—in fact, it is fine if the w -coordinate is zero at some intermediate phase. It is only when we want the ordinary Cartesian coordinates of a point that we need to normalize to an equivalent point that has $w = 1$, which amounts to dividing all the coordinates by w . Once we've done this we are allowed to read off the (x, y, z) -coordinates from the first three components of the homogeneous vector.

7.3 Perspective Projection

The mechanism of projective transformations makes it simple to implement the division by z required to implement perspective. In the 2D example shown in Figure 7.8, we can implement the perspective projection with a matrix transformation

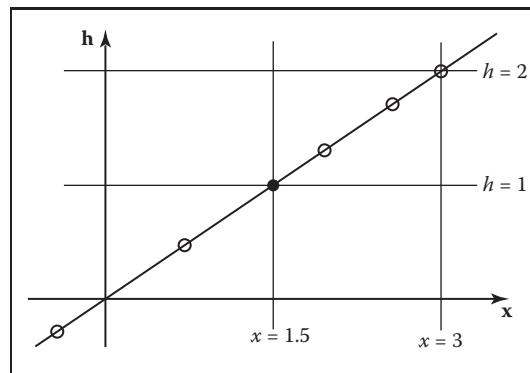


图7.10。点 $x=1.5$ 由线 $x=1.5h$ 上的任何点表示，例如空心圆处的点。然而，在我们将 x 解释为常规笛卡尔坐标之前，我们首先除以 h 以获得 $(x, h) = (1.5, 1)$ ，如黑点所示。

例子。在一维齐次坐标中，我们使用2向量在实线上重新计算发送的点，我们可以表示点 (1.5) 使用homogeneous向量 $[1.5 \ 1]^T$ ，或线 $x=1$ 上的任何其他点。在均匀空间 $5h$ 。（见图7.10。）

在二维同质坐标中，我们使用3向量来表示平面中的点，我们可以表示点 $(-1, -0.5)$ 使用齐次向量 $[-2; -1; 2]^T$ ，或线 $\mathbf{x}=\alpha[-1 \ -0.5 \ 1]^T$ 。线上的任何齐次矢量都可以映射到线与平面 $w=1$ 的交点，以获得其笛卡尔坐标。（见图7.11。）

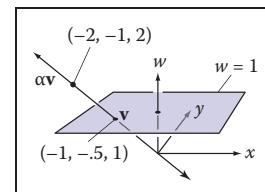
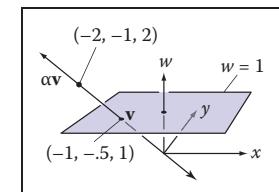


Figure 7.11. A point in homogeneous coordinates is equivalent to any other point on the line through it and the origin, and normalizing the point amounts to intersecting this line with the plane $w = 1$.

根据需要多次变换齐次矢量是可以的，而不必担心 w 坐标的值—事实上，如果 w 坐标在某个中间相位为零就可以了。只有当我们想要一个点的普通笛卡尔坐标时，我们才需要归一化为具有 $w=1$ 的等效点，这相当于将所有坐标除以 w 。完成此操作后，我们可以从齐次矢量的前三个分量中读取 (x, y, z) 坐标。



齐次坐标中的一个点相当于通过它的线和原点上的任何其他点，并且归一化该点相当于将此线与平面 $w=1$ 相交。

7.3 透视投影

射影变换的机制使得实现透视所需的按 z 划分变得简单。在图7.8所示的2D示例中，我们可以通过矩阵变换实现透视投影

as follows:

$$\begin{bmatrix} y_s \\ 1 \end{bmatrix} \sim \begin{bmatrix} d & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix}.$$

This transforms the 2D homogeneous vector $[y; z; 1]^T$ to the 1D homogeneous vector $[dy z]^T$, which represents the 1D point (dy/z) (because it is equivalent to the 1D homogeneous vector $[dy/z 1]^T$). This matches Equation (7.5).

For the “official” perspective projection matrix in 3D, we’ll adopt our usual convention of a camera at the origin facing in the $-z$ direction, so the distance of the point (x, y, z) is $-z$. As with orthographic projection, we also adopt the notion of near and far planes that limit the range of distances to be seen. In this context, we will use the near plane as the projection plane, so the image plane distance is $-n$.

Remember, $n < 0$.

The desired mapping is then $y_s = (n/z)y$, and similarly for x . This transformation can be implemented by the *perspective matrix*:

$$\mathbf{P} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

The first, second, and fourth rows simply implement the perspective equation. The third row, as in the orthographic and viewport matrices, is designed to bring the z -coordinate “along for the ride” so that we can use it later for hidden surface removal. In the perspective projection, though, the addition of a non-constant denominator prevents us from actually preserving the value of z —it’s actually impossible to keep z from changing while getting x and y to do what we need them to do. Instead we’ve opted to keep z unchanged for points on the near or far planes.

More on this later.

There are many matrices that could function as perspective matrices, and all of them nonlinearly distort the z -coordinate. This specific matrix has the nice properties shown in Figures 7.12 and 7.13; it leaves points on the $(z = n)$ -plane entirely alone, and it leaves points on the $(z = f)$ -plane while “squishing” them in x and y by the appropriate amount. The effect of the matrix on a point (x, y, z) is

$$\mathbf{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ z \end{bmatrix} \sim \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n + f - \frac{fn}{z} \\ 1 \end{bmatrix}.$$

as follows:

$$\begin{bmatrix} y_s \\ 1 \end{bmatrix} \sim \begin{bmatrix} d & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix}.$$

这将2D齐次向量 $[y;z;1]^T$ 转换为1D齐次向量 $[dyz]^T$, 其表示1D点 (dyz) (因为它相当于1D齐次向量 $[dyz1]^T$ 。这与等式(7.5)匹配。

对于3D中的“官方”透视投影矩阵, 我们将采用我们通常的惯例, 即在原点面向 z 方向的相机, 因此点 (x, y, z) 的距离是 z 。与正投影一样, 我们也采用了近平面和远平面的概念, 限制了可以看到的距离范围。在此上下文中, 我们将使用近平面作为投影平面, 因此图像平面距离为 $-n$ 。

Remember, $n < 0$.

期望的映射然后是 $ys=(nz)y$, 并且类似地对于 x 。这种变形可以通过透视矩阵来实现:

$$\mathbf{P} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

第一、第二和第四行简单地实现透视方程。第三行, 就像在正交和视口矩阵中一样, 旨在使 z 坐标“沿着行驶”, 以便我们以后可以使用它来移除隐藏的表面。然而, 在透视投影中, 添加一个非恒定分母阻止我们实际保留 z 的值—实际上不可能在让 x 和 y 做我们需要它们做的事情的同时保持 z 的变化。相反, 我们选择保持 z 不变的近点或远点

稍后再谈。

planes.

有许多矩阵可以作为透视矩阵, 并且所有这些矩阵都非线性地扭曲了 z 坐标。这个特定的矩阵具有图7.12和7.13所示的良好属性; 它在 $(z=n)$ 平面上完全单独留下点, 并且在 $(z=f)$ 平面上留下点, 同时在 x 和 y 中“压扁”它们适当的量。矩阵对点 (x, y, z) 的影响为

$$\mathbf{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ z \end{bmatrix} \sim \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n + f - \frac{fn}{z} \\ 1 \end{bmatrix}.$$

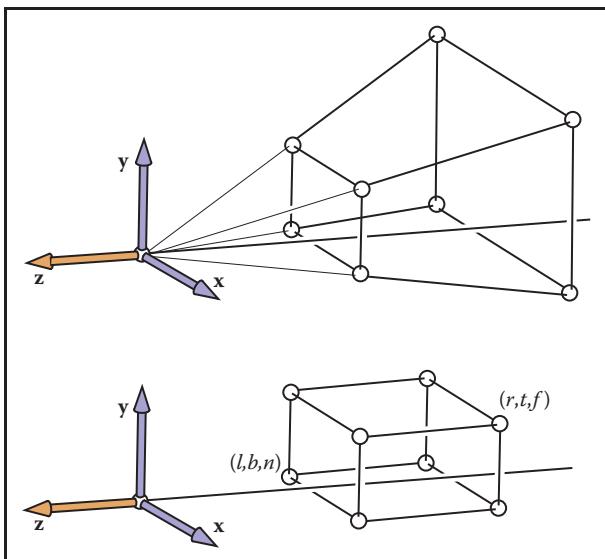


Figure 7.12. The perspective projection leaves points on the $z = n$ plane unchanged and maps the large $z = f$ rectangle at the back of the perspective volume to the small $z = f$ rectangle at the back of the orthographic volume.

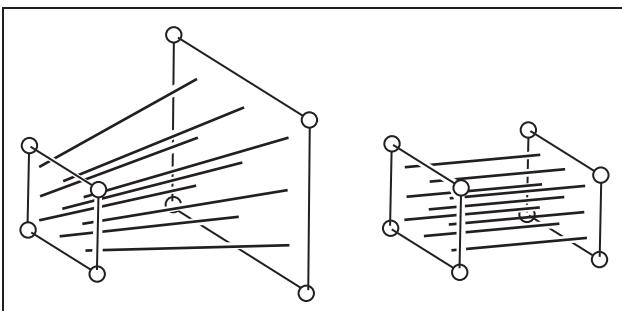


Figure 7.13. The perspective projection maps any line through the origin/eye to a line parallel to the z -axis and without moving the point on the line at $z = n$.

As you can see, x and y are scaled and, more importantly, divided by z . Because both n and z (inside the view volume) are negative, there are no “flips” in x and y . Although it is not obvious (see the exercise at the end of the chapter), the transform also preserves the relative order of z values between $z = n$ and $z = f$, allowing us to do depth ordering after this matrix is applied. This will be important later when we do hidden surface elimination.

Sometimes we will want to take the inverse of \mathbf{P} , for example, to bring a screen coordinate plus z back to the original space, as we might want to do for

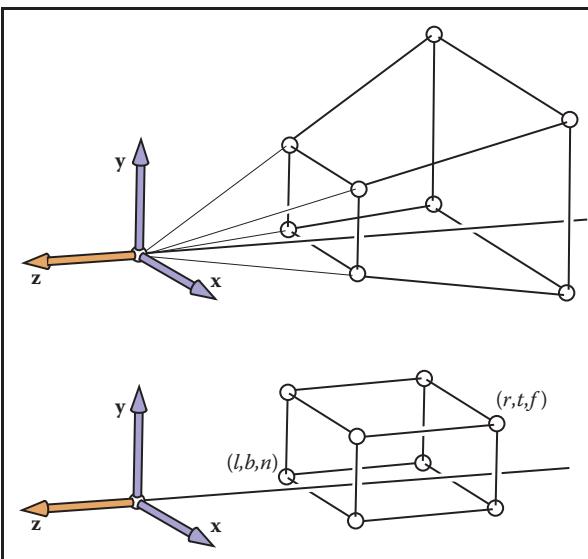
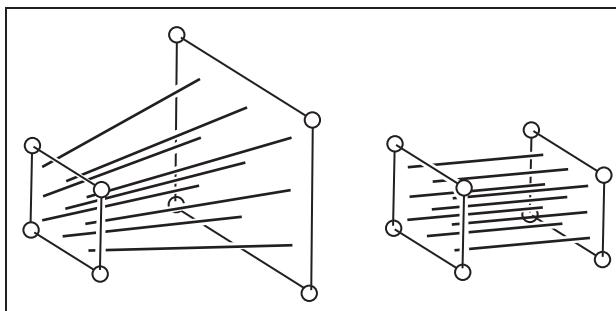


图7.12。 透视投影使 $z=n$ 平面上的点保持不变，并将透视体积后面的大 $z=f$ 矩形映射到正投影体积后面的小 $z=f$ 矩形。



透视投影将通过原点眼睛的任何线映射到平行于 z 轴的线，并且不移动 $z=n$ 处的线上的点。

正如你所看到的， x 和 y 是缩放的，更重要的是，除以 z 。因为 n 和 z （在视图体积内）都是负的，所以 x 和 y 中没有“翻转”。虽然它并不明显（请参阅章节末尾的练习），但变换还保留了 $z=n$ 和 $z=f$ 之间 z 值的相对顺序，允许我们在应用此矩阵之后进行深度排序。当我们做隐藏表面消除时，这将是重要的。

有时我们会想要取 \mathbf{P} 的倒数，例如，将屏幕坐标加上 z 回到原始空间，就像我们可能想要做的那样

picking. The inverse is

$$\mathbf{P}^{-1} = \begin{bmatrix} \frac{1}{n} & 0 & 0 & 0 \\ 0 & \frac{1}{n} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{fn} & \frac{n+f}{fn} \end{bmatrix}.$$

Since multiplying a homogeneous vector by a scalar does not change its meaning, the same is true of matrices that operate on homogeneous vectors. So we can write the inverse matrix in a prettier form by multiplying through by nf :

$$\mathbf{P}^{-1} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & fn \\ 0 & 0 & -1 & n+f \end{bmatrix}.$$

This matrix is not literally the inverse of the matrix \mathbf{P} , but the transformation it describes *is* the inverse of the transformation described by \mathbf{P} .

Taken in the context of the orthographic projection matrix \mathbf{M}_{orth} in Equation (7.3), the perspective matrix simply maps the perspective view volume (which is shaped like a slice, or *frustum*, of a pyramid) to the orthographic view volume (which is an axis-aligned box). The beauty of the perspective matrix is that once we apply it, we can use an orthographic transform to get to the canonical view volume. Thus, all of the orthographic machinery applies, and all that we have added is one matrix and the division by w . It is also heartening that we are not “wasting” the bottom row of our four by four matrices!

Concatenating \mathbf{P} with \mathbf{M}_{orth} results in the *perspective projection matrix*,

$$\mathbf{M}_{\text{per}} = \mathbf{M}_{\text{orth}} \mathbf{P}.$$

One issue, however, is: How are l, r, b, t determined for perspective? They identify the “window” through which we look. Since the perspective matrix does not change the values of x and y on the ($z = n$)-plane, we can specify (l, r, b, t) on that plane.

To integrate the perspective matrix into our orthographic infrastructure, we simply replace \mathbf{M}_{orth} with \mathbf{M}_{per} , which inserts the perspective matrix \mathbf{P} after the camera matrix \mathbf{M}_{cam} has been applied but before the orthographic projection. So the full set of matrices for perspective viewing is

$$\mathbf{M} = \mathbf{M}_{\text{vp}} \mathbf{M}_{\text{orth}} \mathbf{P} \mathbf{M}_{\text{cam}}.$$

The resulting algorithm is:

```
compute Mvp
compute Mper
compute Mcam
```

采摘。相反的是

$$\mathbf{P}^{-1} = \begin{bmatrix} \frac{1}{n} & 0 & 0 & 0 \\ 0 & \frac{1}{n} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{fn} & \frac{n+f}{fn} \end{bmatrix}.$$

由于将齐次向量乘以标量不会改变其含义，因此对齐次向量进行操作的矩阵也是如此。所以我们可以通过乘以 nf 以更漂亮的形式写出逆矩阵：

$$\mathbf{P}^{-1} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & fn \\ 0 & 0 & -1 & n+f \end{bmatrix}.$$

这个矩阵不是字面上的矩阵 \mathbf{P} 的逆，但它描述的变换是由 \mathbf{P} 划线的变换de的逆。

在Equation(7.3)中的正投影矩阵Morth的上下文中，透视矩阵简单地将透视视图体积（其形状像金字塔的切片或截头）映射到正投影视图体积（其是轴对齐的框）。透视矩阵的美妙之处在于，一旦我们应用它，我们就可以使用正交变换来获得规范视图体积。因此，所有的正交机械都适用，我们添加的只是一个矩阵和w的除法。同样令人振奋的是，我们没有“浪费”我们的四乘四矩阵的底行！

将 \mathbf{P} 与 \mathbf{M} 串联，得到透视投影矩阵

$$\mathbf{M}_{\text{每}} = \mathbf{M} \text{ 或 } \mathbf{P}.$$

然而，一个问题是： l, r, b, t 如何确定透视？他们确定了“窗口”，通过它我们看。由于透视矩阵不会改变 ($z=n$) 平面上的x和y的值，我们可以在该平面上指定 (l, r, b, t) 。

要将透视矩阵集成到我们的正投影基础设施中，我们只需将Morth替换为Mper，它将透视矩阵p插入相机矩阵Mcum之后但在正投影之前。所以透视观察的全套矩阵是

$$\mathbf{M} = \mathbf{M}_{\text{vp}} \mathbf{M} = \mathbf{P} \mathbf{M}_{\text{cam}}.$$

所得算法为：

```
计算Mvp计
算m每计算m
cam
```

```

 $M = M_{vp}M_{per}M_{cam}$ 
for each line segment  $(a_i, b_i)$  do
   $p = Ma_i$ 
   $q = Mb_i$ 
  drawline( $x_p/w_p, y_p/w_p, x_q/w_q, y_q/w_q$ )

```

Note that the only change other than the additional matrix is the divide by the homogeneous coordinate w .

Multiplied out, the matrix M_{per} looks like this:

$$M_{per} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

This or similar matrices often appear in documentation, and they are less mysterious when one realizes that they are usually the product of a few simple matrices.

Example. Many APIs such as *OpenGL* (Shreiner, Neider, Woo, & Davis, 2004) use the same canonical view volume as presented here. They also usually have the user specify the absolute values of n and f . The projection matrix for *OpenGL* is

$$M_{OpenGL} = \begin{bmatrix} \frac{2|n|}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2|n|}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{|n|+|f|}{|n|-|f|} & \frac{2|f||n|}{|n|-|f|} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Other APIs send n and f to 0 and 1, respectively. Blinn (J. Blinn, 1996) recommends making the canonical view volume $[0, 1]^3$ for efficiency. All such decisions will change the the projection matrix slightly.

7.4 Some Properties of the Perspective Transform

An important property of the perspective transform is that it takes lines to lines and planes to planes. In addition, it takes line segments in the view volume to line

```

M=mvpMperMcamforeachlinesegment
t(ai bi)dop=Maiq=MbIdrawline(xpwp y
pwp xqwq yqwq)

```

注意，除了附加矩阵之外的唯一变化是除齐次坐标 w 。

相乘出来，矩阵 M_{per} 看起来像这样：

$$M_{per} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

这种或类似的矩阵经常出现在文档中，当人们意识到它们通常是几个简单矩阵的乘积时，它们就不那么神秘了。

例子。许多API如*OpenGL* (Shreiner, Neider, Woo, & Davis, 2004) 使用与此处呈现的相同的规范视图体积。它们通常还让用户指定 n 和 f 的绝对值。*OpenGL*的投影矩阵是

$$M_{OpenGL} = \begin{bmatrix} \frac{2|n|}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2|n|}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{|n|+|f|}{|n|-|f|} & \frac{2|f||n|}{|n|-|f|} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

其他API分别将 n 和 f 发送到0和1。Blinn (J. Blinn, 1996) 建议使规范视图体积 $[0, 1]^3$ 提高效率。所有这些决定都会稍微改变投影矩阵。

7.4 透視變換的一些屬性

透視變換的一個重要屬性是它將線帶到線，將平面帶到平面。此外，它將視圖卷中的線段取為線

segments in the canonical volume. To see this, consider the line segment

$$\mathbf{q} + t(\mathbf{Q} - \mathbf{q}).$$

When transformed by a 4×4 matrix \mathbf{M} , it is a point with possibly varying homogeneous coordinate:

$$\mathbf{M}\mathbf{q} + t(\mathbf{MQ} - \mathbf{Mq}) \equiv \mathbf{r} + t(\mathbf{R} - \mathbf{r}).$$

The homogenized 3D line segment is

$$\frac{\mathbf{r} + t(\mathbf{R} - \mathbf{r})}{w_r + t(w_R - w_r)}. \quad (7.6)$$

If Equation (7.6) can be rewritten in a form

$$\frac{\mathbf{r}}{w_r} + f(t) \left(\frac{\mathbf{R}}{w_R} - \frac{\mathbf{r}}{w_r} \right), \quad (7.7)$$

then all the homogenized points lie on a 3D line. Brute force manipulation of Equation (7.6) yields such a form with

$$f(t) = \frac{w_R t}{w_r + t(w_R - w_r)}. \quad (7.8)$$

It also turns out that the line segments do map to line segments preserving the ordering of the points (Exercise 8), i.e., they do not get reordered or “torn.”

A byproduct of the transform taking line segments to line segments is that it takes the edges and vertices of a triangle to the edges and vertices of another triangle. Thus, it takes triangles to triangles and planes to planes.

7.5 Field-of-View

While we can specify any window using the (l, r, b, t) and n values, sometimes we would like to have a simpler system where we look through the center of the window. This implies the constraint that

$$\begin{aligned} l &= -r, \\ b &= -t. \end{aligned}$$

If we also add the constraint that the pixels are square, i.e., there is no distortion of shape in the image, then the ratio of r to t must be the same as the ratio of the number of horizontal pixels to the number of vertical pixels:

$$\frac{n_x}{n_y} = \frac{r}{t}.$$

规范卷中的分段。要看到这一点，请考虑线段

$$\mathbf{q} + t(\mathbf{Q} - \mathbf{q}).$$

当由 4×4 矩阵m变换时，它是一个具有可能变化的齐次坐标的点：

$$\mathbf{M}\mathbf{q} + t(\mathbf{MQ} - \mathbf{Mq}) \equiv \mathbf{r} + t(\mathbf{R} - \mathbf{r}).$$

匀化后的3D线段为

$$\frac{\mathbf{r} + t(\mathbf{R} - \mathbf{r})}{w_r + t(w_R - w_r)}. \quad (7.6)$$

如果方程 (7.6) 可以改写的形式

$$\frac{\mathbf{r}}{w_r} + f(t) \left(\frac{\mathbf{R}}{w_R} - \frac{\mathbf{r}}{w_r} \right), \quad (7.7)$$

然后所有均匀化的点都位于3D线上。式(7.6)的蛮力操纵产生这样的形式与

$$f(t) = \frac{w_R t}{w_r + t(w_R - w_r)}. \quad (7.8)$$

事实证明，线段确实映射到线段，保留了点的顺序（练习8），即它们不会被重新排序或“撕裂”。

将线段转换为线段的副产品是它将三角形的边和顶点转换为另一个三角形的边和顶点。因此，它需要三角形到三角形和平面到平面。

7.5 Field-of-View

虽然我们可以使用 (l, r, b, t) 和 n 值指定任何窗口，但有时我们希望有一个更简单的系统，我们可以通过窗口的中心查看。这意味着约束

$$\begin{aligned} l &= -r, \\ b &= -t. \end{aligned}$$

如果我们还添加像素为正方形的约束，即图像中没有形状的失真，那么 r 与 t 的比值必须与水平像素数与垂直像素数的比值相同： n_x/n_y

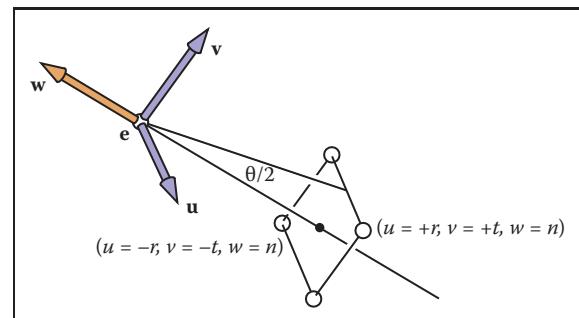


Figure 7.14. The field-of-view θ is the angle from the bottom of the screen to the top of the screen as measured from the eye.

Once n_x and n_y are specified, this leaves only one degree of freedom. That is often set using the *field-of-view* shown as θ in Figure 7.14. This is sometimes called the *vertical field-of-view* to distinguish it from the angle between left and right sides or from the angle between diagonal corners. From the figure we can see that

$$\tan \frac{\theta}{2} = \frac{t}{|n|}.$$

If n and θ are specified, then we can derive t and use code for the more general viewing system. In some systems, the value of n is hard-coded to some reasonable value, and thus we have one fewer degree of freedom.

Frequently Asked Questions

- Is orthographic projection ever useful in practice?

It is useful in applications where relative length judgments are important. It can also yield simplifications where perspective would be too expensive as occurs in some medical visualization applications.

- The tessellated spheres I draw in perspective look like ovals. Is this a bug?

No. It is correct behavior. If you place your eye in the same relative position to the screen as the virtual viewer has with respect to the viewport, then these ovals will look like circles because they themselves are viewed at an angle.

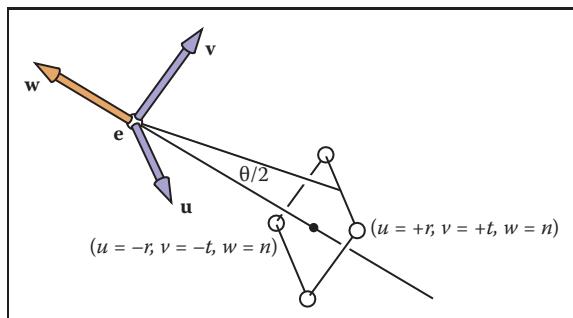


图7.14。视场θ是从眼睛测量的从屏幕底部到屏幕顶部的角度。

一旦指定了nx和ny，这留下一个自由度。这通常是用图7.14中θ所示的视场来设置的。这有时被称为垂直视场，以将其与左右两侧之间的角度或对角角之间的角度区分开来。从图中我们可以看出

$$\tan \frac{\theta}{2} = \frac{t}{|n|}.$$

如果指定了n和θ，那么我们可以推导出t并使用更通用的查看系统的代码。在一些系统中，n的值被硬编码为一些合理的值，因此我们有一个更少的自由度。

常见问题

- 正射投影在实践中有用吗？

它在相对长度判断很重要的应用中很有用。在某些医疗可视化应用中，透视过于昂贵的情况下，它也会产生简化。

- 我在透视中绘制的镶嵌球体看起来像椭圆形。这是bug吗？

非也。这是正确的行为。如果你把你的眼睛放在与屏幕相同的相对位置，就像虚拟观察者相对于视口一样，那么这些椭圆看起来就像圆圈，因为它们本身是在一个角度观看的。

- Does the perspective matrix take negative z values to positive z values with a reversed ordering? Doesn't that cause trouble?

Yes. The equation for transformed z is

$$z' = n + f - \frac{fn}{z}.$$

So $z = +\epsilon$ is transformed to $z' = -\infty$ and $z = -\epsilon$ is transformed to $z = \infty$. So any line segments that span $z = 0$ will be “torn” although all points will be projected to an appropriate screen location. This tearing is not relevant when all objects are contained in the viewing volume. This is usually assured by *clipping* to the view volume. However, clipping itself is made more complicated by the tearing phenomenon as is discussed in Chapter 8.

- The perspective matrix changes the value of the homogeneous coordinate. Doesn't that make the move and scale transformations no longer work properly?

Applying a translation to a homogeneous point we have

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} hx \\ hy \\ hz \\ h \end{bmatrix} = \begin{bmatrix} hx + ht_x \\ hy + ht_y \\ hz + ht_z \\ h \end{bmatrix} \xrightarrow{\text{homogenize}} \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{bmatrix}.$$

Similar effects are true for other transforms (see Exercise 5).

Notes

Most of the discussion of viewing matrices is based on information in *Real-Time Rendering* (Akenine-Möller et al., 2008), the *OpenGL Programming Guide* (Shreiner et al., 2004), *Computer Graphics* (Hearn & Baker, 1986), and *3D Game Engine Design* (Eberly, 2000).

Exercises

1. Construct the viewport matrix required for a system in which pixel coordinates count down from the top of the image, rather than up from the bottom.
2. Multiply the viewport and orthographic projection matrices, and show that the result can also be obtained by a single application of Equation (6.7).

*透视矩阵是否以反向排序将负 z 值变为正 z 值？这不会带来麻烦吗？

是的.变换后的 z 的方程为

$$z' = n + f - \frac{fn}{z}.$$

所以 $z=+\infty$ 转化为 $z'=-\infty$ ， $z=-\infty$ 转化为 $z=\infty$ 。因此，跨越 $z=0$ 的任何线段都将被“撕裂”，尽管所有点都将投影到适当的屏幕位置。当所有对象都包含在查看卷中时，这种撕裂不相关。这通常通过剪切到视图卷来保证。然而，正如第8章所讨论的那样，剪切本身由于撕裂现象而变得更加复杂。

*透视矩阵改变齐次坐标的值。这不会使移动和缩放转换不再正常工作吗？

将翻译应用于同质点我们有

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} hx \\ hy \\ hz \\ h \end{bmatrix} = \begin{bmatrix} hx + ht_x \\ hy + ht_y \\ hz + ht_z \\ h \end{bmatrix} \xrightarrow{\text{homogenize}} \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{bmatrix}.$$

类似的效果也适用于其他变换（请参阅练习5）。

Notes

大多数关于查看矩阵的讨论都是基于实时渲染中的信息（Akenine-Moller et al., 2008），OpenGL编程指南（Shreiner等人。, 2004），计算机图形学（Hearn & Baker, 1986）和3D游戏引擎设计（Eberly, 2000）。

Exercises

1. 构建系统所需的视口矩阵，其中像素坐标从图像的顶部向下计数，而不是从底部向上计数。
2. 乘以视口和正投影矩阵，并表明结果也可以通过等式（6.7）的单一应用获得。



3. Derive the third row of Equation (7.3) from the constraint that z is preserved for points on the near and far planes.
4. Show algebraically that the perspective matrix preserves order of z values within the view volume.
5. For a 4×4 matrix whose top three rows are arbitrary and whose bottom row is $(0, 0, 0, 1)$, show that the points $(x, y, z, 1)$ and (hx, hy, hz, h) transform to the same point after homogenization.
6. Verify that the form of \mathbf{M}_p^{-1} given in the text is correct.
7. Verify that the full perspective to canonical matrix $\mathbf{M}_{\text{projection}}$ takes (r, t, n) to $(1, 1, 1)$.
8. Write down a perspective matrix for $n = 1, f = 2$.
9. For the point $\mathbf{p} = (x, y, z, 1)$, what are the homogenized and unhomogenized results for that point transformed by the perspective matrix in Exercise 6?
10. For the eye position $\mathbf{e} = (0, 1, 0)$, a gaze vector $\mathbf{g} = (0, -1, 0)$, and a view-up vector $\mathbf{t} = (1, 1, 0)$, what is the resulting orthonormal \mathbf{uvw} basis used for coordinate rotations?
11. Show, that for a perspective transform, line segments that start in the view volume do map to line segments in the canonical volume after homogenization. Further, show that the relative ordering of points on the two segments is the same. *Hint:* Show that the $f(t)$ in Equation (7.8) has the properties $f(0) = 0, f(1) = 1$, the derivative of f is positive for all $t \in [0, 1]$, and the homogeneous coordinate does not change sign.

3. 从为近平面和远平面上的点保留 z 的约束导出方程(7.3)的第三行。

4. 以代数方式显示透视矩阵保留视图体积内 z 值的顺序。

5. 对于一个 4×4 矩阵，其顶部三行是任意的，其底部行是 $(0 \ 0 \ 0 \ 1)$ 显示点 $(x, y, z, 1)$ 和 (hx, hy, hz, h) 在均匀化后变换到同一点。

6. 验证 $\mathbf{M}^{-1}\mathbf{p}$ 的形式 文中给出的是正确的。

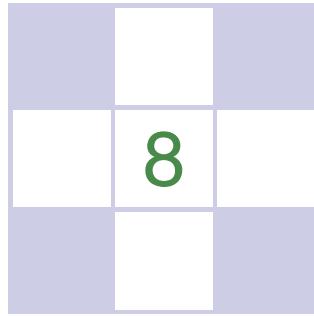
7. 验证规范矩阵 \mathbf{M} 投影的全视角取 $(r \ t \ n)$ 到 $(1 \ 1 \ 1)$ 。

8. 写下 $n=1, f=2$ 的透视矩阵。

9. 对于点 $\mathbf{p}=(x \ y \ z \ 1)$ ，由练习6中的透视矩阵变换的该点的均匀化和非均匀化结果是什么？

10. 对于眼睛位置 $\mathbf{e}=(0 \ 1 \ 0)$ ，一个注视向量 $\mathbf{g}=(0 \ -1 \ 0)$ 和一个viewup向量 $\mathbf{t}=(1 \ 1 \ 0)$ ，什么是产生的正交 \mathbf{uvw} 基用于坐标旋转？

11. 显示，对于透视变换，在视图体积中开始的线段在均匀化后确实映射到规范体积中的线段。另外，表明两段上的点的相对顺序是相同的。提示：表明方程 (7.8) 中的 $f(t)$ 具有属性 $f(0) = 0, f(1) = 1$ ， f 的导数对所有 $t \in [0, 1]$ 都是正的，并且齐次坐标不改变sign。



The Graphics Pipeline

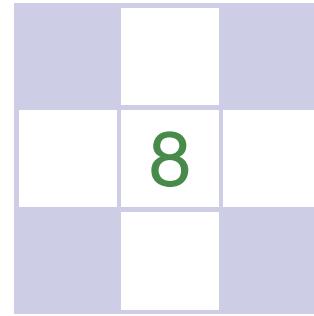
The previous several chapters have established the mathematical scaffolding we need to look at the second major approach to rendering: drawing objects one by one onto the screen, or *object-order rendering*. Unlike in ray tracing, where we consider each pixel in turn and find the objects that influence its color, we'll now instead consider each geometric object in turn and find the pixels that it could have an effect on. The process of finding all the pixels in an image that are occupied by a geometric primitive is called *rasterization*, so object-order rendering can also be called rendering by rasterization. The sequence of operations that is required, starting with objects and ending by updating pixels in the image, is known as the *graphics pipeline*.

Object-order rendering has enjoyed great success because of its efficiency. For large scenes, management of data access patterns is crucial to performance, and making a single pass over the scene visiting each bit of geometry once has significant advantages over repeatedly searching the scene to retrieve the objects required to shade each pixel.

The title of this chapter suggests that there is only one way to do object-order rendering. Of course this isn't true—two quite different examples of graphics pipelines with very different goals are the hardware pipelines used to support interactive rendering via APIs like OpenGL and Direct3D and the software pipelines used in film production, supporting APIs like RenderMan. Hardware pipelines must run fast enough to react in real time for games, visualizations, and user interfaces. Production pipelines must render the highest quality animation and visual effects possible and scale to enormous scenes, but may take much

Any graphics system has one or more types of “primitive object” that it can handle directly, and more complex objects are converted into these “primitives.” Triangles are the most often used primitive.

Rasterization-based systems are also called *scanline renderers*.



图形管道

前面的几章已经建立了数学脚手架，我们需要看第二种主要的渲染方法：将对象一个接一个地绘制到屏幕上，或者对象顺序渲染。在光线追踪中，我们依次考虑每个像素并找到影响其颜色的对象，而在光线追踪中，我们将依次考虑每个几何对象并找到可能对其产生影响的像素。查找图像中被占用的所有像素的过程

几何图元称为光栅化，因此对象顺序渲染也可以称为光栅化渲染。所需的操作顺序

从对象开始到更新图像中的像素结束，称为图形管道。

对象顺序渲染因其效率而获得了巨大的成功。

对于大型场景，数据访问模式的管理对于性能至关重要，并且使在场景上一次访问几何体的每个位具有显着优势，而不是重复搜索场景以检索阴影每个像素所需的对象。

本章的标题表明，只有一种方法可以实现对象顺序渲染。当然，这不是真的—两个完全不同的目标的图形管道的例子是用于通过OpenGL和Direct3D等API支持交互式渲染的硬件管道以及电影制作中使用的软件管道，支持像RenderMan这样的API。硬件管道必须运行得足够快，以实时响应游戏、可视化和用户界面。制作管道必须尽可能呈现最高质量的动画和视觉效果，并扩展到巨大的场景，但可能需要很多

任何图形系统都有一种或多种类型的“基元对象”，它可以直接处理，更复杂的对象被转换成这些“基元”。“三角形是最常用的基元。”

Rasterization-based systems are also called *scanline renderers*.

more time to do so. Despite the different design decisions resulting from these divergent goals, a remarkable amount is shared among most, if not all, pipelines, and this chapter attempts to focus on these common fundamentals, erring on the side of following the hardware pipelines more closely.

The work that needs to be done in object-order rendering can be organized into the task of rasterization itself, the operations that are done to geometry before rasterization, and the operations that are done to pixels after rasterization. The most common geometric operation is applying matrix transformations, as discussed in the previous two chapters, to map the points that define the geometry from object space to screen space, so that the input to the rasterizer is expressed in pixel coordinates, or *screen space*. The most common pixelwise operation is *hidden surface removal* which arranges for surfaces closer to the viewer to appear in front of surfaces farther from the viewer. Many other operations also can be included at each stage, thereby achieving a wide range of different rendering effects using the same general process.

For the purposes of this chapter, we'll discuss the graphics pipeline in terms of four stages (Figure 8.1). Geometric objects are fed into the pipeline from an interactive application or from a scene description file, and they are always described by sets of vertices. The vertices are operated on in the *vertex-processing stage*, then the primitives using those vertices are sent to the *rasterization stage*. The rasterizer breaks each primitive into a number of *fragments*, one for each pixel covered by the primitive. The fragments are processed in the *fragment processing stage*, and then the various fragments corresponding to each pixel are combined in the *fragment blending stage*.

We'll begin by discussing rasterization, then illustrate the purpose of the geometric and pixel-wise stages by a series of examples.

8.1 Rasterization

Rasterization is the central operation in object-order graphics, and the *rasterizer* is central to any graphics pipeline. For each primitive that comes in, the rasterizer has two jobs: it *enumerates* the pixels that are covered by the primitive and it *interpolates* values, called attributes, across the primitive—the purpose for these attributes will be clear with later examples. The output of the rasterizer is a set of *fragments*, one for each pixel covered by the primitive. Each fragment “lives” at a particular pixel and carries its own set of attribute values.

In this chapter, we will present rasterization with a view toward using it to render three-dimensional scenes. The same rasterization methods are used to draw

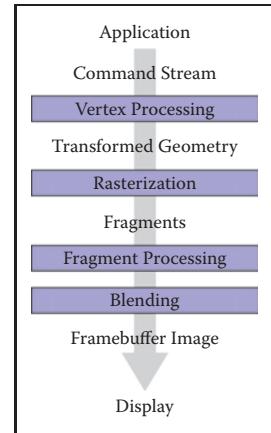
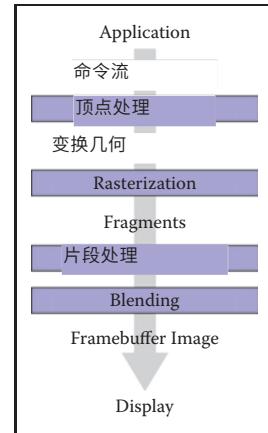


Figure 8.1. The stages of a graphics pipeline.

更多的时间这样做。尽管这些不同的目标导致了不同的设计决策，但大多数（如果不是全部的话）管道之间共享了一个显着的数量，本章试图将重点放在这些共同的基础上，在更密切地遵循硬件管道方面犯了错误。



图形管道的阶段。

在对象顺序渲染中需要完成的工作可以组织成光栅化本身的任务，在光栅化之前对几何体所做的操作，以及在光栅化之后对像素所做的操作。最常见的几何操作是应用矩阵变换，如前两章所讨论的那样，将定义几何的点从对象空间映射到屏幕空间，以便向光栅化器的输入以像素坐标或屏幕空间最常用的pixelwise操作是隐藏表面移除，它安排更靠近查看器的表面出现在距离查看器更远的表面的前面。许多其他操作也可以在每个阶段进行clustered，从而使用相同的一般过程实现广泛的不同渲染效果。

为了本章的目的，我们将从四个阶段来讨论图形管道（图8.1）。几何对象从一个内部输入到管道中

活动应用程序或来自场景描述文件，它们总是由顶点集描述。在顶点处理阶段对顶点进行操作，然后使用这些顶点的基本元被发送到光栅化阶段。光栅化器将每个基本元分成若干片段，每个片段对应基本元复盖的像素。在片段处理阶段对片段进行处理，然后在片段混合阶段对每个像素对应的各个片段进行组合。

我们将首先讨论光栅化，然后通过一系列示例说明几何度量和像素级阶段的目的。

8.1 Rasterization

光栅化是对象顺序图形中的中心操作，而光栅化器是任何图形管道的中心。对于每个进来的基元，光栅化器有两个工作：它枚举基元复盖的像素，并在基元中插值值（称为属性）—这些属性的目的将在以后的示例中清楚。光栅化器的输出是一组片段，一个用于基元复盖的每个像素。每个片段“生活”在一个特定的像素，并携带自己的一组属性值。

在本章中，我们将介绍光栅化，以便使用它来渲染三维场景。相同的光栅化方法用于绘制



lines and shapes in 2D as well—although it is becoming more and more common to use the 3D graphics system “under the covers” to do all 2D drawing.

8.1.1 Line Drawing

Most graphics packages contain a line drawing command that takes two endpoints in screen coordinates (see Figure 3.10) and draws a line between them. For example, the call for endpoints (1,1) and (3,2) would turn on pixels (1,1) and (3,2) and fill in one pixel between them. For general screen coordinate endpoints (x_0, y_0) and (x_1, y_1) , the routine should draw some “reasonable” set of pixels that approximates a line between them. Drawing such lines is based on line equations, and we have two types of equations to choose from: implicit and parametric. This section describes the approach using implicit lines.

Line Drawing Using Implicit Line Equations

The most common way to draw lines using implicit equations is the *midpoint algorithm* (Pitteway (1967); van Aken and Novak (1985)). The midpoint algorithm ends up drawing the same lines as the *Bresenham algorithm* (Bresenham, 1965) but it is somewhat more straightforward.

The first thing to do is find the implicit equation for the line as discussed in Section 2.5.2:

$$f(x, y) \equiv (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0. \quad (8.1)$$

We assume that $x_0 \leq x_1$. If that is not true, we swap the points so that it is true. The slope m of the line is given by

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

The following discussion assumes $m \in (0, 1]$. Analogous discussions can be derived for $m \in (-\infty, -1]$, $m \in (-1, 0]$, and $m \in (1, \infty)$. The four cases cover all possibilities.

For the case $m \in (0, 1]$, there is more “run” than “rise,” i.e., the line is moving faster in x than in y . If we have an API where the y -axis points downward, we might have a concern about whether this makes the process harder, but, in fact, we can ignore that detail. We can ignore the geometric notions of “up” and “down,” because the algebra is exactly the same for the two cases. Cautious readers can confirm that the resulting algorithm works for the y -axis downward case. The key assumption of the midpoint algorithm is that we draw the thinnest line possible

Even though we often use integer-valued endpoints for examples, it's important to properly support arbitrary endpoints.



2d中的线条和形状也是如此—尽管使用3d图形系统“在封面下”进行所有2D绘图变得越来越普遍。

8.1.1 线条画

大多数图形包都包含一个线绘制命令，该命令在屏幕coordinates中获取两个端点（见图3.10）并在它们之间绘制一条线。例如，对端点(1 1)和(3 2)的调用将打开像素(1 1)和(3 2)，并在它们之间填充一个像素。对于一般的屏幕坐标端点 $(x_0 y_0)$ 和 $(x_1 y_1)$ ，例程应该绘制一些“合理”的像素集，以近似它们之间的一条线。绘制这样的线是基于线方程，和

即使我们经常使用整数端点例如，正确支持任意端点是很重要的。

我们有两种类型的方程可供选择：隐式和参数式。本节介绍使用隐式行的方法。

使用隐式线方程绘制线

使用隐式方程绘制线条的最常见方法是中点algorithm (Pitteway (1967) ;vanAken and Novak (1985))。中点算法最终绘制与Bresenham算法 (Bresenham, 1965) 相同的线条，但它更直接。

首先要做的是找到线的隐式方程，如
Section 2.5.2:

$$f(x, y) \equiv (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0. \quad (8.1)$$

我们假设 $x_0 \leq x_1$ 。如果这不是真的，我们交换点，使它是真的。
线的斜率 m 由下式给出

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

下面的讨论假定 $m \in (0, 1]$ 。对于 $m \in (-\infty, -1]$ 、 $m \in (-1, 0]$ 和 $m \in (1, \infty)$ 可以推导出类似的讨论。这四种情况涵盖了所有的可能性。

对于 $m \in (0, 1]$ 的情况，“运行”比“上升”更多，即线在 x 中的移动速度比在 y 中的移动速度更快。如果我们有一个Y轴向下指向的API，我们可能会担心这是否会使过程变得更加困难，但实际上，我们可以忽略这个细节。我们可以忽略“向上”和“向下”的几何概念，因为这两种情况的代数完全相同。谨慎的读者可以确认所得算法适用于y轴向下的情况。中点算法的关键假设是我们绘制尽可能细的线

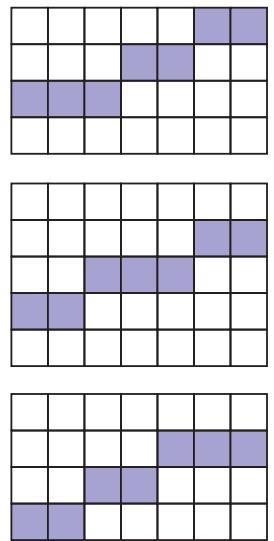


Figure 8.2. Three “reasonable” lines that go seven pixels horizontally and three pixels vertically.

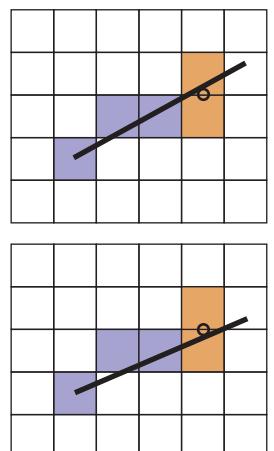


Figure 8.3. Top: the line goes above the midpoint so the top pixel is drawn. Bottom: the line goes below the midpoint so the bottom pixel is drawn.

that has no gaps. A diagonal connection between two pixels is not considered a gap.

As the line progresses from the left endpoint to the right, there are only two possibilities: draw a pixel at the same height as the pixel drawn to its left, or draw a pixel one higher. There will always be exactly one pixel in each column of pixels between the endpoints. Zero would imply a gap, and two would be too thick a line. There may be two pixels in the same row for the case we are considering; the line is more horizontal than vertical so sometimes it will go right, and sometimes up. This concept is shown in Figure 8.2, where three “reasonable” lines are shown, each advancing more in the horizontal direction than in the vertical direction.

The midpoint algorithm for $m \in (0, 1]$ first establishes the leftmost pixel and the column number (x-value) of the rightmost pixel and then loops horizontally establishing the row (y-value) of each pixel. The basic form of the algorithm is:

```

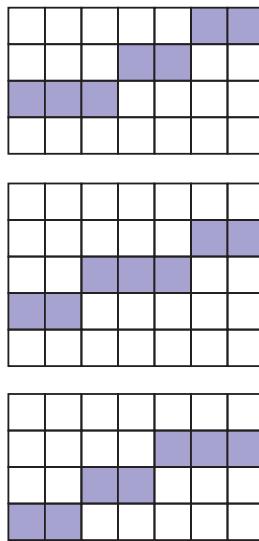
 $y = y_0$ 
for  $x = x_0$  to  $x_1$  do
    draw( $x, y$ )
    if (some condition) then
         $y = y + 1$ 

```

Note that x and y are integers. In words this says, “keep drawing pixels from left to right and sometimes move upward in the y -direction while doing so.” The key is to establish efficient ways to make the decision in the **if** statement.

An effective way to make the choice is to look at the *midpoint* of the line between the two potential pixel centers. More specifically, the pixel just drawn is pixel (x, y) whose center in real screen coordinates is at (x, y) . The candidate pixels to be drawn to the right are pixels $(x + 1, y)$ and $(x + 1, y + 1)$. The midpoint between the centers of the two candidate pixels is $(x + 1, y + 0.5)$. If the line passes below this midpoint we draw the bottom pixel, and otherwise we draw the top pixel (Figure 8.3).

To decide whether the line passes above or below $(x + 1, y + 0.5)$, we evaluate $f(x, y + 0.5)$ in Equation (8.1). Recall from Section 2.5.1 that $f(x, y) = 0$ for points (x, y) on the line, $f(x, y) > 0$ for points on one side of the line, and $f(x, y) < 0$ for points on the other side of the line. Because $-f(x, y) = 0$ and $f(x, y) = 0$ are both perfectly good equations for the line, it is not immediately clear whether $f(x, y)$ being positive indicates that (x, y) is above the line, or whether it is below. However, we can figure it out; the key term in Equation (8.1) is the y term $(x_1 - x_0)y$. Note that $(x_1 - x_0)$ is definitely positive because $x_1 > x_0$. This means that as y increases, the term $(x_1 - x_0)y$ gets larger (i.e., more positive or less negative). Thus, the case $f(x, +\infty)$ is definitely positive, and definitely above the line, implying points above the line are all positive. Another



这是没有差距的。两个像素之间的对角线连接不被认为是间隙。

随着线从左端点向右前进，只有两种可能性：在其左侧绘制的像素相同的高度绘制像素，或者在更高的位置绘制像素。端点之间的每列像素中总是恰好有一个像素。零将意味着一个差距，两个将是太粗的一条线。对于我们正在考虑的情况，同一行中可能有两个像素；该行比垂直更水平，所以有时它会向右，有时会向上。这个概念如图8.2所示，其中显示了三条“合理”线，每条线在水平方向上比在垂直方向上前进更多。

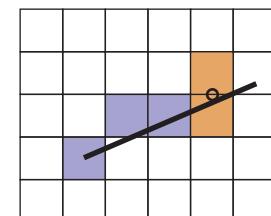
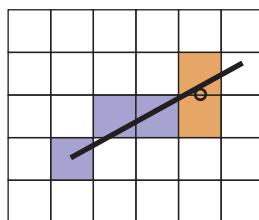
$M [0, 1]$ 的中点算法首先建立最左边的像素和最右边的像素的列号 (x值)，然后循环水平地建立每个像素的行 (y值)。算法的基本形式是：

```

 $y=y_0$  for  $x=x_0$  to  $x_1$  do
    draw( $x, y$ )
    w(x y) if (some condition)
         $heny=y+1$ 

```

三条“理性”线，水平方向七个pixels，垂直方向三个pixels。



顶部：线高于中点，因此绘制顶部像素。
底部：线低于中点，因此底部像素被绘制。

请注意， x 和 y 是整数。换句话说，这说，“保持从左到右绘制像素，有时在 y 方向向上移动，同时这样做。”关键是建立有效的方法来做出if声明中的决定。

做出选择的有效方法是看两个潜在像素中心之间的线的中点。更具体地说，刚刚绘制的像素是其在真实屏幕坐标中的中心在 (x, y) 处的像素 (x, y) 。要向右绘制的候选像素是像素 $(x+1, y)$ 和 $(x+1, y+1)$ 。两个候选像素点的中心之间的中点为 $(x+1, y+0.5)$ 。如果线低于这个中点，我们绘制底部像素，否则我们绘制顶部像素（图8.3）。

来决定该线是通过上方还是下方 $(x+1, y+0.5)$ ，我们评估 $f(x, y+0.5)$ 在等式(8.1)中。从第2.5.1节中回想一下，对于线上的点 (x, y) ， $f(x, y) = 0$ ，对于线的一侧的点， $f(x, y) > 0$ ，对于线的另一侧的点， $f(x, y) < 0$ 。因为 $-f(x, y) = 0$ 和

$f(x, y) = 0$ 都是该线的完美方程，目前尚不清楚 $f(x, y)$ 是否为正表示 (x, y) 在线上方，或者是否低于该线。但是，我们可以弄清楚：方程 (8.1) 中的关键项是 y 项 $(x_1 - x_0)y$ 。请注意， $(x_1 - x_0)$ 肯定是正的，因为 $x_1 > x_0$ 。这意味着随着 y 的增加，项 $(x_1 - x_0)y$ 变得更大（即，更多正或更少负）。因此，情况 $f(x, +\infty)$ 肯定是正的，并且肯定在线上方，意味着在线上方的点都是正的。另一个



way to look at it is that the y component of the gradient vector is positive. So above the line, where y can increase arbitrarily, $f(x, y)$ must be positive. This means we can make our code more specific by filling in the *if* statement:

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

The above code will work nicely for lines of the appropriate slope (i.e., between zero and one). The reader can work out the other three cases which differ only in small details.

If greater efficiency is desired, using an *incremental* method can help. An incremental method tries to make a loop more efficient by reusing computation from the previous step. In the midpoint algorithm as presented, the main computation is the evaluation of $f(x + 1, y + 0.5)$. Note that inside the loop, after the first iteration, either we already evaluated $f(x - 1, y + 0.5)$ or $f(x - 1, y - 0.5)$ (Figure 8.4). Note also this relationship:

$$\begin{aligned}f(x + 1, y) &= f(x, y) + (y_0 - y_1) \\f(x + 1, y + 1) &= f(x, y) + (y_0 - y_1) + (x_1 - x_0).\end{aligned}$$

This allows us to write an incremental version of the code:

```
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)
```

This code should run faster since it has little extra setup cost compared to the non-incremental version (that is not always true for incremental algorithms), but it may accumulate more numeric error because the evaluation of $f(x, y + 0.5)$ may be composed of many adds for long lines. However, given that lines are rarely longer than a few thousand pixels, such an error is unlikely to be critical. Slightly longer setup cost, but faster loop execution, can be achieved by storing $(x_1 - x_0) + (y_0 - y_1)$ and $(y_0 - y_1)$ as variables. We might hope a good compiler would do that for us, but if the code is critical, it would be wise to examine the results of compilation to make sure.

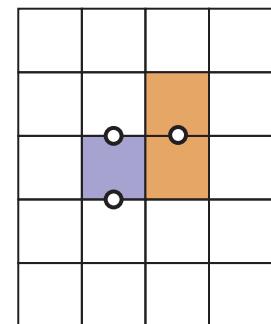


Figure 8.4. When using the decision point shown between the two orange pixels, we just drew the blue pixel, so we evaluated f at one of the two left points shown.

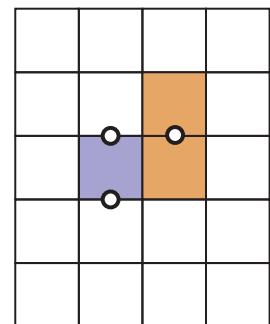


看它的方法是梯度向量的 y 分量是正的。所以在线上方，其中 y 可以任意增加， $f(x, y)$ 必须为正。这意味着我们可以通过填写*if*语句来使我们的代码更具体：

```
如果f(x+1, y+0.5)<0则y
=y+1
```

上面的代码将很好地适用于适当斜率的行（即，在零到一之间）。读者可以计算出其他三种仅在小细节上不同的情况。

如果需要更高的效率，使用增量方法可以提供帮助。增量方法试图通过重用上一步的计算来提高循环的效率。在所提出的中点算法中主要的计算是对 $f(x+1, y+0.5)$ 的评价。请注意，在循环内部，在第一次迭代之后，要么我们已经评估了 $f(x, y+0.5)$ 或 $f(x-1, y+0.5)$ （图8.4）。还要注意这种关系：



当使用所示的决策点补间两个橙色像素时，我们只是绘制了蓝色像素，因此我们在所示的两个左点之一处评估了 f 。

$$\begin{aligned}f(x + 1, y) &= f(x, y) + (y_0 - y_1) \\f(x + 1, y + 1) &= f(x, y) + (y_0 - y_1) + (x_1 - x_0).\end{aligned}$$

这允许我们编写代码的增量版本：

```
y=y0
d=f(x0+1, y0+0.5)对于x=x0
到x1做draw(x, y)如果d<0则y=y+1
d=d+(x1-x0)+(y0-y1)else
```

$$d = d + (y_0 - y_1)$$

此代码应该运行得更快，因为与非增量版本相比，它几乎没有额外的设置成本（对于增量算法来说并不总是如此），但是它可能会累积更多的数字错误，因为 d 可以由许多添加长线组成。然而，鉴于线很少长于几千像素，这样的误差不太可能是关键的。稍长的设置成本，但更快的循环执行，可以通过存储 $(x_1 - x_0) + (y_0 - y_1)$ 和 $(y_0 - y_1)$ 作为变量来实现。我们可能希望一个好的编译器能为我们做到这一点，但是如果代码是关键的，那么检查编译的结果以确保是明智的。

8.1.2 Triangle Rasterization

We often want to draw a 2D triangle with 2D points $\mathbf{p}_0 = (x_0, y_0)$, $\mathbf{p}_1 = (x_1, y_1)$, and $\mathbf{p}_2 = (x_2, y_2)$ in screen coordinates. This is similar to the line drawing problem, but it has some of its own subtleties. As with line drawing, we may wish to interpolate color or other properties from values at the vertices. This is straightforward if we have the barycentric coordinates (Section 2.7). For example, if the vertices have colors \mathbf{c}_0 , \mathbf{c}_1 , and \mathbf{c}_2 , the color at a point in the triangle with barycentric coordinates (α, β, γ) is

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

This type of interpolation of color is known in graphics as *Gouraud* interpolation after its inventor (Gouraud, 1971).

Another subtlety of rasterizing triangles is that we are usually rasterizing triangles that share vertices and edges. This means we would like to rasterize adjacent triangles so there are no holes. We could do this by using the midpoint algorithm to draw the outline of each triangle and then fill in the interior pixels. This would mean adjacent triangles both draw the same pixels along each edge. If the adjacent triangles have different colors, the image will depend on the order in which the two triangles are drawn. The most common way to rasterize triangles that avoids the order problem and eliminates holes is to use the convention that pixels are drawn if and only if their centers are inside the triangle, i.e., the barycentric coordinates of the pixel center are all in the interval $(0, 1)$. This raises the issue of what to do if the center is exactly on the edge of the triangle. There are several ways to handle this as will be discussed later in this section. The key observation is that barycentric coordinates allow us to decide whether to draw a pixel and what color that pixel should be if we are interpolating colors from the vertices. So our problem of rasterizing the triangle boils down to efficiently finding the barycentric coordinates of pixel centers (Pineda, 1988). The brute-force rasterization algorithm is:

```

for all  $x$  do
  for all  $y$  do
    compute  $(\alpha, \beta, \gamma)$  for  $(x, y)$ 
    if  $(\alpha \in [0, 1] \text{ and } \beta \in [0, 1] \text{ and } \gamma \in [0, 1])$  then
       $\mathbf{c} = \alpha\mathbf{c}_0 + \beta\mathbf{c}_1 + \gamma\mathbf{c}_2$ 
      drawpixel  $(x, y)$  with color  $\mathbf{c}$ 

```

The rest of the algorithm limits the outer loops to a smaller set of candidate pixels and makes the barycentric computation efficient.

8.1.2 Triangle Rasterization

我们经常想在屏幕坐标中绘制一个2D点 $p_0= (x_0, y_0)$, $p_1= (x_1, y_1)$ 和 $p_2= (x_2, y_2)$ 的2d三角形。这类似于线条绘制问题，但它有一些自己的微妙之处。与线条绘制一样，我们可能希望从顶点处的值内插颜色或其他属性。如果我们有重心坐标（第2.7节），这很简单。例如，如果顶点具有颜色 c_0 、 c_1 和 c_2 ，则三角形中具有重心坐标(α 、 β 、 γ)的点处的颜色为

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

这种类型的颜色插值在图形中被称为Gouraud插值（Gouraud, 1971）。

光栅化三角形的另一个微妙之处是，我们通常光栅化共享顶点和边的三个角度。这意味着我们想栅格化相邻的三角形，这样就没有洞了。我们可以通过使用中点算法绘制每个三角形的轮廓，然后填充内部像素来做到这一点。这意味着相邻的三角形都沿着每条边绘制相同的像素。如果相邻三角形具有不同的颜色，则图像将取决于绘制两个三角形的顺序。避免顺序问题并消除孔洞的栅格化triangles的最常见方法是使用当且仅当它们的中心在三角形内部时绘制像素的约定，即像素中心的重心坐标都在间隔（这就提出了如果中心正好在三角形的边缘上该怎么办的问题。有几种方法可以处理这个问题，这将在本节后面讨论。关键的观察是，重心坐标允许我们决定是否绘制像素以及如果我们从顶点内插颜色，该像素应该是什么颜色。因此，我们对三角形进行光栅化的问题归结为有效地找到像素中心的重心坐标（Pineda, 1988）。蛮力光栅化算法为：

为所有x做为
所有y做
计算(α β γ)for(x y)if($\alpha \in [0, 1]$ and $\beta \in [0, 1]$ an
 $dy \in [0, 1]$)thenc= $\alpha c_0 + \beta c_1 + \gamma c_2$ drawpixel(x
 y)withcolorc

该算法的其余部分将外部循环限制为较小的候选像素集，并使重心计算效率更高。



We can add a simple efficiency by finding the bounding rectangle of the three vertices and only looping over this rectangle for candidate pixels to draw. We can compute barycentric coordinates using Equation (2.32). This yields the algorithm:

```

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

Here f_{ij} is the line given by Equation (8.1) with the appropriate vertices:

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

Note that we have exchanged the test $\alpha \in (0, 1)$ with $\alpha > 0$ etc., because if all of α, β, γ are positive, then we know they are all less than one because $\alpha + \beta + \gamma = 1$. We could also compute only two of the three barycentric variables

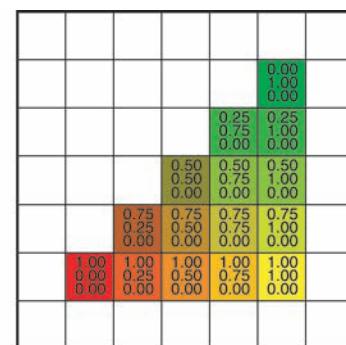
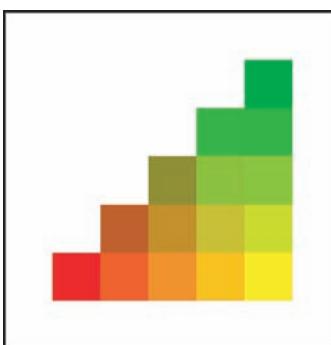


Figure 8.5. A colored triangle with barycentric interpolation. Note that the changes in color components are linear in each row and column as well as along each edge. In fact it is constant along every line, such as the diagonals, as well.



我们可以通过找到三个顶点的边界矩形并仅在该矩形上循环以供候选像素绘制来增加一个简单的效率。我们可以用方程(2.32)计算重心坐标。这产生了算法：

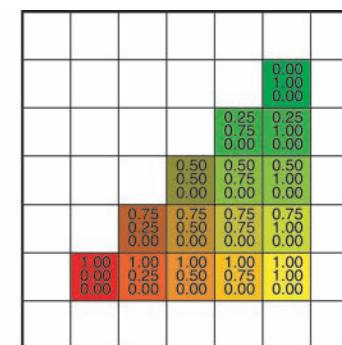
```

xmin=floor(xi)xmax=
ceiling(xi)ymin=floor(
yi)ymax=ceiling(yi)fo
ry=ymintoymaxdo
对于x=xmin到xmaxdo
     $\alpha=f_{12}(x, y)/f_{12}(x_0, y_0)$   $\beta=f_{20}(x, y)/f_{20}(x_1, y_1)$   $\gamma=f_{01}(x, y)/f_{01}(x_2, y_2)$  如果 ( $\alpha>0$ 且 $\beta>0$ 且 $\gamma>0$ ) 则  $c=\alpha c_0+\beta c_1+\gamma c_2$ 
    drawpixel  $(x, y)$  与颜色c
```

这里 f_{ij} 是方程(8.1)给出的具有适当顶点的线：

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

请注意，我们已经将测试 $\alpha \in (0, 1)$ 与 $\alpha > 0$ 等交换。, 因为如果所有的 α, β, γ 都是正的，那么我们知道它们都小于一，因为 $\alpha + \beta + \gamma = 1$ 。我们也只能计算三个重心变量中的两个



带有重心插值的彩色三角形。请注意，颜色分量的变化在每行和每列以及沿每个边缘都是线性的。事实上，它沿着每一条线都是恒定的，例如对角线。

and get the third from that relation, but it is not clear that this saves computation once the algorithm is made incremental, which is possible as in the line drawing algorithms; each of the computations of α , β , and γ does an evaluation of the form $f(x, y) = Ax + By + C$. In the inner loop, only x changes, and it changes by one. Note that $f(x+1, y) = f(x, y) + A$. This is the basis of the incremental algorithm. In the outer loop, the evaluation changes for $f(x, y)$ to $f(x, y+1)$, so a similar efficiency can be achieved. Because α , β , and γ change by constant increments in the loop, so does the color c . So this can be made incremental as well. For example, the red value for pixel $(x+1, y)$ differs from the red value for pixel (x, y) by a constant amount that can be precomputed. An example of a triangle with color interpolation is shown in Figure 8.5.

Dealing with Pixels on Triangle Edges

We have still not discussed what to do for pixels whose centers are exactly on the edge of a triangle. If a pixel is exactly on the edge of a triangle, then it is also on the edge of the adjacent triangle if there is one. There is no obvious way to award the pixel to one triangle or the other. The worst decision would be to not draw the pixel because a hole would result between the two triangles. Better, but still not good, would be to have both triangles draw the pixel. If the triangles are transparent, this will result in a double-coloring. We would really like to award the pixel to exactly one of the triangles, and we would like this process to be simple; which triangle is chosen does not matter as long as the choice is well defined.

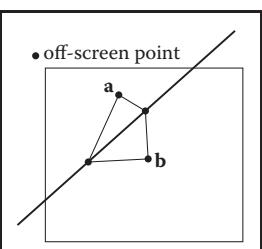


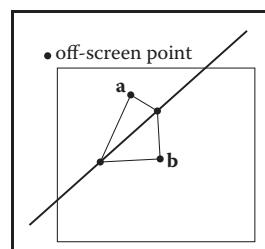
Figure 8.6. The off-screen point will be on one side of the triangle edge or the other. Exactly one of the non-shared vertices a and b will be on the same side.

One approach is to note that any off-screen point is definitely on exactly one side of the shared edge and that is the edge we will draw. For two non-overlapping triangles, the vertices not on the edge are on opposite sides of the edge from each other. Exactly one of these vertices will be on the same side of the edge as the off-screen point (Figure 8.6). This is the basis of the test. The test if numbers p and q have the same sign can be implemented as the test $pq > 0$, which is very efficient in most environments.

Note that the test is not perfect because the line through the edge may also go through the off-screen point, but we have at least greatly reduced the number of problematic cases. Which off-screen point is used is arbitrary, and $(x, y) = (-1, -1)$ is as good a choice as any. We will need to add a check for the case of a point exactly on an edge. We would like this check not to be reached for common cases, which are the completely inside or outside tests. This suggests:

$$\begin{aligned}x_{\min} &= \text{floor}(x_i) \\x_{\max} &= \text{ceiling}(x_i) \\y_{\min} &= \text{floor}(y_i)\end{aligned}$$

并从该关系中获得第三个，但目前尚不清楚，一旦算法是增量的，这就可节省计算，这在画线算法中是可能的； α ， β 和 γ 的每个计算都对形式 $f(x, y) = Ax + By + C$ 进行评估。在内循环中，只有 x 改变，它改变一个。注意 $f(x+1, y) = f(x, y) + A$ 。这是增量算法的基础。在外环中， $f(x, y)$ 的评估变化为 $f(x, y+1)$ ，因此可以实现类似的效率。因为 α ， β 和 γ 在循环中通过恒定增量变化，所以颜色 c 也是如此。所以这也可能是增量的。例如， $\text{pixel}(x+1, y)$ 的红色值与 $\text{pixel}(x, y)$ 的红色值相差一个可以预算的恒定量。带有颜色插值的三角形示例如图8.5所示。



离屏点将在三角形边缘的一侧或另一侧。非共享顶点a和b中的恰好一个将在同一侧。

处理三角形边上的像素

我们还没有讨论如何处理中心正好位于三角形边缘的像素。如果一个像素正好在一个三角形的边缘，那么它也在相邻三角形的边缘，如果有两个。没有明显的方法将像素授予一个三角形或另一个三角形。最糟糕的决定是不绘制像素，因为在两个三角形之间会产生一个洞。更好，但仍然不好，将是让两个三角形绘制像素。如果三角形是透明的，这将导致双重着色。我们真的希望将像素授予其中一个三角形，并且我们希望这个过程很简单；选择哪个三角形并不重要，只要选择是明确的。

一种方法是注意任何屏幕外的点肯定正好位于共享边缘的一侧，这就是我们将要绘制的边缘。对于两个不重叠的三角形，不在边上的顶点在边彼此相对的两侧。这些顶点中的恰好一个将与屏幕外点位于边缘的同一侧（图8.6）。这是测试的基础。如果数字 p 和 q 具有相同的符号，则测试可以实现为测试 $pq > 0$ ，这在大多数环境中非常有效。

请注意，测试并不完美，因为通过边缘的线也可能经过屏幕外的点，但我们至少大大减少了有问题的案例数量。使用哪个屏幕外点是任意的，并且 $(x, y) = (1, 1)$ 与任何一样好的选择。我们需要添加一个检查的情况下，一个点正好在一个边缘。我们希望这个检查不要达到常见的情况，这是完全内部或外部测试。这表明：

$$\begin{aligned}x_{\min} &= \text{地板}(x_i)x_m \\ax &= \text{天花板}(x_i)y_m \\n &= \text{地板}(y_i)\end{aligned}$$

```

 $y_{\max} = \text{ceiling}(y_i)$ 
 $f_\alpha = f_{12}(x_0, y_0)$ 
 $f_\beta = f_{20}(x_1, y_1)$ 
 $f_\gamma = f_{01}(x_2, y_2)$ 
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
           $c = \alpha c_0 + \beta c_1 + \gamma c_2$ 
          drawpixel  $(x, y)$  with color  $c$ 

```

We might expect that the above code would work to eliminate holes and double-draws only if we use exactly the same line equation for both triangles. In fact, the line equation is the same only if the two shared vertices have the same order in the draw call for each triangle. Otherwise the equation might flip in sign. This could be a problem depending on whether the compiler changes the order of operations. So if a robust implementation is needed, the details of the compiler and arithmetic unit may need to be examined. The first four lines in the pseudocode above must be coded carefully to handle cases where the edge exactly hits the pixel center.

In addition to being amenable to an incremental implementation, there are several potential early exit points. For example, if α is negative, there is no need to compute β or γ . While this may well result in a speed improvement, profiling is always a good idea; the extra branches could reduce pipelining or concurrency and might slow down the code. So as always, test any attractive-looking optimizations if the code is a critical section.

Another detail of the above code is that the divisions could be divisions by zero for degenerate triangles, i.e., if $f_\gamma = 0$. Either the floating point error conditions should be accounted for properly, or another test will be needed.

8.1.3 Clipping

Simply transforming primitives into screen space and rasterizing them does not quite work by itself. This is because primitives that are outside the view volume—particularly, primitives that are behind the eye—can end up being rasterized, leading to incorrect results. For instance, consider the triangle shown in Figure 8.7.

```

 $y_{\max} = \text{天花板}(y_i)$ 
 $\alpha = f_{12}(x_0, y_0)$ 
 $\beta = f_{20}(x_1, y_1)$ 
 $\gamma = f_{01}(x_2, y_2)$ 
for  $y = y_{\min}$  to  $y_{\max}$  do
  for  $x = x_{\min}$  to  $x_{\max}$  do
    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
           $c = \alpha c_0 + \beta c_1 + \gamma c_2$ 
          drawpixel  $(x, y)$  with color  $c$ 

```

我们可能期望，只有当我们对两个三角形使用完全相同的线方程时，上面的代码才能消除孔和双绘制。实际上，只有当两个共享顶点在每个三角形的绘制调用中具有相同的顺序时，线方程才是相同的。否则，等式可能会翻转。这可能是一个问题，取决于编译器是否更改了操作的顺序。因此，如果需要一个健壮的实现，编译器和算术单元的细节可能需要检查。上面伪代码中的前四行必须仔细编码，以处理边缘恰好击中像素中心的情况。

除了可以逐步实施之外，还有几个潜在的早期退出点。例如，如果 α 为负，则无需计算 β 或 γ 。虽然这很可能会提高速度，但分析总是一个好主意；额外的分支可能会减少流水线或并发性，并可能减慢代码速度。因此，与往常一样，如果代码是关键部分，请测试任何具有吸引力的优化。

上述代码的另一个细节是，对于退化三角形，即，如果 $f=0$ ，则可以将除法除以零。要么应该正确地考虑浮点误差，要么需要另一个测试。

8.1.3 Clipping

简单地将基元转换为屏幕空间并对其进行光栅化本身并不能正常工作。这是因为视图体积之外的图元（特别是眼睛后面的图元）最终可能会被光栅化，从而导致不正确的结果。例如，考虑图8.7所示的三角形。

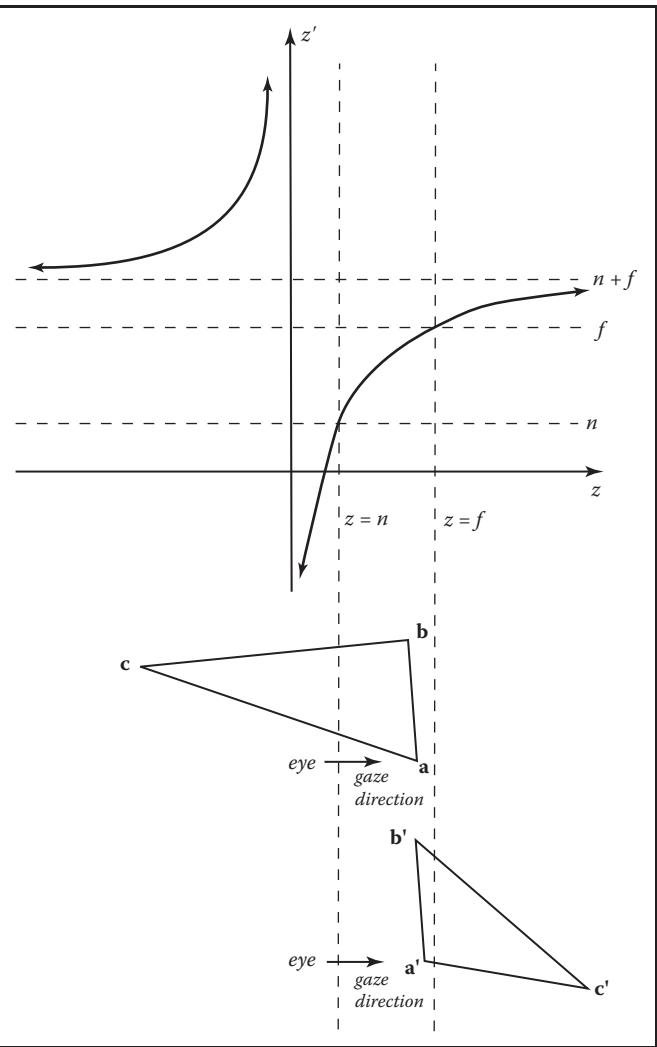


Figure 8.7. The depth z is transformed to the depth z' by the perspective transform. Note that when z moves from positive to negative, z' switches from negative to positive. Thus vertices behind the eye are moved in front of the eye beyond $z' = n + f$. This will lead to wrong results, which is why the triangle is first clipped to ensure all vertices are in front of the eye.

Two vertices are in the view volume, but the third is behind the eye. The projection transformation maps this vertex to a nonsensical location behind the far plane, and if this is allowed to happen the triangle will be rasterized incorrectly. For this reason, rasterization has to be preceded by a *clipping* operation that removes parts of primitives that could extend behind the eye.

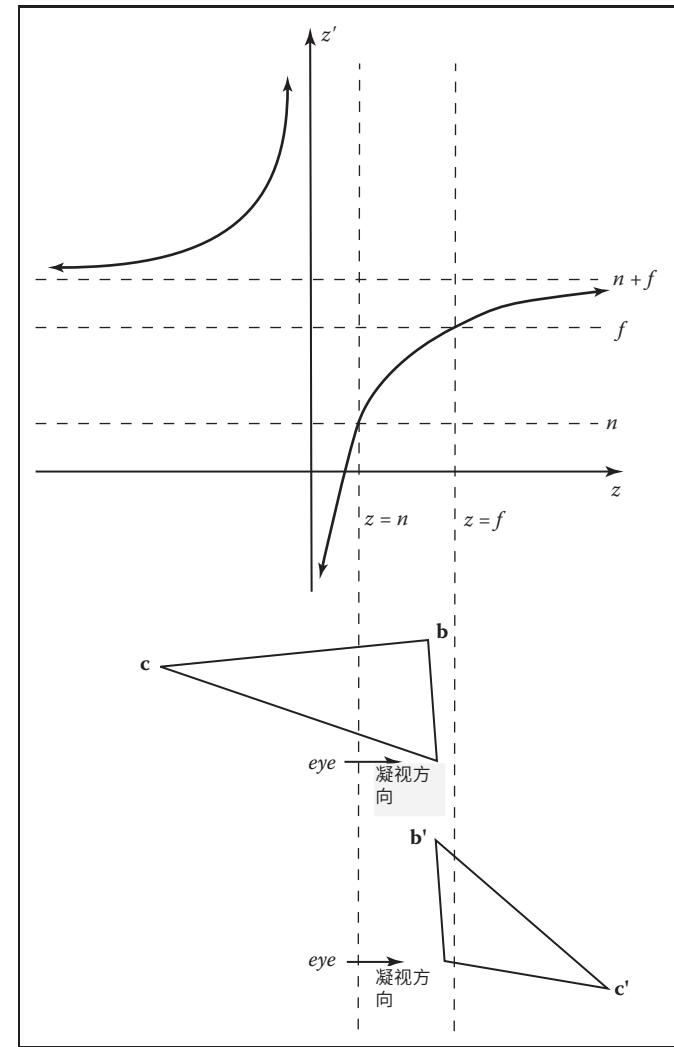


图8.7。通过透视变换将深度 z 变换为深度 z' 。注意，当 z 从正向移动到负向时， z' 从负向切换到正向。因此，眼睛后面的顶点在眼睛前面移动超过 $z'=n+f$ 。这将导致错误的结果，这就是为什么三角形首先被裁剪以确保所有顶点都在眼睛前面。

两个顶点位于视图体积中，但第三个顶点位于眼睛后面。投影变换将此顶点映射到远平面后面的无意义位置，如果允许发生这种情况，则三角形将被错误地栅格化。出于这个原因，光栅化之前必须进行剪切操作，以删除可能延伸到眼睛后面的部分基元。

Clipping is a common operation in graphics, needed whenever one geometric entity “cuts” another. For example, if you clip a triangle against the plane $x = 0$, the plane cuts the triangle into two parts if the signs of the x -coordinates of the vertices are not all the same. In most applications of clipping, the portion of the triangle on the “wrong” side of the plane is discarded. This operation for a single plane is shown in Figure 8.8.

In clipping to prepare for rasterization, the “wrong” side is the side outside the view volume. It is always safe to clip away all geometry outside the view volume—that is, clipping against all six faces of the volume—but many systems manage to get away with only clipping against the near plane.

This section discusses the basic implementation of a clipping module. Those interested in implementing an industrial-speed clipper should see the book by Blinn mentioned in the notes at the end of this chapter.

The two most common approaches for implementing clipping are

1. in world coordinates using the six planes that bound the truncated viewing pyramid,
2. in the 4D transformed space before the homogeneous divide.

Either possibility can be effectively implemented (J. Blinn, 1996) using the following approach for each triangle:

```
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
```

8.1.4 Clipping Before the Transform (Option 1)

Option 1 has a straightforward implementation. The only question is, “What are the six plane equations?” Because these equations are the same for all triangles rendered in the single image, we do not need to compute them very efficiently. For this reason, we can just invert the transform shown in Figure 5.11 and apply

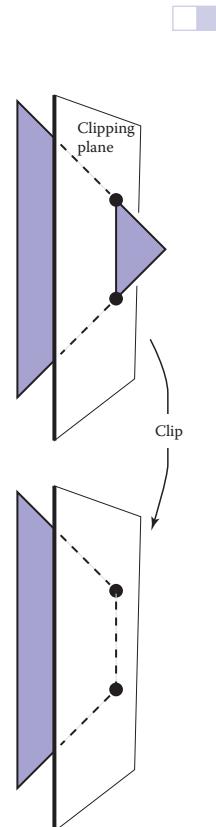


Figure 8.8. A polygon is clipped against a clipping plane. The portion “inside” the plane is retained.

剪切是图形中常见的操作，每当一个几何实体“剪切”另一个几何实体时都需要。例如，如果将三角形夹在平面 $x=0$ 上，则如果顶点的 x 坐标的符号不全部相同，则平面将三角形切割为两部分。在削波的大多数应用中，三角形在平面的“错误”侧上的部分被丢弃。单个平面的此操作如图8.8所示。

在裁剪以准备光栅化时，“错误”的一侧是视图体积之外的一侧。将视图体积之外的所有几何体裁剪掉总是安全的——即对体积的所有六个面进行裁剪——但许多系统只能对近平面进行裁剪。

本节讨论裁剪模块的基本实现。那些对实现工业速度快船感兴趣的人应该看到本章末尾的笔记中提到的Blinn的书。

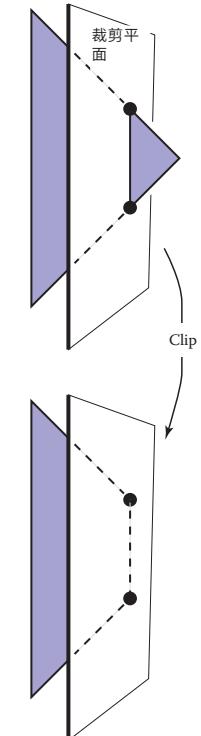
实现裁剪的两种最常见方法是

- 1.在世界坐标中，使用六个平面约束截断的观察金字塔
- 2.在齐次划分之前的4d变换空间中。

任何一种可能性都可以有效地实现 (J.Blinn, 1996) 使用每个三角形的以下方法：

六架飞机中的每一架都有

如果（三角形完全在平面之外），那么打破（三角形不可见）否则如果三角形跨越平面，那么剪辑三角形如果（四边形是左），那么打破成两个三角形



多边形被裁剪到裁剪平面上
平面“内部”的部分被保留。

8.1.4 变换前的裁剪 (选项1)

备选方案1有一个简单的实现。唯一的问题是，“六个平面方程是什么？”因为这些方程对于在单个图像中渲染的所有三角形都是相同的，所以我们不需要非常有效地计算它们。出于这个原因，我们可以反转图5.11所示的变换并应用

it to the eight vertices of the transformed view volume:

$$\begin{aligned}(x, y, z) = & (l, b, n) \\ & (r, b, n) \\ & (l, t, n) \\ & (r, t, n) \\ & (l, b, f) \\ & (r, b, f) \\ & (l, t, f) \\ & (r, t, f).\end{aligned}$$

The plane equations can be inferred from here. Alternatively, we can use vector geometry to get the planes directly from the viewing parameters.

8.1.5 Clipping in Homogeneous Coordinates (Option 2)

Surprisingly, the option usually implemented is that of clipping in homogeneous coordinates before the divide. Here the view volume is 4D, and it is bounded by 3D volumes (hyperplanes). These are

$$\begin{aligned}-x + lw &= 0, \\ x - rw &= 0, \\ -y + bw &= 0, \\ y - tw &= 0, \\ -z + nw &= 0, \\ z - fw &= 0.\end{aligned}$$

These planes are quite simple, so the efficiency is better than for Option 1. They still can be improved by transforming the view volume $[l, r] \times [b, t] \times [f, n]$ to $[0, 1]^3$. It turns out that the clipping of the triangles is not much more complicated than in 3D.

8.1.6 Clipping against a Plane

No matter which option we choose, we must clip against a plane. Recall from Section 2.5.5 that the implicit equation for a plane through point \mathbf{q} with normal \mathbf{n} is

$$f(\mathbf{p}) = \mathbf{n} \cdot (\mathbf{p} - \mathbf{q}) = 0.$$

到变换后的视图体积的八个顶点:

$$\begin{aligned}(x, y, z) = & (l, b, n) \\ & (r, b, n) \\ & (l, t, n) \\ & (r, t, n) \\ & (l, b, f) \\ & (r, b, f) \\ & (l, t, f) \\ & (r, t, f).\end{aligned}$$

从这里可以推断出平面方程。或者，我们可以使用矢量几何直接从查看参数中获取平面。

8.1.5 齐次坐标中的裁剪（选项2）

令人惊讶的是，通常实现的选项是在分割之前在齐次坐标中剪切。这里的视图体积是4D，并且以3d体积（超平面）为界。这些是

$$\begin{aligned}-x + lw &= 0, \\ x - rw &= 0, \\ -y + bw &= 0, \\ y - tw &= 0, \\ -z + nw &= 0, \\ z - fw &= 0.\end{aligned}$$

这些平面相当简单，因此效率优于选项1。它们仍然可以通过将视图体积 $[l, r] \times [b, t] \times [f, n]$ 转换为 $[0, 1]^3$ 来改进。事实证明，三角形的裁剪并不比3D复杂得多。

8.1.6 在飞机上剪

无论我们选择哪个选项，我们都必须夹在飞机上。从第2.5.5节中回想一下，通过点 \mathbf{q} 的平面与法线 \mathbf{n} 的隐含方程是

$$f(\mathbf{p}) = \mathbf{n} \cdot (\mathbf{p} - \mathbf{q}) = 0.$$

This is often written

$$f(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + D = 0. \quad (8.2)$$

Interestingly, this equation not only describes a 3D plane, but it also describes a line in 2D and the volume analog of a plane in 4D. All of these entities are usually called planes in their appropriate dimension.

If we have a line segment between points \mathbf{a} and \mathbf{b} , we can “clip” it against a plane using the techniques for cutting the edges of 3D triangles in BSP tree programs described in Section 12.4.3. Here, the points \mathbf{a} and \mathbf{b} are tested to determine whether they are on opposite sides of the plane $f(\mathbf{p}) = 0$ by checking whether $f(\mathbf{a})$ and $f(\mathbf{b})$ have different signs. Typically $f(\mathbf{p}) < 0$ is defined to be “inside” the plane, and $f(\mathbf{p}) > 0$ is “outside” the plane. If the plane does split the line, then we can solve for the intersection point by substituting the equation for the parametric line,

$$\mathbf{p} = \mathbf{a} + t(\mathbf{b} - \mathbf{a}),$$

into the $f(\mathbf{p}) = 0$ plane of Equation (8.2). This yields

$$\mathbf{n} \cdot (\mathbf{a} + t(\mathbf{b} - \mathbf{a})) + D = 0.$$

Solving for t gives

$$t = \frac{\mathbf{n} \cdot \mathbf{a} + D}{\mathbf{n} \cdot (\mathbf{b} - \mathbf{a})}.$$

We can then find the intersection point and “shorten” the line.

To clip a triangle, we again can follow Section 12.4.3 to produce one or two triangles.

8.2 Operations Before and After Rasterization

Before a primitive can be rasterized, the vertices that define it must be in screen coordinates, and the colors or other attributes that are supposed to be interpolated across the primitive must be known. Preparing this data is the job of the *vertex-processing* stage of the pipeline. In this stage, incoming vertices are transformed by the modeling, viewing, and projection transformations, mapping them from their original coordinates into screen space (where, recall, position is measured in terms of pixels). At the same time, other information, such as colors, surface normals, or texture coordinates, is transformed as needed; we’ll discuss these additional attributes in the examples below.

After rasterization, further processing is done to compute a color and depth for each fragment. This processing can be as simple as just passing through an interpolated color and using the depth computed by the rasterizer; or it can involve

这是经常写的

$$f(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + D = 0. \quad (8.2)$$

有趣的是，这个方程不仅描述了一个3D平面，而且还描述了2D中的一条线和4d中的一个平面的体积模拟。

如果我们在点a和点b之间有一条线段，我们可以使用第12.4.3节中描述的BSP树程序中切割3d三角形边缘的技术将其“剪辑”在平面上。这里，对点a和b进行测试，以通过检查f(a)和f(b)是否具有不同的标志来确定它们是否在平面f(p)=0的相对两侧。通常f(p)<0被定义为在平面的“内部”，并且f(p)>0在平面的“外部”。如果平面确实分割了线，那么我们可以通过用参数线的方程来求解交点

$$\mathbf{p} = \mathbf{a} + t(\mathbf{b} - \mathbf{a}),$$

入式(8.2)的f(p)=0平面。这会产生

$$\mathbf{n} \cdot (\mathbf{a} + t(\mathbf{b} - \mathbf{a})) + D = 0.$$

求解t给出

$$t = \frac{\mathbf{n} \cdot \mathbf{a} + D}{\mathbf{n} \cdot (\mathbf{b} - \mathbf{a})}.$$

然后我们可以找到交点并“缩短”线。

要剪辑一个三角形，我们再次可以按照第12.4.3节产生一个或两个三角形。

8.2 栅格化前后的操作

在对图元进行栅格化之前，定义它的顶点必须位于屏幕坐标中，并且必须知道应该在图元中插值的颜色或其他属性。准备这些数据是管道顶点处理阶段的工作。在这个阶段，传入顶点通过建模、查看和投影变换进行变换，将它们从原始坐标映射到屏幕空间（其中，回想一下，位置是用像素来衡量的）。与此同时，其他信息（如颜色、表面法线或纹理坐标）也会根据需要进行转换；我们将在下面的示例中讨论这些附加属性。

在光栅化之后，进行进一步处理以计算每个片段的颜色和深度。这种处理可以简单到只通过一个在interpolated颜色和使用由光栅化器计算的深度；或者它可以涉及

complex shading operations. Finally, the blending phase combines the fragments generated by the (possibly several) primitives that overlapped each pixel to compute the final color. The most common blending approach is to choose the color of the fragment with the smallest depth (closest to the eye).

The purposes of the different stages are best illustrated by examples.

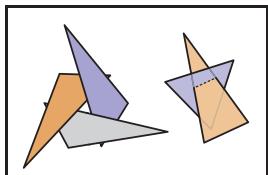


Figure 8.9. Two occlusion cycles, which cannot be drawn in back-to-front order.

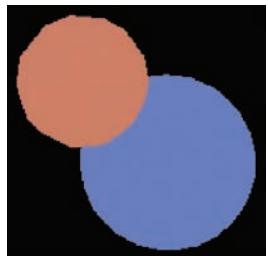


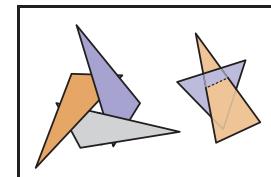
Figure 8.10. The result of drawing two spheres of identical size using the minimal pipeline. The sphere that appears smaller is farther away but is drawn last, so it incorrectly overwrites the nearer one.

复杂的着色操作。最后，混合阶段将由重叠每个像素的（可能是几个）基元生成的片段组合在一起，以混合最终的颜色。最常见的混合方法是选择深度最小的片段的颜色（最接近眼睛）。

不同阶段的目的最好通过实例说明。

8.2.1 Simple 2D Drawing

The simplest possible pipeline does nothing in the vertex or fragment stages, and in the blending stage the color of each fragment simply overwrites the value of the previous one. The application supplies primitives directly in pixel coordinates, and the rasterizer does all the work. This basic arrangement is the essence of many simple, older APIs for drawing user interfaces, plots, graphs, and other 2D content. Solid color shapes can be drawn by specifying the same color for all vertices of each primitive, and our model pipeline also supports smoothly varying color using interpolation.



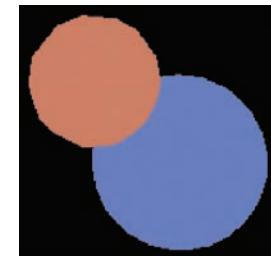
两个遮挡周期，不能以背到前的顺序绘制。

8.2.1 简单的2D绘图

最简单的管道在顶点或片段阶段不做任何事情，在混合阶段，每个片段的颜色只是覆盖前一个片段的值。应用程序直接在像素坐标中提供图元，光栅化器完成所有工作。这种基本的安排是许多简单的，较旧的API的本质，用于绘制用户界面，绘图，图形和其他2D内容。纯色形状可以通过为每个基元的所有顶点指定相同颜色来绘制，我们的模型管道还支持使用插值平滑变化的颜色。

8.2.2 A Minimal 3D Pipeline

To draw objects in 3D, the only change needed to the 2D drawing pipeline is a single matrix transformation: the vertex-processing stage multiplies the incoming vertex positions by the product of the modeling, camera, projection, and viewport matrices, resulting in screen-space triangles that are then drawn in the same way as if they'd been specified directly in 2D.



使用最小管道绘制两个相同尺寸的球体的结果。Ap梨较小的球体距离较远，但最后绘制，因此它正确地覆盖了较近的球体。

8.2.2 最小3D管道

要在3D中绘制对象，2D绘制管道所需的唯一更改是单个矩阵转换：顶点处理阶段将传入的顶点位置乘以建模、相机、投影和视口矩阵的乘积，从而产生屏幕空间三角形，然后以与直接在2D中指定的相同的方式绘制这些三角形。

最小3d管道的一个问题是，为了获得正确的遮挡关系-在更远的对象前面获得更近的对象-必须按照背到前的顺序绘制基元。这被称为画家的隐藏表面去除算法，通过类比先画一幅画的背景，然后画前景。画家的算法是一个完全有效的方法来删除隐藏的表面，但它有几个缺点。它不能处理彼此相交的三角形，因为绘制它们的顺序没有正确的顺序。同样，几个三角形，即使它们不相交，仍然可以在一个遮挡周期中排列，如图8.9所示，另一种情况是不存在背到前的顺序。最重要的是，按深度对基元进行排序很慢，特别是对于大型场景，并且扰乱了使对象顺序渲染如此之快的高效数据流。图8.10显示了当对象未按深度排序时此过程的结果。



8.2.3 Using a z-Buffer for Hidden Surfaces

In practice, the painter's algorithm is rarely used; instead a simple and effective hidden surface removal algorithm known as the *z-buffer* algorithm is used. The method is very simple: at each pixel we keep track of the distance to the closest surface that has been drawn so far, and we throw away fragments that are farther away than that distance. The closest distance is stored by allocating an extra value for each pixel, in addition to the red, green, and blue color values, which is known as the depth, or *z-value*. The *depth buffer*, or *z-buffer*, is the name for the grid of depth values.

The *z-buffer* algorithm is implemented in the fragment blending phase, by comparing the depth of each fragment with the current value stored in the *z-buffer*. If the fragment's depth is closer, both its color and its depth value overwrite the values currently in the color and depth buffers. If the fragment's depth is farther away, it is discarded. To ensure that the first fragment will pass the depth test, the *z-buffer* is initialized to the maximum depth (the depth of the far plane). Irrespective of the order in which surfaces are drawn, the same fragment will win the depth test, and the image will be the same.

The *z-buffer* algorithm requires each fragment to carry a depth. This is done simply by interpolating the *z*-coordinate as a vertex attribute, in the same way that color or other attributes are interpolated.

The *z-buffer* is such a simple and practical way to deal with hidden surfaces in object-order rendering that it is by far the dominant approach. It is much simpler than geometric methods that cut surfaces into pieces that can be sorted by depth, because it avoids solving any problems that don't need to be solved. The depth order only needs to be determined at the locations of the pixels, and that is all that the *z-buffer* does. It is universally supported by hardware graphics pipelines and is also the most commonly used method for software pipelines. Figure 8.11 shows an example result.

Precision Issues

In practice, the *z*-values stored in the buffer are nonnegative integers. This is preferable to true floats because the fast memory needed for the *z-buffer* is somewhat expensive and is worth keeping to a minimum.

The use of integers can cause some precision problems. If we use an integer range having B values $\{0, 1, \dots, B - 1\}$, we can map 0 to the near clipping plane $z = n$ and $B - 1$ to the far clipping plane $z = f$. Note, that for this discussion, we assume z , n , and f are positive. This will result in the same results as the negative case, but the details of the argument are easier to follow. We send each *z*-value to



8.2.3 对隐藏表面使用z缓冲区

在实践中，很少使用painter算法；相反，使用称为z-buffer算法的简单有效的隐藏表面去除算法。该方法非常简单：在每个像素处，我们跟踪到目前为止绘制的最近表面的距离，并且我们扔掉比该距离更远的片段。通过为每个像素分配一个额外的值来存储最近的距离，除了红色、绿色和蓝色的颜色值（称为深度或z值）。深度缓冲区或z缓冲区是深度值网格的名称。

Z-buffer算法在片段混合阶段实现，通过将每个片段的深度与存储在z-buffer中的当前值进行比较。如果片段的深度更接近，则其颜色和深度值都会复盖颜色和深度缓冲区中当前的值。如果碎片的深度更远

离，被丢弃。为了确保第一个片段将通过深度测试，z缓冲器被初始化为最大深度（远平面的深度）。无论绘制曲面的顺序如何，相同的片段都将赢得深度测试，并且图像将是相同的。

Z-buffer算法要求每个片段携带一个深度。这是简单地通过插值z坐标作为顶点属性来完成的，就像插值颜色或其他属性一样。

Z缓冲区是处理对象顺序渲染中隐藏表面的一种简单而实用的方法，到目前为止它是主要的方法。它比将表面切割成可以按深度排序的块的几何方法简单得多，因为它避免了解决任何不需要解决的问题。深度

只需要在像素的位置确定顺序，这就是z缓冲区所做的一切。它受到硬件图形管道的普遍支持，也是软件管道最常用的方法。图8.11显示了一个示例结果。

当然，深度测试中可能存在联系，在这种情况下，顺序可能很重要。

Of course there can be ties in the depth test, in which case the order may well matter.

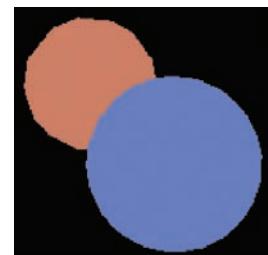
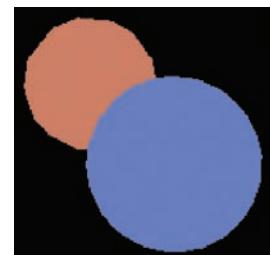


Figure 8.11. The result of drawing the same two spheres using the z-buffer.



使用z缓冲区绘制相同的两个球体的结果。

精度问题

实际上，存储在缓冲区中的z值是非负整数。这比真正的浮点更好，因为z缓冲区所需的快速内存是一些昂贵的，值得保持在最低限度。

整数的使用会导致一些精度问题。如果我们使用具有 b 位的整数范围 $\{0, 1, \dots, B - 1\}$ ，我们可以将0映射到近剪切平面 $z=n$ ，将 $B-1$ 映射到远剪切平面 $z=f$ 。请注意，对于此讨论，我们假设 z ， n 和 f 为正数。这将导致与否定情况相同的结果，但参数的细节更容易遵循。我们将每个z值发送到

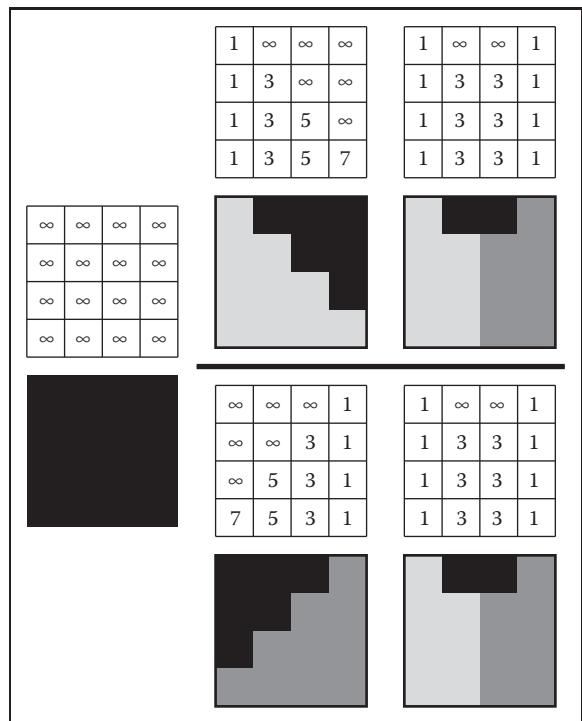


Figure 8.12. A z-buffer rasterizing two triangles in each of two possible orders. The first triangle is fully rasterized. The second triangle has every pixel computed, but for three of the pixels the depth-contest is lost, and those pixels are not drawn. The final image is the same regardless.

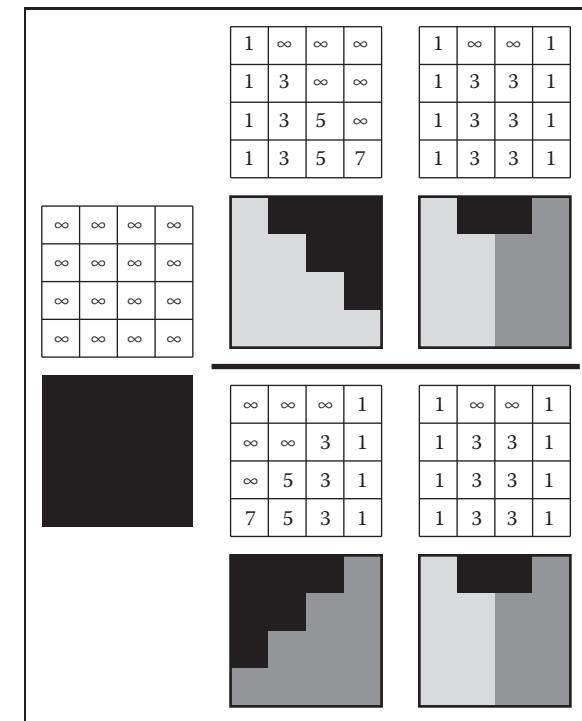
a “bucket” with depth $\Delta z = (f - n)/B$. We would not use the integer z-buffer if memory were not a premium, so it is useful to make B as small as possible.

If we allocate b bits to store the z -value, then $B = 2^b$. We need enough bits to make sure any triangle in front of another triangle will have its depth mapped to distinct depth bins.

For example, if you are rendering a scene where triangles have a separation of at least one meter, then $\Delta z < 1$ should yield images without artifacts. There are two ways to make Δz smaller: move n and f closer together or increase b . If b is fixed, as it may be in APIs or on particular hardware platforms, adjusting n and f is the only option.

The precision of z-buffers must be handled with great care when perspective images are created. The value Δz above is used *after* the perspective divide. Recall from Section 7.3 that the result of the perspective divide is

$$z = n + f - \frac{fn}{z_w}.$$



一个z缓冲区，以两个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个像素，但是对于其中三个像素，深度竞争会丢失，并且这些像素不会被绘制。最终图像无论如何都是相同的。

深度 $\Delta z=(f-n)/B$ 的“桶”。如果内存不是额外的，我们不会使用整数z缓冲区，所以使B尽可能小是有用的。

如果我们分配 b 位来存储 z 值，则 $B=2^b$ 。我们需要足够的位来确保另一个三角形前面的任何三角形将其深度映射到不同的深度区间。

例如，如果您正在渲染一个三角形间隔至少为一米的场景，那么 $\Delta z < 1$ 应该产生没有伪像的图像。有两种方法可以使 Δz 变小：将 n 和 f 移得更近或增加 b 。如果 b 是固定的，因为它可能在API或特定的硬件平台上，调整 n 和 f 是唯一的选择。

创建透视图像时，必须非常小心地处理z缓冲区的精度。上述值 Δz 在透视分割后使用。从第7.3节回想一下，透视线划分的结果是

$$z = n + f - \frac{fn}{z_w}.$$

The actual bin depth is related to z_w , the world depth, rather than z , the post-perspective divide depth. We can approximate the bin size by differentiating both sides:

$$\Delta z \approx \frac{fn\Delta z_w}{z_w^2}.$$

Bin sizes vary in depth. The bin size in world space is

$$\Delta z_w \approx \frac{z_w^2 \Delta z}{fn}.$$

Note that the quantity Δz is as previously discussed. The biggest bin will be for $z' = f$, where

$$\Delta z_w^{\max} \approx \frac{f \Delta z}{n}.$$

Note that choosing $n = 0$, a natural choice if we don't want to lose objects right in front of the eye, will result in an infinitely large bin—a very bad condition. To make Δz_w^{\max} as small as possible, we want to minimize f and maximize n . Thus, it is always important to choose n and f carefully.

8.2.4 Per-vertex Shading

So far the application sending triangles into the pipeline is responsible for setting the color; the rasterizer just interpolates the colors and they are written directly into the output image. For some applications this is sufficient, but in many cases we want 3D objects to be drawn with shading, using the same illumination equations that we used for image-order rendering in Chapter 4. Recall that these equations require a light direction, an eye direction, and a surface normal to compute the color of a surface.

One way to handle shading computations is to perform them in the vertex stage. The application provides normal vectors at the vertices, and the positions and colors of the lights are provided separately (they don't vary across the surface, so they don't need to be specified for each vertex). For each vertex, the direction to the viewer and the direction to each light are computed based on the positions of the camera, the lights, and the vertex. The desired shading equation is evaluated to compute a color, which is then passed to the rasterizer as the vertex color. Per-vertex shading is sometimes called *Gouraud shading*.

One decision to be made is the coordinate system in which shading computations are done. World space or eye space are good choices. It is important to choose a coordinate system that is orthonormal when viewed in world space, because shading equations depend on angles between vectors, which are

实际的bin深度与 z_w ，世界深度有关，而不是 z ，postperspectivedivide深度。我们可以通过区分两边来近似bin大小：

$$\Delta z \approx \frac{z_w^2 \Delta z}{fn}.$$

垃圾箱尺寸在深度上有所不同。世界空间中的bin大小为

$$\Delta z_w \approx \frac{z_w^2 \Delta z}{fn}.$$

注意，量 Δz 如先前讨论的。最大的bin将是 $z'=f$ ，其中

$$\Delta z_w^{\max} \approx \frac{f \Delta z}{n}.$$

请注意，如果我们不想在眼睛前面丢失物体，选择 $n=0$ 是一个自然的选择，将导致一个无限大的bin—一个非常糟糕的条件。使 Δz 最大 w 尽可能小，我们希望最小化 f 并最大化 n 。因此，仔细选择 n 和 f 总是很重要的。

8.2.4 Per-vertex Shading

到目前为止，将三角形发送到管道中的应用程序负责设置颜色；光栅化器只是插值颜色，它们直接写入输出图像。对于某些应用来说，这就足够了，但在许多情况下，我们希望使用与第4章中用于图像顺序渲染的相同的照明等效来绘制3d对象。回想一下，这些等距需要一个光方向、一个眼睛方向和一个表面法线来计算表面的颜色。

处理着色计算的一种方法是在顶点阶段执行它们。该应用程序在顶点处提供法向量，并且灯的位置和颜色是单独提供的（它们不会在整个表面上变化，因此不需要为每个顶点指定它们）。对于每个顶点，根据相机、光源和顶点的位置计算到查看器的方向和到每个光源的方向。计算所需的着色方程以计算颜色，然后将其作为顶点颜色传递给光栅化器。每个顶点着色有时称为Gouraud着色。

要做出的一个决定是完成着色com放置的坐标系。世界空间或眼睛空间都是不错的选择。选择一个在世界空间中观察时正交的坐标系是很重要的，因为着色方程依赖于矢量之间的角度，这些矢量是

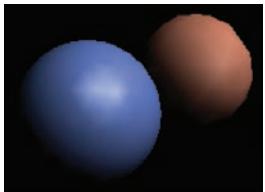


Figure 8.13. Two spheres drawn using per-vertex (Gouraud) shading. Because the triangles are large, interpolation artifacts are visible.

Per-vertex shading has the disadvantage that it cannot produce any details in the shading that are smaller than the primitives used to draw the surface, because it only computes shading once for each vertex and never in between vertices. For instance, in a room with a floor that is drawn using two large triangles and illuminated by a light source in the middle of the room, shading will be evaluated only at the corners of the room, and the interpolated value will likely be much too dark in the center. Also, curved surfaces that are shaded with specular highlights must be drawn using primitives small enough that the highlights can be resolved.

Figure 8.13 shows our two spheres drawn with per-vertex shading.

8.2.5 Per-fragment Shading

Per-fragment shading is sometimes called Phong shading, which is confusing because the same name is attached to the Phong illumination model.

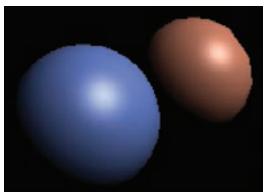


Figure 8.14. Two spheres drawn using per-fragment shading. Because the triangles are large, interpolation artifacts are visible.

Figure 8.14 shows our two spheres drawn with per-fragment shading.

8.2.6 Texture Mapping

Textures (discussed in Chapter 11) are images that are used to add extra detail to the shading of surfaces that would otherwise look too homogeneous and artificial. The idea is simple: each time shading is computed, we read one of the values used in the shading computation—the diffuse color, for instance—from a texture instead of using the attribute values that are attached to the geometry being rendered. This operation is known as a *texture lookup*: the shading code specifies a *texture coordinate*, a point in the domain of the texture, and the texture-mapping



Figure 8.13. 两个球体使用每顶点 (Gouraud) 着色。因为三角形很大，所以可以看到极化间伪像。

不被像在modeling变换中经常使用的非均匀尺度这样的操作保留，或者在对规范视图体积的投影中经常使用的透视投影。眼睛空间中的阴影的优点是我们不需要跟踪相机位置，因为相机总是在眼睛空间中的原点，在透视投影中，或者在正投影中视图方向总是 $+z$ 。

每个顶点着色的缺点是它不能在着色中产生比用于绘制表面的基元更小的任何细节，因为它只为每个顶点计算一次着色，而从不在顶点之间计算着色。

例如，在使用两个大三角形绘制并由房间中间的光源照明的地板的房间中，将仅在房间的角落评估阴影，并且插值值可能在中心太暗。此外，必须使用足够小的图元绘制带有镜面高光阴影的曲面，以便可以解析高光。图8.13显示了我们用每顶点着色绘制的两个球体。

Per-fragment shading is
称为Phong阴影，这是混淆，因为相同的名称附加到Phong照明模型。

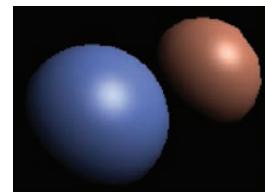


Figure 8.14. 两个球体
drawn using per-fragment
阴影。因为triangles很大，
所以插值伪影是可见的。

8.2.5 Per-fragment Shading

为了避免与每个顶点着色相关的插值伪影，我们可以通过在片段阶段的interpolation之后执行着色计算来避免插值颜色。在每个片段着色中，计算相同的着色方程，但使用相互关联的矢量对每个片段进行评估，而不是使用应用程序中的矢量对每个顶点进行评估。在每片段着色中，着色所需的几何信息作为属性通过光栅化器，因此顶点阶段必须与片段阶段协调以适当地准备数据。一种方法是在 interpolate眼空间表面法线和眼空间顶点位置，然后可以像在每个顶点着色中一样使用它们。

图8.14显示了我们用每个片段着色绘制的两个球体。

8.2.6 纹理映射

纹理（在第11章中讨论）是用于为表面阴影添加额外细节的图像，否则这些表面看起来会过于均匀和人为。这个想法很简单：每次计算着色时，我们都会从纹理中读取着色计算中使用的值之一—例如漫反射颜色，而不是使用附加到正在渲染的几何体的属性值。此操作称为纹理查找：着色代码指定纹理坐标、纹理域中的点以及纹理映射



system finds the value at that point in the texture image and returns it. The texture value is then used in the shading computation.

The most common way to define texture coordinates is simply to make the texture coordinate another vertex attribute. Each primitive then knows where it lives in the texture.

8.2.7 Shading Frequency

The decision about where to place shading computations depends on how fast the color changes—the *scale* of the details being computed. Shading with large-scale features, such as diffuse shading on curved surfaces, can be evaluated fairly infrequently and then interpolated: it can be computed with a low *shading frequency*. Shading that produces small-scale features, such as sharp highlights or detailed textures, needs to be evaluated at a high shading frequency. For details that need to look sharp and crisp in the image, the shading frequency needs to be at least one shading sample per pixel.

So large-scale effects can safely be computed in the vertex stage, even when the vertices defining the primitives are many pixels apart. Effects that require a high shading frequency can also be computed at the vertex stage, as long as the vertices are close together in the image; alternatively, they can be computed at the fragment stage when primitives are larger than a pixel.

For example, a hardware pipeline as used in a computer game, generally using primitives that cover several pixels to ensure high efficiency, normally does most shading computations per fragment. On the other hand, the PhotoRealistic RenderMan system does all shading computations per vertex, after first subdividing, or *dicing*, all surfaces into small quadrilaterals called *micropolygons* that are about the size of pixels. Since the primitives are small, per-vertex shading in this system achieves a high shading frequency that is suitable for detailed shading.

8.3 Simple Antialiasing

Just as with ray tracing, rasterization will produce jagged lines and triangle edges if we make an all-or-nothing determination of whether each pixel is inside the primitive or not. In fact, the set of fragments generated by the simple triangle rasterization algorithms described in this chapter, sometimes called standard or *aliased* rasterization, is exactly the same as the set of pixels that would be mapped to that triangle by a ray tracer that sends one ray through the center of each pixel.



系统在纹理图像中找到该点的值并返回它。然后将纹理值用于阴影计算。

定义纹理坐标的最常见方法是简单地使纹理坐标成为另一个顶点属性。然后，每个基元都知道它在纹理中的位置。

8.2.7 遮光频率

关于在何处放置着色计算的决定取决于颜色变化的速度—计算的细节的比例。具有大规模特征的着色，例如曲面上的漫反射着色，可以相当精确地评估，然后进行插值：可以用低着色频率计算。产生小尺度特征（如尖锐高光或细节纹理）的着色需要以较高的着色频率进行评估。对于需要在图像中看起来锐利和清晰的细节，阴影频率需要为每个像素至少一个阴影样本。

因此，即使定义图元的顶点相距许多像素，也可以在顶点阶段安全地计算大规模效果。需要高着色频率的效果也可以在顶点阶段计算，只要顶点在图像中靠近在一起；或者，当基元大于像素时，可以在片段阶段计算。

例如，在计算机游戏中使用的硬件管道，通常使用复盖几个像素以确保高效率的ing基元，通常对每个片段进行大多数着色计算。另一方面，Photo Realistic RenderMan系统在对每个顶点进行第一次细分或划片后，将所有表面划分为称为micropolygons的小四边形，这些四边形大约是像素的大小。由于图元很小，因此该系统中的每顶点着色实现了适合于详细着色的高着色频率。

8.3 Simple Antialiasing

就像光线追踪一样，如果我们确定每个像素是否在基元内部，光栅化将产生锯齿线和三角形边缘。实际上，本章中描述的简单三角形光栅化算法生成的片段集（有时称为标准或混叠光栅化）与光线跟踪器将映射到该三角形的像素集完全相同，该像素集通过每个像素的中心发送一条光线。

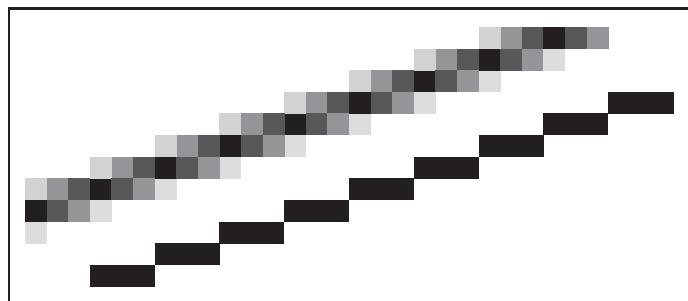


Figure 8.15. An antialiased and a jaggy line viewed at close range so individual pixels are visible.

Also as in ray tracing, the solution is to allow pixels to be partly covered by a primitive (Crow, 1978). In practice, this form of blurring helps visual quality, especially in animations. This is shown as the top line of Figure 8.15.

There are a number of different approaches to antialiasing in rasterization applications. Just as with a ray tracer, we can produce an antialiased image by setting each pixel value to the average color of the image over the square area belonging to the pixel, an approach known as *box filtering*. This means we have to think of all drawable entities as having well-defined areas. For example, the line in Figure 8.15 can be thought of as approximating a one-pixel-wide rectangle.

The easiest way to implement box-filter antialiasing is by *supersampling*: create images at very high resolutions and then downsample. For example, if our goal is a 256×256 pixel image of a line with width 1.2 pixels, we could rasterize a rectangle version of the line with width 4.8 pixels on a 1024×1024 screen, and then average 4×4 groups of pixels to get the colors for each of the 256×256 pixels in the “shrunken” image. This is an approximation of the actual box-filtered image, but works well when objects are not extremely small relative to the distance between pixels.

Supersampling is quite expensive, however. Because the very sharp edges that cause aliasing are normally caused by the edges of primitives, rather than sudden variations in shading within a primitive, a widely used optimization is to sample visibility at a higher rate than shading. If information about coverage and depth is stored for several points within each pixel, very good antialiasing can be achieved even if only one color is computed. In systems like RenderMan that use per-vertex shading, this is achieved by rasterizing at high resolution: it is inexpensive to do so because shading is simply interpolated to produce colors for the many fragments, or visibility samples. In systems with per-fragment shading, such as hardware pipelines, *multisample antialiasing* is achieved by storing for each fragment a single color plus a coverage mask and a set of depth values.

There are better filters than the box, but a box filter will suffice for all but the most demanding applications.

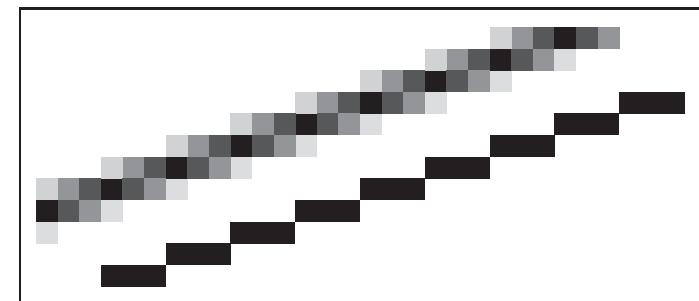


图8.15。在近距离观察的抗锯齿和锯齿线，因此单个像素可见。

与光线追踪一样，解决方案是允许像素部分被基元复盖（Crow, 1978）。在实践中，这种模糊形式有助于视觉质量，特别是在动画中。这显示为图8.15的顶线。

在光栅化应用中有许多不同的抗锯齿方法。就像光线追踪器一样，我们可以通过将每个像素值设置为图像在属于像素的正方形区域上的平均颜色来产生抗锯齿图像，这种方法称为框滤波。这意味着我们有

有比盒子更好的过滤器，但盒子过滤器将满足所有，但最苛刻的应用。

将所有可绘制实体视为具有明确定义的区域。例如，图8.15中的线可以被认为是一个像素宽的矩形。
实现盒式滤波器抗锯齿的最简单方法是通过超采样：以非常高的分辨率对图像进行采样，然后进行下采样。例如，如果我们的目标是一个宽度为1.2像素的线的 256×256 像素图像，我们可以在 1024×1024 屏幕上栅格化宽度为4.8像素的线的矩形版本，然后平均 4×4 组像素以获得“缩小”图像中每个 256×256 像素的颜色。这是实际框过滤图像的近似值，但是当对象相对于像素之间的距离不是极小时，效果很好。

然而，超级采样相当昂贵。由于导致混叠的非常尖锐的边缘通常是由图元的边缘引起的，而不是图元内阴影的突然变化，因此广泛使用的优化是以比阴影更高的速率采样可见性。如果为每个像素内的几个点存储有关复盖和深度的信息，即使只计算一种颜色，也可以实现非常好的抗锯齿。在像RenderMan这样使用每顶点着色的系统中，这是通过高分辨率光栅化来实现的：这样做很便宜，因为着色只是简单地插值以产生许多片段或可见性样本的颜在具有每个片段着色的系统中，例如硬件管道，通过为每个片段存储单一颜色加上复盖遮罩和一组深度值来实现多采样抗锯齿。



8.4 Culling Primitives for Efficiency

The strength of object-order rendering, that it requires a single pass over all the geometry in the scene, is also a weakness for complex scenes. For instance, in a model of an entire city, only a few buildings are likely to be visible at any given time. A correct image can be obtained by drawing all the primitives in the scene, but a great deal of effort will be wasted processing geometry that is behind the visible buildings, or behind the viewer, and therefore doesn't contribute to the final image.

Identifying and throwing away invisible geometry to save the time that would be spent processing it is known as *culling*. Three commonly implemented culling strategies (often used in tandem) are

- **view volume culling**—the removal of geometry that is outside the view volume;
- **occlusion culling**—the removal of geometry that may be within the view volume but is obscured, or occluded, by other geometry closer to the camera;
- **backface culling**—the removal of primitives facing away from the camera.

We will briefly discuss view volume culling and backface culling, but culling in high performance systems is a complex topic; see (Akenine-Möller et al., 2008) for a complete discussion and for information about occlusion culling.

8.4.1 View Volume Culling

When an entire primitive lies outside the view volume, it can be culled, since it will produce no fragments when rasterized. If we can cull many primitives with a quick test, we may be able to speed up drawing significantly. On the other hand, testing primitives individually to decide exactly which ones need to be drawn may cost more than just letting the rasterizer eliminate them.

View volume culling, also known as *view frustum culling*, is especially helpful when many triangles are grouped into an object with an associated bounding volume. If the bounding volume lies outside the view volume, then so do all the triangles that make up the object. For example, if we have 1000 triangles bounded by a single sphere with center c and radius r , we can check whether the sphere lies outside the clipping plane,

$$(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0,$$



8.4 为提高效率而剔除基元

对象顺序渲染的优势在于它需要在场景中的所有几何体上进行一次遍历，这对于复杂场景来说也是一个弱点。例如，在整个城市的模型中，只有少数建筑物可能在任何给定时间可见。通过绘制场景中的所有图元可以获得正确的图像，但是大量的努力将浪费在处理可见建筑物后面或观察者后面的几何体上，因此对最终图像没有贡献。

识别和丢弃不可见的几何体以节省处理它所花费的时间，这被称为剔除。三种常用的剔除策略（通常同时使用）是

*视图体积剔除—移除视图体积之外的几何体；

*遮挡剔除—移除可能位于视图体积内但被靠近相机的其他几何体遮挡或遮挡的几何体；

*背面剔除—去除远离相机的基元。

我们将简要讨论视图体积剔除和背面剔除，但在高性能系统中剔除是一个复杂的话题；参见(Akenine-Möller et al., 2008) 进行完整的讨论和有关遮挡剔除的信息。

8.4.1 查看卷剔除

当整个图元位于视图体积之外时，可以对其进行剔除，因为它在栅格化时不会产生任何片段。如果我们可以快速测试剔除许多图元，我们可能能够显著加快绘图速度。另一方面，单独测试基元以确定需要绘制哪些基元可能比让光栅化器消除它们花费更多。

视图体积剔除（Viewvolumeculling），也称为视图截头剔除（viewfrustumculling），在将许多三角形分组到具有关联边界体积的对象中时尤其有帮助。如果边界体积位于视图体积之外，那么构成对象的所有三角形也是如此。例如，如果我们有1000个三角形，由中心 c 和半径 r 的单个球体限定，我们可以检查球体是否位于裁剪平面之外

$$(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0,$$

where a is a point on the plane, and p is a variable. This is equivalent to checking whether the signed distance from the center of the sphere c to the plane is greater than $+r$. This amounts to the check that

$$\frac{(c - a) \cdot n}{\|n\|} > r.$$

Note that the sphere may overlap the plane even in a case where all the triangles do lie outside the plane. Thus, this is a conservative test. How conservative the test depends on how well the sphere bounds the object.

The same idea can be applied hierarchically if the scene is organized in one of the spatial data structures described in Chapter 12.

8.4.2 Backface Culling

When polygonal models are closed, i.e., they bound a closed space with no holes, then they are often assumed to have outward facing normal vectors as discussed in Chapter 10. For such models, the polygons that face away from the eye are certain to be overdrawn by polygons that face the eye. Thus, those polygons can be culled before the pipeline even starts. The test for this condition is the same one used for silhouette drawing given in Section 10.3.1.

Frequently Asked Questions

- I've often seen clipping discussed at length, and it is a much more involved process than that described in this chapter. What is going on here?

The clipping described in this chapter works, but lacks optimizations that an industrial-strength clipper would have. These optimizations are discussed in detail in Blinn's definitive work listed in the chapter notes.

- How are polygons that are not triangles rasterized?

These can either be done directly scan-line by scan-line, or they can be broken down into triangles. The latter appears to be the more popular technique.

- Is it always better to antialias?

No. Some images look crisper without antialiasing. Many programs use unantialiased "screen fonts" because they are easier to read.

其中 a 是平面上的点， p 是变量。这相当于检查球体中心 c 到平面的带符号距离是否大于 $+r$ 。这相当于支票

$$\frac{(c - a) \cdot n}{\|n\|} > r.$$

请注意，即使在所有三角形都位于平面之外的情况下，球体也可能与平面重叠。因此，这是一个保守的测试。测试的保守程度取决于球体对物体的边界程度。

如果场景组织在第12章中描述的空间数据结构之一中，则可以分层应用相同的想法。

8.4.2 Backface Culling

当多边形模型是封闭的，即它们绑定了一个没有孔的封闭空间，那么它们通常被假定为具有面向外的法向量，如第10章所讨论的。对于这样的模型，朝向远离眼睛的多边形肯定会被朝向眼睛的多边形过度绘制。因此，这些多边形可以在管道开始之前被剔除。此条件的测试与第10.3.1节中给出的用于轮廓绘制的测试相同。

常见问题

- 我经常看到剪辑被详细讨论，这是一个比本章描述的更复杂的过程。这是怎么回事？

本章中描述的裁剪工作，但缺乏工业强度裁剪器所具有的优化。这些优化在章节笔记中列出的Blinn的权威工作中详细讨论。

- *不是三角形的多边形如何栅格化？

这些可以直接逐行扫描，也可以分解成三角形。后者似乎是更流行的技术。

- *Antialias总是更好吗？

非也。有些图像看起来更清晰，没有抗锯齿。许多程序使用未经修改的"屏幕字体"，因为它们更容易阅读。



- The documentation for my API talks about “scene graphs” and “matrix stacks.” Are these part of the graphics pipeline?

The graphics pipeline is certainly designed with these in mind, and whether we define them as part of the pipeline is a matter of taste. This book delays their discussion until Chapter 12.

- Is a uniform distance z-buffer better than the standard one that includes perspective matrix nonlinearities?

It depends. One “feature” of the nonlinearities is that the z-buffer has more resolution near the eye and less in the distance. If a level-of-detail system is used, then geometry in the distance is coarser and the “unfairness” of the z-buffer can be a good thing.

- Is a software z-buffer ever useful?

Yes. Most of the movies that use 3D computer graphics have used a variant of the software z-buffer developed by Pixar (Cook, Carpenter, & Catmull, 1987).

Notes

A wonderful book about designing a graphics pipeline is *Jim Blinn's Corner: A Trip Down the Graphics Pipeline* (J. Blinn, 1996). Many nice details of the pipeline and culling are in *3D Game Engine Design* (Eberly, 2000) and *Real-Time Rendering* (Akenine-Möller et al., 2008).

Exercises

1. Suppose that in the perspective transform we have $n = 1$ and $f = 2$. Under what circumstances will we have a “reversal” where a vertex before and after the perspective transform flips from in front of to behind the eye or vice versa?
2. Is there any reason not to clip in x and y after the perspective divide (see Figure 11.2, stage 3)?
3. Derive the incremental form of the midpoint line-drawing algorithm with colors at endpoints for $0 < m \leq 1$.



- 我的API的文档谈论“场景图”和“矩阵堆栈。”这些是图形管道的一部分吗？

图形管道的设计当然考虑到了这些，我们是否将它们定义为管道的一部分是一个品味问题。这本书将他们的讨论推迟到第12章。

- *均匀距离z缓冲区是否优于包含透视矩阵非线性的标准缓冲区？

这得看情况而定。非线性的一个“特征”是z缓冲区在眼睛附近有更多的分辨率，在距离上有更少的分辨率。如果使用细节级别系统，则距离中的几何形状更粗糙，z缓冲区的“不公平”可能是一件好事。

- *软件z-buffer是否有用？

是的。大多数使用3d计算机图形的电影都使用了皮克斯开发的软件z-buffer的变体 (Cook, Carpenter, & Catmull, 1987)。

Notes

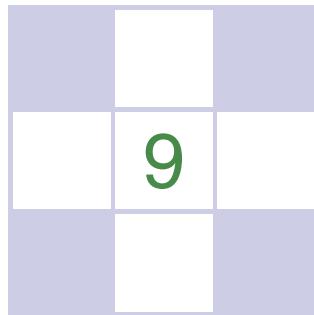
关于设计图形管道的一本精彩的书是 JimBlinn'SCorner:ATripDownTheGraphicsPipeline (J.Blinn, 1996)。管道和剔除的许多不错的细节都在3D游戏引擎设计 (Eberly, 2000) 和实时渲染 (Akenine-Möller et al. 2008)。

Exercises

1. 假设在透视变换中，我们有 $n=1$ 和 $f=2$ 。在什么情况下，我们会有一个“反转”，其中透视变换前后的顶点从眼睛前面翻转到眼睛后面，反之亦然？
2. 是否有任何理由不在透视分割后的x和y中剪辑（见图11.2，第3阶段）？
3. 推导出端点颜色为 $0 < m \leq 1$ 的中点画线算法的增量形式。

4. Modify the triangle-drawing algorithm so that it will draw exactly one pixel for points on a triangle edge which goes through $(x, y) = (-1, -1)$.
5. Suppose you are designing an integer z-buffer for flight simulation where all of the objects are at least one meter thick, are never closer to the viewer than 4 meters, and may be as far away as 100 km. How many bits are needed in the z-buffer to ensure there are no visibility errors? Suppose that visibility errors only matter near the viewer, i.e., for distances less than 100 meters. How many bits are needed in that case?

- 4.修改三角形绘制算法，以便它将为经过 (x, y) 的三角形边上的点绘制一个像素)=(1 1).
- 5.假设您正在为飞行模拟设计一个整数z缓冲区，其中所有对象都至少有一米厚，距离观察者永远不会超过4米，并且可能距离100公里。Z-buffer中需要多少位才能确保没有可见性错误？假设可见性误差仅在观看者附近重要，即对于小于100米的距离。在这种情况下需要多少位？

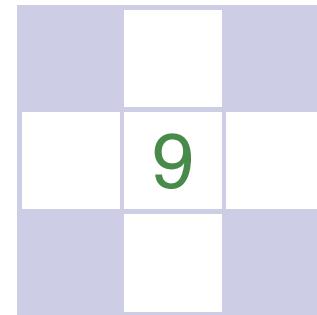


Signal Processing

In graphics, we often deal with functions of a continuous variable: an image is the first example you have seen, but you will encounter many more as you continue your exploration of graphics. By their nature, continuous functions can't be directly represented in a computer; we have to somehow represent them using a finite number of bits. One of the most useful approaches to representing continuous functions is to use *samples* of the function: just store the values of the function at many different points and *reconstruct* the values in between when and if they are needed.

You are by now familiar with the idea of representing an image using a two-dimensional grid of pixels—so you have already seen a sampled representation! Think of an image captured by a digital camera: the actual image of the scene that was formed by the camera's lens is a continuous function of the position on the image plane, and the camera converted that function into a two-dimensional grid of samples. Mathematically, the camera converted a function of type $\mathbb{R}^2 \rightarrow \mathbf{C}$ (where \mathbf{C} is the set of colors) to a two-dimensional array of color samples, or a function of type $\mathbb{Z}^2 \rightarrow \mathbf{C}$.

Another example of a sampled representation is a 2D digitizing tablet, such as the screen of a tablet computer or a separate pen tablet used by an artist. In this case, the original function is the motion of the stylus, which is a time-varying 2D position, or a function of type $\mathbb{R} \rightarrow \mathbb{R}^2$. The digitizer measures the position of the stylus at many points in time, resulting in a sequence of 2D coordinates, or a function of type $\mathbb{Z} \rightarrow \mathbb{R}^2$. A *motion capture* system does exactly the same thing



信号处理

在图形学中，我们经常处理连续变量的函数：图像是你见过的第一个例子，但是当你探索图形时，你会遇到更多的例子。就其性质而言，连续函数不能在计算机中直接表示；我们必须以某种方式使用有限数量的位来表示它们。表示微小函数的最有用的方法之一是使用函数的样本：只需将函数的值存储在许多不同的点上，并在需要时和需要时重建它们之间的值。

您现在已经熟悉了使用二维像素网格表示图像的想法——所以您已经看到了一个采样表示！想象一下数码相机拍摄的图像：由相机镜头形成的场景的实际图像是图像平面上位置的连续函数，相机将该函数转换为样本的二维网格。在数学上，相机将类型为 $\mathbb{R}^2 \rightarrow \mathbf{C}$ 的函数（其中 \mathbf{C} 是颜色集）转换为颜色样本的二维数组，或类型为 $\mathbb{Z}^2 \rightarrow \mathbf{C}$ 的函数。

采样表示的另一示例是2D数字化平板，例如平板计算机的屏幕或艺术家使用的单独的笔平板。在这种情况下，原始函数是触控笔的运动，这是一个时变的2D位置，或者是 $\mathbb{R} \rightarrow \mathbb{R}^2$ 类型的函数。数字化仪测量手写笔在许多时间点的位置，从而产生一系列2D坐标，或 $\mathbb{Z} \rightarrow \mathbb{R}^2$ 类型的函数。动作捕捉系统做同样的事情

for a special marker attached to an actor's body: it takes the 3D position of the marker over time ($\mathbb{R} \rightarrow \mathbb{R}^3$) and makes it into a series of instantaneous position measurements ($\mathbb{Z} \rightarrow \mathbb{R}^3$).

Going up in dimension, a medical CT scanner, used to non-invasively examine the interior of a person's body, measures density as a function of position inside the body. The output of the scanner is a 3D grid of density values: it converts the density of the body ($\mathbb{R}^3 \rightarrow \mathbb{R}$) to a 3D array of real numbers ($\mathbb{Z}^3 \rightarrow \mathbb{R}$).

These examples seem different, but in fact they can all be handled using exactly the same mathematics. In all cases a function is being sampled at the points of a *lattice* in one or more dimensions, and in all cases we need to be able to reconstruct that original continuous function from the array of samples.

From the example of a 2D image, it may seem that the pixels are enough, and we never need to think about continuous functions again once the camera has discretized the image. But what if we want to make the image larger or smaller on the screen, particularly by non-integer scale factors? It turns out that the simplest algorithms to do this perform badly, introducing obvious visual artifacts known as *aliasing*. Explaining why aliasing happens and understanding how to prevent it require the mathematics of sampling theory. The resulting algorithms are rather simple, but the reasoning behind them, and the details of making them perform well, can be subtle.

Representing continuous functions in a computer is, of course, not unique to graphics; nor is the idea of sampling and reconstruction. Sampled representations are used in applications from digital audio to computational physics, and graphics is just one (and by no means the first) user of the related algorithms and mathematics. The fundamental facts about how to do sampling and reconstruction have been known in the field of communications since the 1920s and were stated in exactly the form we use them by the 1940s (Shannon & Weaver, 1964).

This chapter starts by summarizing sampling and reconstruction using the concrete one-dimensional example of digital audio. Then, we go on to present the basic mathematics and algorithms that underlie sampling and reconstruction in one and two dimensions. Finally, we go into the details of the frequency-domain viewpoint, which provides many insights into the behavior of these algorithms.

9.1 Digital Audio: Sampling in 1D

Although sampled representations had already been in use for years in telecommunications, the introduction of the compact disc in 1982, following the increased use of digital recording for audio in the previous decade, was the first highly visible consumer application of sampling.

对于附着在演员身体上的特殊标记：它获取标记随时间的3D位置 ($R \rightarrow R^3$)，并使其成为一系列瞬时位置测量 ($Z \rightarrow R^3$)。

在维度上升，一个医疗CT扫描仪，用于非侵入性地检查一个人的身体内部，测量密度作为身体内部位置的函数。扫描仪的输出是密度值的3D网格：它将身体的密度 ($R^3 \rightarrow R$) 转换为实数的3d数组 ($Z^3 \rightarrow R$)。

这些例子看起来不同，但实际上它们都可以使用exactly相同的数学来处理。在所有情况下，函数都是在一个或多个维度的晶格点上采样的，在所有情况下，我们都需要能够从样本数组中重建原始连续函数。

从2D图像的例子来看，像素似乎已经足够了，一旦相机对图像进行了离散化，我们就再也不需要考虑连续函数了。但是，如果我们想在屏幕上使图像变大或变小，特别是通过非整数比例因子呢？事实证明，这样做的最简单算法表现不佳，引入了明显的视觉伪像，称为混叠。解释混叠发生的原因并理解如何防止它需要抽样理论的数学。由此产生的算法相当简单，但它们背后的推理以及使它们表现良好的细节可能是微妙的。

在计算机中表示连续函数当然不是图形所独有的；采样和重建的想法也不是。采样表示用于从数字音频到计算物理的应用，而图形只是相关算法和mathematics的一个（绝不是第一个）用户。自20世纪20年代以来，关于如何进行采样和重建的基本事实已经在通信领域为人所知，并且在20世纪40年代以我们使用它们的形式陈述（Shannon & Weaver, 1964）。

本章首先使用数字音频的具体一维示例总结采样和重建。然后，我们继续介绍基本的数学和算法，这些算法是一维和二维采样和重建的基础。最后，我们进入频域视点的细节，它为这些算法的行为提供了许多见解。

9.1 数字音频：一维采样

虽然采样表示已经在电信中使用多年，但在过去十年中音频数字记录的使用增加之后，1982年推出了光盘，这是采样的第一个高度可见的消费者应用。

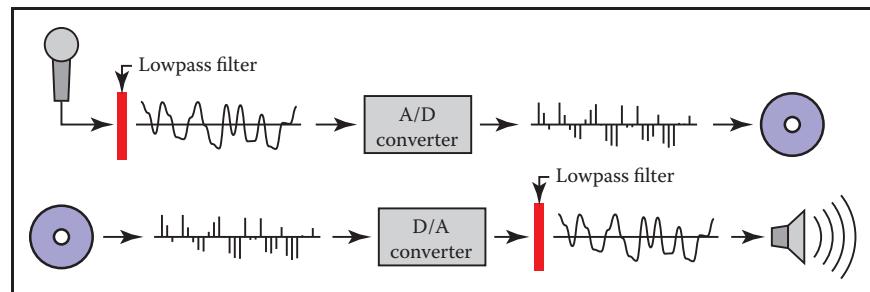


Figure 9.1. Sampling and reconstruction in digital audio.

In audio recording, a microphone converts sound, which exists as pressure waves in the air, into a time-varying voltage that amounts to a measurement of the changing air pressure at the point where the microphone is located. This electrical signal needs to be stored somehow so that it may be played back at a later time and sent to a loudspeaker that converts the voltage back into pressure waves by moving a diaphragm in synchronization with the voltage.

The digital approach to recording the audio signal (Figure 9.1) uses sampling: an *analog-to-digital converter* (*A/D converter*, or *ADC*) measures the voltage many thousand times per second, generating a stream of integers that can easily be stored on any number of media, say a disk on a computer in the recording studio, or transmitted to another location, say the memory in a portable audio player. At playback time, the data is read out at the appropriate rate and sent to a *digital-to-analog converter* (*D/A converter*, or *DAC*). The DAC produces a voltage according to the numbers it receives, and, provided we take enough samples to fairly represent the variation in voltage, the resulting electrical signal is, for all practical purposes, identical to the input.

It turns out that the number of samples per second required to end up with a good reproduction depends on how high-pitched the sounds are that we are trying to record. A sample rate that works fine for reproducing a string bass or a kick drum produces bizarre-sounding results if we try to record a piccolo or a cymbal; but those sounds are reproduced just fine with a higher sample rate. To avoid these *undersampling artifacts* the digital audio recorder *filters* the input to the ADC to remove high frequencies that can cause problems.

Another kind of problem arises on the output side. The DAC produces a voltage that changes whenever a new sample comes in, but stays constant until the next sample, producing a stair-step shaped graph. These stair-steps act like noise, adding a high-frequency, signal-dependent buzzing sound. To remove this *reconstruction artifact*, the digital audio player filters the output from the DAC to smooth out the waveform.

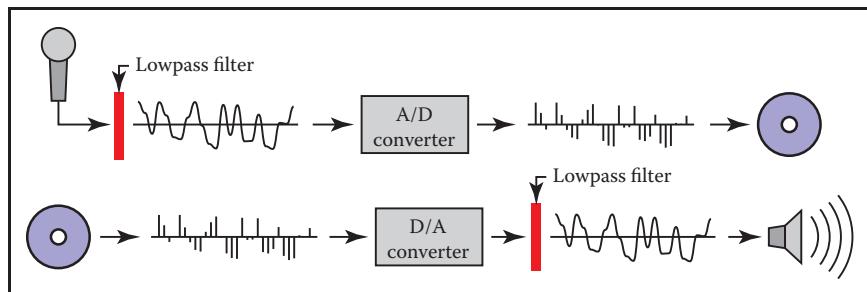


图9.1。数字音频中的采样和重建。

在音频录制中，麦克风将空气中作为压力波存在的声音转换为时变电压，该电压相当于麦克风所在点处不断变化的气压的测量。这种电信号需要以某种方式存储，以便它可以在稍后的时间回放，并发送到扬声器，该扬声器通过与电压同步移动隔膜将电压转换回压力波。

记录音频信号的数字方法（图9.1）使用采样：模数转换器（D转换器或ADC）每秒测量数千次电压，生成一个整数流，可以存储在任何数量的媒体上，比如录音室计算机上的磁盘，或者传输到另一个位置，比如便携式音频播放器中的内存。在回放时，数据以适当的速率读出并被发送到数模转换器（Da转换器，或DAC）。DAC根据它接收到的数字产生一个伏龄，并且，如果我们采取足够的样本来公平地表示电压的变化，就所有实际目的而言，产生的电信号与输入相同。

事实证明，以良好的再现结束所需的每秒样本数量取决于我们试图记录的声音有多高。如果我们尝试录制短笛或铙钹，可以很好地再现弦低音或大鼓的采样率会产生听起来奇怪的结果；但这些声音以更高的采样率再现得很好。为了避免这些欠采样伪影，数字音频记录器对ADC的输入进行滤波，以消除可能导致问题的高频率。

另一种问题出现在输出端。DAC产生的电压在新样品进入时会发生变化，但直到下一个样品保持不变，从而产生阶梯形图形。这些楼梯台阶就像噪音一样，增加了高频率，信号依赖的嗡嗡声。为了消除这种重建伪影，数字音频播放器对DAC的输出进行滤波，以平滑波形。

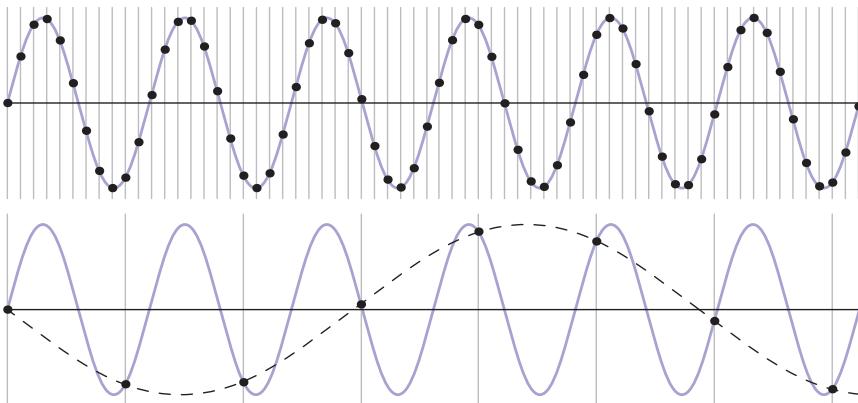


Figure 9.2. A sine wave (blue curve) sampled at two different rates. Top: at a high sample rate, the resulting samples (black dots) represent the signal well. Bottom: a lower sample rate produces an ambiguous result: the samples are exactly the same as would result from sampling a wave of much lower frequency (dashed curve).

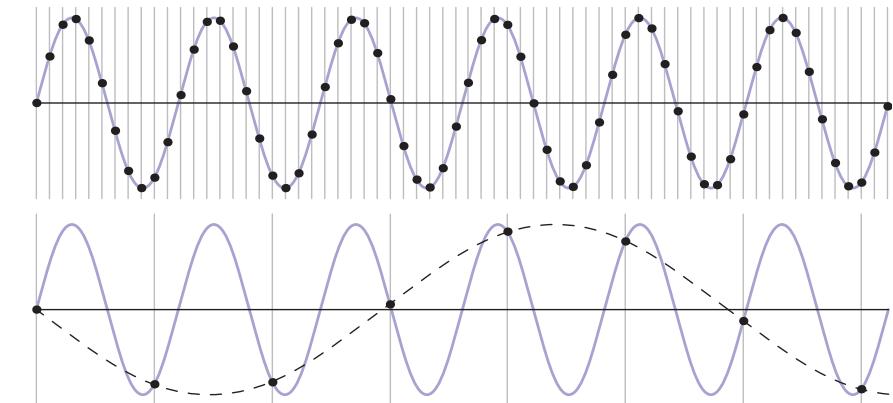
9.1.1 Sampling Artifacts and Aliasing

The digital audio recording chain can serve as a concrete model for the sampling and reconstruction processes that happen in graphics. The same kind of undersampling and reconstruction artifacts also happen with images or other sampled signals in graphics, and the solution is the same: filtering before sampling and filtering again during reconstruction.

A concrete example of the kind of artifacts that can arise from too-low sample frequencies is shown in Figure 9.2. Here we are sampling a simple sine wave using two different sample frequencies: 10.8 samples per cycle on the top and 1.2 samples per cycle on the bottom. The higher rate produces a set of samples that obviously capture the signal well, but the samples resulting from the lower sample rate are indistinguishable from samples of a low-frequency sine wave—in fact, faced with this set of samples the low-frequency sinusoid seems the more likely interpretation.

Once the sampling has been done, it is impossible to know which of the two signals—the fast or the slow sine wave—was the original, and therefore there is no single method that can properly reconstruct the signal in both cases. Because the high-frequency signal is “pretending to be” a low-frequency signal, this phenomenon is known as *aliasing*.

Aliasing shows up whenever flaws in sampling and reconstruction lead to artifacts at surprising frequencies. In audio, aliasing takes the form of odd-sounding extra tones—a bell ringing at 10KHz, after being sampled at 8KHz, turns into a



以两种不同速率采样的正弦波（蓝色曲线）。顶部：在高采样率下，得到的样本（黑点）很好地代表了信号。底部：较低的采样率会产生一个模糊的结果：样本与采样频率低得多的波（虚线曲线）的结果完全相同。

9.1.1 采样伪像和混叠

数字音频记录链可以作为图形中发生的采样和重建过程的具体模型。同样的欠采样和重建伪像也发生在图形中的图像或其他采样信号中，解决方案是相同的：在采样之前进行滤波并在重建期间再次进行滤波。

图9.2显示了样本频率太低可能产生的伪像的具体例子。在这里，我们使用两个不同的采样频率对一个简单的正弦波进行采样：顶部每个周期10.8个采样，底部每个周期1.2个采样。较高的速率产生了一组明显很好地捕获信号的样本，但较低的样本速率产生的样本与低频正弦波的样本无法区分—事实上，面对这组样本，低频正弦波似乎更有可能解释。

一旦采样完成，就不可能知道两个信号中的哪一个—快速或慢速正弦波—是原始的，因此没有一种方法可以在这两种情况下正确地重建信号。因为高频信号是“假装成”低频信号，所以这种相位混叠被称为混叠。

每当采样和重建中的缺陷以令人惊讶的频率导致错误事实时，就会出现混叠。在音频中，混叠采用听起来奇怪的额外音调的形式——在8kHz采样后，10kHz的钟声会变成一个



6kHz tone. In images, aliasing often takes the form of *moiré patterns* that result from the interaction of the sample grid with regular features in an image, for instance the window blinds in Figure 9.34.

Another example of aliasing in a synthetic image is the familiar stair-stepping on straight lines that are rendered with only black and white pixels (Figure 9.34). This is an example of small-scale features (the sharp edges of the lines) creating artifacts at a different scale (for shallow-slope lines the stair steps are very long).

The basic issues of sampling and reconstruction can be understood simply based on features being too small or too large, but some more quantitative questions are harder to answer:

- What sample rate is high enough to ensure good results?
- What kinds of filters are appropriate for sampling and reconstruction?
- What degree of smoothing is required to avoid aliasing?

Solid answers to these questions will have to wait until we have developed the theory fully in Section 9.5

9.2 Convolution

Before we discuss algorithms for sampling and reconstruction, we'll first examine the mathematical concept on which they are based—*convolution*. Convolution is a simple mathematical concept that underlies the algorithms that are used for sampling, filtering, and reconstruction. It also is the basis of how we will analyze these algorithms later in the chapter.

Convolution is an operation on functions: it takes two functions and combines them to produce a new function. In this book, the convolution operator is denoted by a star: the result of applying convolution to the functions f and g is $f \star g$. We say that f is convolved with g , and $f \star g$ is the convolution of f and g .

Convolution can be applied either to continuous functions (functions $f(x)$ that are defined for any real argument x) or to discrete sequences (functions $a[i]$ that are defined only for integer arguments i). It can also be applied to functions defined on one-dimensional, two-dimensional, or higher-dimensional domains (that is, functions of one, two, or more arguments). We will start with the discrete, one-dimensional case first, then continue to continuous functions and two- and three-dimensional functions.

For convenience in the definitions, we generally assume that the functions' domains go on forever, though of course in practice they will have to stop somewhere, and we have to handle the endpoints in a special way.



6khz的音调。在图像中，混叠通常采用莫尔图案的形式，这些图案从样本网格与图像中常规特征的相互作用中重新分离出来，例如图9.34中的百叶窗。

合成图像中混叠的另一个例子是熟悉的楼梯-踩在只有黑白像素渲染的直线上（图9.34）。这是一个小尺度特征（线条的尖锐边缘）在不同尺度上创建伪像的例子（对于浅斜坡线，楼梯台阶非常长）。采样和重建的基本问题可以简单地根据特征太小或太大来理解，但一些更定量的问题更难回答：

- *什么采样率足够高，以确保良好的结果？
- *什么样的滤波器适合采样和重建？
- *需要多大程度的平滑才能避免混叠？

这些问题的可靠答案将不得不等待，直到我们在第9.5节中充分发展了理论

9.2 Convolution

在我们讨论采样和重建算法之前，我们将首先检查它们所基于的数学概念—卷积。卷积是一个简单的数学概念，它是用于采样、滤波和重建的算法的基础。这也是我们将在本章后面分析这些算法的基础。

卷积是对函数的操作：它采用两个函数并将它们组合以产生一个新函数。在本书中，卷积运算符用星形表示：对函数 f 和 g 应用卷积的结果是 $f \star g$ 。我们说 f 与 g 卷积， $f \star g$ 是 f 与 g 的卷积。

卷积可以应用于连续函数（为任何实变量 x 定义的函数 $f(x)$ ）或离散序列（仅为整数参数 i 定义的函数 $a[i]$ ）。它也可以应用于一维、二维或更高维域上的函数（即一个、两个或多个自变量的函数）。我们将首先从离散的一维情况开始，然后继续连续函数和两个连续的三维函数。

为了方便定义，我们通常假设函数的域永远持续下去，当然在实践中它们必须停止某些地方，我们必须以特殊的方式处理端点。

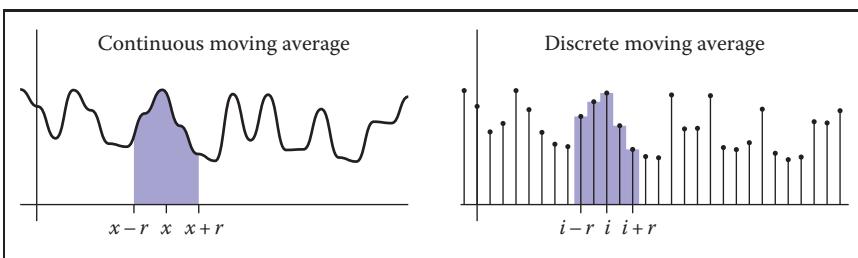


Figure 9.3. Smoothing using a moving average.

9.2.1 Moving Averages

To get a basic picture of convolution, consider the example of smoothing a 1D function using a moving average (Figure 9.3). To get a smoothed value at any point, we compute the average of the function over a range extending a distance r in each direction. The distance r , called the *radius* of the smoothing operation, is a parameter that controls how much smoothing happens.

We can state this idea mathematically for discrete or continuous functions. If we're smoothing a continuous function $g(x)$, averaging means integrating g over an interval and then dividing by the length of the interval:

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

On the other hand, if we're smoothing a discrete function $a[i]$, averaging means summing a for a range of indices and dividing by the number of values:

$$c[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} a[j]. \quad (9.1)$$

In each case, the normalization constant is chosen so that if we smooth a constant function the result will be the same function.

This idea of a moving average is the essence of convolution; the only difference is that in convolution the moving average is a weighted average.

9.2.2 Discrete Convolution

We will start with the most concrete case of convolution: convolving a discrete sequence $a[i]$ with another discrete sequence $b[i]$. The result is a discrete sequence $(a * b)[i]$. The process is just like smoothing a with a moving average, but this

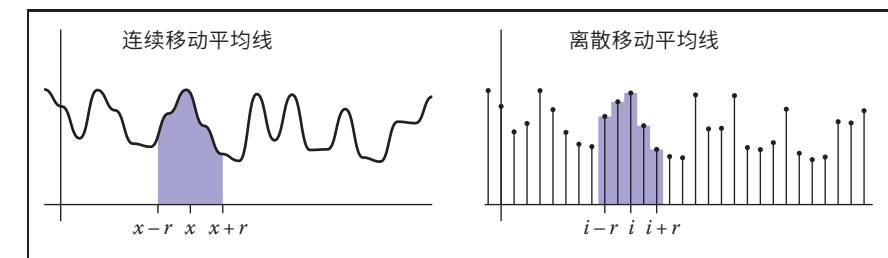


图9.3。使用移动平均线平滑。

9.2.1 移动平均线

要获得卷积的基本图像，请考虑使用移动平均线平滑1D函数的示例（图9.3）。为了在任何点获得平滑值，我们计算函数在每个方向延伸距离 r 的范围内的平均值。距离 r ，称为平滑操作的半径，是控制平滑发生多少的参数。

对于离散或连续函数，我们可以用数学方法陈述这个想法。如果我们平滑一个连续函数 $g(x)$ ，平均意味着在一个区间上积分 g ，然后除以区间的长度：

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

另一方面，如果我们平滑离散函数 $a[i]$ ，平均意味着对一系列指数求和并除以值的数量：

$$c[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} a[j]. \quad (9.1)$$

在每种情况下，选择归一化常数，以便如果我们平滑一个常数函数，结果将是相同的函数。

移动平均线的这种想法是卷积的本质；唯一不同的是，在卷积中，移动平均线是加权平均值。

9.2.2 离散卷积

我们将从最具体的卷积案例开始：将一个离散序列 $a[i]$ 与另一个离散序列 $b[i]$ 卷积。结果是离散序列 $(a * b)[i]$ 。这个过程就像用移动平均线平滑 a 一样，但是这

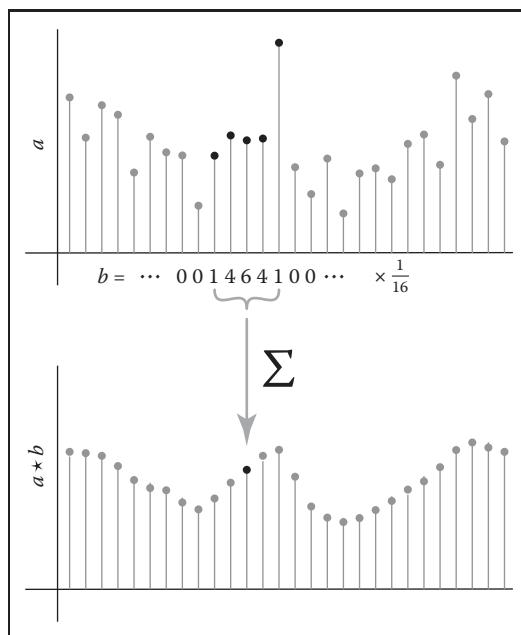


Figure 9.4. Computing one value in the discrete convolution of a sequence a with a filter b that has support five samples wide. Each sample in $a \star b$ is an average of nearby samples in a , weighted by the values of b .

time instead of equally weighting all samples within a distance r , we use a second sequence b to give a weight to each sample (Figure 9.4). The value $b[i - j]$ gives the weight for the sample at position j , which is at a distance $i - j$ from the index i where we are evaluating the convolution. Here is the definition of $(a \star b)$, expressed as a formula:

$$(a \star b)[i] = \sum_j a[j]b[i - j]. \quad (9.2)$$

By omitting bounds on j , we indicate that this sum runs over all integers (that is, from $-\infty$ to $+\infty$). Figure 9.4 illustrates how one output sample is computed, using the example of $b = \frac{1}{16}[\dots, 0, 1, 4, 6, 4, 1, 0, \dots]$ —that is, $b[0] = \frac{6}{16}$, $a[\pm 1] = \frac{4}{16}$, etc.

In graphics, one of the two functions will usually have *finite support* (as does the example in Figure 9.4), which means that it is nonzero only over a finite interval of argument values. If we assume that b has finite support, there is some *radius* r such that $b[k] = 0$ whenever $|k| > r$. In that case, we can write the sum

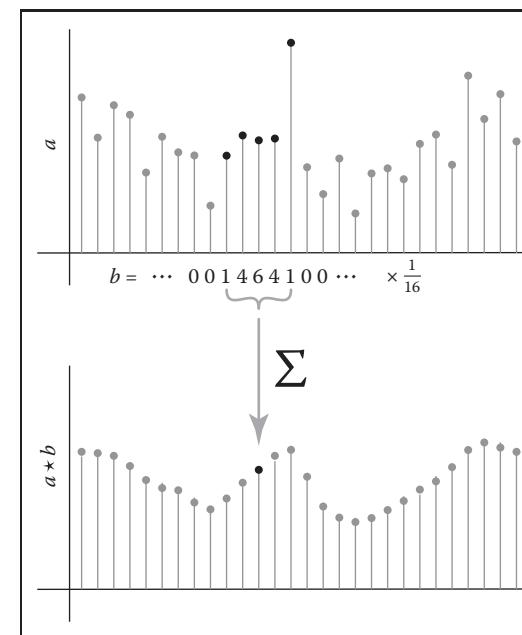


图9.4。 用支持五个样本宽的滤波器b计算序列a的离散卷积中的一个值。A∈b中的每个样本是a中附近样本的平均值，由b的值加权。

时间而不是平均加权距离 r 内的所有样本，我们使用第二个序列**b**给每个样本赋予权重（图9.4）。值 $b[i - j]$ 给出了位置*j*处样本的权重，该位置与我们评估卷积的索引*i*相距*i-j*。这里是 $(a \star b)$ 的定义，用公式表示：

$$(a \star b)[i] = \sum_j a[j]b[i - j]. \quad (9.2)$$

通过省略j的边界，我们表明这个总和在所有整数上运行（即从 $-\infty$ 到 $+\infty$ ）。图9.4用b=的例子说明了一个输出样本是如何被压缩的

$$a[-1] = \frac{4}{16}, \text{etc.}$$

在图形中，两个函数中的一个通常具有有限支撑（如图9.4中的示例），这意味着它仅在参数值的有限区间内非零。如果我们假设**b**具有有限支撑，则存在一些半径 r ，使得每当 $k>r$ 时 $b[k]=0$ 。在这种情况下，我们可以写出总和

above as

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

and we can express the definition in code as

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

Convolution Filters

Convolution is important because we can use it to perform filtering. Looking back at our first example of filtering, the moving average, we can now reinterpret that smoothing operation as convolution with a particular sequence. When we compute an average over some limited range of indices, that is the same as weighting the points in the range all identically and weighting the rest of the points with zeros. This kind of filter, which has a constant value over the interval where it is nonzero, is known as a *box filter* (because it looks like a rectangle if you draw its graph—see Figure 9.5). For a box filter of radius r the weight is $1/(2r+1)$:

$$b[k] = \begin{cases} \frac{1}{2r+1} & -r \leq k \leq r, \\ 0 & \text{otherwise.} \end{cases}$$

If you substitute this filter into Equation (9.2), you will find that it reduces to the moving average in Equation (9.1).

As in this example, convolution filters are usually designed so that they sum to 1. That way, they don't affect the overall level of the signal.

Example (Convolution of a box and a step). For a simple example of filtering, let the signal be the *step function*

$$a[i] = \begin{cases} 1 & i \geq 0, \\ 0 & i < 0, \end{cases}$$

and the filter be the five-point box filter centered at zero,

$$b[k] = \frac{1}{5} \begin{cases} 1 & -2 \leq k \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

以上为

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

我们可以将代码中的定义表示为

```
函数卷积 (序列a, 滤波器b, inti) s=0
=b.半径为j=i-r到i+r做s=s+a[j]b[i-j]返
回s
```

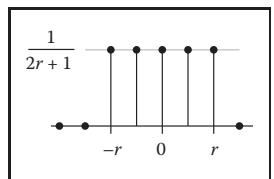


Figure 9.5. A discrete box filter.

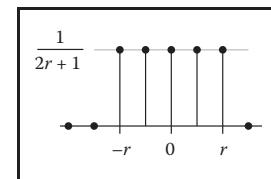


Figure 9.5. 一个离散的盒子滤波器。

卷积滤波器

卷积很重要，因为我们可以用它来执行过滤。回顾我们的第一个滤波示例，即移动平均线，我们现在可以将平滑操作重新解释为与特定序列的卷积。当我们有一些有限的指数范围内计算平均值时，这与对范围内的点进行相同的加权，并用零对其余点进行加权相同。这种过滤器在非零的区间内具有恒定值，被称为框过滤器（因为如果你绘制它的矩形，它看起来像一个矩形）。

图-见图9.5）。对于半径为r的箱形滤波器，权重为1(2r+1):

$$b[k] = \begin{cases} \frac{1}{2r+1} & -r \leq k \leq r, \\ 0 & \text{otherwise.} \end{cases}$$

如果将这个滤波器代入公式 (9.2)，你会发现它减少到公式 (9.1) 中的移动平均线。

如在这个例子中，卷积滤波器通常被设计成使它们总和为1。这样，它们不会影响信号的整体电平。

例 (一个框和一个步骤的卷积)。 对于一个简单的滤波示例，设信号为阶跃函数

$$a[i] = \begin{cases} 1 & i \geq 0, \\ 0 & i < 0, \end{cases}$$

并且滤波器是以零为中心的五点盒滤波器

$$b[k] = \frac{1}{5} \begin{cases} 1 & -2 \leq k \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

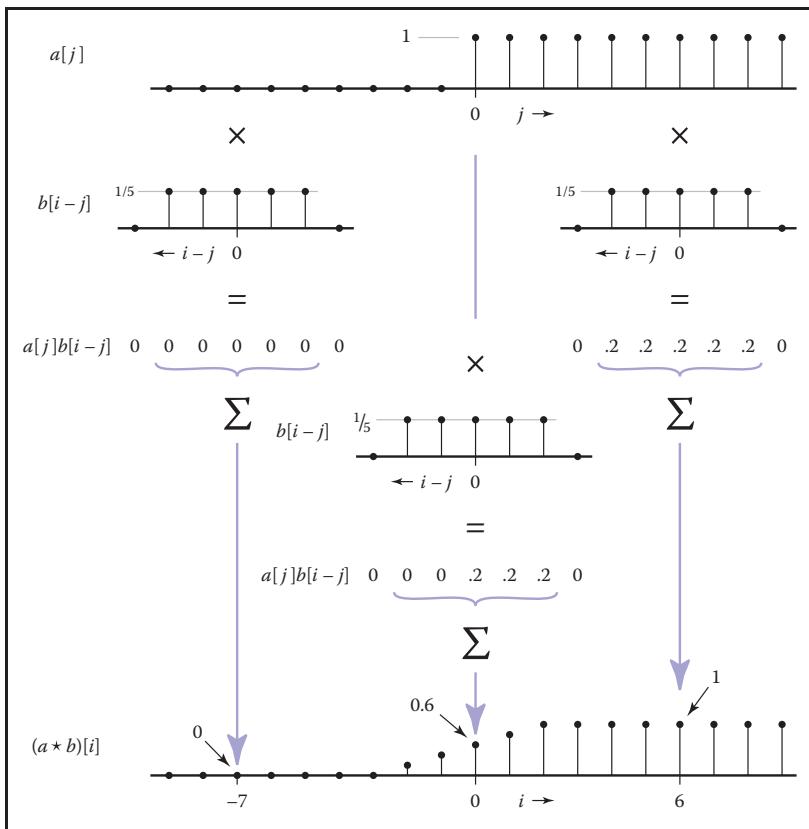


Figure 9.6. Discrete convolution of a box function with a step function.

What is the result of convolving a and b ? At a particular index i , as shown in Figure 9.6, the result is the average of the step function over the range from $i - 2$ to $i + 2$. If $i < -2$, we are averaging all zeros and the result is zero. If $i \geq 2$, we are averaging all ones and the result is one. In between there are $i + 3$ ones, resulting in the value $\frac{i+3}{5}$. The output is a linear ramp that goes from 0 to 1 over five samples: $\frac{1}{5}[\dots, 0, 0, 1, 2, 3, 4, 5, 5, \dots]$.

Properties of Convolution

The way we've written it so far, convolution seems like an asymmetric operation: a is the sequence we're smoothing, and b provides the weights. But one of the nice properties of convolution is that it actually doesn't make any difference which is which: the filter and the signal are interchangeable. To see this, just rethink the sum in Equation (9.2) with the indices counting from the origin of the filter b ,

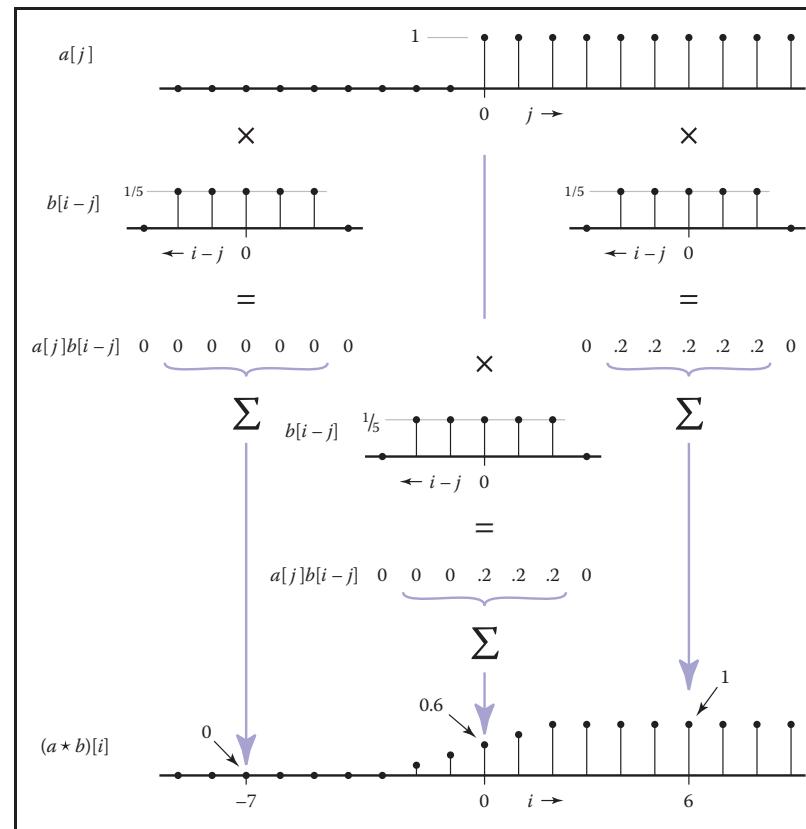


图9.6。具有阶跃函数的盒函数的离散卷积。

卷积a和b的结果是什么？在特定指数*i*处，如图9.6所示，结果是阶跃函数在从*i*-2到*i*+2范围内的平均值。如果*i*<-2，我们平均所有零，结果为零。如果*i*≥2，我们平均所有的，结果是一个。之间有*i*+3个，导致值*i*+3

5.输出是一个线性斜坡，从0到1超过
five samples: $\frac{1}{5}[\dots, 0, 0, 1, 2, 3, 4, 5, 5, \dots]$.

卷积的属性

到目前为止，卷积似乎是一个不对称的操作： a 是我们平滑的序列， b 提供权重。但是卷积的一个很好的特性是它实际上并没有任何区别：滤波器和信号是可以互换的。为了看到这一点，只需重新考虑公式（9.2）中的总和，其中索引从滤波器**b**的原点开始计数

rather than from the origin of a . That is, we replace j with $i - k$. The result of this change of variable is

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

This is exactly the same as Equation (9.2) but with a acting as the filter and b acting as the signal. So for any sequences a and b , $(a * b) = (b * a)$, and we say that convolution is a *commutative* operation.¹

More generally, convolution is a “multiplication-like” operation. Like multiplication or addition of numbers or functions, neither the order of the arguments nor the placement of parentheses affects the result. Also, convolution relates to addition in the same way that multiplication does. To be precise, convolution is *commutative* and *associative*, and it is *distributive* over addition.

commutative:	$(a * b)[i] = (b * a)[i]$
associative:	$(a * (b * c))[i] = ((a * b) * c)[i]$
distributive:	$(a * (b + c))[i] = (a * b + a * c)[i]$

These properties are very natural if we think of convolution as being like multiplication, and they are very handy to know about because they can help us save work by simplifying convolutions before we actually compute them. For instance, suppose we want to take a sequence a and convolve it with three filters, b_1 , b_2 , and b_3 —that is, we want $((a * b_1) * b_2) * b_3$. If the sequence is long and the filters are short (that is, they have small radii), it is much faster to first convolve the three filters together (computing $b_1 * b_2 * b_3$) and finally to convolve the result with the signal, computing $a * (b_1 * b_2 * b_3)$, which we know from associativity gives the same result.

A very simple filter serves as an *identity* for discrete convolution: it is the discrete filter of radius zero, or the sequence $d[i] = \dots, 0, 0, 1, 0, 0, \dots$ (Figure 9.7). If we convolve d with a signal a , there will be only one nonzero term in the sum:

$$\begin{aligned}(a * d)[i] &= \sum_{j=i}^{j=i} a[j]d[i - j] \\ &= a[i].\end{aligned}$$

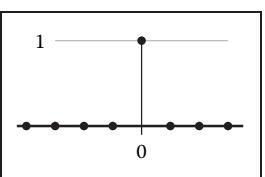


Figure 9.7. The discrete identity filter.

¹You may have noticed that one of the functions in the convolution sum seems to be flipped over—that is, $b[k]$ gives the weight for the sample k units *earlier* in the sequence, while $b[-k]$ gives the weight for the sample k units *later* in the sequence. The reason for this has to do with ensuring associativity; see Exercise 4. Most of the filters we use are symmetric, so you hardly ever need to worry about this.

而不是来自a的起源。也就是说，我们用 $i - k$ 替换 j 。变量的这种变化的结果是

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

这与公式(9.2)完全相同，但a充当滤波器，b充当信号。因此，对于任何序列a和b， $(a \in b) = (b \in a)$ ，我们说卷积是一个交换操作。¹

更一般地说，卷积是一种“类似乘法”的操作。就像数字或函数的乘法或加法一样，参数的顺序和括号的位置都不影响结果。此外，卷积与加法的关系与乘法的关系相同。确切地说，卷积是交换和关联的，并且它是分布的而不是加法。

commutative:	$(a * b)[i] = (b * a)[i]$
associative:	$(a * (b * c))[i] = ((a * b) * c)[i]$
distributive:	$(a * (b + c))[i] = (a * b + a * c)[i]$

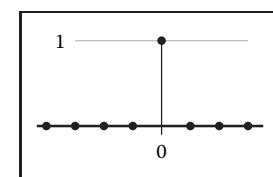


Figure 9.7. 离散的身份过滤器。

如果我们认为卷积就像乘法一样，这些属性是非常自然的，而且它们非常方便，因为它们可以帮助我们在实际计算卷积之前简化卷积，从而节省工作。例如，假设我们想取一个序列a，并用三个过滤器卷积它， b_1 ， b_2 和 b_3 —也就是说，我们想要 $((a * b_1) * b_2) * b_3$ 。如果序列很长，滤波器很短（也就是说，它们的半径很小），那么首先将三个滤波器卷积在一起（计算 $b_1 * b_2 * b_3$ ），最后将结果与信号卷积，计算 $a * (b_1 * b_2 * b_3)$ ，我们

一个非常简单的滤波器作为离散卷积的身份：它是半径为零的离散滤波器，或序列 $d[i] = \dots, 0, 0, 1, 0, 0, \dots$ （图9.7）。如果我们用信号a卷积d，则总和中只有一个非零项：

$$\begin{aligned}(a * d)[i] &= \sum_{j=i}^{j=i} a[j]d[i - j] \\ &= a[i].\end{aligned}$$

¹您可能已经注意到卷积和中的一个函数似乎被翻转了—也就是说， $b[k]$ 给出了序列中较早的样本 k 个单位的权重，而 $b[-k]$ 给出了序列中较晚的样本 k 这样做的原因与确保联想性有关；请参阅练习4。我们使用的大多数滤波器都是对称的，所以你几乎不需要担心这一点。



So clearly, convolving a with d just gives back a again. The sequence d is known as the *discrete impulse*. It is occasionally useful in expressing a filter: for instance, the process of smoothing a signal a with a filter b and then subtracting that from the original could be expressed as a single convolution with the filter $d - b$:

$$c = a - a * b = a * d - a * b = a * (d - b).$$

9.2.3 Convolution as a Sum of Shifted Filters

There is a second, entirely equivalent, way of interpreting Equation (9.2). Looking at the samples of $a * b$ one at a time leads to the weighted-average interpretation that we have already seen. But if we omit the $[i]$, we can instead think of the sum as adding together entire sequences. One piece of notation is required to make this work: if b is a sequence, then the same sequence shifted to the right by j places is called $b_{\rightarrow j}$ (Figure 9.8):

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

Then, we can write Equation (9.2) as a statement about the whole sequence $(a * b)$ rather than element-by-element:

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

Looking at it this way, the convolution is a sum of shifted copies of b , weighted by the entries of a (Figure 9.9). Because of commutativity, we can pick either a

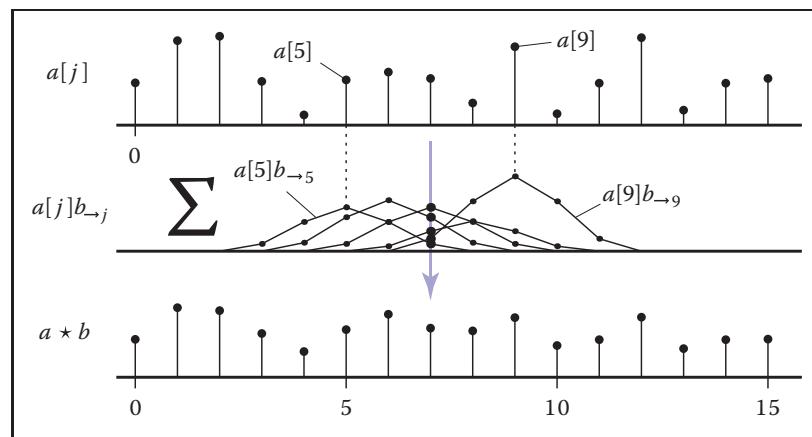


Figure 9.9. Discrete convolution as a sum of shifted copies of the filter.

所以很明显，用 d 卷积 a 只是再次返回 a 。序列 d 被称为离散嵌入。它有时在表达滤波器时很有用：例如，用滤波器 b 平滑信号 a ，然后从原始信号中减去该信号的过程可以表示为与滤波器 $d-b$ 的单个卷积：

$$c = a - a * b = a * d - a * b = a * (d - b).$$

9.2.3 作为移位滤波器之和的卷积

还有第二种完全等价的解释方程 (9.2) 的方法。一次查看一个 b 的样本会导致我们已经看到的加权平均解释。但是，如果我们省略 $[i]$ ，我们可以将总和视为将整个序列加在一起。需要一个符号来使这个工作：如果 b 是一个序列，那么由 j 个地方向右移位的相同序列称为 $b_{\rightarrow j}$ (图9.8)：

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

然后，我们可以将方程 (9.2) 写成关于整个序列 ($a \in b$) 的语句而不是逐个元素：

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

这样看，卷积是 b 的移位副本的总和，由 a 的条目加权 (图9.9)。由于交换性，我们可以选择

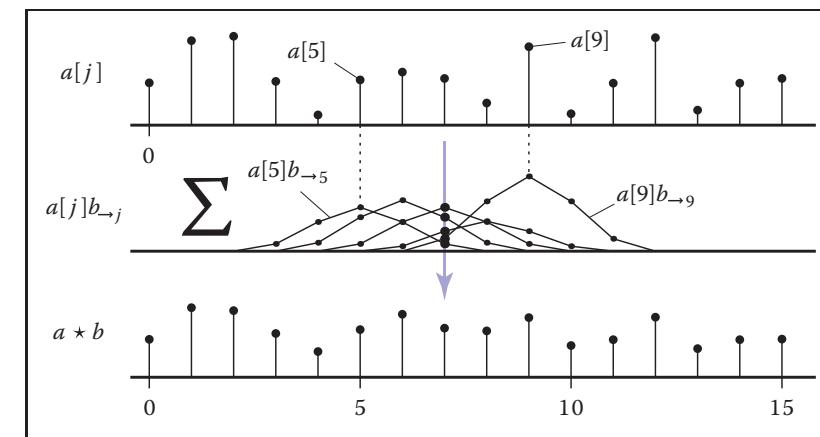
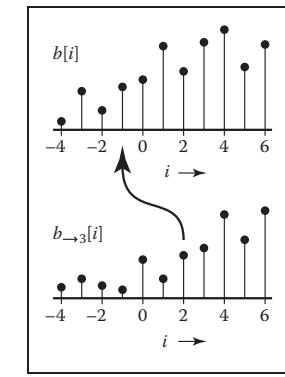


图9.9。离散卷积作为滤波器的移位副本的总和。



转移a sequence b 以获得 $b_{\rightarrow j}$ 。

or b as the filter; if we choose b , then we are adding up one copy of the filter for every sample in the input.

9.2.4 Convolution with Continuous Functions

While it is true that discrete sequences are what we actually work with in a computer program, these sampled sequences are supposed to represent continuous functions, and often we need to reason mathematically about the continuous functions in order to figure out what to do. For this reason, it is useful to define convolution between continuous functions and also between continuous and discrete functions.

The convolution of two continuous functions is the obvious generalization of Equation (9.2), with an integral replacing the sum:

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(t)g(x-t) dt. \quad (9.3)$$

One way of interpreting this definition is that the convolution of f and g , evaluated at the argument x , is the area under the curve of the product of the two functions

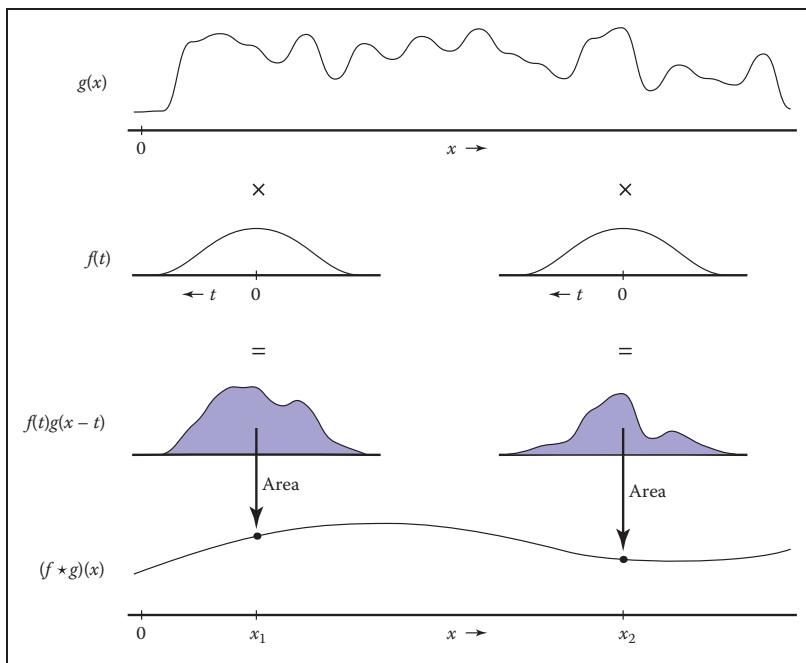


Figure 9.10. Continuous convolution.

或者b作为过滤器；如果我们选择b，那么我们将为输入中的每个样本添加一个过滤器副本。

9.2.4 具有连续函数的卷积

虽然离散序列确实是在computer程序中实际使用的，但这些采样序列应该表示连续函数，我们经常需要对连续函数进行数学推理以确定该怎么做。因此，在连续函数之间以及在连续函数和离散函数之间定义convolution是有用的。

两个连续函数的卷积是明显的泛化方程 (9.2)，用积分代替总和：

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(t)g(x-t) dt. \quad (9.3)$$

解释这个定义的一种方法是，在自变量x处评估的f和g的卷积是两个函数乘积的曲线下面积

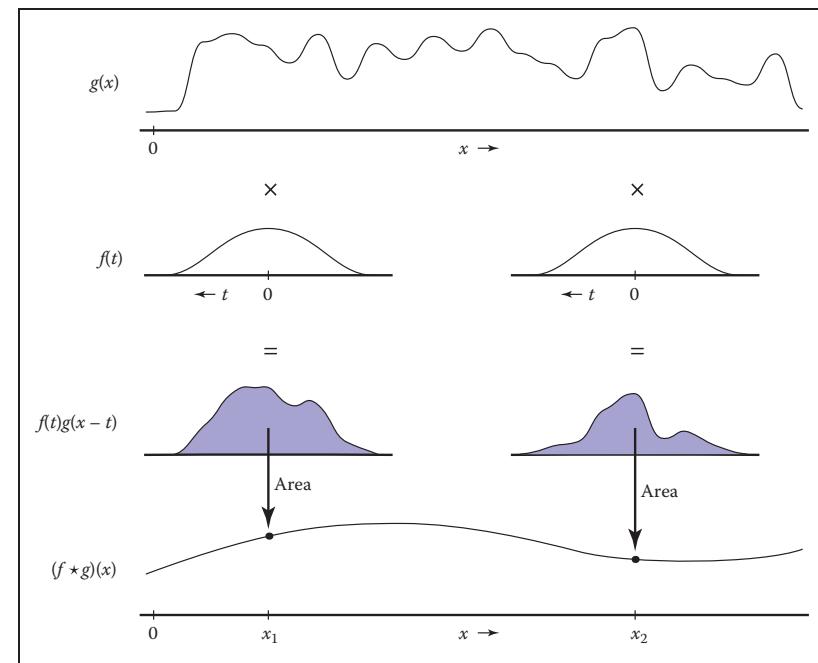


图9.10。连续卷积。



after we shift g so that $g(0)$ lines up with $f(t)$. Just like in the discrete case, the convolution is a moving average, with the filter providing the weights for the average (see Figure 9.10).

Like discrete convolution, convolution of continuous functions is commutative and associative, and it is distributive over addition. Also as with the discrete case, the continuous convolution can be seen as a sum of copies of the filter rather than the computation of weighted averages. Except, in this case, there are infinitely many copies of the filter g :

$$(f \star g) = \int_{-\infty}^{+\infty} f(t)g_{\rightarrow t} dt.$$

Example (Convolution of two box functions). Let f be a box function:

$$f(x) = \begin{cases} 1 & -\frac{1}{2} \leq x < \frac{1}{2}, \\ 0 & \text{otherwise.} \end{cases}$$

Then what is $f \star f$? The definition (Equation 9.3) gives

$$(f \star f)(x) = \int_{-\infty}^{\infty} f(t)f(x-t) dt.$$

Figure 9.11 shows the two cases of this integral. The two boxes might have zero overlap, which happens when $x \leq -1$ or $x \geq 1$; in this case the result is zero. When $-1 < x < 1$, the overlap depends on the separation between the two boxes,

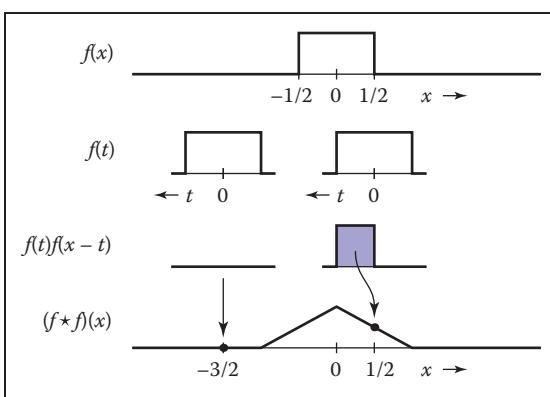


Figure 9.11. Convolving two boxes yields a tent function.



在我们移位g之后，使 $g(0)$ 与 $f(t)$ 对齐。就像在离散情况下一样，卷积是一个移动平均线，滤波器提供平均值的权重（见图9.10）。

与离散卷积一样，连续函数的卷积是commutative和associative，并且它是分布的而不是加法。与离散情况一样，连续卷积可以被看作是滤波器副本的总和，而不是加权平均的计算。除了，在这种情况下，有在finitely许多过滤器的副本g:

$$(f \star g) = \int_{-\infty}^{+\infty} f(t)g_{\rightarrow t} dt.$$

例（两个框函数的卷积）。设 f 为框函数：

$$f(x) = \begin{cases} 1 & -\frac{1}{2} \leq x < \frac{1}{2}, \\ 0 & \text{otherwise.} \end{cases}$$

那么什么是 $f \star f$? 定义（公式9.3）给出

$$(f \star f)(x) = \int_{-\infty}^{\infty} f(t)f(x-t) dt.$$

图9.11显示了这种积分的两种情况。这两个框可能有零重叠，当 $x \leq -1$ 或 $x \geq 1$ 时发生；在这种情况下，结果为零。当 $-1 < x < 1$ 时，重叠取决于两个框之间的分离

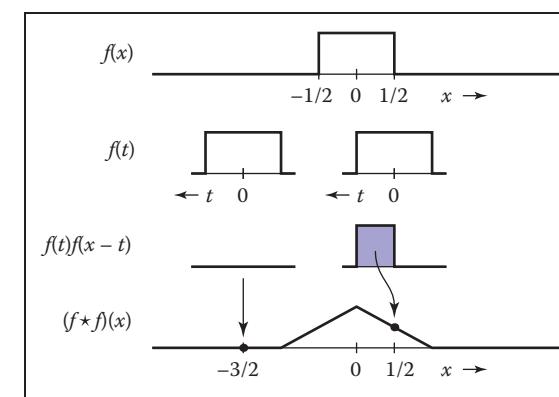


图9.11。卷积两个盒子产生一个帐篷功能。

which is $|x|$; the result is $1 - |x|$. So

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

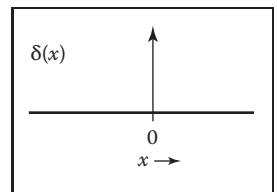


Figure 9.12. The Dirac delta function $\delta(x)$.

This function, known as the *tent function*, is another common filter (see Section 9.3.1).

The Dirac Delta Function

In discrete convolution, we saw that the discrete impulse d acted as an identity: $d * a = a$. In the continuous case, there is also an identity function, called the *Dirac impulse* or *Dirac delta* function, denoted $\delta(x)$.

Intuitively, the delta function is a very narrow, very tall spike that has infinitesimal width but still has area equal to 1 (Figure 9.12). The key defining property of the delta function is that multiplying it by a function selects out the value exactly at zero:

$$\int_{-\infty}^{\infty} \delta(x)f(x)dx = f(0).$$

The delta function does not have a well-defined value at 0 (you can think of its value loosely as $+\infty$), but it does have the value $\delta(x) = 0$ for all $x \neq 0$.

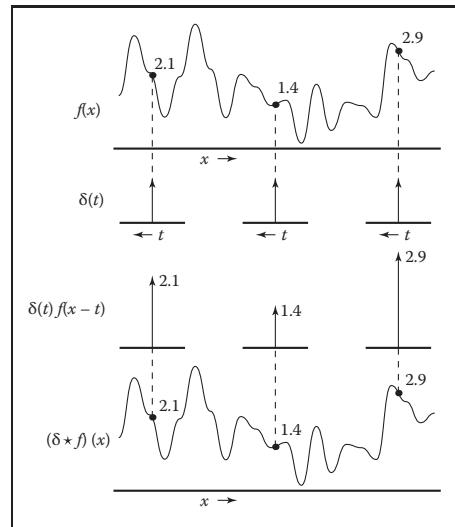
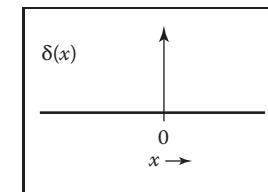


Figure 9.13. Convolving a function with $\delta(x)$ returns a copy of the same function.

这是 x ; 结果是 $1 - x$ 。所以

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



狄拉克δ函数 $\delta(x)$ 。

这个函数称为帐篷函数 是另一个常见的过滤器(见第9.3.1节).

狄拉克德尔塔函数

在离散卷积中，我们看到离散脉冲d充当身份： $d \in a=a$ 。在连续的情况下，还有一个恒等函数，称为狄拉克脉冲或狄拉克三角洲函数，表示 $\delta(x)$ 。

直观地说，delta函数是一个非常窄，非常高的尖峰，具有infinitesimal宽度，但仍然具有等于1的面积（图9.12）。的关键定义属性delta函数是将其乘以一个函数选择出恰好在零处的值：

$$\int_{-\infty}^{\infty} \delta(x)f(x)dx = f(0).$$

Delta函数在0处没有明确定义的值（您可以将其值松散地视为 $+\infty$ ），但它确实具有所有 $x=0$ 的值 $\delta(x)=0$ 。

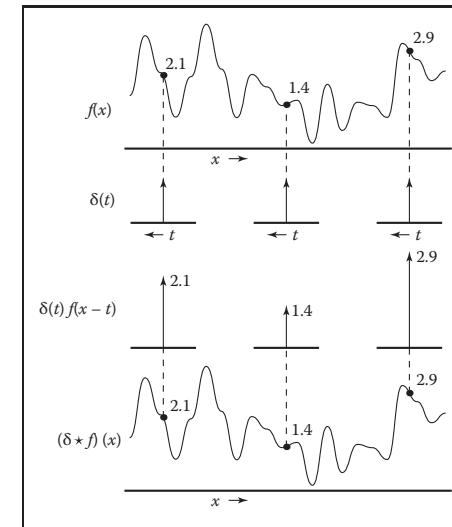


图9.13。用 $\delta(x)$ 对函数进行卷积会返回相同函数的副本。

From this property of selecting out single values, it follows that the delta function is the identity for continuous convolution (Figure 9.13), because convolving δ with any function f yields

$$(\delta * f)(x) = \int_{-\infty}^{\infty} \delta(t)f(x-t)dt = f(x).$$

So $\delta * f = f$ (and because of commutativity $f * \delta = f$ also).

9.2.5 Discrete-Continuous Convolution

There are two ways to connect the discrete and continuous worlds. One is sampling: we convert a continuous function into a discrete one by writing down the function's value at all integer arguments and forgetting about the rest. Given a continuous function $f(x)$, we can sample it to convert to a discrete sequence $a[i]$:

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

Going the other way, from a discrete function, or sequence, to a continuous function, is called *reconstruction*. This is accomplished using yet another form of convolution, the discrete-continuous form. In this case, we are filtering a discrete sequence $a[i]$ with a continuous filter $f(x)$:

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

The value of the reconstructed function $a * f$ at x is a weighted sum of the samples $a[i]$ for values of i near x (Figure 9.14). The weights come from the filter f , which

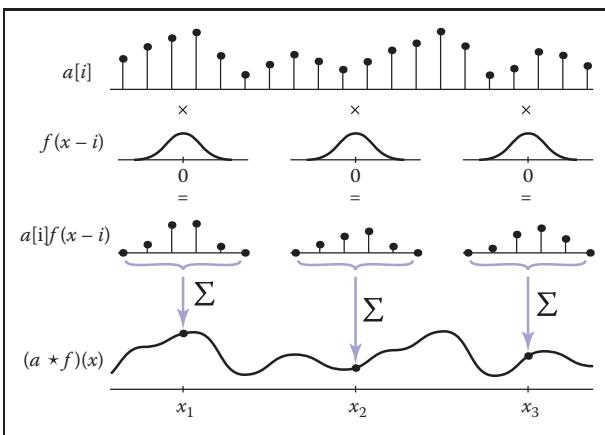


Figure 9.14. Discrete-continuous convolution.

从这个选择单个值的属性中，可以看出delta函数是连续卷积的同一性（图9.13），因为用任何函数f卷积δ会产生

$$(\delta * f)(x) = \int_{-\infty}^{\infty} \delta(t)f(x-t)dt = f(x).$$

所以 $\delta \in f = f$ （并且由于交换性 $f \in \delta = f$ 也）。

9.2.5 Discrete-Continuous Convolution

有两种方法可以连接离散和连续的世界。一个是采样：我们通过在所有整数参数处写下函数的值并忘记其余部分来将连续函数转换为离散函数。给定一个连续函数 $f(x)$ ，我们可以对其进行采样以转换为离散序列 $a[i]$ ：

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

相反，从离散函数或序列到连续函数，称为重构。这是使用另一种卷积形式，即离散-连续形式来实现的。在这种情况下，我们正在用连续滤波器 $f(x)$ 过滤离散序列 $a[i]$ ：

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

X处重建函数 $a \in f$ 的值是x附近i值的样本 $a[i]$ 的加权和（图9.14）。权重来自滤波器 f ，其

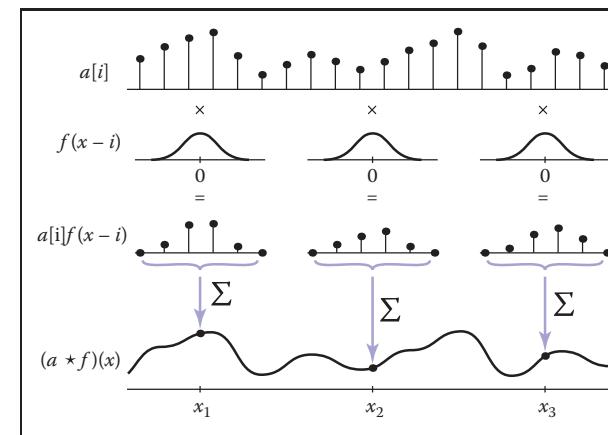


图9.14。离散-连续卷积。

is evaluated at a set of points spaced one unit apart. For example, if $x = 5.3$ and f has radius 2, f is evaluated at 1.3, 0.3, -0.7, and -1.7. Note that for discrete-continuous convolution we generally write the sequence first and the filter second, so that the sum is over integers.

As with discrete convolution, we can put bounds on the sum if we know the filter's radius, r , eliminating all points where the difference between x and i is at least r :

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

Note, that if a point falls exactly at distance r from x (i.e., if $x - r$ turns out to be an integer), it will be left out of the sum. This is in contrast to the discrete case, where we included the point at $i - r$.

Expressed in code, this is:

```
function reconstruct(sequence a, filter f, real x)
    s = 0
    r = f.radius
    for i =  $\lceil x - r \rceil$  to  $\lfloor x + r \rfloor$  do
        s = s + a[i]f(x - i)
    return s
```

As with the other forms of convolution, discrete-continuous convolution may be seen as summing shifted copies of the filter (Figure 9.15):

$$(a * f) = \sum_i a[i]f_{\rightarrow i}.$$

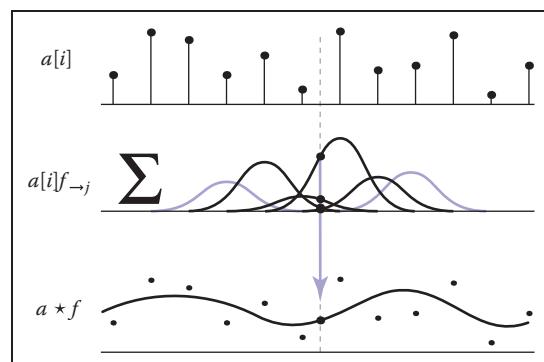


Figure 9.15. Reconstruction (discrete-continuous convolution) as a sum of shifted copies of the filter.

在间隔一个单位的一组点处被评估。例如，如果 $x=5.3$ 和 f 具有半径 2, f 在 1.3、0.3、-0.7, 和 -1.7. 请注意，对于离散连续卷积，我们通常首先编写序列，然后编写过滤器，以便总和超过整数。

与离散卷积一样，如果我们知道滤波器的半径 r ，我们可以在总和上设置边界，从而消除 x 和 i 之间的差值至少为 r 的所有点：

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

请注意，如果一个点正好落在距离 x 的距离 r 处（即，如果 $x - r$ 结果是一个整数），它将被排除在总和之外。这与离散情况相反，我们在 $i = r$ 处包含了点。用代码表示，这是：

```
函数重构 (序列a, 滤波器f, 实数x) s=0;r=
f.半径为i= x rto到 x+r 做s=s+a[i]f(x - i)返回s
```

与其他形式的卷积一样，离散-连续卷积可以被看作是对滤波器的移位副本进行求和（图9.15）：

$$(a * f) = \sum_i a[i]f_{\rightarrow i}.$$

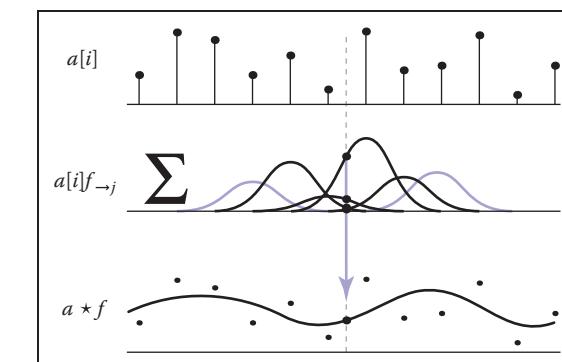


Figure 9.15. 重建（离散-连续卷积）作为 filter.

Discrete-continuous convolution is closely related to splines. For uniform splines (a uniform B-spline, for instance), the parameterized curve for the spline is exactly the convolution of the spline's basis function with the control point sequence (see Section 15.6.2).

9.2.6 Convolution in More Than One Dimension

So far, everything we have said about sampling and reconstruction has been one-dimensional: there has been a single variable x or a single sequence index i . Many of the important applications of sampling and reconstruction in graphics, though, are applied to two-dimensional functions—in particular, to 2D images. Fortunately, the generalization of sampling algorithms and theory from 1D to 2D, 3D, and beyond is conceptually very simple.

Beginning with the definition of discrete convolution, we can generalize it to two dimensions by making the sum into a double sum:

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

If b is a finitely supported filter of radius r (that is, it has $(2r + 1)^2$ values), then we can write this sum with bounds (Figure 9.16):

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

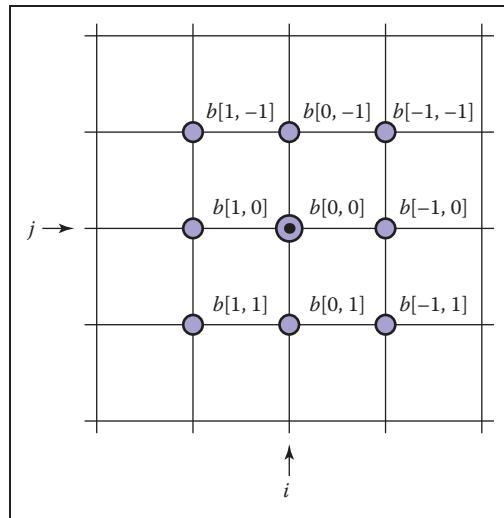


Figure 9.16. The weights for the nine input samples that contribute to the discrete convolution at point (i, j) with a filter b of radius 1.

离散-连续卷积与样条密切相关。对于均匀样条（例如均匀的B样条），样条的参数化曲线正是样条基函数与控制点序列的卷积（见第15.6.2节）。

9.2.6 多个维度的卷积

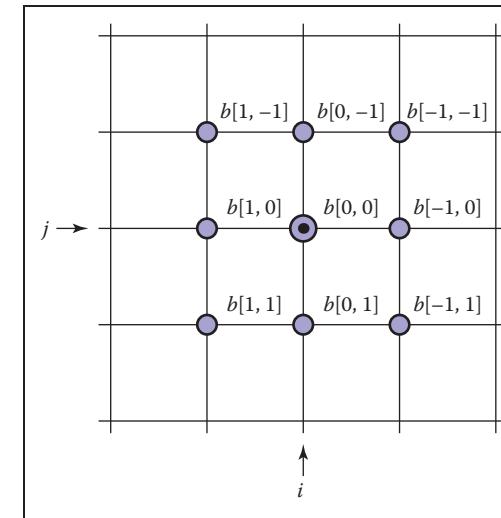
到目前为止，我们所说的关于采样和重建的一切都是一维的：出现了单个变量 x 或单个序列索引*i*。然而，采样和重建在图形中的许多重要应用都应用于二维函数，特别是二维图像。幸运的是，从1D到2D, 3D以及更远的采样算法和理论的概括在概念上非常简单。

从离散卷积的定义开始，我们可以将其概括为两个维度，方法是将总和做成双和：

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

如果 b 是半径 r 的有限支持滤波器（即它具有 $(2r+1)^2$ 个值），那么我们可以用边界写出这个总和（图9.16）：

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



在具有半径1的滤波器 b 的点 (i, j) 处有助于离散卷积的九个输入样本的权重。

and express it in code:

```
function convolve2d(sequence2d a, filter2d b, int i, int j)
    s = 0
    r = b.radius
    for i' = i - r to i + r do
        for j' = j - r to j + r do
            s = s + a[i'][j']b[i - i'][j - j']
    return s
```

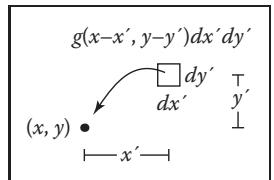


Figure 9.17. The weight for an infinitesimal area in the input signal resulting from continuous convolution at (x, y) .

This definition can be interpreted in the same way as in the 1D case: each output sample is a weighted average of an area in the input, using the 2D filter as a "mask" to determine the weight of each sample in the average.

Continuing the generalization, we can write continuous-continuous (Figure 9.17) and discrete-continuous (Figure 9.18) convolutions in 2D as well:

$$(f * g)(x, y) = \int \int f(x', y')g(x - x', y - y') dx' dy';$$

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

In each case, the result at a particular point is a weighted average of the input near that point. For the continuous-continuous case, it is a weighted integral over a region centered at that point, and in the discrete-continuous case it is a weighted average of all the samples that fall near the point.

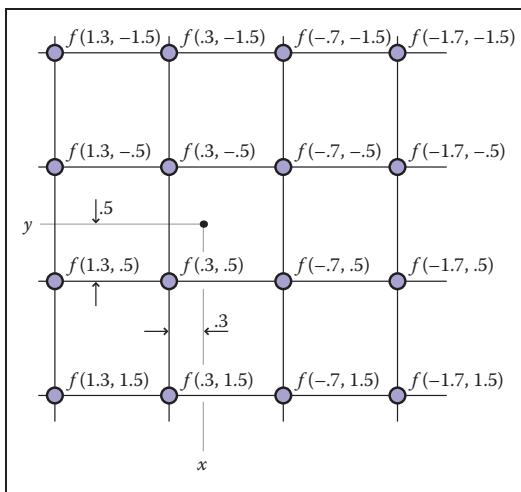


Figure 9.18. The weights for the 16 input samples that contribute to the discrete-continuous convolution at point (x, y) for a reconstruction filter of radius 2.

并用代码表达:

```
函数convolve2d(sequence2da filter2db inti intj)
=0r=b。半径为i'=i-r到i+rdo
```

对于j'=j-r到j+rdo

$[j - j']$

return s

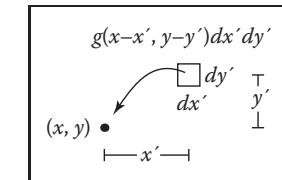


图9.17。输入信号中由 (x, y) 处的连续卷积产生的无穷小区域的权重。

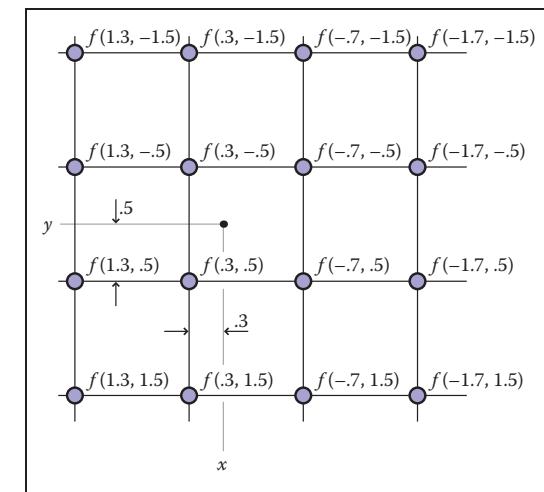
该定义可以以与1D情况中相同的方式解释：每个输出样本是输入中一个区域的加权平均值，使用2D滤波器作为“掩码”来确定平均值中每个样本的权重。

继续概括，我们也可以在2D中编写连续-连续（图9.17）和离散-连续（图9.18）卷积：

$$(f * g)(x, y) = \int \int f(x', y')g(x - x', y - y') dx' dy';$$

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

在每种情况下，在特定点处的结果是该点附近的输入的加权平均值。对于连续-连续的情况，它是在以该点为中心的区域上的加权积分，而在离散-连续的情况下，它是落在该点附近的所有样本的加权平均值。



半径为2的重建滤波器的16个输入样本在点 (x, y) 处对离散连续卷积有贡献的权重。



Once we have gone from 1D to 2D, it should be fairly clear how to generalize further to 3D or even to higher dimensions.

9.3 Convolution Filters

Now that we have the machinery of convolution, let's examine some of the particular filters commonly used in graphics.

Each of the following filters has a natural radius, which is the default size to be used for sampling or reconstruction when samples are spaced one unit apart. In this section filters are defined at this natural size: for instance, the box filter has a natural radius of $\frac{1}{2}$, and the cubic filters have a natural radius of 2. We also arrange for each filter to integrate to 1: $\int_{x=0}^{\infty} f(x)dx = 1$, as required for sampling and reconstruction without changing a signal's average value.

As we will see in Section 9.4.3, some applications require filters of different sizes, which can be obtained by scaling the basic filter. For a filter $f(x)$, we can define a version of scale s :

$$f_s(x) = \frac{f(x/s)}{s}.$$

The filter is stretched horizontally by a factor of s , and then squashed vertically by a factor $\frac{1}{s}$ so that its area is unchanged. A filter that has a natural radius of r and is used at scale s has a radius of support sr (see Figure 9.20 below).

9.3.1 A Gallery of Convolution Filters

The Box Filter

The box filter (Figure 9.19) is a piecewise constant function whose integral is equal to one. As a discrete filter, it can be written as

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

Note that for symmetry we include both endpoints.

As a continuous filter, we write

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

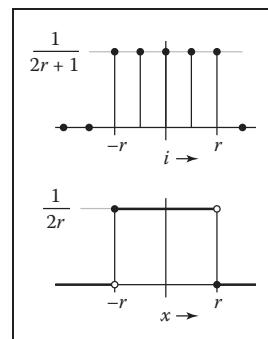


Figure 9.19. The discrete and continuous box filters.

一旦我们从1D到2D，应该很清楚如何进一步推广到3D甚至更高维度。

9.3 卷积滤波器

现在我们已经有了卷积的机制，让我们来看看一些常用于图形的particular过滤器。

以下每个过滤器都有一个自然半径，这是当样本间隔一个单位时用于采样或重建的默认大小。在本节中，筛选器按此自然大小定义：例如，框筛选器的自然半径为1

在不改变信号平均值的情况下进行采样和重建所需要的增益 $\Delta x = 0 f(x)dx = 1$ 。

正如我们将在第9.4.3节中看到的，一些应用程序需要不同大小的过滤器，这可以通过缩放基本过滤器来获得。对于一个过滤器 $f(x)$ ，我们可以定义一个scales的版本：

$$f_s(x) = \frac{f(x/s)}{s}.$$

滤波器水平拉伸因子s，然后垂直压扁因子1/s使其面积不变。一个自然半径为r并在尺度s上使用的滤波器的支撑半径为sr（见下图9.20）。

9.3.1 卷积滤波器图库

箱式过滤器

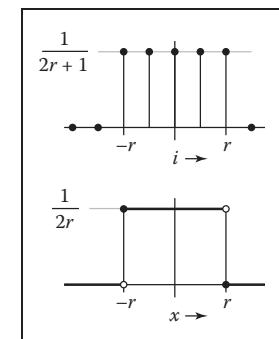
盒式滤波器（图9.19）是一个分段常数函数，其积分等于1。作为离散滤波器，可以写成

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

请注意，对于对称性，我们包括两个端点。

作为一个连续的过滤器，我们写

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



离散和连续箱过滤器。

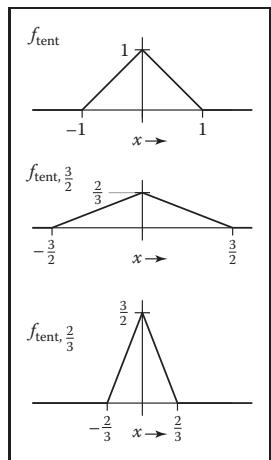


Figure 9.20. The tent filter and two scaled versions.

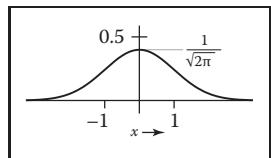


Figure 9.21. The Gaussian filter.

In this case, we exclude one endpoint, which makes the box of radius 0.5 usable as a reconstruction filter. It is because the box filter is discontinuous that these boundary cases are important, and so for this particular filter we need to pay attention to them. We write just f_{box} for the natural radius of $r = \frac{1}{2}$.

The Tent Filter

The tent, or linear filter (Figure 9.20), is a continuous, piecewise linear function:

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

Its natural radius is 1. For filters, such as this one, that are at least C^0 (that is, there are no sudden jumps in the value, as there are with the box), we no longer need to separate the definitions of the discrete and continuous filters: the discrete filter is just the continuous filter sampled at the integers.

The Gaussian Filter

The Gaussian function (Figure 9.21), also known as the normal distribution, is an important filter theoretically and practically. We'll see more of its special properties as the chapter goes on:

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

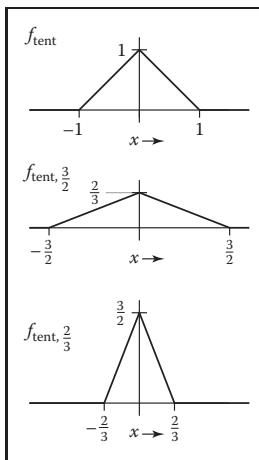
The parameter σ is called the standard deviation. The Gaussian makes a good sampling filter because it is very smooth; we'll make this statement more precise later in the chapter.

The Gaussian filter does not have any particular natural radius; it is a useful sampling filter for a range of σ . The Gaussian also does not have a finite radius of support, although because of the exponential decay, its values rapidly become small enough to ignore. When necessary, then, we can trim the tails from the function by setting it to zero outside some radius r , resulting in a *trimmed Gaussian*. This means that the filter's width and natural radius can vary depending on the application, and a trimmed Gaussian scaled by s is the same as an unscaled trimmed Gaussian with standard deviation $s\sigma$ and radius sr . The best way to handle this in practice is to let σ and r be set as properties of the filter, fixed when the filter is specified, and then scale the filter just like any other when it is applied.

Good starting points are $\sigma = 1$ and $r = 3$.

The B-Spline Cubic Filter

Many filters are defined as piecewise polynomials, and cubic filters with four pieces (natural radius of 2) are often used as reconstruction filters. One such filter



帐篷过滤器和两个缩放版本。

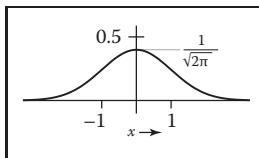


图9.21. 该高斯滤波器。

在这种情况下，我们排除一个端点，这使得半径为0的框。5可用作重建滤波器。正是因为盒式滤波器是不连续的，所以这些边界情况很重要，因此对于这个特定的滤波器，我们需要注意它们。我们只为r=1的自然半径写f框

帐篷过滤器

帐篷，或线性滤波器（图9.20），是一个连续的，分段线性函数：

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

其自然半径为1。对于至少为C0的滤波器，例如这个滤波器（即值中没有突然跳跃，就像盒子一样），我们不再需要分离离散滤波器和连续滤波器的定义：离散滤波器只是在整数处采样的连续滤波器。

高斯滤波器

高斯函数（图9.21），也称为正态分布，在理论上和实践上都是一个重要的滤波器。随着本章的继续，我们将看到更多它的特殊属性：

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

参数 σ 称为标准偏差。高斯是一个很好的采样滤波器，因为它非常平滑；我们将在本章的后面使这个语句更精确。

高斯滤波器没有任何特定的自然半径；对于 σ 范围来说，它是一个有用的采样滤波器。高斯也没有有限的支持半径，尽管由于指数衰减，它的值迅速变得小到足以忽略。必要时，我们可以通过在某些半径 r 之外将其设置为零来修剪函数中的尾部，从而产生修剪的Gaussian。这意味着滤镜的宽度和自然半径可能因以下情况而异

好的起点是 $\sigma=1$ 和 $r=3$ 。

应用程序，并且由s缩放的修剪高斯与具有标准偏差 $s\sigma$ 和半径 sr 的未缩放的修剪高斯相同。在实践中处理此问题的最佳方法是将 σ 和 r 设置为过滤器的属性，在指定过滤器时固定，然后在应用过滤器时缩放过滤器，就像任何其他过滤器一样。

B样条三次滤波器

许多滤波器被定义为分段多项式，并且具有四个片（自然半径为2）的三次滤波器通常用作重建滤波器。一个这样的过滤器



is known as the B-spline filter (Figure 9.22) because of its origins as a blending function for spline curves (see Chapter 15):

$$f_B(x) = \frac{1}{6} \begin{cases} -3(1-|x|)^3 + 3(1-|x|)^2 + 3(1-|x|) + 1 & -1 \leq x \leq 1, \\ (2-|x|)^3 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

Among piecewise cubics, the B-spline is special because it has continuous first and second derivatives—that is, it is C^2 . A more concise way of defining this filter is $f_B = f_{\text{box}} * f_{\text{box}} * f_{\text{box}} * f_{\text{box}}$; proving that the longer form above is equivalent is a nice exercise in convolution (see Exercise 3).

The Catmull-Rom Cubic Filter

Another piecewise cubic filter named for a spline, the Catmull-Rom filter (Figure 9.23), has the value zero at $x = -2, -1, 1$, and 2 , which means it will *interpolate* the samples when used as a reconstruction filter (Section 9.3.2):

$$f_C(x) = \frac{1}{2} \begin{cases} -3(1-|x|)^3 + 4(1-|x|)^2 + (1-|x|) & -1 \leq x \leq 1, \\ (2-|x|)^3 - (2-|x|)^2 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

The Mitchell-Netravali Cubic Filter

For the all-important application of resampling images, Mitchell and Netravali (Mitchell & Netravali, 1988) made a study of cubic filters and recommended one partway between the previous two filters as the best all-around choice (Figure 9.24). It is simply a weighted combination of the previous two filters:

$$\begin{aligned} f_M(x) &= \frac{1}{3} f_B(x) + \frac{2}{3} f_C(x) \\ &= \frac{1}{18} \begin{cases} -21(1-|x|)^3 + 27(1-|x|)^2 + 9(1-|x|) + 1 & -1 \leq x \leq 1, \\ 7(2-|x|)^3 - 6(2-|x|)^2 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

9.3.2 Properties of Filters

Filters have some traditional terminology that goes with them, which we use to describe the filters and compare them to one another.

The *impulse response* of a filter is just another name for the function: it is the response of the filter to a signal that just contains an impulse (and recall that convolving with an impulse just gives back the filter).

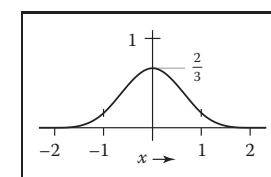


Figure 9.22. The B-spline filter.

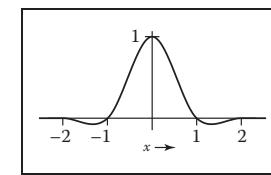


Figure 9.23. The Catmull-Rom filter.

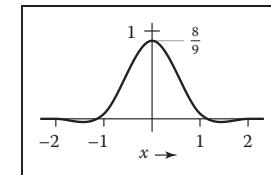


Figure 9.24. The Mitchell-Netravali filter.



被称为B样条滤波器（图9.22），因为它起源于样条曲线的混合函数（见第15章）：

$$f_B(x) = \frac{1}{6} \begin{cases} -3(1-|x|)^3 + 3(1-|x|)^2 + 3(1-|x|) + 1 & -1 \leq x \leq 1, \\ (2-|x|)^3 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

在分段立方体中，B样条是特殊的，因为它具有连续的一阶和二阶导数—即它是C2。定义这个过滤器的更简洁的方法是 $f_B=f_{\text{box}} * f_{\text{box}} * f_{\text{box}}$ ；证明上面的较长形式是等价的是卷积中的一个很好的练习（见练习3）。

Catmull-Rom立方滤波器

另一个以样条曲线命名的分段三次滤波器Catmull-Rom滤波器（图9.23）在 $x=-2, -1, 1$ 和 2 处的值为零，这意味着它将在用作重建滤波器时对样本进行插值（第9.3.2节）：

$$f_C(x) = \frac{1}{2} \begin{cases} -3(1-|x|)^3 + 4(1-|x|)^2 + (1-|x|) & -1 \leq x \leq 1, \\ (2-|x|)^3 - (2-|x|)^2 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

米切尔-Netravali立方过滤器

对于重采样图像的所有重要应用，Mitchell和Netravali (Mitchell & Netravali, 1988) 对立方滤波器进行了研究，并推荐了前两个滤波器之间的一个partway作为最佳全能选择（图9.24）。它只是前面两个滤波器的加权组合：

$$\begin{aligned} f_M(x) &= \frac{1}{3} f_B(x) + \frac{2}{3} f_C(x) \\ &= \frac{1}{18} \begin{cases} -21(1-|x|)^3 + 27(1-|x|)^2 + 9(1-|x|) + 1 & -1 \leq x \leq 1, \\ 7(2-|x|)^3 - 6(2-|x|)^2 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

9.3.2 过滤器的属性

过滤器有一些与它们相关的传统术语，我们用它们来描述过滤器并将它们相互比较。

滤波器的脉冲响应只是该函数的另一个名称：它是滤波器对仅包含一个impulse的信号的响应（并且回想一下，用脉冲进行卷积只是返回滤波器）。

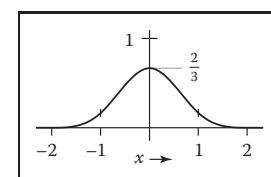


Figure 9.22. The B-spline filter.

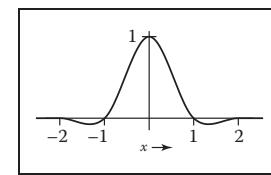


Figure 9.23. The Catmull-Rom filter.

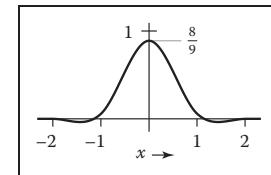


Figure 9.24. 米切尔-Netravali filter.

A continuous filter is *interpolating* if, when it is used to reconstruct a continuous function from a discrete sequence, the resulting function takes on exactly the values of the samples at the sample points—that is, it “connects the dots” rather than producing a function that only goes near the dots. Interpolating filters are exactly those filters f for which $f(0) = 1$ and $f(i) = 0$ for all nonzero integers i (Figure 9.25).

A filter that takes on negative values has *ringing* or *overshoot*: it will produce extra oscillations in the value around sharp changes in the value of the function being filtered.

For instance, the Catmull-Rom filter has negative lobes on either side, and if you filter a step function with it, it will exaggerate the step a bit, resulting in function values that undershoot 0 and overshoot 1 (Figure 9.26).

A continuous filter is *ripple free* if, when used as a reconstruction filter, it will reconstruct a constant sequence as a constant function (Figure 9.27). This is equivalent to the requirement that the filter sum to one on any integer-spaced grid:

$$\sum_i f(x+i) = 1 \quad \text{for all } x.$$

All the filters in Section 9.3.1 are ripple free at their natural radii, except the

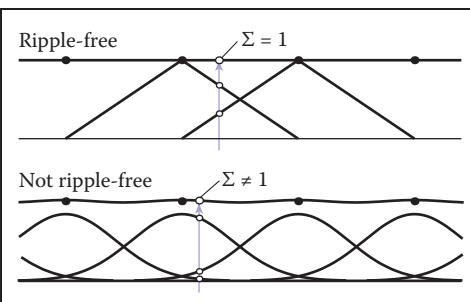


Figure 9.25. An interpolating filter reconstructs the sample points exactly because it has the value zero at all nonzero integer offsets from the center.

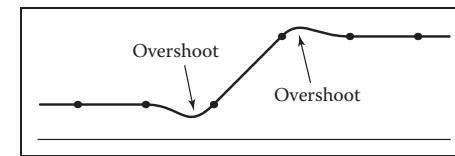


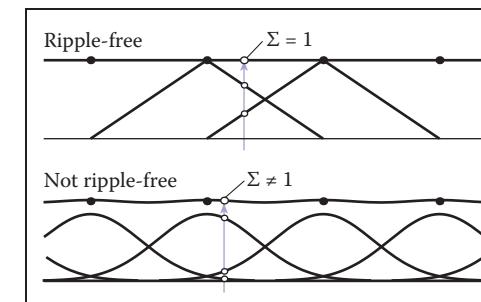
Figure 9.26. A filter with negative lobes will always produce some overshoot when filtering or reconstructing a sharp discontinuity.

插值滤波器重建样本点，正是因为它在距中心的所有非零整数偏移处的值为零。如果使用连续滤波器从离散序列重建连续函数，则生成的函数恰好在样本点处接受样本的值—也就是说，它“连接点”，而不是产生只靠近点的函数，则连续滤插值滤波器正是所有非零整数*i*的 $f(0) = 1$ 和 $f(i) = 0$ 的滤波器 f （图9.25）。

具有负波瓣的滤波器在滤波或重建尖锐不连续性时总是会产生一些过冲。取负值的滤波器有振铃或过冲：它会在被滤波函数值急剧变化的周围产生额外的振荡。例如，Catmull-Rom滤波器的任一侧都有负波瓣，如果用它过滤阶跃函数，它会使阶跃夸大一点，从而产生函数值下冲0和过冲1（图9.26）。连续滤波器是无纹波的，如果当用作重建滤波器时，它将重建一个恒定序列作为一个恒定函数（图9.27）。这相当于要求滤波器在任何整数间隔的网格上总和为1：

$$\sum_i f(x+i) = 1 \quad \text{对于所有 } x.$$

第9.3.1节中的所有滤波器在其自然半径处均无纹波，但



半径1的tent滤波器是无波纹重建滤波器；标准差1的高斯滤波器2不是。



Gaussian, but none of them are necessarily ripple free when they are used at a noninteger scale. If it is necessary to eliminate ripple in discrete-continuous convolution, it is easy to do so: divide each computed sample by the sum of the weights used to compute it:

$$(\overline{a \star f})(x) = \frac{\sum_i a[i]f(x-i)}{\sum_i a[i]}. \quad (9.4)$$

This expression can still be interpreted as convolution between a and a filter \bar{f} (see Exercise 6).

A continuous filter has a *degree of continuity*, which is the highest-order derivative that is defined everywhere. A filter, like the box filter, that has sudden jumps in its value is not continuous at all. A filter that is continuous but has sharp corners (discontinuities in the first derivative), such as the tent filter, has order of continuity zero, and we say it is C^0 . A filter that has a continuous derivative (no sharp corners), such as the piecewise cubic filters in the previous section, is C^1 ; if its second derivative is also continuous, as is true of the B-spline filter, it is C^2 . The order of continuity of a filter is particularly important for a reconstruction filter because the reconstructed function inherits the continuity of the filter.

Separable Filters

So far we have only discussed filters for 1D convolution, but for images and other multidimensional signals we need filters too. In general, any 2D function could be a 2D filter, and occasionally it is useful to define them this way. But, in most cases, we can build suitable 2D (or higher-dimensional) filters from the 1D filters we have already seen.

The most useful way of doing this is by using a *separable* filter. The value of a separable filter $f_2(x, y)$ at a particular x and y is simply the product of f_1 (the 1D filter) evaluated at x and at y :

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

Similarly, for discrete filters,

$$b_2[i, j] = b_1[i]b_1[j].$$

Any horizontal or vertical slice through f_2 is a scaled copy of f_1 . The integral of f_2 is the square of the integral of f_1 , so in particular if f_1 is normalized, then so is f_2 .



高斯，但当它们以非整数尺度使用时，它们都不一定是无波纹的。如果需要消除离散-连续卷积中的波纹，那么很容易做到：将每个计算样本除以用于计算它的权重之和：

$$(\overline{a \star f})(x) = \frac{\sum_i a[i]f(x-i)}{\sum_i a[i]}. \quad (9.4)$$

这个表达式仍然可以解释为 a 和滤波器 f 之间的卷积（见练习6）。

连续滤波器具有一定程度的连续性，这是到处定义的最高阶导数。一个过滤器，就像盒子过滤器一样，在它的值中有sudden跳跃的过滤器根本不是连续的。连续但具有尖角（一阶导数中的不连续性）的滤波器，例如帐篷滤波器，具有连续性零阶数，我们说它是 C^0 。具有连续导数（无尖角）的滤波器，如上一节中的分段三次滤波器，为 C^1 ；如果其二阶导数也是连续的，如B样条滤波器一样，则为 C^2 。滤波器的连续性顺序对于重构滤波器特别重要，因为重构函数继承了滤波器的连续性。

Separable Filters

到目前为止，我们只讨论了一维卷积的滤波器，但对于图像和其他多维信号，我们也需要滤波器。一般来说，任何2D函数都可以是2D过滤器，有时以这种方式定义它们是有用的。但是，在大多数情况下，我们可以从我们已经看到的1D过滤器中构建合适的2D（或更高维度）过滤器。

最有用的方法是使用可分离过滤器。可分离滤波器 $f_2(x, y)$ 在特定 x 和 y 处的值只是在 x 和 y 处评估的 f_1 （一维滤波器）的乘积：

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

同样，对于离散滤波器

$$b_2[i, j] = b_1[i]b_1[j].$$

通过 f_2 的任何水平或垂直切片都是 f_1 的缩放副本。 F_2 的积分是 f_1 的积分的平方，因此特别是如果 f_1 被归一化，那么 f_2 也是如此。

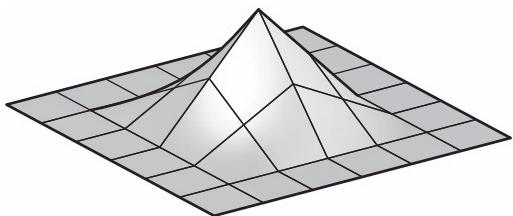


Figure 9.28. The separable 2D tent filter.

Example (The separable tent filter). If we choose the tent function for f_1 , the resulting piecewise bilinear function (Figure 9.28) is

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

The profiles along the coordinate axes are tent functions, but the profiles along the diagonals are quadratics (for instance, along the line $x = y$ in the positive quadrant, we see the quadratic function $(1 - x)^2$). □

Example (The 2D Gaussian filter). If we choose the Gaussian function for f_1 , the resulting 2D function (Figure 9.29) is

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

Notice that this is (up to a scale factor) the same function we would get if we revolved the 1D Gaussian around the origin to produce a circularly symmetric

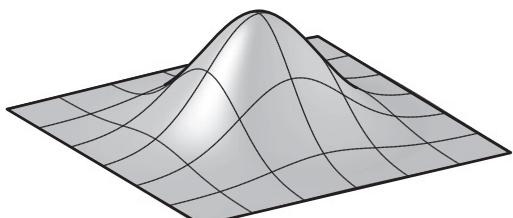


Figure 9.29. The 2D Gaussian filter, which is both separable and radially symmetric.

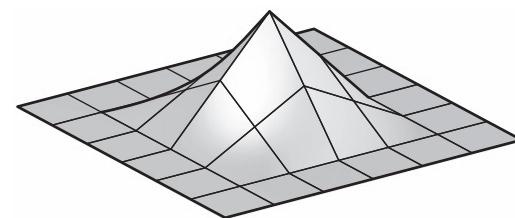


图9.28。可分离的2d帐篷过滤器。

例（可分离帐篷过滤器）。如果我们为f1选择tent函数，则得到的分段双线性函数（图9.28）为

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

沿着坐标轴的轮廓是帐篷函数，但沿着对角线的轮廓是象限（例如，沿着正象限中的线 $x=y$ ，我们看到二次函数 $(1 - x)^2$ ）。

例（2d高斯滤波器）。如果我们为f1选择高斯函数，则得到的2D函数（图9.29）是

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

请注意，这是（最多一个比例因子），如果我们围绕原点旋转1D高斯以产生圆对称，我们将得到相同的函数

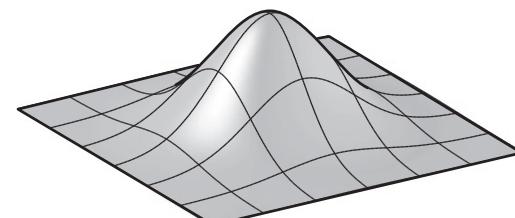


图9.29。的二维高斯滤波器，其既可分离又径向对称。

function. The property of being both circularly symmetric and separable at the same time is unique to the Gaussian function. The profiles along the coordinate axes are Gaussians, but so are the profiles along any direction at any offset from the center.

The key advantage of separable filters over other 2D filters has to do with efficiency in implementation. Let's substitute the definition of a_2 into the definition of discrete convolution:

$$(a * b_2)[i, j] = \sum_{i'} \sum_{j'} a[i', j'] b_1[i - i'] b_1[j - j'].$$

Note that $b_1[i - i']$ does not depend on j' and can be factored out of the inner sum:

$$= \sum_{i'} b_1[i - i'] \sum_{j'} a[i', j'] b_1[j - j'].$$

Let's abbreviate the inner sum as $S[i']$:

$$\begin{aligned} S[i'] &= \sum_{j'} a[i', j'] b_1[j - j']; \\ (a * b_2)[i, j] &= \sum_{i'} b_1[i - i'] S[i']. \end{aligned} \quad (9.5)$$

With the equation in this form, we can first compute and store $S[i']$ for each value of i' , and then compute the outer sum using these stored values. At first glance this does not seem remarkable, since we still had to do work proportional to $(2r + 1)^2$ to compute all the inner sums. However, it's quite different if we want to compute the value at many points $[i, j]$.

Suppose we need to compute $a * b_2$ at $[2, 2]$ and $[3, 2]$, and b_1 has a radius of 2. Examining Equation 9.5, we can see that we will need $S[0], \dots, S[4]$ to compute the result at $[2, 2]$, and we will need $S[1], \dots, S[5]$ to compute the result at $[3, 2]$. So, in the separable formulation, we can just compute all six values of S and share $S[1], \dots, S[4]$ (Figure 9.30).

This savings has great significance for large filters. Filtering an image with a filter of radius r in the general case requires computation of $(2r + 1)^2$ products per pixel, while filtering the image with a separable filter of the same size requires $2(2r + 1)$ products (at the expense of some intermediate storage). This change in asymptotic complexity from $O(r^2)$ to $O(r)$ enables the use of much larger filters.

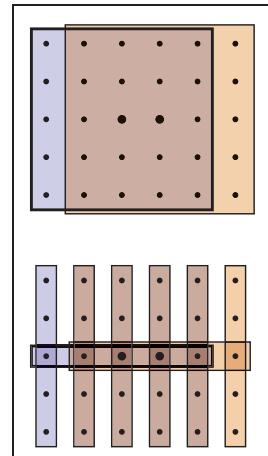


Figure 9.30. Computing two output points using separate 2D arrays of 25 samples (above) vs. filtering once along the columns, then using separate 1D arrays of five samples (below).

函数。在同一时间圆对称和可分离的性质是高斯函数所特有的。沿坐标轴的轮廓是高斯，但沿任何方向的轮廓也是如此，在任何偏移中心。

与其他2D滤波器相比，可分离滤波器的关键优势与实现效率有关。让我们将 a_2 的定义替换为离散卷积的定义：

$$(a * b_2)[i, j] = \sum_{i'} \sum_{j'} a[i', j'] b_1[i - i'] b_1[j - j'].$$

请注意， $b_1[i - i']$ 不依赖于 j' ，并且可以从内部总和中考虑：

$$= \sum_{i'} b_1[i - i'] \sum_{j'} a[i', j'] b_1[j - j'].$$

让我们将内和缩写为 $S[i']$ ：

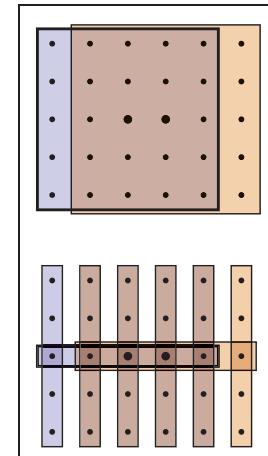
$$\begin{aligned} S[i'] &= \sum_{j'} a[i', j'] b_1[j - j']; \\ (a * b_2)[i, j] &= \sum_{i'} b_1[i - i'] S[i']. \end{aligned} \quad (9.5)$$

使用这种形式的方程，我们可以首先为 i' 的每个值计算并存储 $S[i']$ ，然后使用这些存储的值计算外和。乍一看，这似乎并不显着，因为我们仍然必须做与 $(2r+1)^2$ 成比例的工作

来计算所有的内和。但是，如果我们要计算许多点 $[i, j]$ 的值，则完全不同。

假设我们需要在 $[2, 2]$ 和 $[3, 2]$ 处计算 $a * b_2$ ，并且 b_1 的半径为2。检查公式9.5，我们可以看到我们将需要 $S[0], \dots, S[4]$ 来计算 $[2, 2]$ 处的结果，我们将需要 $S[1], \dots, S[5]$ 来计算 $[3, 2]$ 处的结果。因此，在可分离公式中，我们可以计算 S 的所有六个值并共享 $S[1], \dots, S[4]$ (Figure 9.30)。

这种节省对于大型过滤器具有重大意义。一般情况下用半径为 r 的滤波器滤波图像需要每像素计算 $(2r+1)^2$ 个乘积，而用相同大小的可分离滤波器滤波图像需要 $2(2r+1)$ 个乘积（代价是一些中渐近复杂度从 $O(r^2)$ 到 $O(r)$ 的这种变化使得能够使用大得多的滤波器）。



使用25个samples的separate 2D阵列计算两个输出点（上图），而不是沿着列过滤一次，然后使用5个samples的单独1D阵列（下图）。

The algorithm is:

```

function filterImage(image I, filter b)
    r = b.radius
    nx = I.width
    ny = I.height
    allocate storage array S[0 ... nx - 1]
    allocate image Iout[r ... nx - r - 1, r ... ny - r - 1]
    initialize S and Iout to all zero
    for j = r to ny - r - 1 do
        for i' = 0 to nx - 1 do
            S[i'] = 0
            for j' = j - r to j + r do
                S[i'] = S[i'] + I[i', j']b[j - j']
            for i = r to nx - r - 1 do
                for i' = i - r to i + r do
                    Iout[i, j] = Iout[i, j] + S[i']b[i - i']
    return Iout
```

For simplicity, this function avoids all questions of boundaries by trimming r pixels off all four sides of the output image. In practice there are various ways to handle the boundaries; see Section 9.4.3.

9.4 Signal Processing for Images

We have discussed sampling, filtering, and reconstruction in the abstract so far, using mostly 1D signals for examples. But as we observed at the beginning of the chapter, the most important and most common application of signal processing in graphics is for sampled images. Let us look carefully at how all this applies to images.

9.4.1 Image Filtering Using Discrete Filters

Perhaps the simplest application of convolution is processing images using discrete convolution. Some of the most widely used features of image manipulation programs are simple convolution filters. Blurring of images can be achieved by convolving with many common lowpass filters, ranging from the box to the Gaussian (Figure 9.31). A Gaussian filter creates a very smooth-looking blur and is commonly used for this purpose.

The opposite of blurring is sharpening, and one way to do this is by using the “unsharp mask” procedure: subtract a fraction α of a blurred image from the

该算法是：

```

函数filterImage(imageI filterb)r=b.radiusnx=I.widthny=I.height
存储数组S[0...nx-1]分配图像Iout[r...nx-r-1, r...ny-r-1]
初始化S和Iout为全零forj=r to ny-r-1 do
fori'=0 to nx-1 do
    S[i']=0
    forj'=j-r to j+r do
        S[i']=S[i']+I[i', j']b[j-j']
    fori=r to nx-r-1 do
        fori'=i-r to i+r do
            Iout[i, j]=Iout[i, j]+S[i']b[i-i']
returnIout
```

为简单起见，该函数通过修剪输出图像的所有四个边的 r 像素来避免边界的所有问题。在实践中，有各种方法来处理边界；请参阅第9.4.3节。

9.4 图像信号处理

到目前为止，我们已经在摘要中讨论了采样、滤波和重建，主要使用一维信号作为示例。但正如我们在本章开头所观察到的，信号处理在图形中最重要也是最常见的应用是采样图像。让我们仔细看看这一切如何适用于图像。

9.4.1 使用离散滤波器进行图像滤波

也许卷积最简单的应用是使用离散卷积处理图像。一些最广泛使用的图像处理程序的功能是简单的卷积滤波器。图像的模糊可以通过使用许多常见的低通滤波器进行卷积来实现，范围从盒子到Gaussian（图9.31）。高斯滤波器创建一个非常平滑的模糊，通常用于此目的。

与模糊相反的是锐化，其中一种方法是使用“unsharp mask”程序：从模糊图像中减去一小部分 α 。



Figure 9.31. Blurring an image by convolution with each of three different filters.

original. With a rescaling to avoid changing the overall brightness, we have

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

where $f_{g,\sigma}$ is the Gaussian filter of width σ . Using the discrete impulse d and the distributive property of convolution, we were able to write this whole process as a single filter that depends on both the width of the blur and the degree of sharpening (Figure 9.32).



Figure 9.32. Sharpening an image using a convolution filter.

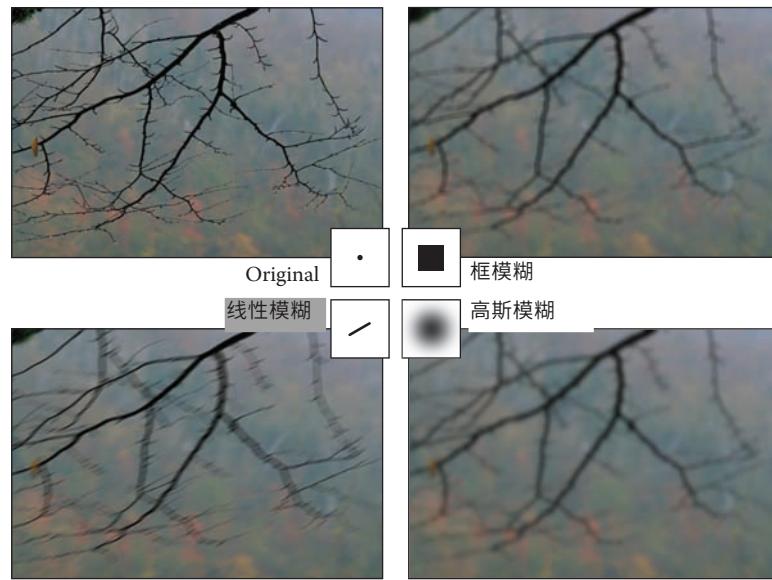


图9.31。通过与三个不同滤波器中的每一个进行卷积来模糊图像。

原创。通过重新调整以避免改变整体亮度，我们有

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

其中 f_g , σ 是宽度 σ 的高斯滤波器。使用离散内隐 d 和卷积的分布属性，我们能够将整个过程写成一个单一的过滤器，它取决于模糊的宽度和锐化程度（图9.32）。



图9.32。 使用卷积滤波器锐化图像。

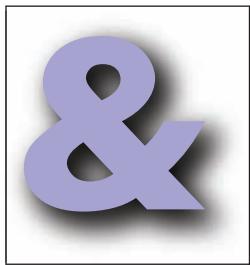


Figure 9.33. A soft drop shadow.

Another example of combining two discrete filters is a drop shadow. It's common to take a blurred, shifted copy of an object's outline to create a soft drop shadow (Figure 9.33). We can express the shifting operation as convolution with an off-center impulse:

$$d_{m,n}(i,j) = \begin{cases} 1 & i = m \text{ and } j = n, \\ 0 & \text{otherwise.} \end{cases}$$

Shifting, then blurring, is achieved by convolving with both filters:

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

Here we have used associativity to group the two operations into a single filter with three parameters.

9.4.2 Antialiasing in Image Sampling

In image synthesis, we often have the task of producing a sampled representation of an image for which we have a continuous mathematical formula (or at least a procedure we can use to compute the color at any point, not just at integer pixel positions). Ray tracing is a common example; more about ray tracing and the specific methods for antialiasing is in Chapter 4. In the language of signal processing, we have a continuous 2D signal (the image) that we need to sample on a regular 2D lattice. If we go ahead and sample the image without any special measures, the result will exhibit various aliasing artifacts (Figure 9.34). At sharp edges in the image, we see stair-step artifacts known as "jaggies." In areas where there are repeating patterns, we see wide bands known as *moiré patterns*.

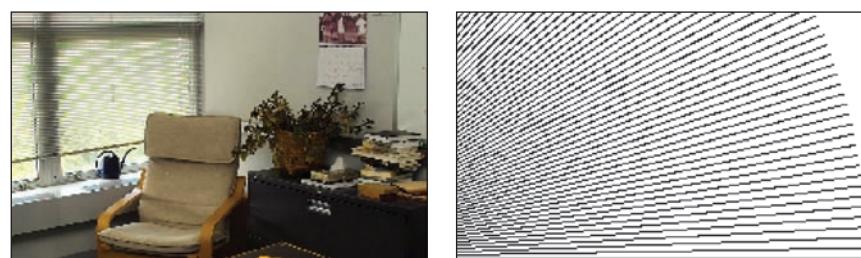


Figure 9.34. Two artifacts of aliasing in images: moiré patterns in periodic textures (left), and "jaggies" on straight lines (right).

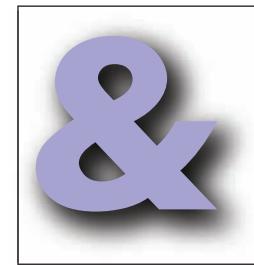


Figure 9.33. 轻轻一滴 shadow.

组合两个离散滤镜的另一个示例是投影。通常采用对象轮廓的模糊移动副本来创建柔和的阴影（图9.33）。我们可以将移位操作表示为卷积 an off-center impulse:

$$d_{m,n}(i,j) = \begin{cases} 1 & i = m \text{ and } j = n, \\ 0 & \text{otherwise.} \end{cases}$$

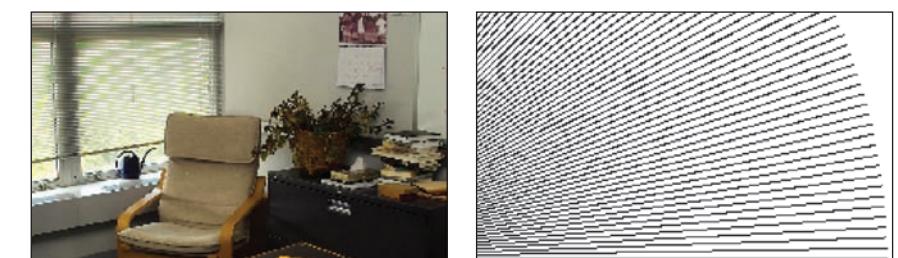
通过使用两个滤波器进行卷积来实现移位，然后模糊：

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

在这里，我们使用关联性将两个操作分组到具有三个参数的单个过滤器中。

9.4.2 图像采样中的抗锯齿

在图像合成中，我们经常需要生成图像的采样表示，我们有一个连续的数学公式（或者至少是一个我们可以用来计算任何点上的颜色的过程，而不仅仅是在整数像素位置）。光线追踪是一个常见的例子；更多关于光线追踪和抗锯齿的具体方法在第4章中。在信号处理语言中，我们有一个连续的2D信号（图像），我们需要在一个规则的2d晶格上采样。如果我们在没有任何特殊措施的情况下继续对图像进行采样，结果将呈现各种混叠伪像（图9.34）。在图像的尖锐边缘，我们看到被称为“jaggies”的楼梯级伪像。“在有重复图案的区域，我们看到被称为云纹图案的宽带。



图像中混叠的两个伪像：周期性纹理中的莫尔图案（左）和直线上的“jaggies”（右）。

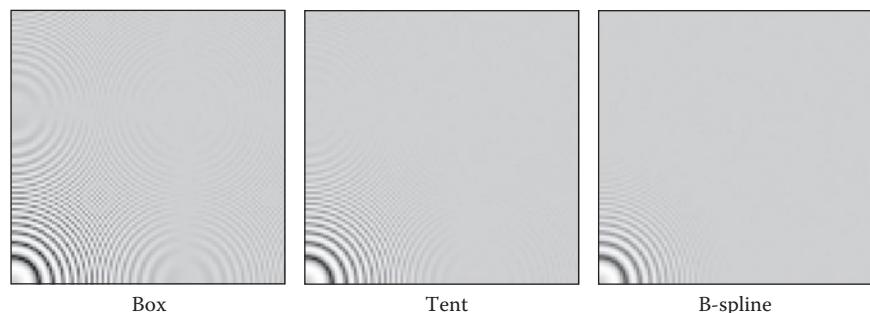


Figure 9.35. A comparison of three different sampling filters being used to antialias a difficult test image that contains circles that are spaced closer and closer as they get larger.

The problem here is that the image contains too many small-scale features; we need to smooth it out by filtering it before sampling. Looking back at the definition of continuous convolution in Equation (9.3), we need to average the image over an area around the pixel location, rather than just taking the value at a single point. The specific methods for doing this are discussed in Chapter 4. A simple filter like a box will improve the appearance of sharp edges, but it still produces some moiré patterns (Figure 9.35). The Gaussian filter, which is very smooth, is much more effective against the moiré patterns, at the expense of overall somewhat more blurring. These two examples illustrate the tradeoff between sharpness and aliasing that is fundamental to choosing antialiasing filters.

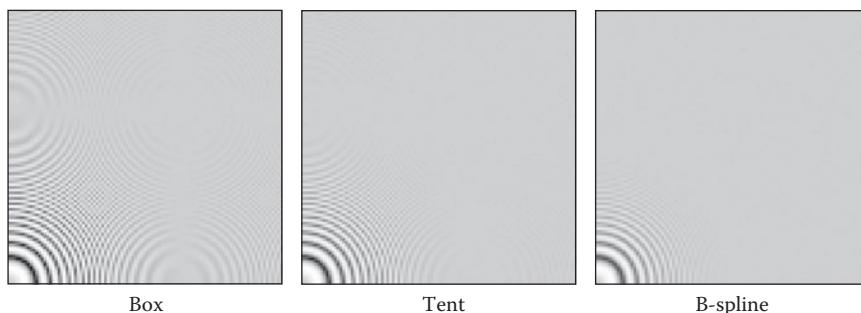
9.4.3 Reconstruction and Resampling

One of the most common image operations where careful filtering is crucial is *resampling*—changing the sample rate, or changing the image size.

Suppose we have taken an image with a digital camera that is 3000 by 2000 pixels in size, and we want to display it on a monitor that has only 1280 by 1024 pixels. In order to make it fit, while maintaining the 3:2 aspect ratio, we need to resample it to 1278 by 852 pixels. How should we go about this?

One way to approach this problem is to think of the process as dropping pixels: the size ratio is between 2 and 3, so we'll have to drop out one or two pixels between pixels that we keep. It's possible to shrink an image in this way, but the quality of the result is low—the images in Figure 9.34 were made using pixel dropping. Pixel dropping is very fast, however, and it is a reasonable choice to make a preview of the resized image during an interactive manipulation.

The way to think about resizing images is as a *resampling* operation: we want a set of samples of the image on a particular grid that is defined by the new



三个不同的采样滤波器的比较被用来反锯齿一个困难的测试图像，其中包含的圆圈，间隔越来越近，因为他们变得更大。

这里的问题是图像包含太多的小尺度特征;我们需要在采样之前通过过滤来平滑它。回顾方程 (9.3) 中连续卷积的定义，我们需要在像素位置周围的一个区域上对图像进行平均，而不仅仅是在单个点上取值。执行此操作的具体方法在第4章中讨论。像盒子这样的简单过滤器会改善尖锐边缘的外观，但它仍然会产生一些云纹图案（图9.35）。高斯滤波器，这是非常平滑的，是更有效的莫尔模式，以牺牲整体更模糊。这两个例子说明了锐度和混叠之间的权衡，这是选择抗混叠滤波器的基础。

9.4.3 重建和重采样

仔细滤波至关重要的最常见图像操作之一是重采样—更改采样率或更改图像大小。

假设我们用3000x2000像素大小的数码相机拍摄了一张图像，我们希望将其显示在只有1280x1024像素的显示器上。为了使其适合，同时保持3: 2宽高比，我们需要将其重新采样到1278乘852像素。我们该怎么做呢？

解决这个问题的一种方法是将该过程视为丢弃像素：大小比率在2到3之间，因此我们必须在保留的像素之间丢弃一个或两个像素。可以通过这种方式缩小图像，但结果的质量很低—图9.34中的图像是使用像素下降制作的。然而，像素下降速度非常快，在交互操作期间预览调整大小的图像是一个合理的选择。

考虑调整图像大小的方法是作为一个重采样操作：我们想要一组由新定义的特定网格上的图像样本

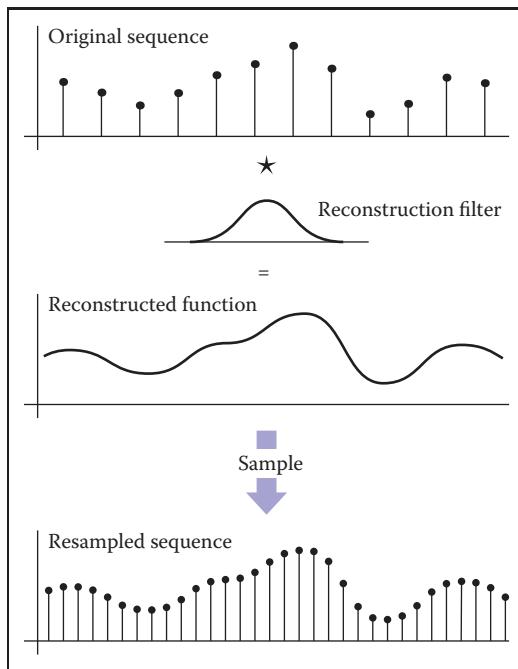


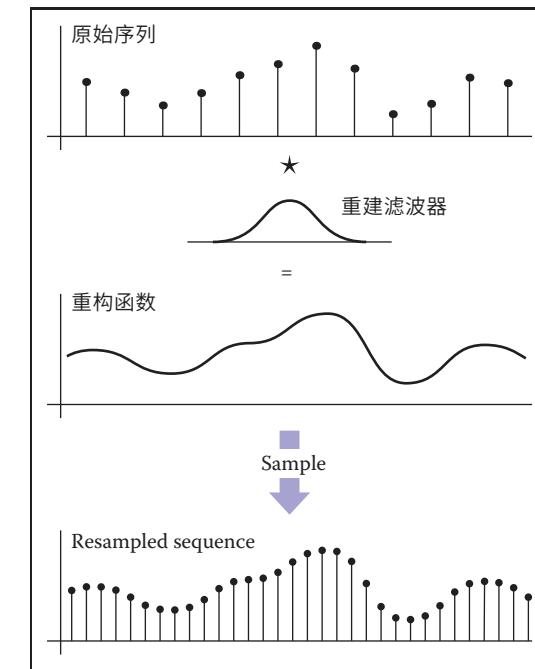
Figure 9.36. Resampling an image consists of two logical steps that are combined into a single operation in code. First, we use a reconstruction filter to define a smooth, continuous function from the input samples. Then, we sample that function on a new grid to get the output samples.

image dimensions, and we get them by sampling a continuous function that is reconstructed from the input samples (Figure 9.36). Looking at it this way, it's just a sequence of standard image processing operations: first we reconstruct a continuous function from the input samples, and then we sample that function just as we would sample any other continuous image. To avoid aliasing artifacts, appropriate filters need to be used at each stage.

A small example is shown in Figure 9.37: if the original image is 12×9 pixels and the new one is 8×6 pixels, there are $2/3$ as many output pixels as input pixels in each dimension, so their spacing across the image is $3/2$ the spacing of the original samples.

In order to come up with a value for each of the output samples, we need to somehow compute values for the image in between the samples. The pixel-dropping algorithm gives us one way to do this: just take the value of the closest sample in the input image and make that the output value. This is exactly equivalent to reconstructing the image with a 1-pixel-wide (radius one-half) box filter and then point sampling.

Of course, if the main reason for choosing pixel dropping or other very simple filtering is performance, one would never *implement* that method as a special



对图像进行重采样由两个逻辑步骤组成，这些步骤在代码中组合为单个操作。首先，我们使用重建滤波器从输入样本中定义一个平滑的连续函数。然后，我们在一个新的网格上对该函数进行采样以获得输出样本。

图像尺寸，我们通过采样从输入样本重建的连续函数来获得它们（图9.36）。这样看，它只是一系列标准的图像处理操作：首先我们从输入样本中重建一个连续函数，然后我们对该函数进行采样，就像对任何其他连续图像进行采样一样。为了避免混叠伪影，需要在每个阶段使用适当的滤波器。

一个小例子如图9.37所示：如果原始图像是 12×9 像素，而新图像是 8×6 像素，则在每个维度中有23作为输入像素的输出像素，因此它们在图像上的间距是原始样本间距的32。

为了得出每个输出样本的值，我们需要以某种方式计算样本之间图像的值。像素丢弃算法为我们提供了一种方法：只需取输入图像中最接近样本的值，并使其成为输出值。这完全等同于用1像素宽（半径二分之一）盒滤波器重建图像，然后进行点采样。

当然，如果选择像素丢弃或其他非常简单的滤波的主要原因是性能，则永远不会将该方法作为特殊方法实现。

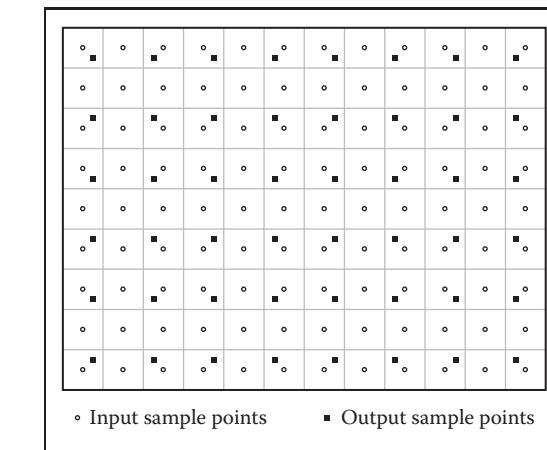


Figure 9.37. The sample locations for the input and output grids in resampling a 12 by 9 image to make an 8 by 6 one.

case of the general reconstruction-and-resampling procedure. In fact, because of the discontinuities, it's difficult to make box filters work in a general framework. But, for high-quality resampling, the reconstruction/sampling framework provides valuable flexibility.

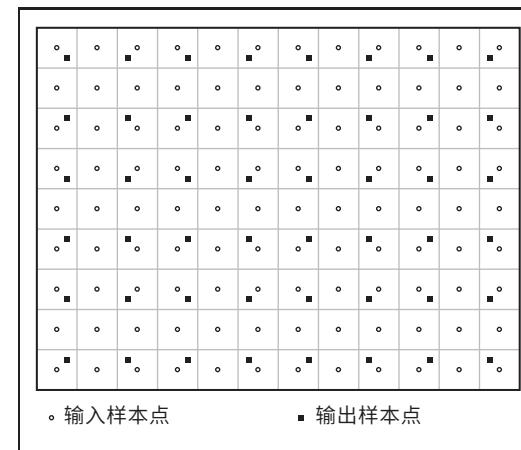
To work out the algorithmic details, it's simplest to drop down to 1D and discuss resampling a sequence. The simplest way to write an implementation is in terms of the *reconstruct* function we defined in Section 9.2.5.

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

The parameter x_0 gives the position of the first sample of the new sequence in terms of the samples of the old sequence. That is, if the first output sample falls midway between samples 3 and 4 in the input sequence, x_0 is 3.5.

This procedure reconstructs a continuous image by convolving the input sequence with a continuous filter and then point samples it. That's not to say that these two operations happen sequentially—the continuous function exists only in principle and its values are computed only at the sample points. But mathematically, this function computes a set of point samples of the function $a * f$.

This point sampling seems wrong, though, because we just finished saying that a signal should be sampled with an appropriate smoothing filter to avoid aliasing. We should be convolving the reconstructed function with a sampling filter g and point sampling $g * (f * a)$. But since this is the same as $(g * f) * a$,



输入和输出网格的示例位置在重新采样一个12乘9的图像，使一个8乘6的一个。

一般重建和重采样程序的情况。实际上，由于不连续性，很难使盒式过滤器在一般的框架工作中工作。但是，对于高质量重采样，重建采样框架提供了宝贵的灵活性。

为了计算算法细节，最简单的方法是下拉到一维并讨论重新采样序列。编写实现的最简单方法是根据我们在第9.2.5节中定义的重构函数。

```
function resample(sequence a float x0 float ΔX int n filter f)
    f)为i=0到n-1创建长度为n的序列b。
    b[i]=reconstruct(a f x0+IΔx)返回b
```

参数x0给出新序列的第一个样本在旧序列的样本方面的位置。也就是说，如果第一个输出样本落在输入序列中的样本3和4之间，则x0为3.5。

该过程通过用连续滤波器对输入sequence进行卷积，然后对其进行点采样来重建连续图像。这并不是说这两个操作顺序发生—连续函数仅在原理上存在，其值仅在样本点计算。但是mathematically，这个函数计算函数 $a \in f$ 的一组点样本。

然而，这个点采样似乎是错误的，因为我们刚刚说完应该用适当的平滑滤波器对信号进行采样以避免混叠。我们应该用采样滤波器 g 和点采样 $g * (f * a)$ 对重构函数进行卷积。但由于这与 $(g * f) * a$ 相同

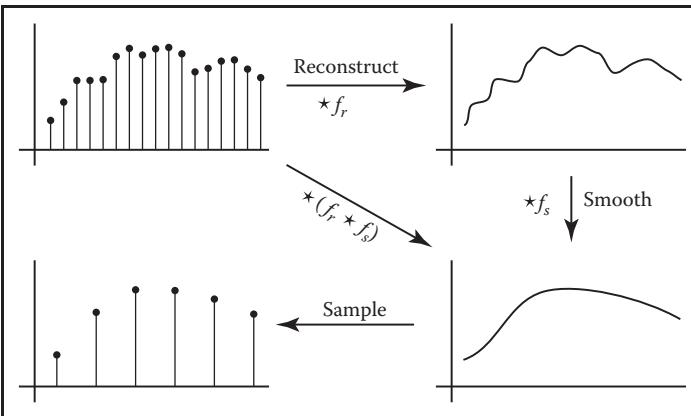


Figure 9.38. Resampling involves filtering for reconstruction and for sampling. Since two convolution filters applied in sequence can be replaced with a single filter, we only need one resampling filter, which serves the roles of reconstruction and sampling.

we can roll the sampling filter together with the reconstruction filter; one convolution operation is all we need (Figure 9.38). This combined reconstruction and sampling filter is known as a *resampling filter*.

When resampling images, we usually specify a *source rectangle* in the units of the old image that specifies the part we want to keep in the new image. For example, using the pixel sample positioning convention from Chapter 3, the rectangle we'd use to resample the entire image is $(-0.5, n_x^{\text{old}} - 0.5) \times (-0.5, n_y^{\text{old}} - 0.5)$. Given a source rectangle $(x_l, x_h) \times (y_l, y_h)$, the sample spacing for the new image is $\Delta x = (x_h - x_l)/n_x^{\text{new}}$ in x and $\Delta y = (y_h - y_l)/n_y^{\text{new}}$ in y . The lower-left sample is positioned at $(x_l + \Delta x/2, y_l + \Delta y/2)$.

Modifying the 1D pseudocode to use this convention, and expanding the call to the reconstruct function into the double loop that is implied, we arrive at:

```
function resample(sequence a, float xl, float xh, int n, filter f)
    create sequence b of length n
    r = f.radius
    x0 = xl + Δx/2
    for i = 0 to n - 1 do
        s = 0
        x = x0 + iΔx
        for j = ⌈x - r⌉ to ⌈x + r⌉ do
            s = s + a[j]f(x - j)
        b[i] = s
    return b
```

This routine contains all the basics of resampling an image. One last issue that

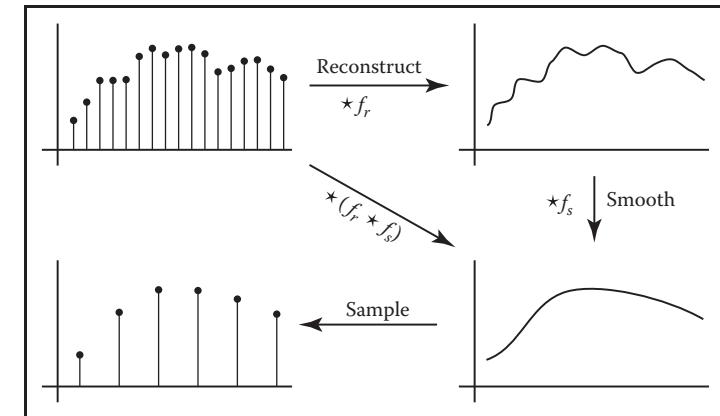


图9.38。重采样涉及用于重建和用于采样的滤波。由于两个按顺序应用的卷积滤波器可以用一个滤波器代替，我们只需要一个重采样滤波器，它起到重建和采样的作用。

我们可以将采样滤波器与重建滤波器一起滚动；一个卷积运算就是我们所需要的（图9.38）。这种组合的重构和采样滤波器被称为重采样滤波器。

在重新采样图像时，我们通常以旧图像的单位指定一个源矩形，该矩形指定了我们希望保留在新图像中的部分。例如，使用第3章的像素样本定位约定，我们用于重新采样整个图像的矩形是 $(-0.5, n_x^{\text{old}} - 0.5) \times (-0.5, n_y^{\text{old}} - 0.5)$ 。

给定一个源矩形 $(xl, xh) \times (yl, yh)$ ，新im年龄的样本间距为 $\Delta x = (xh - xl)/n_x^{\text{new}}$

在x和 $\Delta y = (yh - yl)/n_y^{\text{new}}$ 在y。左下方

样品定位在 $(xl + \Delta x/2, yl + \Delta y/2)$ 。

修改1D伪代码以使用此约定，并将对reconstruct函数的调用扩展到隐含的双循环中，我们到达：

```
函数重采样 (序列a, float xl, float xh, int n, filter f)
    创建长度为n的序列b
    r = f.radius
    x0 = xl + Δx/2
    for i = 0 to n - 1 do
        s = 0
        for j = ⌈x - r⌉ to ⌈x + r⌉ do
            s = s + a[j]f(x - j)
        b[i] = s
    return b
```

此例程包含重新采样图像的所有基础知识。最后一个问题是



remains to be addressed is what to do at the edges of the image, where the simple version here will access beyond the bounds of the input sequence. There are several things we might do:

- Just stop the loop at the ends of the sequence. This is equivalent to padding the image with zeros on all sides.
- Clip all array accesses to the end of the sequence—that is, return $a[0]$ when we would want to access $a[-1]$. This is equivalent to padding the edges of the image by extending the last row or column.
- Modify the filter as we approach the edge so that it does not extend beyond the bounds of the sequence.

The first option leads to dim edges when we resample the whole image, which is not really satisfactory. The second option is easy to implement; the third is probably the best performing. The simplest way to modify the filter near the edge of the image is to *renormalize* it: divide the filter by the sum of the part of the filter that falls within the image. This way, the filter always adds up to 1 over the actual image samples, so it preserves image intensity. For performance, it is desirable to handle the band of pixels within a filter radius of the edge (which require this renormalization) separately from the center (which contains many more pixels and does not require renormalization).

The choice of filter for resampling is important. There are two separate issues: the shape of the filter and the size (radius). Because the filter serves both as a reconstruction filter and a sampling filter, the requirements of both roles affect the choice of filter. For reconstruction, we would like a filter smooth enough to avoid aliasing artifacts when we enlarge the image, and the filter should be ripple-free. For sampling, the filter should be large enough to avoid undersampling and smooth enough to avoid moiré artifacts. Figure 9.39 illustrates these two different needs.

Generally, we will choose one filter shape and scale it according to the relative resolutions of the input and output. The lower of the two resolutions determines the size of the filter: when the output is more coarsely sampled than the input (downsampling, or shrinking the image), the smoothing required for proper sampling is greater than the smoothing required for reconstruction, so we size the filter according to the output sample spacing (radius 3 in Figure 9.39). On the other hand, when the output is more finely sampled (upsampling, or enlarging the image) then the smoothing required for reconstruction dominates (the reconstructed function is already smooth enough to sample at a higher rate than it started), so the size of the filter is determined by the input sample spacing (radius 1 in Figure 9.39).



仍然需要解决的是在图像的边缘做什么，这里的简单版本将访问超出输入序列的边界。我们可以做几件事：

- *只需在序列结束时停止循环即可。这相当于在所有边上用零填充图像。
- *剪辑所有数组访问到序列的末尾—也就是说，当我们想要访问 $a[-1]$ 时返回 $a[0]$ 。这相当于通过扩展最后一行或列来填充图像的边缘。
- *当我们接近边缘时修改过滤器，使其不会超出序列的边界。

当我们重新采样整个图像时，第一个选项会导致暗淡的边缘，这并不是真正令人满意的。第二个选项很容易实现；第三个可能是性能最好的。修改图像边缘附近的滤镜的最简单方法是将其重新规范化：将滤镜除以滤镜落在图像内的部分的总和。这样，过滤器总是在实际图像样本上加起来最多1，因此它保留了图像强度。为了性能，期望处理边缘的一个滤波器半径内的像素的带（其需要这种重归一化）与中心分开（其包含许多更多的像素并且不需要重归一化）。

用于重采样的滤波器的选择是重要的。有两个单独的问题：过滤器的形状和大小（半径）。因为滤波器既充当重构滤波器又充当采样滤波器，所以这两个角色的要求都影响滤波器的选择。对于重建，我们希望有一个足够平滑的滤波器，以避免在放大图像时出现混叠伪像，并且滤波器应该是无纹波的。对于采样，滤波器应该足够大以避免欠采样并且足够平滑以避免莫尔伪影。图9.39说明了这两种不同的需求。

通常，我们会选择一种滤波器形状，并根据输入和输出的相对分辨率对其进行缩放。两个分辨率中较低的一个决定了滤波器的大小：当输出比输入更粗采样（下采样或缩小图像）时，适当的sampling所需的平滑大于重建所需的平滑，因此我们根据输出样本间距（图9.39中的半径3）来调整滤波器的大小。另一方面，当输出被更精细地采样（上采样或放大im年龄）时，重建所需的平滑占主导地位（重建函数已经足够平滑，可以以比它开始的更高的速率采样），因此滤波器的大小由输入样本间距（图9.39中的半径1）决定。

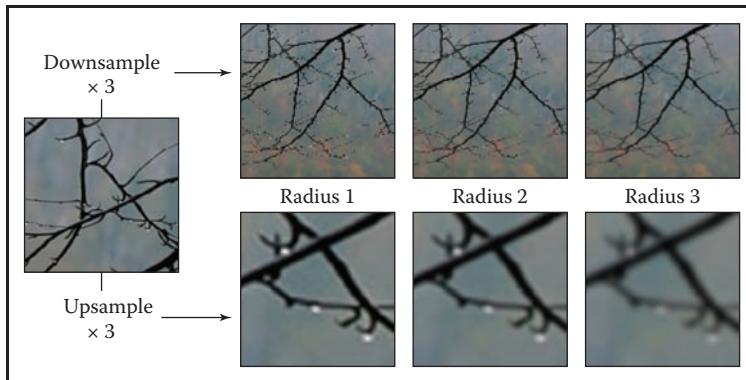


Figure 9.39. The effects of using different sizes of a filter for upsampling (enlarging) or downsampling (reducing) an image.

Choosing the filter itself is a tradeoff between speed and quality. Common choices are the box filter (when speed is paramount), the tent filter (moderate quality), or a piecewise cubic (excellent quality). In the piecewise cubic case, the degree of smoothing can be adjusted by interpolating between f_B and f_C ; the Mitchell-Netravali filter is a good choice.

Just as with image filtering, separable filters can provide a significant speed-up. The basic idea is to resample all the rows first, producing an image with changed width but not height, then to resample the columns of that image to produce the final result (Figure 9.40). Modifying the pseudocode given earlier so that it takes advantage of this optimization is reasonably straightforward.

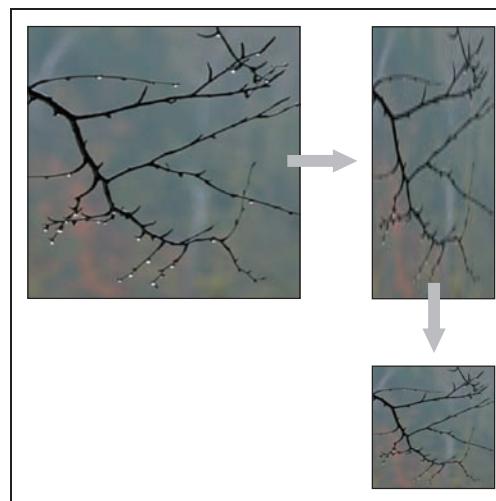


Figure 9.40. Resampling an image using a separable approach.

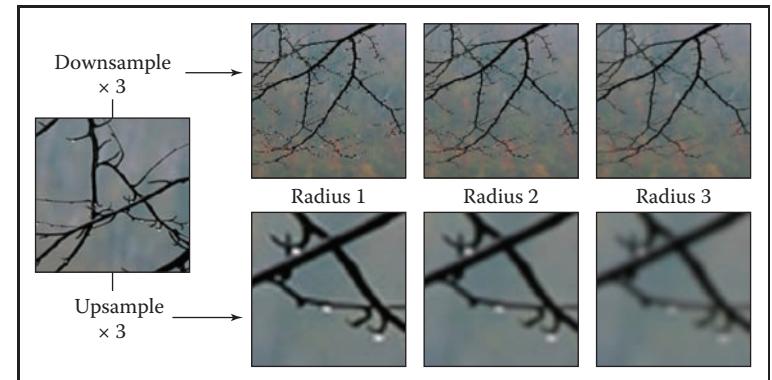


图9.39。 使用不同大小的滤波器对图像进行上采样（放大）或下采样（缩小）的效果。

选择过滤器本身是速度和质量之间的权衡。常见的选择是箱式过滤器（当速度至关重要时），帐篷过滤器（中等质量）或分段立方（卓越质量）。在分段三次情况下，可以通过在fb和fc之间插值来调整平滑程度；Mitchell-Netravali滤波器是一个很好的选择。

就像图像过滤一样，可分离过滤器可以提供显着的加速。基本思想是首先对所有行进行重新采样，生成一个宽度改变但高度改变的图像，然后对该图像的列进行重新采样以产生最终结果（图9.40）。修改前面给出的伪代码以便利用这种优化是相当简单的。

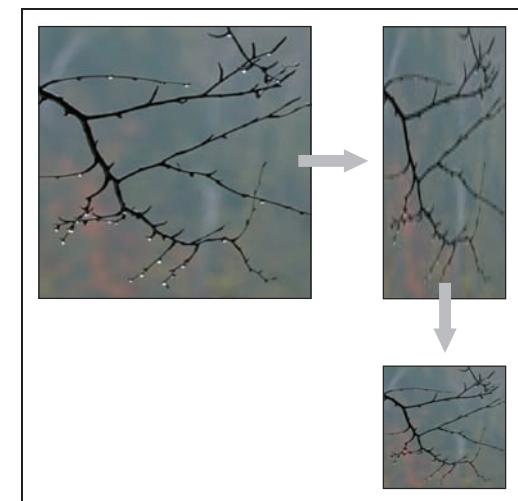


图9.40。 使用可分离方法重新采样图像。



9.5 Sampling Theory

If you are only interested in implementation, you can stop reading here; the algorithms and recommendations in the previous sections will let you implement programs that perform sampling and reconstruction and achieve excellent results. However, there is a deeper mathematical theory of sampling with a history reaching back to the first uses of sampled representations in telecommunications. Sampling theory answers many questions that are difficult to answer with reasoning based strictly on scale arguments.

But most important, sampling theory gives valuable insight into the workings of sampling and reconstruction. It gives the student who learns it an extra set of intellectual tools for reasoning about how to achieve the best results with the most efficient code.

9.5.1 The Fourier Transform

The Fourier transform, along with convolution, is the main mathematical concept that underlies sampling theory. You can read about the Fourier transform in many math books on analysis, as well as in books on signal processing.

The basic idea behind the Fourier transform is to express any function by adding together sine waves (sinusoids) of all frequencies. By using the appropriate weights for the different frequencies, we can arrange for the sinusoids to add up to any (reasonable) function we want.

As an example, the square wave in Figure 9.41 can be expressed by a sequence of sine waves:

$$\sum_{n=1,3,5,\dots}^{\infty} \frac{4}{\pi n} \sin 2\pi n x.$$

This *Fourier series* starts with a sine wave ($\sin 2\pi x$) that has frequency 1.0—same as the square wave—and the remaining terms add smaller and smaller corrections to reduce the ripples and, in the limit, reproduce the square wave exactly. Note that all the terms in the sum have frequencies that are integer multiples of the frequency of the square wave. This is because other frequencies would produce results that don't have the same period as the square wave.

A surprising fact is that a signal does not have to be periodic in order to be expressed as a sum of sinusoids in this way: a non-periodic signal just requires more sinusoids. Rather than summing over a discrete sequence of sinusoids, we will instead integrate over a continuous family of sinusoids. For instance, a box



9.5 抽样理论

如果您只对实现感兴趣，可以在此处停止阅读：前面各节中的algorithms和建议将让您实现执行采样和重建并取得优异结果的程序。然而，有一个更深层次的采样数学理论，其历史可以追溯到采样表示在电信中的首次使用。山姆普林理论回答了许多问题，这些问题很难用严格基于尺度论证的推理来回答。

但最重要的是，采样理论对采样和重建的工作原理提供了宝贵的见解。它为学习它的学生提供了一套额外的智力工具，用于推理如何用最有效的代码实现最佳结果。

9.5.1 傅立叶变换

傅立叶变换和卷积是采样理论的主要数学概念。您可以在许多关于分析的数学书籍以及信号处理的书籍中阅读有关傅立叶变换的信息。

傅立叶变换背后的基本思想是通过将所有频率的正弦波（正弦）加在一起表达任何函数。通过对不同频率使用适当的权重，我们可以安排正弦曲线加起来成为我们想要的任何（合理的）函数。

例如，图9.41中的方波可以用正弦波序列表示：

$$\sum_{n=1,3,5,\dots}^{\infty} \frac{4}{\pi n} \sin 2\pi n x.$$

这个傅立叶级数从频率为1.0的正弦波($\sin 2\pi n x$)开始—与方波相同—其余项添加越来越小的校正以减少波纹，并且在极限中精确地再现方波。请注意，总和中的所有项的频率都是方波频率的整数倍。这是因为其他频率会产生与方波不同周期的结果。

一个令人惊讶的事实是，信号不必是周期性的，以便以这种方式表示为正弦的总和：非周期性信号只需要更多的正弦。而不是总结在一个离散的正弦序列，我们将集成在一个连续的正弦家族。例如，一个盒子

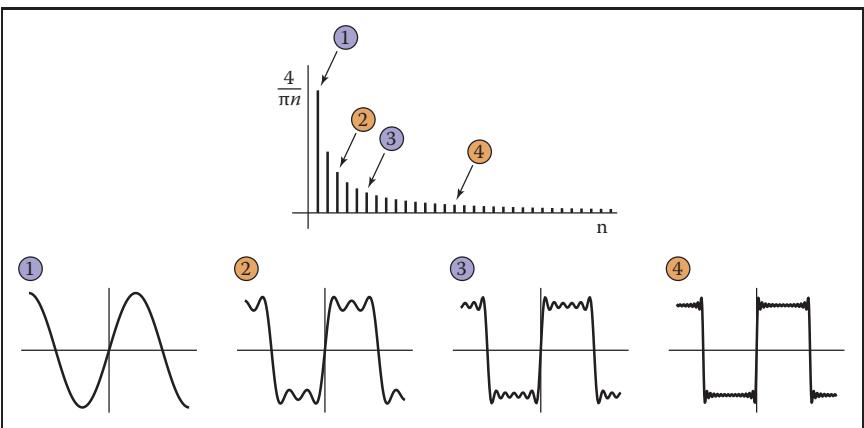


Figure 9.41. Approximating a square wave with finite sums of sines.

function can be written as the integral of a family of cosine waves:

$$\int_{-\infty}^{\infty} \frac{\sin \pi u}{\pi u} \cos 2\pi ux du. \quad (9.6)$$

This integral in Equation (9.6) is adding up infinitely many cosines, weighting the cosine of frequency u by the weight $(\sin \pi u)/\pi u$. The result, as we include higher and higher frequencies, converges to the box function (see Figure 9.42). When a function f is expressed in this way, this weight, which is a function of the frequency u , is called the *Fourier transform* of f , denoted \hat{f} . The function \hat{f} tells us how to build f by integrating over a family of sinusoids:

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(u) e^{2\pi iux} du. \quad (9.7)$$

Equation (9.7) is known as the *inverse Fourier transform* (IFT) because it starts with the Fourier transform of f and ends up with f .²

Note that in Equation (9.7) the complex exponential $e^{2\pi iux}$ has been substituted for the cosine in the previous equation. Also, \hat{f} is a complex-valued function. The machinery of complex numbers is required to allow the phase, as well as the frequency, of the sinusoids to be controlled; this is necessary to represent any functions that are not symmetric across zero. The magnitude of \hat{f} is known as the *Fourier spectrum*, and, for our purposes, this is sufficient—we won't need to worry about phase or use any complex numbers directly.

²Note that the term “Fourier transform” is used both for the function \hat{f} and for the operation that computes \hat{f} from f . Unfortunately, this rather ambiguous usage is standard.

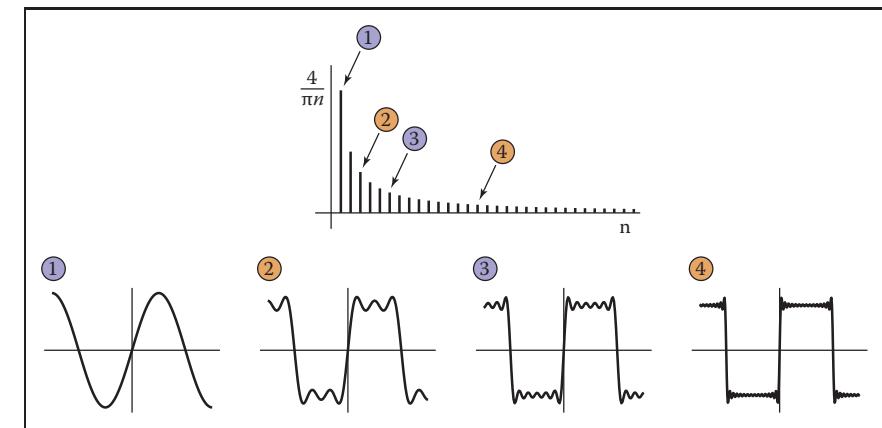


图9.41。用有限的正弦和近似方波。

函数可以写成余弦波家族的积分:

$$\int_{-\infty}^{\infty} \frac{\sin \pi u}{\pi u} \cos 2\pi nux du. \quad (9.6)$$

方程 (9.6) 中的这个积分是将无限多的余弦加起来，用权重 $(\sin \pi u)/\pi u$ 加权频率 u 的余弦。结果，因为我们包括更高和更高的频率，收敛到箱函数 (见图9.42)。当函数 f 以这种方式表示时，这个权重，它是频率 u 的函数，称为 f 的傅立叶变换，记为 $\triangle f$ 。函数 f 告诉我们如何通过在一个正弦家族上积分来建立 f :

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(u) e^{2\pi iux} du. \quad (9.7)$$

方程 (9.7) 被称为逆傅立叶变换 (ift)，因为它从 f 的傅立叶变换开始，以 f 结束。2

请注意，在等式 (9.7) 中，复数指数 $e^{2\pi iux}$ 已被替换为前一个等式中的余弦。此外， f 是一个复值函数。需要复数的机制来控制正弦曲线的相位和频率；这是表示任何在零上对称的函数所必需的。 f 的幅度被称为傅立叶谱，对于我们的目的，这就足够了—我们不需要担心相位或直接使用任何复数。

2请注意，术语“傅立叶变换”既用于函数 f ，也用于从 f 计算 \hat{f} 的运算。不幸的是，这种相当模糊的用法是标准的。

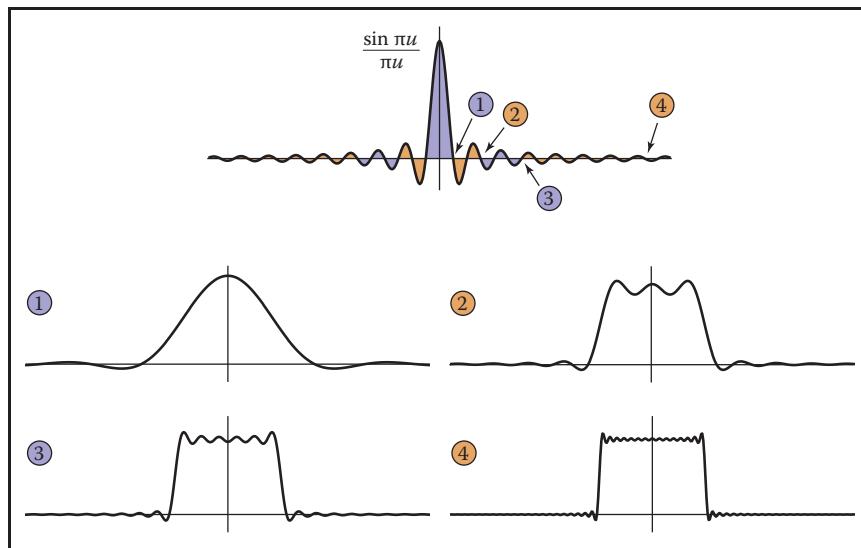


Figure 9.42. Approximating a box function with integrals of cosines up to each of four cutoff frequencies.

It turns out that computing \hat{f} from f looks very much like computing f from \hat{f} :

$$\hat{f}(u) = \int_{-\infty}^{\infty} f(x)e^{-2\pi i ux} dx. \quad (9.8)$$

Equation (9.8) is known as the (forward) *Fourier transform* (FT). The sign in the exponential is the only difference between the forward and inverse Fourier transforms, and it is really just a technical detail. For our purposes, we can think of the FT and IFT as the same operation.

Sometimes the $f - \hat{f}$ notation is inconvenient, and then we will denote the Fourier transform of f by $\mathcal{F}\{f\}$ and the inverse Fourier transform of \hat{f} by $\mathcal{F}^{-1}\{\hat{f}\}$.

A function and its Fourier transform are related in many useful ways. A few facts (most of them easy to verify) that we will use later in the chapter are:

- A function and its Fourier transform have the same squared integral:

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

The physical interpretation is that the two have the same energy (Figure 9.43).

In particular, scaling a function up by a also scales its Fourier transform by a . That is, $\mathcal{F}\{af\} = a\mathcal{F}\{f\}$.

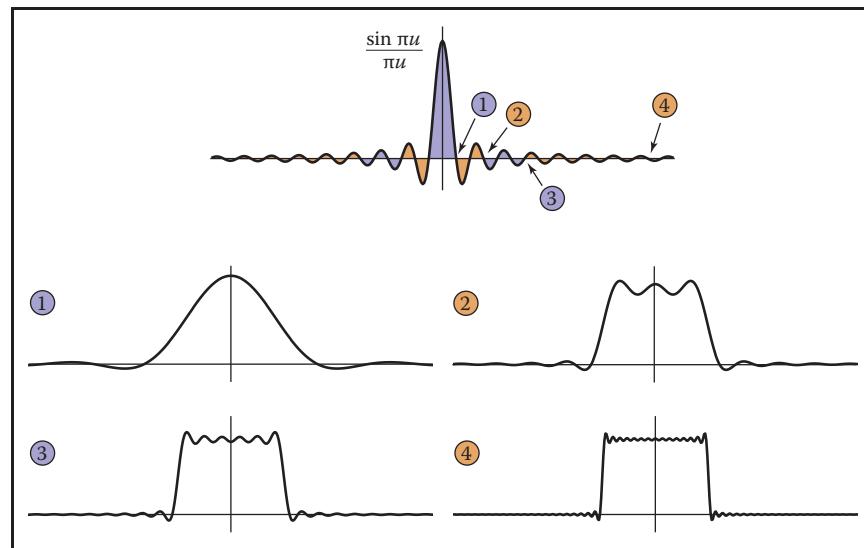


Figure 9.42. 用余弦的积分近似一个箱函数，最多可达四个截止值中的每一个 frequencies.

事实证明，从 f 计算 \hat{f} 看起来非常像从 \hat{f} 计算 f :

$$\hat{f}(u) = \int_{-\infty}^{\infty} f(x)e^{-2\pi i ux} dx. \quad (9.8)$$

公式 (9.8) 被称为 (正向) 傅立叶变换 (FT)。指数中的符号是正向和逆傅立叶变换之间的唯一区别，它实际上只是一个技术细节。出于我们的目的，我们可以将FT和IFT视为相同的操作。

有时 $f - \hat{f}$ 表示法是不方便的，然后我们将通过 $\mathcal{F}\{f\}$ 表示 f 的傅立叶变换和 $\mathcal{F}^{-1}\{\hat{f}\}$ 表示 \hat{f} 的逆傅立叶变换。

函数及其傅立叶变换以许多有用的方式相关。我们将在本章后面使用的一些事实 (其中大多数易于验证) 是：

- *函数及其傅立叶变换具有相同的平方积分：

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

物理解释是两者具有相同的能量 (图9.43)。

特别是，按 a 缩放函数也按 A 缩放其傅立叶变换。即 $\mathcal{F}\{af\} = a\mathcal{F}\{f\}$ 。

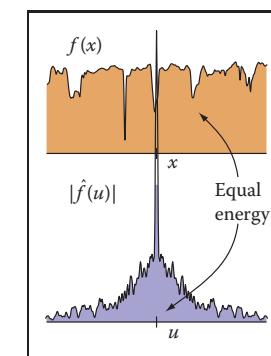


Figure 9.43. The Fourier transform preserves the squared integral of the signal.

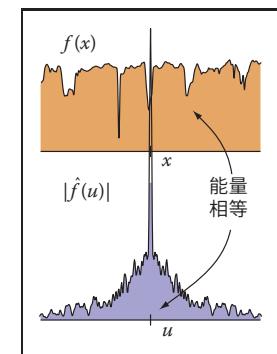


Figure 9.43. 傅立叶保留信号的平方积分。

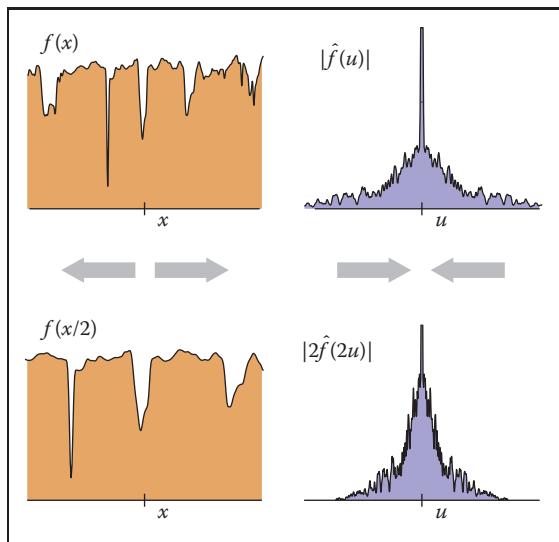


Figure 9.44. Scaling a signal along the x -axis in the space domain causes an inverse scale along the u -axis in the frequency domain.

- Stretching a function along the x -axis squashes its Fourier transform along the u -axis by the same factor (Figure 9.44):

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

(The renormalization by b is needed to keep the energy the same.)

This means that if we are interested in a family of functions of different width and height (say all box functions centered at zero), then we only need to know the Fourier transform of one canonical function (say the box function with width and height equal to one), and we can easily know the Fourier transforms of all the scaled and dilated versions of that function. For example, we can instantly generalize Equation (9.6) to give the Fourier transform of a box of width b and height a :

$$ab \frac{\sin \pi bu}{\pi bu}.$$

- The average value of f is equal to $\hat{f}(0)$. This makes sense since $\hat{f}(0)$ is supposed to be the zero-frequency component of the signal (the DC component if we are thinking of an electrical voltage).
- If f is real (which it always is for us), \hat{f} is an even function—that is, $\hat{f}(u) = \hat{f}(-u)$. Likewise, if f is an even function then \hat{f} will be real (this is not

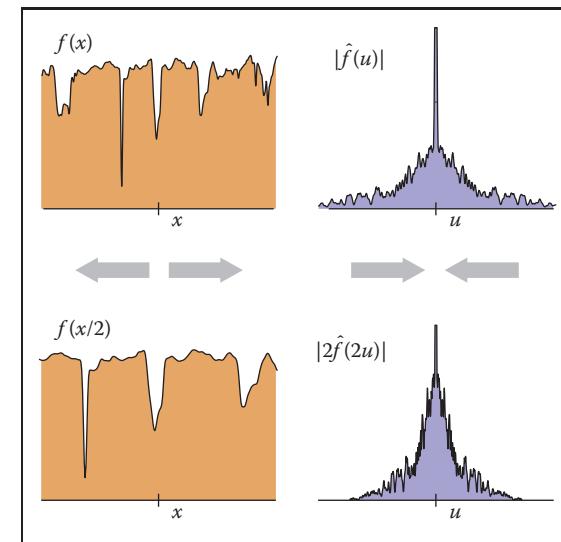


图9.44。在空间域中沿x轴缩放信号导致在频域中沿u轴的逆标度。

- 沿x轴拉伸一个函数将其傅立叶变换沿u轴以相同的因子平方（图9.44）：

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

（需要b的重新规范化以保持能量不变。）

这意味着，如果我们对一系列不同宽度和高度的函数感兴趣（比如所有以零为中心的框函数），那么我们只需要知道一个规范函数的傅立叶变换（比如宽度和高度等于一的框函数），我们就可以很容易地知道该函数的所有缩放和扩张版本的傅立叶变换。例如，我们可以立即推广方程（9.6）给出宽度b和高度a的框的傅立叶变换：

absinnbu

*F的平均值等于 $\Delta f(0)$ 。这是有道理的，因为 $f(0)$ 应该是信号的零频率分量（如果我们考虑的是电压，则为直流分量）。

•如果f是真实的（它总是对我们来说），f是一个偶数函数—即 $f(u) = f(-u)$ 。同样，如果f是偶数函数，那么f将是真实的（这不是

usually the case in our domain, but remember that we really are only going to care about the magnitude of \hat{f}).

9.5.2 Convolution and the Fourier Transform

One final property of the Fourier transform that deserves special mention is its relationship to convolution (Figure 9.45). Briefly,

$$\mathcal{F}\{f \star g\} = \hat{f}\hat{g}.$$

The Fourier transform of the convolution of two functions is the product of the Fourier transforms. Following the by now familiar symmetry,

$$\hat{f} \star \hat{g} = \mathcal{F}\{fg\},$$

The convolution of two Fourier transforms is the Fourier transform of the product of the two functions. These facts are fairly straightforward to derive from the definitions.

This relationship is the main reason Fourier transforms are useful in studying the effects of sampling and reconstruction. We've seen how sampling, filtering, and reconstruction can be seen in terms of convolution; now the Fourier transform gives us a new domain—the frequency domain—in which these operations are simply products.

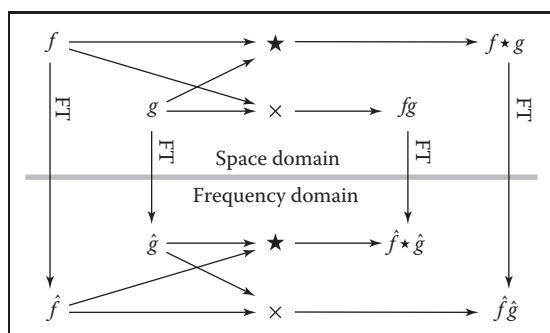


Figure 9.45. A commutative diagram to show visually the relationship between convolution and multiplication. If we multiply f and g in space, then transform to frequency, we end up in the same place as if we transformed f and g to frequency and then convolved them. Likewise, if we convolve f and g in space and then transform into frequency, we end up in the same place as if we transformed f and g to frequency, then multiplied them.

通常情况下在我们的领域，但请记住，我们真的只会关心magnitude f的大小)。

9.5.2 卷积和傅里叶变换

值得特别提及的傅立叶变换的最后一个属性是它与卷积的关系（图9.45）。简言之

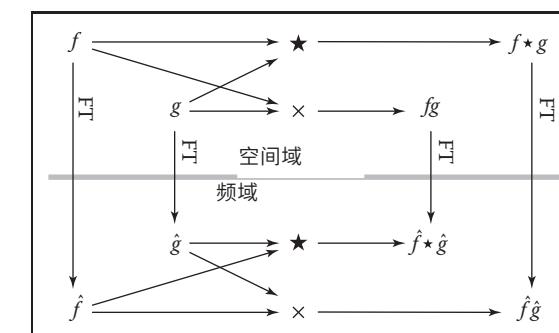
$$\mathcal{F}\{f \star g\} = \hat{f}\hat{g}.$$

两个函数的卷积的傅立叶变换是
傅立叶变换。遵循现在熟悉的对称性

$$\hat{f} \star \hat{g} = \mathcal{F}\{fg\},$$

两个傅立叶变换的卷积是两个函数乘积的傅立叶变换。从定义中得出这些事实相当简单。

这种关系是傅立叶变换在研究采样和重建效应方面有用的主要原因。我们已经看到了采样，滤波和重建如何从卷积的角度来看待；现在傅立叶变换为我们提供了一个新的域—频率域—其中这些操作只是产品。



一个交换图，直观地显示卷积和乘法之间的关系。如果我们在空间中乘以 f 和 g ，然后转换为频率，我们最终会在同一个地方，就像我们将 f 和 g 转换为频率，然后对它们进行卷积一样。同样，如果我们在空间中对 f 和 g 进行卷积，然后转换为频率，我们最终会在同一个地方，就像我们将 f 和 g 转换为频率，然后将它们相乘一样。

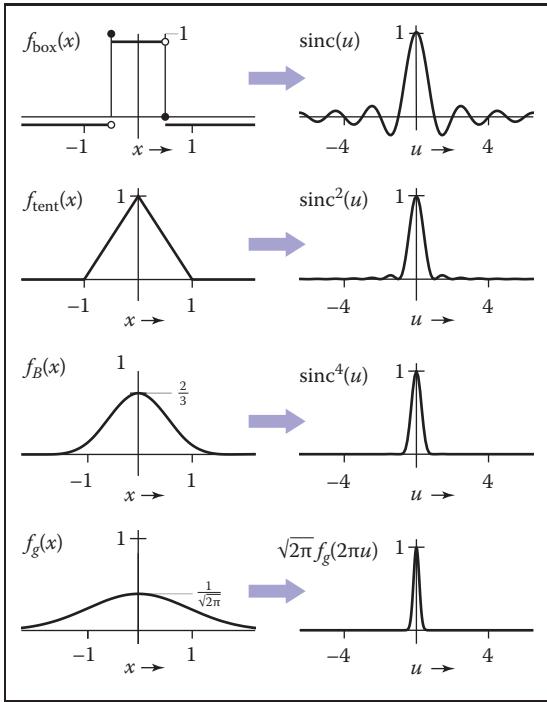


Figure 9.46. The Fourier transforms of the box, tent, B-spline, and Gaussian filters.

9.5.3 A Gallery of Fourier Transforms

Now that we have some facts about Fourier transforms, let's look at some examples of individual functions. In particular, we'll look at some filters from Section 9.3.1, which are shown with their Fourier transforms in Figure 9.46. We have already seen the box function:

$$\mathcal{F}\{f_{\text{box}}\} = \frac{\sin \pi u}{\pi u} = \text{sinc } \pi u.$$

The function³ $\sin x/x$ is important enough to have its own name, $\text{sinc } x$.

The tent function is the convolution of the box with itself, so its Fourier transform is just the square of the Fourier transform of the box function:

$$\mathcal{F}\{f_{\text{tent}}\} = \frac{\sin^2 \pi u}{\pi^2 u^2} = \text{sinc}^2 \pi u.$$

³You may notice that $\sin \pi u / \pi u$ is undefined for $u = 0$. It is, however, continuous across zero, and we take it as understood that we use the limiting value of this ratio, 1, at $u = 0$.

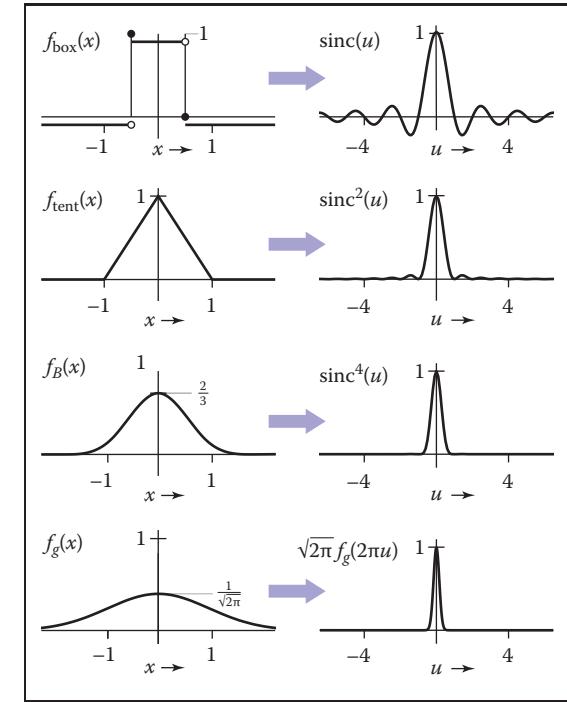


图9.46。箱、帐篷、b样条和高斯滤波器的傅立叶变换。

9.5.3 傅立叶变换图库

现在我们已经有了傅立叶变换的一些事实，让我们来看看个别函数的一些例子。特别是，我们将从第9.3.1节看到一些滤波器，它们在图9.46中用它们的傅立叶变换显示。我们已经看到了box功能：

$$\mathcal{F}\{f_{\text{box}}\} = \text{sinc } \pi u = \text{sinc } \pi u.$$

函数3sinxx足够重要，有它自己的名字，sincx。

箱函数是箱与自身的卷积，所以它的傅立叶反式形式只是箱函数的傅立叶变换的平方：

$$\mathcal{F}\{f_{\text{帐篷}}\} = \text{sinc}^2 \pi u = \text{sinc}^2 \pi u.$$

³您可能会注意到sinnnu对于u=0是未定义的。然而，它在零上是连续的，我们认为我们使用这个比率的限制值，1，在u=0。

We can continue this process to get the Fourier transform of the B-spline filter (see Exercise 3):

$$\mathcal{F}\{f_B\} = \frac{\sin^4 \pi u}{\pi^4 u^4} = \text{sinc}^4 \pi u.$$

The Gaussian has a particularly nice Fourier transform:

$$\mathcal{F}\{f_G\} = e^{-(2\pi u)^2/2}.$$

It is another Gaussian! The Gaussian with standard deviation 1.0 becomes a Gaussian with standard deviation $1/(2\pi)$.

9.5.4 Dirac Impulses in Sampling Theory

The reason impulses are useful in sampling theory is that we can use them to talk about samples in the context of continuous functions and Fourier transforms. We represent a sample, which has a position and a value, by an impulse translated to that position and scaled by that value. A sample at position a with value b is represented by $b\delta(x - a)$. This way we can express the operation of sampling the function $f(x)$ at a as multiplying f by $\delta(x - a)$. The result is $f(a)\delta(x - a)$.

Sampling a function at a series of equally spaced points is therefore expressed as multiplying the function by the sum of a series of equally spaced impulses, called an *impulse train* (Figure 9.47). An impulse train with period T , meaning that the impulses are spaced a distance T apart is

$$s_T(x) = \sum_{i=-\infty}^{\infty} \delta(x - Ti).$$

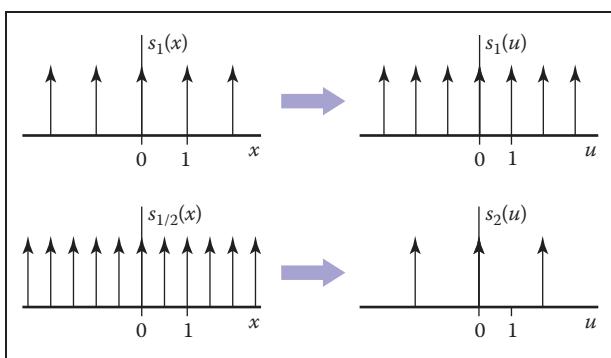


Figure 9.47. Impulse trains. The Fourier transform of an impulse train is another impulse train. Changing the period of the impulse train in space causes an inverse change in the period in frequency.

我们可以继续这个过程，得到B样条滤波器的傅立叶变换（见练习3）：

$$\mathcal{F}\{f_B\} = \frac{\sin^4 \pi u}{\pi^4 u^4} = \text{sinc}^4 \pi u.$$

高斯有一个特别好的傅立叶变换：

$$\mathcal{F}\{f_G\} = e^{-(2\pi u)^2/2}.$$

又是一个高斯！标准差1.0的高斯变为标准差 $1/(2\pi)$ 的高斯。

9.5.4 采样理论中的狄拉克脉冲

脉冲在采样理论中有用的原因是我们可以用它们来谈论连续函数和傅里叶变换背景下的样本。我们表示一个样本，它有一个位置和一个值，由一个脉冲转换到该位置并按该值缩放。具有值 b 的位置 a 处的样品由 $b\delta(x - a)$ 表示。这样我们就可以将在 a 处对函数 $f(x)$ 进行采样的操作表示为将 f 乘以 $\delta(x - a)$ 。结果是 $f(a)\delta(x - a)$ 。

因此，在一系列等间隔点处对函数进行采样表示为将函数乘以一系列等间隔脉冲的总和，称为脉冲列（图9.47）。一个周期为 T 的脉冲列，意味着脉冲间隔一段距离为 T

$$s_T(x) = \sum_{i=-\infty}^{\infty} \delta(x - Ti).$$

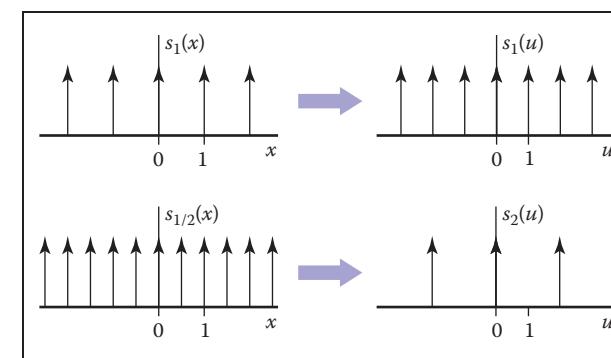


Figure 9.47. 冲动列车。脉冲列的傅立叶变换是另一个脉冲列。在空间中改变脉冲序列的周期会导致频率周期的反向变化。

The Fourier transform of s_1 is the same as s_1 : a sequence of impulses at all integer frequencies. You can see why this should be true by thinking about what happens when we multiply the impulse train by a sinusoid and integrate. We wind up adding up the values of the sinusoid at all the integers. This sum will exactly cancel to zero for non-integer frequencies, and it will diverge to $+\infty$ for integer frequencies.

Because of the dilation property of the Fourier transform, we can guess that the Fourier transform of an impulse train with period T (which is like a dilation of s_1) is an impulse train with period $1/T$. Making the sampling finer in the space domain makes the impulses farther apart in the frequency domain.

9.5.5 Sampling and Aliasing

Now that we have built the mathematical machinery, we need to understand the sampling and reconstruction process from the viewpoint of the frequency domain. The key advantage of introducing Fourier transforms is that it makes the effects of convolution filtering on the signal much clearer, and it provides more precise explanations of why we need to filter when sampling and reconstructing.

We start the process with the original, continuous signal. In general its Fourier transform could include components at any frequency, although for most kinds of signals (especially images), we expect the content to decrease as the frequency gets higher. Images also tend to have a large component at zero frequency—remember that the zero-frequency, or DC, component is the integral of the whole image, and since images are all positive values this tends to be a large number.

Let's see what happens to the Fourier transform if we sample and reconstruct without doing any special filtering (Figure 9.48). When we sample the signal, we model the operation as multiplication with an impulse train; the sampled signal is f_{sT} . Because of the multiplication-convolution property, the FT of the sampled signal is $\hat{f} * \hat{s}_T = \hat{f} * s_{1/T}$.

Recall that δ is the identity for convolution. This means that

$$(\hat{f} * s_{1/T})(u) = \sum_{i=-\infty}^{\infty} \hat{f}(u - i/T);$$

that is, convolving with the impulse train makes a whole series of equally spaced copies of the spectrum of f . A good intuitive interpretation of this seemingly odd result is that all those copies just express the fact (as we saw back in Section 9.1.1) that frequencies that differ by an integer multiple of the sampling frequency are indistinguishable once we have sampled—they will produce exactly the same set

s_1 的傅立叶变换与 s_1 相同：所有整数频率的脉冲序列。你可以通过思考当我们把脉冲列乘以正弦并积分时会发生什么来理解为什么这应该是真实的。我们最终将所有整数的正弦值相加。对于非整数频率，此总和将完全抵消为零，对于整数频率，它将发散为 $+\infty$ 。

由于傅立叶变换的膨胀性质，我们可以猜测周期为 T 的脉冲列的傅立叶变换（就像 s_1 的膨胀一样）是周期为 $1/T$ 的脉冲列。在空间域中使采样更精细使得脉冲在频域中相距更远。

9.5.5 采样和混叠

现在我们已经建立了数学机器，我们需要从频域的角度理解采样和重建过程。引入傅立叶变换的关键优势在于它使卷积滤波对信号的影响更加清晰，并且它提供了更精确的解释，说明为什么我们在采样和重建时需要滤波。

我们用原始的连续信号开始这个过程。一般来说，它的傅立叶变换可以包括任何频率的分量，尽管对于大多数类型的信号（特别是图像），我们预计内容会随着频率变高而减少。图像在零频率下也往往具有很大的分量—请记住，零频率或DC分量是整个图像的积分，并且由于图像都是正值，因此这往往是一个很大的数字。

让我们看看如果我们在不做任何特殊滤波的情况下采样和重建傅立叶变换会发生什么（图9.48）。当我们对信号进行采样时，我们将运算建模为与脉冲串的乘法；采样信号为 f_{sT} 。由于乘法-卷积属性，采样信号的FT为 f

$sT = f * s_{1/T}$ 。回想一下， δ 是卷积的身份。这意味着

$$(\hat{f} * s_{1/T})(u) = \sum_{i=-\infty}^{\infty} \hat{f}(u - i/T);$$

也就是说，与脉冲列车卷积使得 f 的频谱的整个系列等间隔的副本。对这个看似奇怪的结果的一个很好的直观解释是，所有这些副本都只是表达了一个事实（正如我们在第9.1.1节中看到的那样），即采样频率整数倍的频率

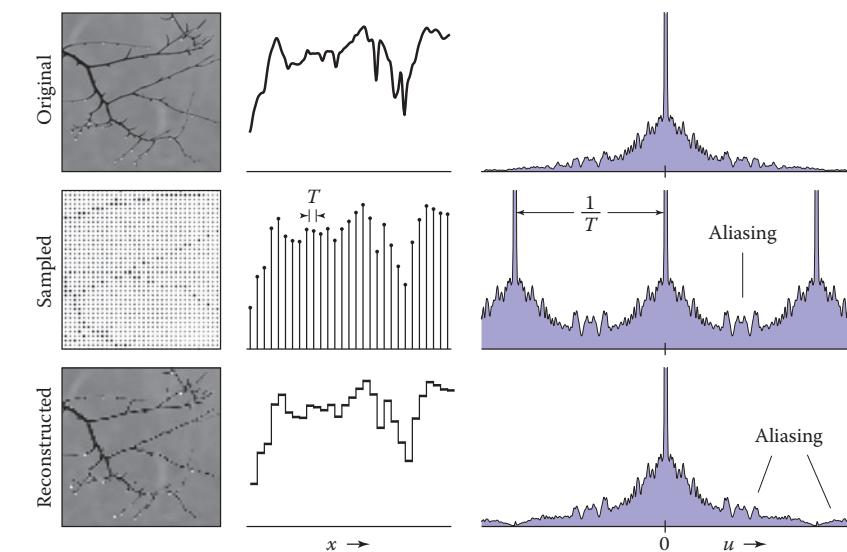
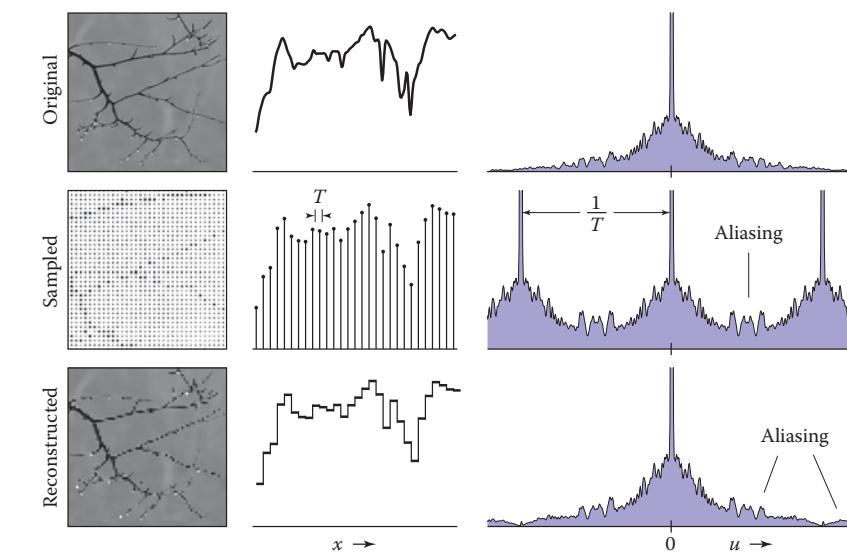


Figure 9.48. Sampling and reconstruction with no filtering. Sampling produces alias spectra that overlap and mix with the base spectrum. Reconstruction with a box filter collects even more information from the alias spectra. The result is a signal that has serious aliasing artifacts.

of samples. The original spectrum is called the *base spectrum* and the copies are known as *alias spectra*.

The trouble begins if these copies of the signal's spectrum overlap, which will happen if the signal contains any significant content beyond half the sample frequency. When this happens, the spectra add, and the information about different frequencies is irreversibly mixed up. This is the first place aliasing can occur, and if it happens here, it's due to undersampling—using too low a sample frequency for the signal.

Suppose we reconstruct the signal using the nearest-neighbor technique. This is equivalent to convolving with a box of width 1. (The discrete-continuous convolution used to do this is the same as a continuous convolution with the series of impulses that represent the samples.) The convolution-multiplication property means that the spectrum of the reconstructed signal will be the product of the spectrum of the sampled signal and the spectrum of the box. The resulting reconstructed Fourier transform contains the base spectrum (though somewhat attenuated at higher frequencies), plus attenuated copies of all the alias spectra. Because the box has a fairly broad Fourier transform, these attenuated bits of alias spectra are significant, and they are the second form of aliasing, due to an inadequate reconstruction filter. These alias components manifest themselves in the image as the pattern of squares that is characteristic of nearest-neighbor reconstruction.



没有滤波的采样和重建。采样产生与基础光谱重叠和混合的别名光谱。使用盒式滤波器重建可从别名光谱中收集更多信息。结果是具有严重混叠伪影的信号。

的样品。原始光谱称为基础光谱，副本称为别名光谱。

麻烦开始，如果这些副本的信号的频谱重叠，这将发生，如果信号包含任何显着的内容超过一半的样本频率。当这种情况发生时，频谱增加，关于不同频率的信息不可逆地混合起来。这是可能发生混叠的第一个地方，如果它发生在这里，这是由于欠采样—为信号使用太低的采样频率。

假设我们使用最近邻技术重建信号。这相当于用宽度为1的盒子进行卷积。（用于执行此操作的离散-连续卷积与代表样本的一系列脉冲的连续卷积相同。）卷积-乘法属性意味着重建信号的频谱将是采样信号的频谱与盒的频谱的乘积。所得到的侦察结构傅立叶变换包含基础频谱（虽然在较高频率有所衰减），加上所有别名频谱的衰减副本。因为盒具有相当宽的傅立叶变换，别名谱的这些衰减位是显着的，并且它们是混叠的第二种形式，由于不充分的重建滤波器。这些别名成分在图像中表现为正方形的模式，这是最近邻重建的特征。

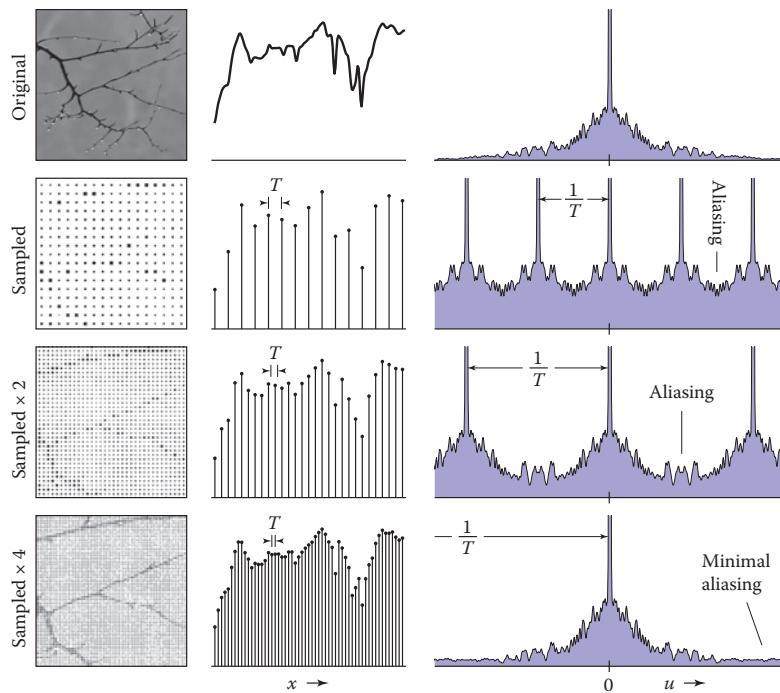
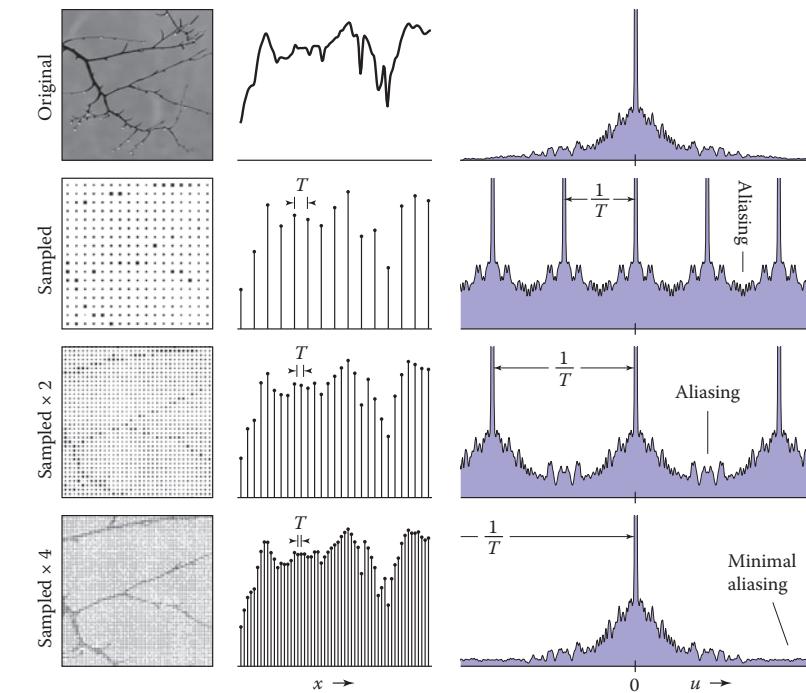


Figure 9.49. The effect of sample rate on the frequency spectrum of the sampled signal. Higher sample rates push the copies of the spectrum apart, reducing problems caused by overlap.

Preventing Aliasing in Sampling

To do high quality sampling and reconstruction, we have seen that we need to choose sampling and reconstruction filters appropriately. From the standpoint of the frequency domain, the purpose of lowpass filtering when sampling is to limit the frequency range of the signal so that the alias spectra do not overlap the base spectrum. Figure 9.49 shows the effect of sample rate on the Fourier transform of the sampled signal. Higher sample rates move the alias spectra farther apart, and eventually whatever overlap is left does not matter.

The key criterion is that the width of the spectrum must be less than the distance between the copies—that is, the highest frequency present in the signal must be less than half the sample frequency. This is known as the *Nyquist criterion*, and the highest allowable frequency is known as the *Nyquist frequency* or *Nyquist limit*. The *Nyquist-Shannon sampling theorem* states that a signal whose frequencies do not exceed the Nyquist limit (or, said another way, a signal that is bandlimited to the Nyquist frequency) can, in principle, be reconstructed exactly from samples.



采样率对采样信号频谱的影响。更高的采样率会将频谱副本分开，从而减少由重叠引起的问题。

防止采样中的混叠

要做高质量的采样和重建，我们已经看到需要适当地选择采样和重建滤波器。从频域的角度来看，采样时低通滤波的目的是限制信号的频率范围，使混叠谱不与基谱重叠。图9.49显示了采样率对采样信号傅立叶变换的影响。更高的采样速率将别名谱移动得更远，最终留下的任何重叠都无关紧要。

关键标准是频谱的宽度必须小于副本之间的distance—即信号中存在的最高频率必须小于样本频率的一半。这被称为奈奎斯特criterion，最高允许频率被称为奈奎斯特频率或奈奎斯特极限。奈奎斯特-香农采样定理指出，频率不超过奈奎斯特极限的信号（或者，换句话说，带限到奈奎斯特频率的信号）原则上可以完全从样本中重建。

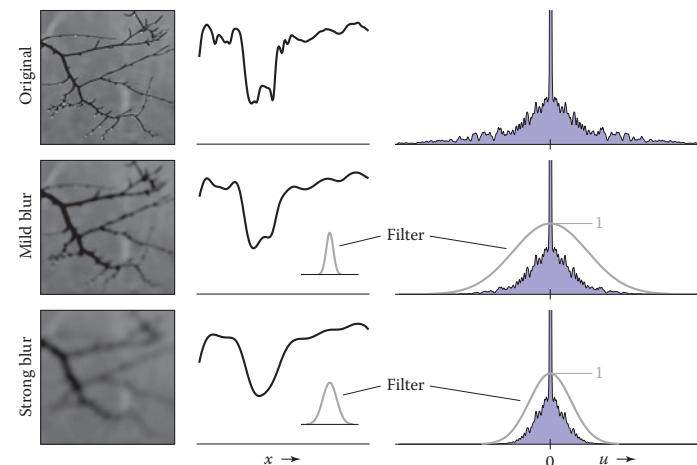


Figure 9.50. Applying lowpass (smoothing) filters narrows the frequency spectrum of a signal.

With a high enough sample rate for a particular signal, we don't need to use a sampling filter. But if we are stuck with a signal that contains a wide range of frequencies (such as an image with sharp edges in it), we must use a sampling filter to bandlimit the signal before we can sample it. Figure 9.50 shows the effects of three lowpass (smoothing) filters in the frequency domain, and Figure 9.51 shows the effect of using these same filters when sampling. Even if the spectra overlap without filtering, convolving the signal with a lowpass filter can narrow the spectrum enough to eliminate overlap and produce a well-sampled representation of the filtered signal. Of course, we have lost the high frequencies, but that's better than having them get scrambled with the signal and turn into artifacts.

Preventing Aliasing in Reconstruction

From the frequency domain perspective, the job of a reconstruction filter is to remove the alias spectra while preserving the base spectrum. In Figure 9.48, we can see that the crudest reconstruction filter, the box, does attenuate the alias spectra. Most important, it completely blocks the DC spike for all the alias spectra. This is a characteristic of all reasonable reconstruction filters: they have zeroes in frequency space at all multiples of the sample frequency. This turns out to be equivalent to the ripple-free property in the space domain.

So a good reconstruction filter needs to be a good lowpass filter, with the added requirement of completely blocking all multiples of the sample frequency.

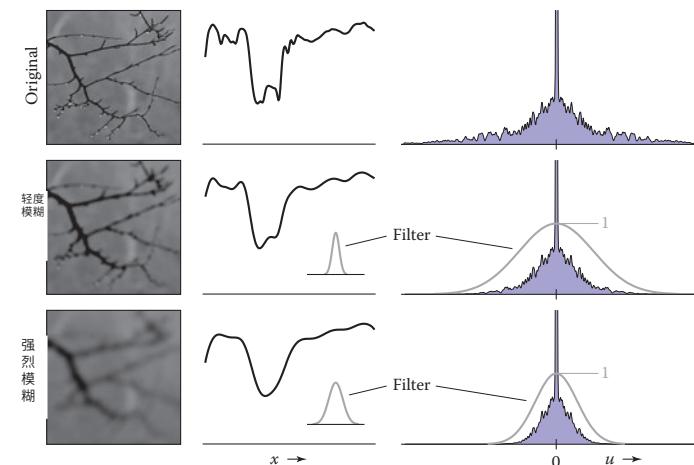


图9.50。应用低通（平滑）滤波器会缩小信号的频谱。

对于特定信号具有足够高的采样率，我们不需要使用采样滤波器。但是，如果我们被一个包含很宽频率范围的信号所困住（例如其中有尖锐边缘的图像），我们必须使用采样滤波器对信号进行带限，然后才能对其进行采样。图9.50显示了三个低通（平滑）滤波器在频域中的效果，图9.51显示了采样时使用这些相同滤波器的效果。即使频谱重叠而没有滤波，用低通滤波器对信号进行卷积也可以使频谱缩小到足以消除重叠并产生滤波信号的良好采样表示。当然，我们已经失去了高频率，但这比让它们被信号扰乱并变成伪像要好。

防止重建中的混叠

从频域的角度来看，重建滤波器的工作是在保留基础频谱的同时重新移动别名频谱。在图9.48中，我们可以看到最粗糙的重建滤波器，盒子，确实衰减了别名规范tra。最重要的是，它完全阻断了所有别名谱的直流尖峰。这是所有合理重建滤波器的一个特征：它们在采样频率的所有倍数处的频率空间中具有零。事实证明，这相当于空间域中的无纹波属性。

因此，一个好的重建滤波器需要是一个好的低通滤波器，并增加了完全阻塞采样频率的所有倍数的要求。

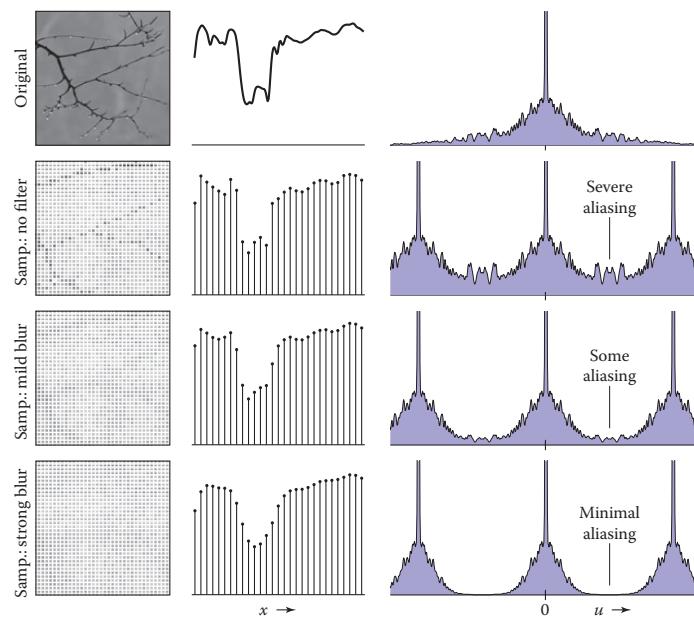


Figure 9.51. How the lowpass filters from Figure 9.50 prevent aliasing during sampling. Lowpass filtering narrows the spectrum so that the copies overlap less, and the high frequencies from the alias spectra interfere less with the base spectrum.

The purpose of using a reconstruction filter different from the box filter is to more completely eliminate the alias spectra, reducing the leakage of high-frequency artifacts into the reconstructed signal, while disturbing the base spectrum as little as possible. Figure 9.52 illustrates the effects of different filters when used during reconstruction. As we have seen, the box filter is quite “leaky” and results in plenty of artifacts even if the sample rate is high enough. The tent filter, resulting in linear interpolation, attenuates high frequencies more, resulting in milder artifacts, and the B-spline filter is very smooth, controlling the alias spectra very effectively. It also smooths the base spectrum some—this is the tradeoff between smoothing and aliasing that we saw earlier.

Preventing Aliasing in Resampling

When the operations of reconstruction and sampling are combined in resampling, the same principles apply, but with one filter doing the work of both reconstruction and sampling. Figure 9.53 illustrates how a resampling filter must remove the alias spectra *and* leave the spectrum narrow enough to be sampled at the new sample rate.

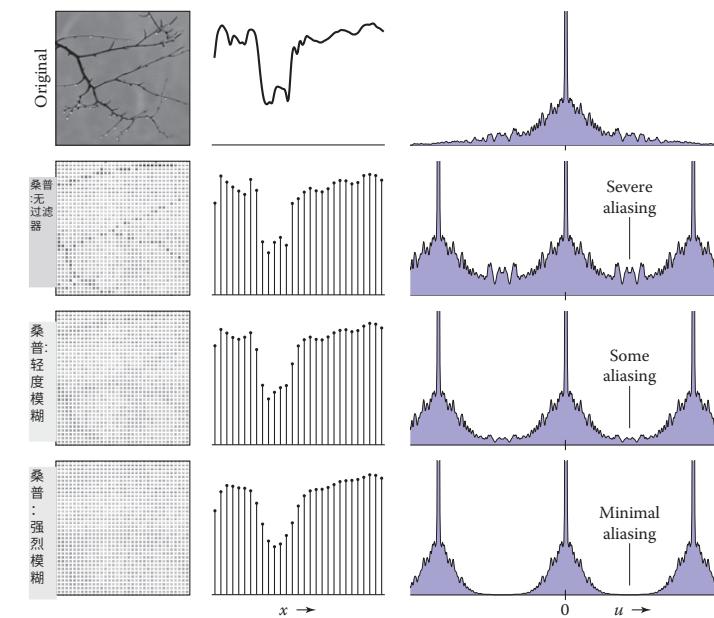


图9.50中的低通滤波器如何防止采样过程中的混叠。低通滤波使频谱变窄，使得副本重叠较少，并且来自别名频谱的高频对基础频谱的干扰较小。

使用不同于盒滤波器的重构滤波器的目的是更完全地消除混叠谱，减少高仿像到重构信号中的泄漏，同时尽可能少地干扰基谱。图9.52说明了在重建过程中使用不同滤波器时的效果。正如我们所看到的，盒式过滤器相当“泄漏”，即使采样率足够高，也会导致大量仿像。Tent滤波器，导致线性插值，更多地衰减高频，导致更温和的仿像，并且B样条滤波器非常平滑，非常有效地控制别名谱。它还平滑了基本频谱—这是我们之前看到的平滑和混叠之间的权衡。

防止重采样中的混叠

当重建和采样的操作在重采样中结合时，同样的原则也适用，但是一个滤波器同时完成重建和采样的工作。图9.53说明了重采样滤波器如何去除别名谱，并使谱足够窄，以便以新的采样率进行采样。

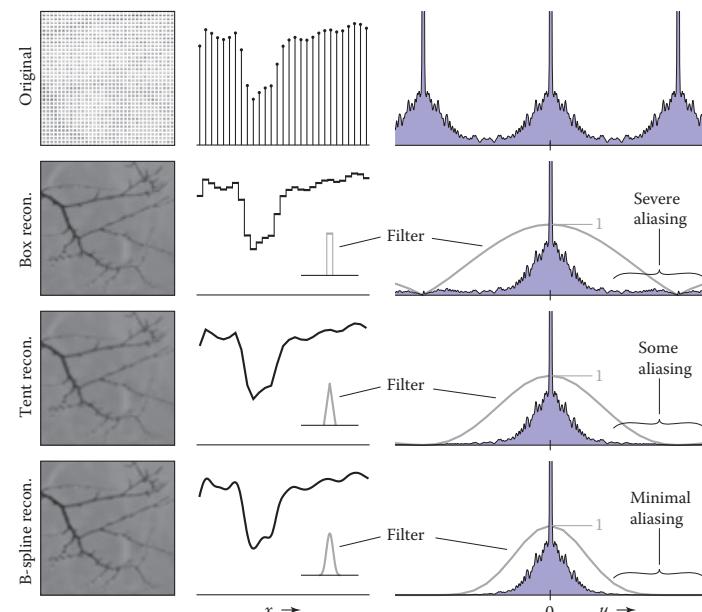


Figure 9.52. The effects of different reconstruction filters in the frequency domain. A good reconstruction filter attenuates the alias spectra effectively while preserving the base spectrum.

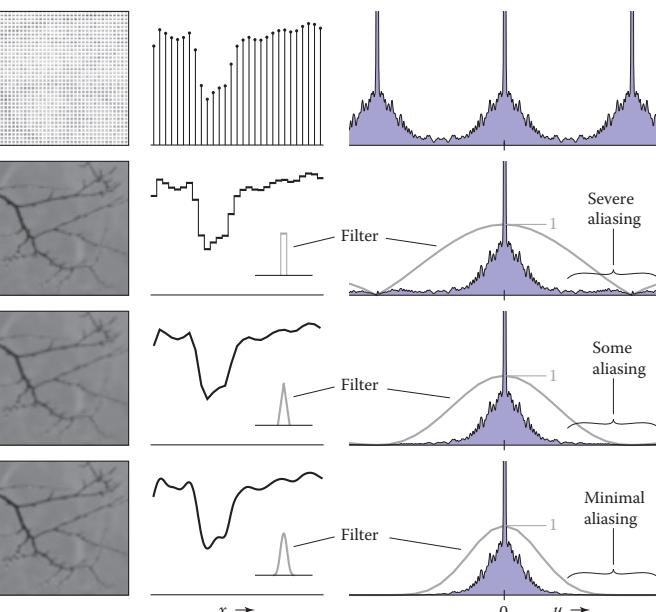
9.5.6 Ideal Filters vs. Useful Filters

Following the frequency domain analysis to its logical conclusion, a filter that is exactly a box in the frequency domain is ideal for both sampling and reconstruction. Such a filter would prevent aliasing at both stages without diminishing the frequencies below the Nyquist frequency at all.

Recall that the inverse and forward Fourier transforms are essentially identical, so the spatial domain filter that has a box as its Fourier transform is the function $\sin \pi x / \pi x = \text{sinc } \pi x$.

However, the sinc filter is not generally used in practice, either for sampling or for reconstruction, because it is impractical and because, even though it is optimal according to the frequency domain criteria, it doesn't produce the best results for many applications.

For sampling, the infinite extent of the sinc filter, and its relatively slow rate of decrease with distance from the center, is a liability. Also, for some kinds of sampling, the negative lobes are problematic. A Gaussian filter makes an excellent sampling filter even for difficult cases where high-frequency patterns must be removed from the input signal, because its Fourier transform falls off exponentially.



频域中不同重建滤波器的影响。一个好的重建滤波器在保留基谱的同时有效地衰减别名谱。

9.5.6 理想的过滤器与有用的过滤器

根据频域分析得出的逻辑结论，一个在频域中完全是一个盒子的滤波器对于采样和重构都是理想的。这样的滤波器可以防止两级的混叠，而不会使频率降低到奈奎斯特频率以下。

回想一下，逆傅立叶变换和正向傅立叶变换本质上是identical，因此具有框作为其傅立叶变换的空间域滤波器是函数 $\text{sinn}nx=\text{sinc}nx$ 。

然而，sinc滤波器在实践中通常不被使用，无论是用于采样还是用于重建，因为它是不切实际的，并且因为，即使根据频域准则它是最优的，但它对于许多应用

对于采样来说，sinc滤波器的无限程度以及其相对较慢的随距中心距离的减小速率是一种负担。此外，对于某些类型的采样，负瓣是有问题的。高斯滤波器使得即使对于必须从输入信号中去除高频图案的困难情况，也可以使用excel采样滤波器，因为它的傅立叶变换从指数上脱落

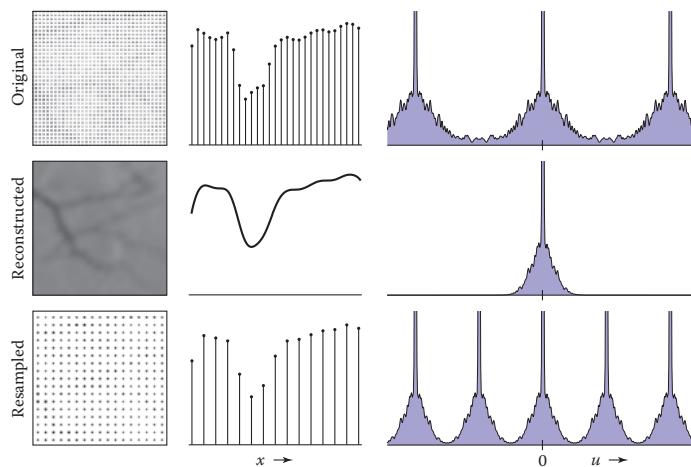


Figure 9.53. Resampling viewed in the frequency domain. The resampling filter both reconstructs the signal (removes the alias spectra) and bandlimits it (reduces its width) for sampling at the new rate.

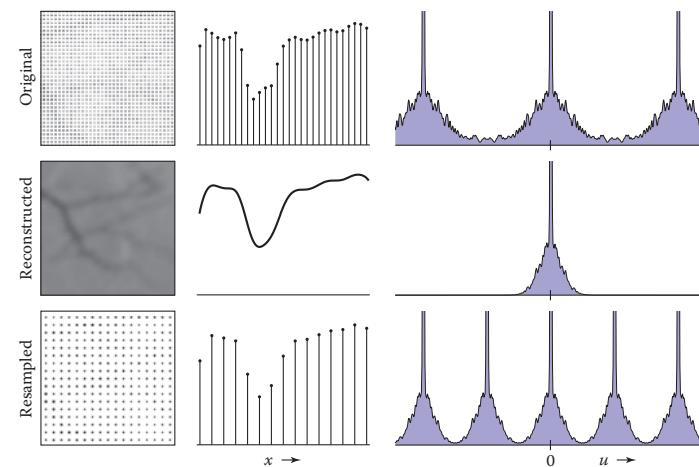
tially, with no bumps that tend to let aliases leak through. For less difficult cases, a tent filter generally suffices.

For reconstruction, the size of the sinc function again creates problems, but even more importantly, the many ripples create “ringing” artifacts in reconstructed signals.

Exercises

1. Show that discrete convolution is commutative and associative. Do the same for continuous convolution.
2. Discrete-continuous convolution can't be commutative, because its arguments have two different types. Show that it is associative, though.
3. Prove that the B-spline is the convolution of four box functions.
4. Show that the “flipped” definition of convolution is necessary by trying to show that convolution is commutative and associative using this (incorrect) definition (see the footnote on page 192):

$$(a \star b)[i] = \sum_j a[j]b[i + j]$$



频域中查看的重采样。重采样滤波器既重建信号（去除别名谱），又对其进行带限（减小其宽度），以便以新速率进行采样。

最重要的是，没有容易让别名泄漏的凸起。对于不太困难的情况，帐篷过滤器通常就足够了。

对于重建，sinc函数的大小再次产生问题，但更重要的是，许多波纹在重建信号中产生“振铃”伪影。

Exercises

1. 表明离散卷积是交换和关联的。对连续卷积做同样的事情。
2. 离散-连续卷积不能是交换的，因为它的参数有两种不同的类型。显示它是联想，虽然。
3. 证明B样条是四个盒函数的卷积。
4. 显示卷积的“翻转”定义是必要的，通过尝试使用这个（不正确的）定义来显示卷积是交换和关联的（请参阅第192页的脚注）：

$$(a \star b)[i] = \sum_j a[j]b[i + j]$$

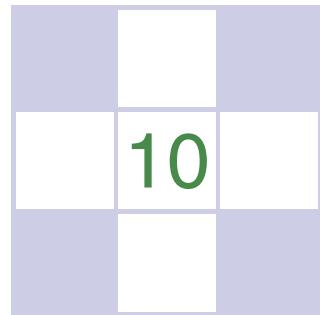


5. Prove that $\mathcal{F}\{f * g\} = \hat{f}\hat{g}$ and $\hat{f} * \hat{g} = \mathcal{F}\{fg\}$.

6. Equation 9.4 can be interpreted as the convolution of a with a filter \bar{f} . Write a mathematical expression for the “de-rippled” filter \bar{f} . Plot the filter that results from de-rippling the box, tent, and B-spline filters scaled to $s = 1.25$.

5.证明 $\mathcal{F}\{f * g\} = \hat{f}\hat{g}$ 和 $f * g = \mathcal{F}\{fg\}$ 。

6.公式9.4可以解释为a与滤波器f的卷积。为“去波纹”滤波器f写一个数学表达式。绘制对缩放为s=1的box、tent和B样条滤波器进行去波纹处理所产生的滤波器。25.

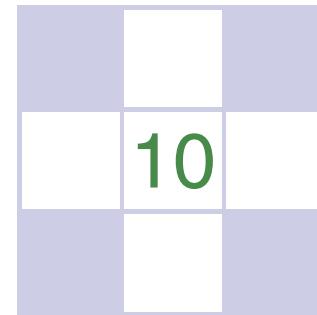


Surface Shading

To make objects appear to have more volume, it can help to use *shading*, i.e., the surface is “painted” with light. This chapter presents the most common heuristic shading methods. The first two, diffuse and Phong shading, were developed in the 1970s and are available in most graphics libraries. The last, artistic shading, uses artistic conventions to assign color to objects. This creates images reminiscent of technical drawings, which is desirable in many applications.

10.1 Diffuse Shading

Many objects in the world have a surface appearance loosely described as “matte,” indicating that the object is not at all shiny. Examples include paper, unfinished wood, and dry, unpolished stones. To a large degree, such objects do not have a color change with a change in viewpoint. For example, if you stare at a particular point on a piece of paper and move while keeping your gaze fixed on that point, the color at that point will stay relatively constant. Such matte objects can be considered as behaving as *Lambertian* objects. This section discusses how to implement the shading of such objects. A key point is that all formulas in this chapter should be evaluated in world coordinates and not in the warped coordinates after the perspective transform is applied. Otherwise, the angles between normals are changed and the shading will be inaccurate.



表面阴影

为了使物体看起来有更多的体积，它可以帮助使用阴影，即表面被“涂”光。本章介绍了最常见的启发式着色方法。前两个，漫射和Phong阴影，是在20世纪70年代开发的，在大多数图形库中可用。最后一个，艺术阴影，使用艺术约定为对象分配颜色。这会创建让人想起技术图纸的图像，这在许多应用中是可取的。

10.1 漫射阴影

世界上许多物体的表面外观松散地描述为“哑光”，表明物体根本没有光泽。例子包括纸张，未完成的木材和干燥，未抛光的石头。在很大程度上，这样的对象不具有随视点变化的颜色变化。例如，如果您盯着一张纸上的特定点并移动，同时将目光固定在该点上，则该点的颜色将保持相对恒定。这样的哑光物体可以被认为表现为朗伯物体。本节讨论如何实现此类对象的着色。一个关键点是，在应用透视变换后，本章中的所有公式都应该在世界坐标中进行评估，而不是在扭曲的坐标中进行评估。否则，法线之间的角度会发生变化，阴影将不准确。

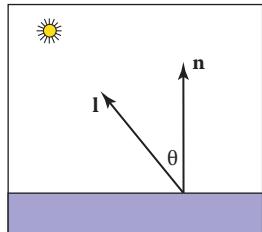


Figure 10.1. The geometry for Lambert's law. Both \mathbf{n} and \mathbf{l} are unit vectors.

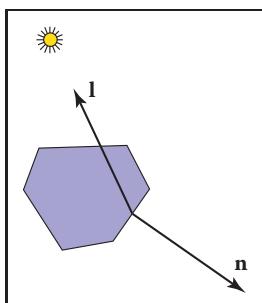


Figure 10.2. When a surface points away from the light, it should receive no light. This case can be verified by checking whether the dot product of \mathbf{l} and \mathbf{n} is negative.

$$c \propto c_r \mathbf{n} \cdot \mathbf{l}. \quad (10.1)$$

The right-hand side of Equation (10.1) is an RGB color with all RGB components in the range $[0, 1]$. We would like to add the effects of light intensity while keeping the RGB components in the range $[0, 1]$. This suggests adding an RGB intensity term c_l which itself has components in the range $[0, 1]$:

$$c = c_r c_l \mathbf{n} \cdot \mathbf{l}. \quad (10.2)$$

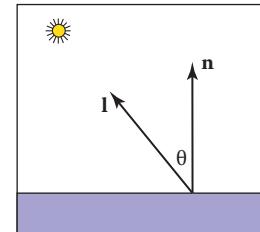
This is a very convenient form, but it can produce RGB components for c that are outside the range $[0, 1]$, because the dot product can be negative. The dot product is negative when the surface is pointing away from the light as shown in Figure 10.2.

The “max” function can be added to Equation (10.2) to test for that case:

$$c = c_r c_l \max(0, \mathbf{n} \cdot \mathbf{l}). \quad (10.3)$$

Another way to deal with the “negative” light is to use an absolute value:

$$c = c_r c_l |\mathbf{n} \cdot \mathbf{l}|. \quad (10.4)$$



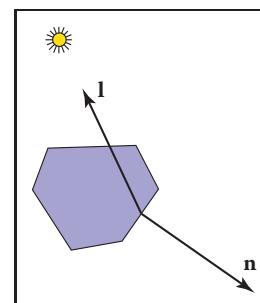
兰伯特定律的几何。N和l都是单位向量。

$$c \propto \cos \theta,$$

或矢量形式

$$c \propto \mathbf{n} \cdot \mathbf{l},$$

其中n和l如图10.1所示。因此，表面上的颜色将根据表面法线与光方向之间的角度的余弦而变化。注意，向量l典型地假定不依赖于对象的位置。这个假设相当于假设光线相对于物体大小“遥远”。这样的“远处”光通常被称为定向光，因为其位置仅由一个方向指定。



当一个表面指向远离光线时，它应该不接收光线。这种情况可以通过检查l和n的点积是否为negative来验证。

$$c \propto c_r \mathbf{n} \cdot \mathbf{l}. \quad (10.1)$$

等式 (10.1) 的右侧是RGB颜色，所有RGB分量都在 $[0, 1]$ 范围内。我们希望添加光强度的效果，同时将RGB分量保持在 $[0, 1]$ 范围内。这建议添加RGB强度项 c_l ，其本身具有范围 $[0, 1]$ 中的分量：

$$c = c_r c_l \mathbf{n} \cdot \mathbf{l}. \quad (10.2)$$

这是一种非常方便的形式，但它可以为 c 产生范围 $[0, 1]$ 之外的RGB分量，因为点积可以是负数。如图10.2所示，当表面指向远离光线时，点积为负数。

可以将“max”函数添加到公式 (10.2) 中以测试该情况：

$$c = c_r c_l \max(0, \mathbf{n} \cdot \mathbf{l}). \quad (10.3)$$

处理“负”光的另一种方法是使用绝对值：

$$c = c_r c_l |\mathbf{n} \cdot \mathbf{l}|. \quad (10.4)$$

While Equation (10.4) may seem physically implausible, it actually corresponds to Equation (10.3) with two lights in opposite directions. For this reason it is often called *two-sided* lighting (Figure 10.3).

10.1.2 Ambient Shading

One problem with the diffuse shading of Equation (10.3) is that any point whose normal faces away from the light will be black. In real life, light is reflected all over, and some light is incident from every direction. In addition, there is often skylight giving “ambient” lighting. One way to handle this is to use several light sources. A common trick is to always put a dim source at the eye so that all visible points will receive some light. Another way is to use two-sided lighting as described by Equation (10.4). A more common approach is to add an ambient term (Gouraud, 1971). This is just a constant color term added to Equation (10.3):

$$c = c_r (c_a + c_l \max(0, \mathbf{n} \cdot \mathbf{l})) .$$

Intuitively, you can think of the ambient color c_a as the average color of all surfaces in the scene. If you want to ensure that the computed RGB color stays in the range $[0, 1]^3$, then $c_a + c_l \leq (1, 1, 1)$. Otherwise your code should “clamp” RGB values above one to have the value one.

10.1.3 Vertex-Based Diffuse Shading

If we apply Equation (10.1) to an object made up of triangles, it will typically have a faceted appearance. Often, the triangles are an approximation to a smooth surface. To avoid the faceted appearance, we can place surface normal vectors at the vertices of the triangles (Phong, 1975), and apply Equation (10.3) at each of the vertices using the normal vectors at the vertices (see Figure 10.4). This will give a color at each triangle vertex, and this color can be interpolated using the barycentric interpolation described in Section 8.1.2.

One problem with shading at triangle vertices is that we need to get the normals from somewhere. Many models will come with normals supplied. If you tessellate your own smooth model, you can create normals when you create the triangles. If you are presented with a polygonal model that does not have normals at vertices and you want to shade it smoothly, you can compute normals by a variety of heuristic methods. The simplest is to just average the normals of the triangles that share each vertex and use this average normal at the vertex. This average normal will not automatically be of unit length, so you should convert it to a unit vector before using it for shading.

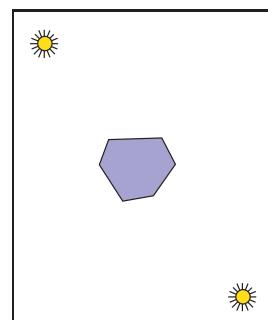


Figure 10.3. Using Equation (10.4), the two-sided lighting formula, is equivalent to assuming two opposing light sources of the same color.

10.1.2 环境阴影

方程 (10.3) 的漫射阴影的一个问题是，任何法线朝向远离光线的点都将是黑色的。在现实生活中，光线被反射到各处，有些光线从各个方面入射。此外，经常有天窗给予“环境”照明。处理这种情况的一种方法是使用多个光源。一个常见的技巧是总是在眼睛上放置一个昏暗的光源，以便所有可见点都会收到一些光线。另一种方式是如等式(10.4)所述使用双面照明。更常见的方法是添加环境术语 (Gouraud, 1971)。这只是添加到等式 (10.3) 中的恒定颜色项：

$$c = c_r (c_a + c_l \max(0, \mathbf{n} \cdot \mathbf{l})) .$$

直观地，您可以将环境颜色 c_a 视为场景中所有表面的平均颜色。如果要确保计算的RGB颜色保持在 $[0, 1]^3$ 范围内，则 $c_a + c_l \leq (1, 1, 1)$ 。否则，您的代码应该将RGB值“钳位”在一个以上以具有值一。

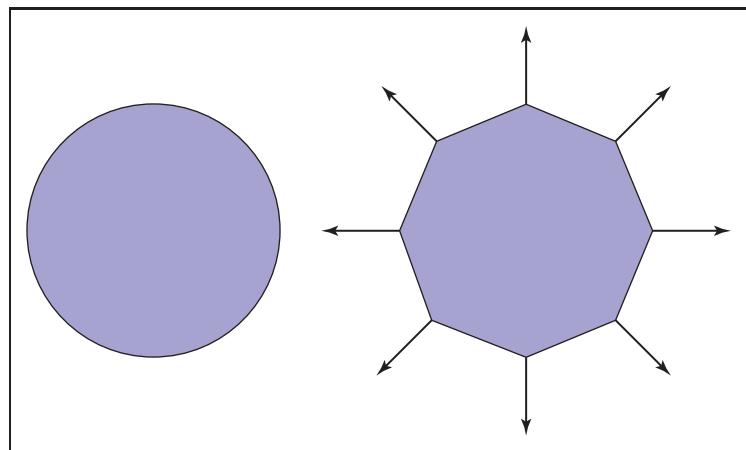


Figure 10.4. A circle (left) is approximated by an octagon (right). Vertex normals record the surface normal of the original curve.

10.2 Phong Shading

Some surfaces are essentially like matte surfaces, but they have *highlights*. Examples of such surfaces include polished tile floors, gloss paint, and whiteboards. Highlights move across a surface as the viewpoint moves. This means that we must add a unit vector e toward the eye into our equations. If you look carefully at highlights, you will see that they are really reflections of the light; sometimes these reflections are blurred. The color of these highlights is the color of the light—the surface color seems to have little effect. This is because the reflection occurs at the object’s surface, and the light that penetrates the surface and picks up the object’s color is scattered diffusely.

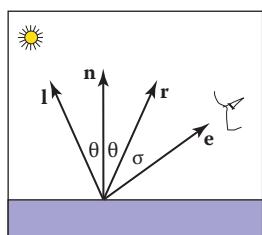


Figure 10.5. The geometry for the Phong illumination model. The eye should see a highlight if σ is small.

10.2.1 Phong Lighting Model

We want to add a fuzzy “spot” the same color as the light source in the right place. The center of the dot should be drawn where the direction e to the eye “lines” up with the natural direction of reflection r as shown in Figure 10.5. Here “lines up” is mathematically equivalent to “where σ is zero.” We would like to have the highlight have some nonzero area, so that the eye sees some highlight wherever σ is small.

Given r , we’d like a heuristic function that is bright when $e = r$ and falls off gradually when e moves away from r . An obvious candidate is the cosine of the

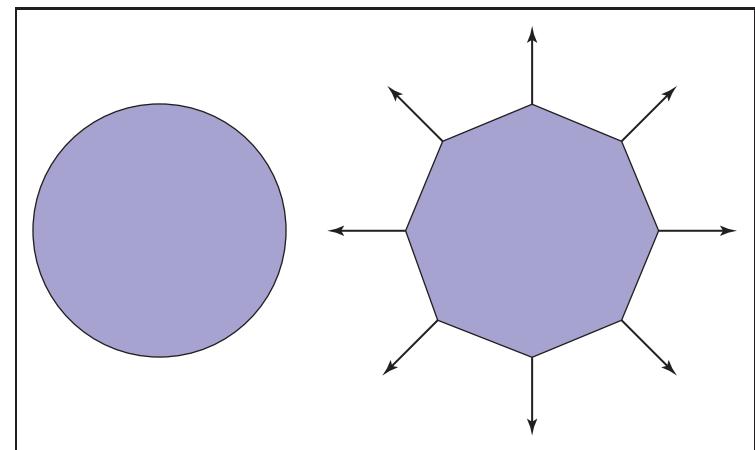
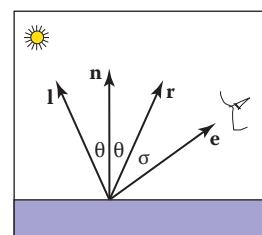


图10.4。圆（左）由八边形（右）近似。顶点法线记录原始曲线的表面法线。

10.2 Phong Shading

有些表面上类似于哑光表面，但它们具有高光。这种表面的实例包括抛光瓷砖地板、光泽漆和白板。高光在视点移动时在表面上移动。这意味着我们必须在方程中添加一个朝向眼睛的单位向量 e 。如果你仔细观察高光，你会发现它们确实是光线的反射；有时这些反射是模糊的。这些亮点的颜色是光线的颜色—表面颜色似乎几乎没有影响。这是因为反射发生在物体表面，穿透表面并拾取物体颜色的光被扩散地散射。



Geome尝试Phong照明模型。
如果 σ 很小眼睛应该看到一个亮点。

10.2.1 Phong照明模型

我们想在正确的地方添加一个与光源相同颜色的模糊“斑点”。点的中心应该画在眼睛的方向 e 与自然反射方向 r “线”的地方，如图10.5所示。这里“lines up”在数学上等同于“其中 σ 为零。”我们希望高光具有一些非零区域，以便眼睛在 σ 很小的地方看到一些高光。

给定 r ，我们想要一个启发式函数，当 $e=r$ 时是明亮的，当 e 远离 r 时逐渐脱落。一个明显的候选人是余弦的



angle between them:

$$c = c_l(\mathbf{e} \cdot \mathbf{r}).$$

There are two problems with using this equation. The first is that the dot product can be negative. This can be solved computationally with an “if” statement that sets the color to zero when the dot product is negative. The more serious problem is that the highlight produced by this equation is much wider than that seen in real life. The maximum is in the right place and it is the right color, but it is just too big. We can narrow it without reducing its maximum color by raising to a power:

$$c = c_l \max(0, \mathbf{e} \cdot \mathbf{r})^p. \quad (10.5)$$

Here p is called the *Phong exponent*; it is a positive real number (Phong, 1975). The effect that changing the Phong exponent has on the highlight can be seen in Figure 10.6.

To implement Equation (10.5), we first need to compute the unit vector \mathbf{r} . Given unit vectors \mathbf{l} and \mathbf{n} , \mathbf{r} is the vector \mathbf{l} reflected about \mathbf{n} . Figure 10.7 shows

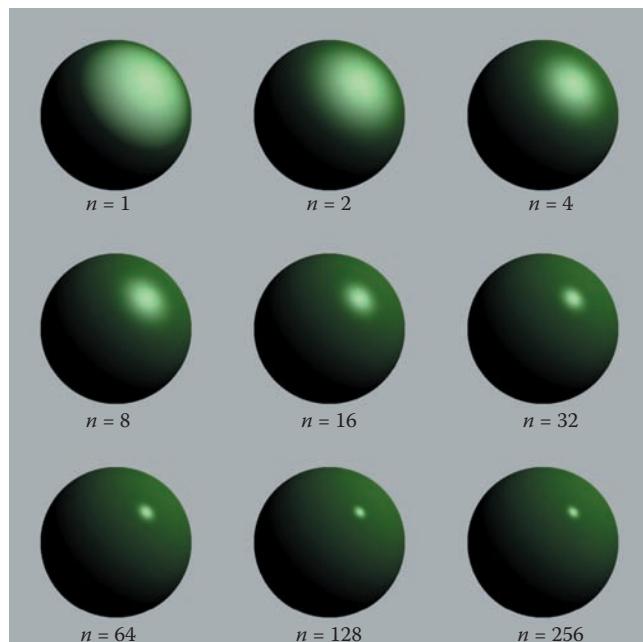


Figure 10.6. The effect of the Phong exponent on highlight characteristics. This uses Equation (10.5) for the highlight. There is also a diffuse component, giving the objects a shiny but non-metallic appearance. *Image courtesy Nate Robins.*



之间的角度:

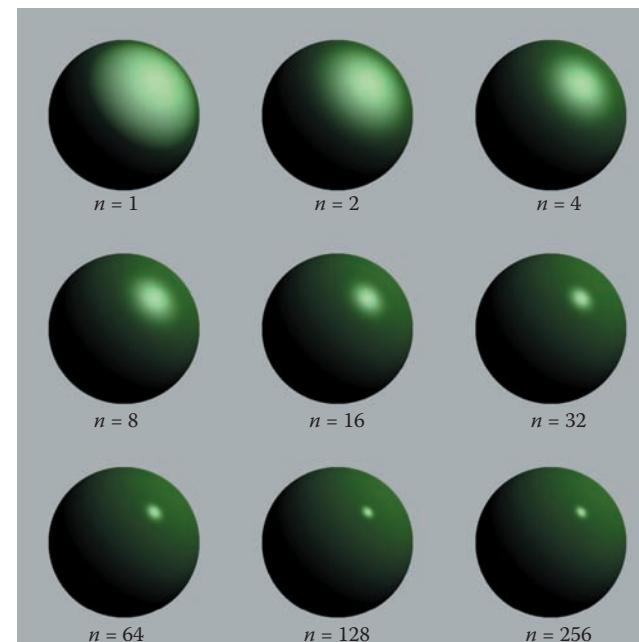
$$c = c_l(\mathbf{e} \cdot \mathbf{r}).$$

使用这个方程有两个问题。首先是点积可以是负数。这可以通过“if”语句在点积为负时将颜色设置为零来计算解决。更严重的问题是，这个等式产生的亮点比现实生活中看到的要广泛得多。最大值在正确的地方，它是正确的颜色，但它太大了。我们可以通过提高到幂来缩小它而不降低它的最大颜色：

$$c = c_l \max(0, \mathbf{e} \cdot \mathbf{r})^p. \quad (10.5)$$

这里p被称为Phong指数;它是一个正实数 (Phong, 1975)。改变Phong指数对高光的影响可以在 Figure 10.6.

为了实现方程 (10.5) , 我们首先需要计算单位向量r。给定单位向量l和n, r是关于n反映的向量l。图10.7显示



Phong指数对高光特征的影响。这使用Equation(10.5)作为亮点。还有一个漫反射组件，使物体具有光泽但非金属的外观。图片由内特·罗宾斯提供。

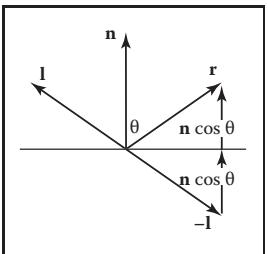


Figure 10.7. The geometry for calculating the vector \mathbf{r} .

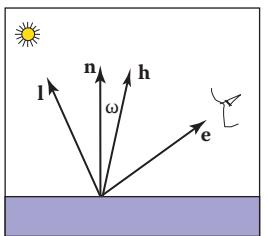


Figure 10.8. The unit vector \mathbf{h} is halfway between \mathbf{l} and \mathbf{e} .

that this vector can be computed as

$$\mathbf{r} = -\mathbf{l} + 2(\mathbf{l} \cdot \mathbf{n})\mathbf{n}, \quad (10.6)$$

where the dot product is used to compute $\cos \theta$.

An alternative heuristic model based on Equation (10.5) eliminates the need to check for negative values of the number used as a base for exponentiation (Warn, 1983). Instead of \mathbf{r} , we compute \mathbf{h} , the unit vector halfway between \mathbf{l} and \mathbf{e} (Figure 10.8):

$$\mathbf{h} = \frac{\mathbf{e} + \mathbf{l}}{\|\mathbf{e} + \mathbf{l}\|}.$$

The highlight occurs when \mathbf{h} is near \mathbf{n} , i.e., when $\cos \omega = \mathbf{h} \cdot \mathbf{n}$ is near 1. This suggests the rule:

$$c = c_l(\mathbf{h} \cdot \mathbf{n})^p. \quad (10.7)$$

The exponent p here will have analogous control behavior to the exponent in Equation (10.5), but the angle between \mathbf{h} and \mathbf{n} is half the size of the angle between \mathbf{e} and \mathbf{r} , so the details will be slightly different. The advantage of using the cosine between \mathbf{n} and \mathbf{h} is that it is always positive for eye and light above the plane. The disadvantage is that a square root and divide is needed to compute \mathbf{h} .

In practice, we want most materials to have a diffuse appearance in addition to a highlight. We can combine Equations (10.3) and (10.7) to get

$$c = c_r(c_a + c_l \max(0, \mathbf{n} \cdot \mathbf{l})) + c_l(\mathbf{h} \cdot \mathbf{n})^p. \quad (10.8)$$

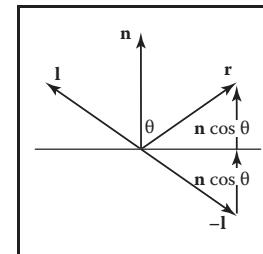
If we want to allow the user to dim the highlight, we can add a control term c_p :

$$c = c_r(c_a + c_l \max(0, \mathbf{n} \cdot \mathbf{l})) + c_l c_p (\mathbf{h} \cdot \mathbf{n})^p. \quad (10.9)$$

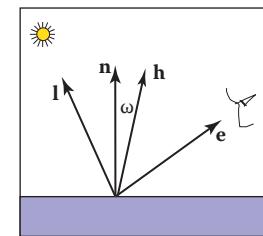
The term c_p is a RGB color, which allows us to change highlight colors. This is useful for metals where $c_p = c_r$, because highlights on metal take on a metallic color. In addition, it is often useful to make c_p a neutral value less than one, so that colors stay below one. For example, setting $c_p = 1 - M$ where M is the maximum component of c_r will keep colors below one for one light source and no ambient term.

10.2.2 Surface Normal Vector Interpolation

Smooth surfaces with highlights tend to change color quickly compared to Lambertian surfaces with the same geometry. Thus, shading at the normal vectors can generate disturbing artifacts.



用于计算矢量r的几何形状。



单位矢量h介于l和e之间。

这个向量可以计算为

$$\mathbf{r} = -\mathbf{l} + 2(\mathbf{l} \cdot \mathbf{n})\mathbf{n}, \quad (10.6)$$

其中点积用于计算 $\cos\theta$ 。

基于等式 (10.5) 的替代启发式模型消除了检查用作指数基的数字的负值的需要 (Warn, 1983)。而不是 \mathbf{r} , 我们计算 \mathbf{h} , \mathbf{l} 和 \mathbf{e} 之间的单位矢量 (图 10.8) :

$$\mathbf{h} = \frac{\mathbf{e} + \mathbf{l}}{\|\mathbf{e} + \mathbf{l}\|}.$$

当 \mathbf{h} 在 \mathbf{n} 附近时, 即当 $\cos\omega=\mathbf{h} \cdot \mathbf{n}$ 在1附近时, 高亮发生。这建议规则:

$$c = c_l(\mathbf{h} \cdot \mathbf{n})^p. \quad (10.7)$$

这里的指数 p 将具有类似于方程 (10.5) 中的指数的控制行为, 但 \mathbf{h} 和 \mathbf{n} 之间的角度是 \mathbf{e} 和 \mathbf{r} 之间角度大小的一半, 因此细节会略有不同。使用 \mathbf{n} 和 \mathbf{h} 之间的余弦的优点是它对于平面上方的眼睛和光线总是积极的。缺点是需要平方根和除法来计算 \mathbf{h} 。实际上, 我们希望大多数材质除了高光外, 还具有漫反射外观。我们可以结合方程 (10.3) 和 (10.7) 得到

$$c = c_r(c_a + c_l \max(0, \mathbf{n} \cdot \mathbf{l})) + c_l(\mathbf{h} \cdot \mathbf{n})^p. \quad (10.8)$$

如果我们想让用户调暗高光, 我们可以添加一个控制项 c_p :

$$c = c_r(c_a + c_l \max(0, \mathbf{n} \cdot \mathbf{l})) + c_l c_p (\mathbf{h} \cdot \mathbf{n})^p. \quad (10.9)$$

术语 c_p 是RGB颜色, 它允许我们更改高亮颜色。这对于 $c_p=c_r$ 的金属很有用, 因为金属上的高光呈现金属色。此外, 使 c_p 的中性值小于1通常是有用的, 以便颜色保持在1以下。例如, 设置 $c_p=1-M$, 其中 M 是 c_r 的最大分量, 将使一个光源的颜色保持在一个以下, 并且没有环境项。

10.2.2 表面法向矢量插值

与具有相同几何形状的朗伯表面相比, 具有高光的光滑表面往往会快速改变颜色。因此, 法向量处的阴影会产生干扰伪像。

These problems can be reduced by interpolating the normal vectors across the polygon and then applying Phong shading at each pixel. This allows you to get good images without making the size of the triangles extremely small. Recall from Chapter 3, that when rasterizing a triangle, we compute barycentric coordinates (α, β, γ) to interpolate the vertex colors c_0, c_1, c_2 :

$$c = \alpha c_0 + \beta c_1 + \gamma c_2. \quad (10.10)$$

We can use the same equation to interpolate surface normals $\mathbf{n}_0, \mathbf{n}_1$, and \mathbf{n}_2 :

$$\mathbf{n} = \alpha \mathbf{n}_0 + \beta \mathbf{n}_1 + \gamma \mathbf{n}_2. \quad (10.11)$$

And Equation (10.9) can then be evaluated for the \mathbf{n} computed at each pixel. Note that the \mathbf{n} resulting from Equation (10.11) is usually not a unit normal. Better visual results will be achieved if it is converted to a unit vector before it is used in shading computations. This type of normal interpolation is often called *Phong normal interpolation* (Phong, 1975).

10.3 Artistic Shading

The Lambertian and Phong shading methods are based on heuristics designed to imitate the appearance of objects in the real world. Artistic shading is designed to mimic drawings made by human artists (Yessios, 1979; Dooley & Cohen, 1990; Saito & Takahashi, 1990; L. Williams, 1991). Such shading seems to have advantages in many applications. For example, auto manufacturers hire artists to draw diagrams for car owners' manuals. This is more expensive than using much more "realistic" photographs, so there is probably some intrinsic advantage to the techniques of artists when certain types of communication are needed. In this section, we show how to make subtly shaded line drawings reminiscent of human-drawn images. Creating such images is often called *non-photorealistic rendering*, but we will avoid that term because many non-photorealistic techniques are used for efficiency that are not related to any artistic practice.

10.3.1 Line Drawing

The most obvious thing we see in human drawings that we don't see in real life is *silhouettes*. When we have a set of triangles with shared edges, we should draw an edge as a silhouette when one of the two triangles sharing an edge faces toward the viewer, and the other triangle faces away from the viewer. This condition can be tested for two normals \mathbf{n}_0 and \mathbf{n}_1 by

draw silhouette if $(\mathbf{e} \cdot \mathbf{n}_0)(\mathbf{e} \cdot \mathbf{n}_1) \leq 0$.

通过在多边形上插值法向量，然后在每个像素处应用Phong阴影，可以减少这些问题。这使您可以获得良好的图像，而不会使三角形的大小非常小。回顾第3章，当栅格化三角形时，我们计算重心坐标 (α, β, γ) 来插值顶点颜色 c_0, c_1, c_2 : $c = \alpha c_0 + \beta c_1 + \gamma c_2$ 。

我们可以使用相同的方程来插值表面法线 $\mathbf{n}_0, \mathbf{n}_1$ 和 \mathbf{n}_2 :

$$\mathbf{n} = \alpha \mathbf{n}_0 + \beta \mathbf{n}_1 + \gamma \mathbf{n}_2. \quad (10.11)$$

然后可以针对在每个像素处计算的 \mathbf{n} 来评估等式 (10.9)。请注意，由等式 (10.11) 产生的 \mathbf{n} 通常不是单位法线。如果在用于着色计算之前将其转换为单位向量，则会获得更好的视觉效果。这种类型的正态插值通常被称为 Phong 正态插值 (Phong, 1975)。

10.3 艺术阴影

Lambertian 和 Phong 着色方法基于启发法，旨在模仿现实世界中物体的外观。艺术阴影旨在模仿人类艺术家的绘画 (Yessios, 1979; Dooley & Cohen, 1990; Saito & Takahashi, 1990; L. Williams, 1991)。这样的阴影似乎在许多应用中具有优势。例如，汽车制造商聘请艺术家为车主手册绘制图表。这比使用更"真实"的照片要昂贵得多，所以当需要某些类型的交流时，艺术家的技术可能会有一些内在的优势。在本节中，我们将展示如何制作微妙的阴影线条图，让人联想到人类绘制的图像。创建这样的图像通常被称为非真实感渲染，但我们将避免这个术语，因为许多非真实感技术用于与任何艺术实践无关的效率。

10.3.1 线条画

我们在人类绘画中看到的最明显的东西，我们在现实生活中看不到的是剪影。当我们有一组具有共享边的三角形时，当共享一条边的两个三角形中的一个面向观看者，而另一个三角形面向远离观看者时，我们应该绘制一条边作为轮廓。此条件可以通过以下方法测试两个法线 \mathbf{n}_0 和 \mathbf{n}_1

绘制轮廓 if $(\mathbf{e} \cdot \mathbf{n}_0)(\mathbf{e} \cdot \mathbf{n}_1) \leq 0$ 。

Here \mathbf{e} is a vector from the edge to the eye. This can be any point on the edge or either of the triangles. Alternatively, if $f_i(\mathbf{p}) = 0$ are the implicit plane equations for the two triangles, the test can be written

$$\text{draw silhouette if } f_0(\mathbf{e})f_1(\mathbf{e}) \leq 0.$$

We would also like to draw visible edges of a polygonal model. To do this, we can use either of the hidden surface methods of Chapter 12 for drawing in the background color and then draw the outlines of each triangle in black. This, in fact, will also capture the silhouettes. Unfortunately, if the polygons represent a smooth surface, we really don't want to draw most of those edges. However, we might want to draw all *creases* where there really is a corner in the geometry. We can test for creases by using a heuristic threshold:

$$\text{draw crease if } (\mathbf{n}_0 \cdot \mathbf{n}_1) \leq \text{threshold}.$$

This combined with the silhouette test will give nice-looking line drawings.

10.3.2 Cool-to-Warm Shading

When artists shade line drawings, they often use low intensity shading to give some impression of curve to the surface and to give colors to objects (Gooch, Gooch, Shirley, & Cohen, 1998). Surfaces facing in one direction are shaded with a cool color, such as a blue, and surfaces facing in the opposite direction are shaded with a warm color, such as orange. Typically these colors are not very saturated and are also not dark. That way, black silhouettes show up nicely. Overall this gives a cartoon-like effect. This can be achieved by setting up a direction to a “warm” light \mathbf{l} and using the cosine to modulate color, where the warmth constant k_w is defined on $[0, 1]$:

$$k_w = \frac{1 + \mathbf{n} \cdot \mathbf{l}}{2}.$$

The color c is then just a linear blend of the cool color c_c and the warm color c_w :

$$c = k_w c_w + (1 - k_w) c_c.$$

There are many possible c_w and c_c that will produce reasonable looking results. A good starting place for a guess is

$$\begin{aligned} c_c &= (0.4, 0.4, 0.7), \\ c_c &= (0.8, 0.6, 0.6). \end{aligned}$$

Figure 10.9 shows a comparison between traditional Phong lighting and this type of artistic shading.

这里 \mathbf{e} 是从边缘到眼睛的矢量。这可以是边缘上的任何点，也可以是三角形中的任何一个。或者，如果 $f_i(p) = 0$ 是两个三角形的隐式平面方程，则可以编写测试

如果 $f_0(\mathbf{e})f_1(\mathbf{e}) \leq 0$ ，则绘制轮廓。

我们还想绘制多边形模型的可见边缘。为此，我们可以使用第12章的隐藏表面方法中的任何一种在背景颜色中绘制，然后用黑色绘制每个三角形的轮廓。事实上，这也将捕捉轮廓。不幸的是，如果多边形代表一个光滑的表面，我们真的不希望绘制这些边缘的大部分。但是，我们可能希望在几何中确实有一个角落的地方绘制所有折痕。我们可以使用启发式阈值来测试折痕：

绘制折痕 $\text{if } (\mathbf{n}_0 \cdot \mathbf{n}_1) \leq \text{threshold}$ 。

这与轮廓测试相结合将给出好看的线条图。

10.3.2 Cool-to-Warm Shading

当艺术家为线条画着色时，他们经常使用低强度阴影来给表面留下一些曲线的印象，并为物体赋予颜色 (Gooch, Gooch, Shirley, & Cohen, 1998)。面向一个方向的表面用冷色（如蓝色）阴影，面向相反方向的表面用暖色（如橙色）阴影。通常这些颜色不是很饱和，也不暗。这样，黑色的轮廓就会很好地显现出来。总体而言，这给出了一个卡通般的效果。这可以通过设置“暖”光的方向并使用余弦调制颜色来实现，其中暖度常数 k_w 在 $[0, 1]$ 上定义：

$$k_w = \frac{1 + \mathbf{n} \cdot \mathbf{l}}{2}.$$

然后，颜色 c 只是冷色 c_c 和暖色 c_w 的线性混合：

$$c = k_w c_w + (1 - k_w) c_c.$$

有许多可能的 c_w 和 c_c 会产生合理的外观结果。
一个猜测的好起点是

$$\begin{aligned} c_c &= (0.4, 0.4, 0.7), \\ c_c &= (0.8, 0.6, 0.6). \end{aligned}$$

图10.9显示了传统Phong照明和这种艺术阴影的比较。

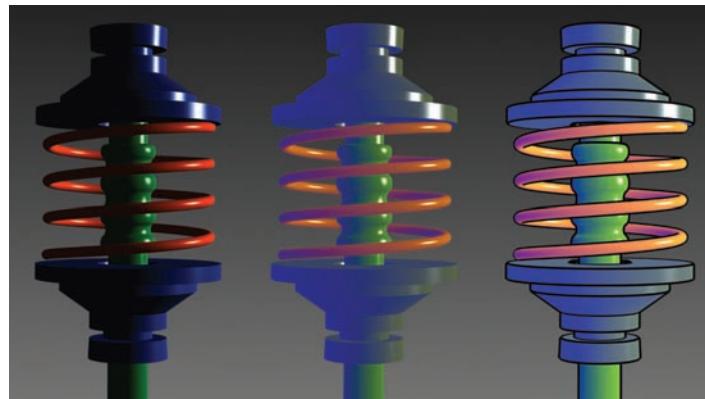


Figure 10.9. Left: a Phong-illuminated image. Middle: cool-to-warm shading is not useful without silhouettes. Right: cool-to-warm shading plus silhouettes. *Image courtesy Amy Gooch.*

Frequently Asked Questions

- All of the shading in this chapter seems like enormous hacks. Is that true?

Yes. However, they are carefully designed hacks that have proven useful in practice. In the long run, we will probably have better-motivated algorithms that include physics, psychology, and tone-mapping. However, the improvements in image quality will probably be incremental.

- I hate calling pow(). Is there a way to avoid it when doing Phong lighting?

A simple way is to only have exponents that are themselves a power of two, i.e., 2, 4, 8, 16, In practice, this is not a problematic restriction for most applications. A look-up table is also possible, but will often not give a large speed-up.

Exercises

1. The moon is poorly approximated by diffuse or Phong shading. What observations tell you that this is true?
2. Velvet is poorly approximated by diffuse or Phong shading. What observations tell you that this is true?

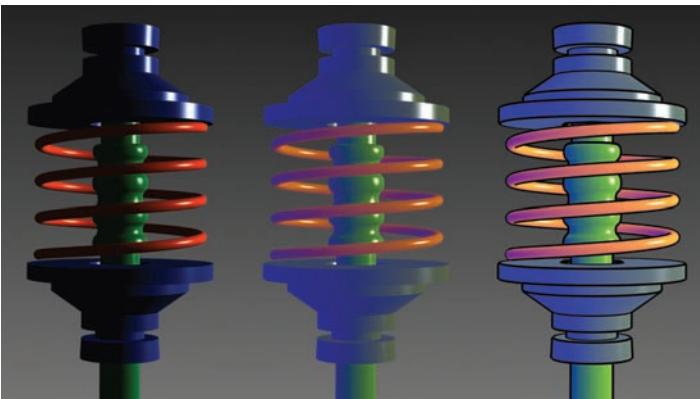


图10.9。左图：一幅Phong-illuminated图像。中间：没有剪影，冷到暖的阴影是没有用的。右：凉爽到温暖的阴影加上轮廓。图片由AmyGooch提供。

常见问题

- 本章中的所有阴影似乎都是巨大的黑客。这是真的吗？

是的.然而，它们是精心设计的黑客，在实践中被证明是有用的。从长远来看，我们可能会有更好的动机算法，包括物理学，心理学和色调映射。然而，图像质量的改进可能是渐进的。

- 我讨厌打电话给pow () 。在做Phong照明时，有没有办法避免它？

一个简单的方法是只有指数本身是二的幂，即 2 4 8 16 ...在实践中，对于大多数应用程序来说，这不是一个有问题的限制。查找表也是可能的，但通常不会给出很大的加速。

Exercises

1. 月亮很难被漫射或Phong阴影所接近.什么观察告诉你这是真的?
2. 绒被漫射或Phong阴影很差地近似。什么观察告诉你这是真的?



242

10. Surface Shading

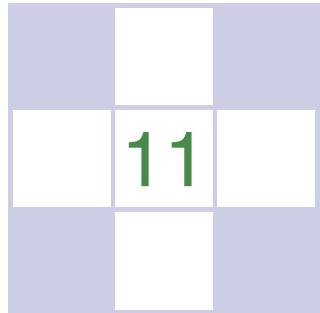
3. Why do most highlights on plastic objects look white, while those on gold metal look gold?



242

10. 表面阴影

- 3.为什么塑料物体上的大部分亮点看起来是白色的，而金色金属上的亮点看起来是金色的？



Texture Mapping

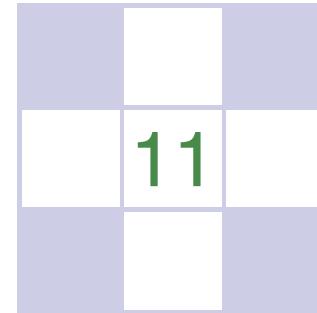
When trying to replicate the look of the real world, one quickly realizes that hardly any surfaces are featureless. Wood grows with grain; skin grows with wrinkles; cloth shows its woven structure; paint shows the marks of the brush or roller that laid it down. Even smooth plastic is made with bumps molded into it, and smooth metal shows the marks of the machining process that made it. Materials that were once featureless quickly become covered with marks, dents, stains, scratches, fingerprints, and dirt.

In computer graphics we lump all these phenomena under the heading of “spatially varying surface properties”—attributes of surfaces that vary from place to place but don’t really change the *shape* of the surface in a meaningful way. To allow for these effects, all kinds of modeling and rendering systems provide some means for *texture mapping*: using an image, called a *texture map*, *texture image*, or just a *texture*, to store the details that you want to go on a surface, then mathematically “mapping” the image onto the surface.

As it turns out, once the mechanism to map images onto surfaces exists, there are many less obvious ways it can be used that go beyond the basic purpose of introducing surface detail. Textures can be used to make shadows and reflections, to provide illumination, even to define surface shape. In sophisticated interactive programs, textures are used to store all kinds of data that doesn’t even have anything to do with pictures!

This chapter discusses the use of texture for representing surface detail, shadows, and reflections. While the basic ideas are simple, several practical problems

This is mapping in the sense of Section 2.1.



纹理映射

当试图复制现实世界的外观时，人们很快意识到几乎没有任何表面是无特征的。木材与谷物一起生长；皮肤与皱纹一起生长；布显示其编织结构；油漆显示放下它的刷子或滚筒的痕迹。即使是光滑的塑料也是用凸块成型的，光滑的金属显示了制造它的加工过程的痕迹。曾经没有特色的材料很快就会被标记、凹痕、污点、划痕、指纹和污垢所复盖。

在计算机图形学中，我们将所有这些现象放在“spatially varying surface properties”的标题下—表面的属性因地而异，但并没有真正以有意义的方式改变表面的形状。为了实现这些效果，各种建模和渲染系统都为纹理映射提供了一些方法：使用称为纹理映射、纹理图像或仅仅是纹理的图像来存储要在表面上的细节，然后将图像“映射”到表面上。

事实证明，一旦将图像映射到表面上的机制存在，就有许多不太明显的方法可以使用，这些方法超出了引入表面细节的基本目的。纹理可以用来制作阴影和反射，提供照明，甚至定义表面形状。在复杂的交互式程序中，纹理被用来存储各种甚至与图片无关的数据！

本章讨论如何使用纹理来表示表面细节、阴影和反射。虽然基本思想很简单，但几个实际问题

这是2.1节意义上的映射。

complicate the use of textures. First of all, textures easily become distorted, and designing the functions that map textures onto surfaces is challenging. Also, texture mapping is a resampling process, just like rescaling an image, and as we saw in Chapter 9, resampling can very easily introduce aliasing artifacts. The use of texture mapping and animation together readily produces truly dramatic aliasing, and much of the complexity of texture mapping systems is created by the *antialiasing* measures that are used to tame these artifacts.

11.1 Looking Up Texture Values

To start off, let's consider a simple application of texture mapping. We have a scene with a wood floor, and we would like the diffuse color of the floor to be controlled by an image showing floorboards with wood grain. Regardless of whether we are using ray tracing or rasterization, the shading code that computes the color for a ray-surface intersection point or for a fragment generated by the rasterizer needs to know the color of the texture at the shading point, in order to use it as the diffuse color in the Lambertian shading model from Chapter 10.

To get this color, the shader performs a *texture lookup*: it figures out the location, in the coordinate system of the texture image, that corresponds to the shading point, and it reads out the color at that point in the image, resulting in the *texture sample*. That color is then used in shading, and since the texture lookup happens at a different place in the texture for every pixel that sees the floor, a pattern of different colors shows up in the image. The code might look like this:

```
Color texture_lookup(Texture t, float u, float v) {
    int i = round(u * t.width() - 0.5)
    int j = round(v * t.height() - 0.5)
    return t.get_pixel(i, j)
}
Color shade_surface_point(Surface s, Point p, Texture t) {
    Vector normal = s.get_normal(p)
    (u, v) = s.get_texcoord(p)
    Color diffuse_color = texture_lookup(u, v)
    // compute shading using diffuse_color and normal
    // return shading result
}
```

In this code, the shader asks the surface where to look in the texture, and somehow every surface that we want to shade using a texture needs to be able to answer this query. This brings us to the first key ingredient of texture mapping: we need a function that maps from the surface to the texture that we can easily compute for every pixel. This is the *texture coordinate function* (Figure 11.1) and we say

使纹理的使用复杂化。首先，纹理很容易变形，设计将纹理映射到表面上的功能具有挑战性。此外，纹理映射是一个重采样过程，就像重新缩放图像一样，正如我们在第9章中看到的那样，重采样可以非常容易地引入混叠伪像。纹理映射和动画的结合使用很容易产生真正戏剧性的混叠，纹理映射系统的大部分复杂性是由用于驯服这些工件的抗混叠措施创建的。

11.1 查找纹理值

首先，让我们考虑一个纹理映射的简单应用。我们有一个木地板的场景，我们希望地板的漫反射颜色由一个显示有木纹的地板的图像控制。无论我们使用的是光线追踪还是光栅化，计算光线表面交叉点或光栅化器生成的片段颜色的着色代码都需要知道着色点处纹理的颜色，以便将其用作

为了获得这种颜色，着色器执行纹理查找：它在纹理图像的坐标系中找出与着色点相对应的位置，然后读出图像中该点的颜色，从而产生纹理样本。然后在着色中使用该颜色，并且由于纹理查找发生在纹理中每个看到地板的像素的不同位置，因此图像中会显示不同颜色的图案。代码可能如下所示：

```
Colortexture_lookup(Texture t, float u, float v) {
    int i = round(u * t.width() - 0.5)
    int j = round(v * t.height() - 0.5)
    return t.get_pixel(i, j)
}
Color shade_surface_point(Surface s, Point p, Texture t) {
    Vector normal = s.get_normal(p)
    (u, v) = s.get_texcoord(p)
    Color diffuse_color = texture_lookup(u, v)
    // 使用 diffuse_color 和法线返回着色结果
    // 计算着色
}
```

在这段代码中，着色器询问表面在纹理中的位置，不知何故，我们想要使用纹理着色的每个表面都需要能够回答这个查询。这就引出了纹理映射的第一个关键要素：我们需要一个从表面映射到纹理的函数，我们可以轻松地计算每个像素。这是纹理坐标函数（图11.1），我们说

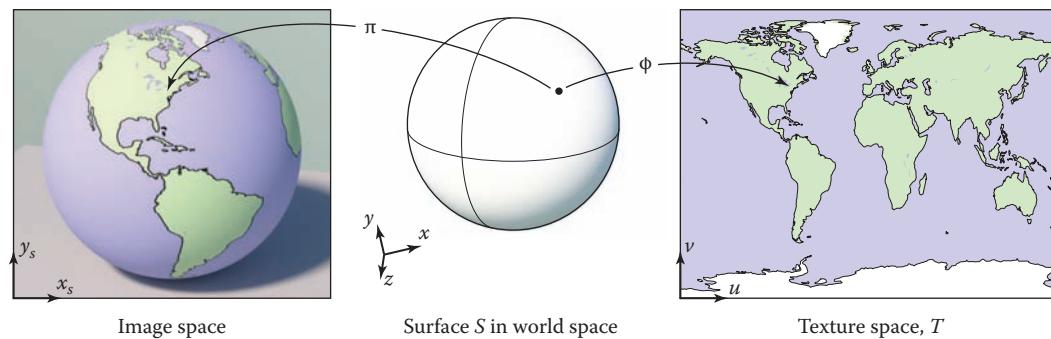


Figure 11.1. Just like the viewing projection π maps every point on an object's surface, S , to a point in the image, the texture coordinate function ϕ maps every point on the object's surface to a point in the texture map, T . Appropriately defining this function ϕ is fundamental to all applications of texture mapping.

that it assigns texture coordinates to every point on the surface. Mathematically it is a mapping from the surface S to the domain of the texture, T :

$$\begin{aligned}\phi : S &\rightarrow T \\ &: (x, y, z) \mapsto (u, v).\end{aligned}$$

The set T , often called “texture space,” is usually just a rectangle that contains the image; it is common to use the unit square $(u, v) \in [0, 1]^2$ (in this book we'll use the names u and v for the two *texture coordinates*). In many ways it's similar to the viewing projection discussed in Chapter 7, called π in this chapter, which maps points on surfaces in the scene to points in the image; both are 3D-to-2D mappings, and both are needed for rendering—one to know where to get the texture value from, and one to know where to put the shading result in the image. But there are some important differences, too: π is almost always a perspective or orthographic projection, whereas ϕ can take on many forms; and there is only one viewing projection for an image, whereas each object in the scene is likely to have a completely separate texture coordinate function.

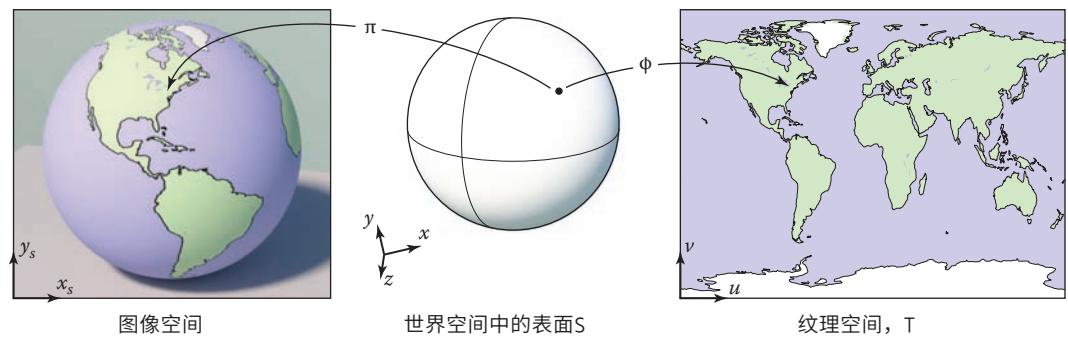
It may seem surprising that ϕ is a mapping *from* the surface *to* the texture, when our goal is to put the texture onto the surface, but this is the function we need.

For the case of the wood floor, if the floor happens to be at constant z and aligned to the x and y axes, we could just use the mapping

$$u = ax; \quad v = by,$$

for some suitably chosen scale factors a and b , to assign texture coordinates (s, t) to the point $(x, y, z)_{\text{floor}}$, and then use the value of the texture pixel, or *texel*,

So ... the first thing you have to learn is how to think backwards?



就像观看投影 π 将物体表面上的每个点 s 映射到图像中的一个点一样，纹理坐标函数 ϕ 将物体表面上的每个点映射到纹理映射中的一个点 T 。适当地定义此函数 ϕ 对于纹理映射的所有应用都是基础的。

它为表面上的每个点分配纹理坐标。从数学上讲，它是从表面 S 到纹理域的映射， T :

$$\begin{aligned}\phi : S &\rightarrow T \\ &: (x, y, z) \mapsto (u, v).\end{aligned}$$

集合 T ，通常称为“纹理空间”，通常只是一个包含图像的矩形；通常使用单位square $(u, v) \in [0, 1]^2$ （在本书中，我们将使用名称 u 和 v 来表示两个纹理坐标）。在许多方面，它类似于第7章中讨论的查看投影，在本章中称为 π ，它将场景中的表面上的点映射到图像中的点；两者都是3D到2D映射，并且两者都是但也有一些重要的区别： π 几乎总是透视投影或正投影，而 ϕ 可以有多种形式；一个图像只有一个观察投影，而场景中的每个对象都可能有一个完全独立的纹理坐标函数。

当我们的目标是将纹理放到表面上时， ϕ 是从表面到纹理的映射似乎令人惊讶，但这是我们需要的功能。

对于木地板的情况，如果地板恰好处于恒定的 z 并且与 x 和 y 轴对齐，我们可以使用映射

$$u = ax; \quad v = by,$$

对于一些适当选择的比例因子 a 和 b ，将纹理坐标 (s, t) 分配给点 (x, y, z) 地板，然后使用纹理像素或纹素的值

所以...你要学会的第一件事是如何向后思考？

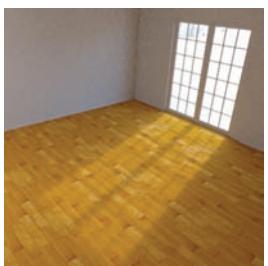


Figure 11.2. A wood floor, textured using a texture coordinate function that simply uses the x and y coordinates of points directly.

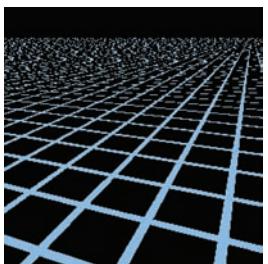


Figure 11.3. A large horizontal plane, textured in the same way as in Figure 11.2 and displaying severe aliasing artifacts.

closest to (u, v) as the texture value at (x, y) . In this way we rendered the image in Figure 11.2.

This is pretty limiting, though: what if the room is modeled at an angle to the x and y axes, or what if we want the wood texture on the curved back of a chair? We will need some better way to compute texture coordinates for points on the surface.

Another problem that arises from the simplest form of texture mapping is illustrated dramatically by rendering at a high contrast texture from a very grazing angle into a low-resolution image. Figure 11.3 shows a larger plane textured using the same approach but with a high contrast grid pattern and a view toward the horizon. You can see it contains aliasing artifacts (stairsteps in the foreground, wavy and glittery patterns in the distance) similar to the ones that arise in image resampling (Chapter 9) when appropriate filters are not used. Although it takes an extreme case to make these artifacts so obvious in a tiny still image printed in a book, in animations these patterns move around and are very distracting even when they are much more subtle.

We have now seen the two primary issues in basic texture mapping:

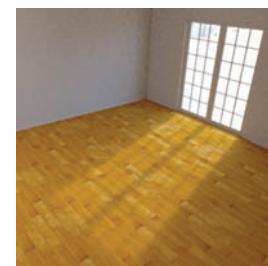
- defining texture coordinate functions, and
- looking up texture values without introducing too much aliasing.

These two concerns are fundamental to all kinds of applications of texture mapping and are discussed in Sections 11.2 and 11.3. Once you understand them and some of the solutions to them, then you understand texture mapping. The rest is just how to apply the basic texturing machinery for a variety of different purposes, which is discussed in Section 11.4.

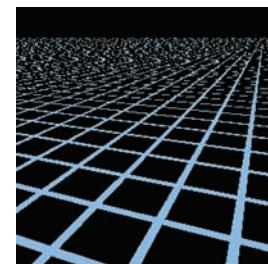
11.2 Texture Coordinate Functions

Designing the texture coordinate function ϕ well is a key requirement for getting good results with texture mapping. You can think of this as deciding how you are going to deform a flat, rectangular image so that it conforms to the 3D surface you want to draw. Or alternatively, you are taking the surface and gently flattening it, without letting it wrinkle, tear, or fold, so that it lies flat on the image. Sometimes this is easy: maybe the 3D surface is already a flat rectangle! In other cases it's very tricky: the 3D shape might be very complicated, like the surface of a character's body.

The problem of defining texture coordinate functions is not new to computer graphics. Exactly the same problem is faced by cartographers when designing



木地板，纹理使用纹理坐标函数，simply直接使用点的 x 和 y 坐标。



一个大的horizontal平面，以与图11.2相同的方式纹理，并显示严重的混叠伪像。

最接近(u, v)作为(x, y)处的纹理值。通过这种方式，我们渲染了图11.2中的图像。

但是，这是非常有限的：如果房间建模与 x 和 y 轴成一个角度，或者如果我们想要弯曲的椅背上的木材纹理怎么办？我们需要一些更好的方法来计算表面上点的纹理坐标。

从最简单的纹理映射形式产生的另一个问题是通过从非常掠角的高对比度纹理渲染到低分辨率图像来显着说明的。图11.3显示了使用相同方法但具有高对比度网格图案和朝向地平线的视图的较大平面纹理。您可以看到它包含混叠伪像（前景中的阶梯，距离中的波浪和闪光图案），类似于未使用适当滤镜时图像重采样（第9章）中出现的伪像。虽然在一本书中打印的微小静止图像中使这些工件如此明显需要极端的情况，但在动画中，这些图案四处移动，即使它们更加微妙，也会非常分散注意力。

我们现在已经看到了基本纹理映射中的两个主要问题：

- *定义纹理坐标函数，以及
- *查找纹理值而不引入太多的别名。

这两个问题对纹理映射的各种应用都是基本的，并在第11.2和11.3节中讨论。一旦你理解了它们以及它们的一些解决方案，那么你就理解了纹理映射。其余的只是如何将基本的纹理机制应用于各种不同的目的，这在第11.4节中讨论。

11.2 纹理坐标函数

设计好纹理坐标函数 ϕ 是纹理映射获得良好结果的关键要求。您可以将其视为决定如何变形平面矩形图像，使其符合您要绘制的3D表面。或者，您正在拍摄表面并轻轻压平它，而不让它起皱，撕裂或折叠，使其平放在图像上。有时这很容易：也许3D表面已经是一个平坦的矩形！在其他情况下，这是非常棘手的：3D形状可能非常复杂，就像角色身体的表面一样。

定义纹理坐标函数的问题对计算机图形学来说并不新鲜。制图师在设计时也面临同样的问题



maps that cover large areas of the Earth’s surface: the mapping from the curved globe to the flat map inevitably causes distortion of areas, angles, and/or distances that can easily make maps very misleading. Many map projections have been proposed over the centuries, all balancing the same competing concerns—of minimizing various kinds of distortion while covering a large area in one contiguous piece—that are faced in texture mapping.

In some applications (as we’ll see later in this chapter) there’s a clear reason to use a particular map. But in most cases, designing the texture coordinate map is a delicate task of balancing competing concerns, which skilled modelers put considerable effort into.

You can define ϕ in just about any way you can dream up. But there are several competing goals to consider:

- **Bijectivity.** In most cases you’d like ϕ to be bijective (see Section 2.1.1), so that each point on the surface maps to a *different* point in texture space. If several points map to the same texture space point, the value at one point in the texture will affect several points on the surface. In cases where you want a texture to repeat over a surface (think of wallpaper or carpet with their repeating patterns), it makes sense to deliberately introduce a many-to-one mapping from surface points to texture points, but you don’t want this to happen by accident.
- **Size distortion.** The scale of the texture should be approximately constant across the surface. That is, close-together points anywhere on the surface that are about the same distance apart should map to points about the same distance apart in the texture. In terms of the function ϕ , the magnitude of the derivatives of ϕ should not vary too much.
- **Shape distortion.** The texture should not be very distorted. That is, a small circle drawn on the surface should map to a reasonably circular shape in texture space, rather than an extremely squashed or elongated shape. In terms of ϕ , the derivative of ϕ should not be too different in different directions.
- **Continuity.** There should not be too many seams: neighboring points on the surface should map to neighboring points in the texture. That is, ϕ should be continuous, or have as few discontinuities as possible. In most cases, some discontinuities are inevitable, and we’d like to put them in inconspicuous locations.

Surfaces that are defined by parametric equations (Section 2.5.8) come with a built-in choice for the texture coordinate function: simply invert the function

“UV mapping” or “surface parameterization” are other names you may encounter for the texture coordinate function.



复盖地球表面大面积的地图：从曲面地球到平面地图的映射不可避免地会导致区域，角度和/或距离的扭曲，这些扭曲很容易使地图变得非常误导。几个世纪以来提出了许多地图投影，所有这些都平衡了纹理映射中面临的相同的竞争问题—在一个连续的部分中最大限度地模仿各种失真，同时复盖大面积。

在某些应用程序中（我们将在本章后面看到），有一个明确的理由使用特定的映射。但在大多数情况下，设计纹理坐标图是一个微妙的任务，平衡竞争的关注，熟练的建模者付出了相当大的努力。

你可以用任何你能想象的方式来定义 ϕ 。但有几个相互竞争的目标需要考虑：

*双射性。在大多数情况下，您希望 ϕ 是双射的（请参阅第2.1.1节），以便表面上的每个点映射到纹理空间中的不同点。如果有几个点映射到同一个纹理空间点，则纹理中某一点处的值将影响表面上的几个点。在您希望纹理在表面上重复的情况下（想想壁纸或地毯及其重复图案），故意从表面点到纹理点引入many-to-one映射是有意义的，但您不希望偶然发生这种情况。

*尺寸失真。纹理的尺度应该在整个表面上大致恒定。也就是说，表面上任何位置相距相同距离的紧密点应映射到纹理中相距相同距离的点。就函数 ϕ 而言， ϕ 的导数的量值不应相差太大。

*形状失真。纹理不应该非常扭曲。也就是说，在表面上绘制的小圆圈应该映射到纹理空间中合理的圆形形状，而不是极其压扁或细长的形状。就 ϕ 而言， ϕ 的导数在不同方向上不应相差太大。

*连续性。不应该有太多的接缝：表面上的邻近点应该映射到纹理中的邻近点。也就是说， ϕ 应该是连续的，或者具有尽可能少的不连续性。在大多数情况下，一些不连续性是不可避免的，我们希望将它们放在不显眼的位置。

由参数方程定义的曲面（第2.5.8节）带有纹理坐标函数的内置选择：简单地反转函数

“UV映射”或“表面参数化”是您可能会遇到的纹理坐标函数的其他名称。

that defines the surface, and use the two parameters of the surface as texture coordinates. These texture coordinates may or may not have desirable properties, depending on the surface, but they do provide a mapping.

But for surfaces that are defined implicitly, or are just defined by a triangle mesh, we need some other way to define the texture coordinates, without relying on an existing parameterization. Broadly speaking, the two ways to define texture coordinates are to compute them geometrically, from the spatial coordinates of the surface point, or, for mesh surfaces, to store values of the texture coordinates at vertices and interpolate them across the surface. Let's look at these options one at a time.

11.2.1 Geometrically Determined Coordinates

Geometrically determined texture coordinates are used for simple shapes or special situations, as a quick solution, or as a starting point for designing a hand-tweaked texture coordinate map.

We will illustrate the various texture coordinate functions by mapping the test image in Figure 11.4 onto the surface. The numbers in the image let you read the approximate (u, v) coordinates out of the rendered image, and the grid lets you see how distorted the mapping is.

09	19	29	39	49	59	69	79	89	99
08	18	28	38	48	58	68	78	88	98
07	17	27	37	47	57	67	77	87	97
06	16	26	36	46	56	66	76	86	96
05	15	25	35	45	55	65	75	85	95
04	14	24	34	44	54	64	74	84	94
03	13	23	33	43	53	63	73	83	93
02	12	22	32	42	52	62	72	82	92
01	11	21	31	41	51	61	71	81	91
00	10	20	30	40	50	60	70	80	90

Figure 11.4. Test image.

Planar Projection

Probably the simplest mapping from 3D to 2D is a parallel projection—the same mapping as used for orthographic viewing (Figure 11.5). The machinery we developed already for viewing (Section 7.1) can be re-used directly for defining texture coordinates: just as orthographic viewing boils down to multiplying by a matrix and discarding the z component, generating texture coordinates by planar projection can be done with a simple matrix multiply:

$$\phi(x, y, z) = (u, v) \quad \text{where} \quad \begin{bmatrix} u \\ v \\ * \\ 1 \end{bmatrix} = M_t \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

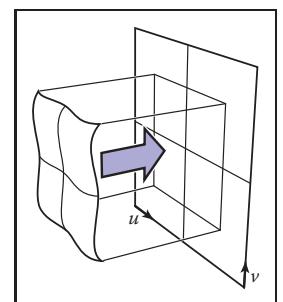


Figure 11.5. Planar projection makes a useful parameterization for objects or parts of objects that are nearly flat, without too much variation in surface normal, and a good projection direction can be found by taking the average normal. For any kind of closed shape, though, a planar projection will

where the texturing matrix M_t represents an affine transformation, and the asterisk indicates that we don't care what ends up in the third coordinate.

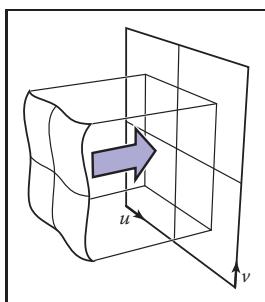
This works quite well for surfaces that are mostly flat, without too much variation in surface normal, and a good projection direction can be found by taking the average normal. For any kind of closed shape, though, a planar projection will

即定义曲面，并将曲面的两个参数作为纹理坐标。这些纹理坐标可能具有也可能不具有期望的属性，这取决于表面，但它们确实提供了映射。

但是对于隐式定义的曲面，或者只是由三角形网格定义的曲面，我们需要其他方式来定义纹理坐标，而不依赖于现有的参数化。广义地说，定义纹理坐标的两种方法是从表面点的空间坐标进行几何计算，或者对于网格表面，将纹理坐标的值存储在顶点并在表面上插值。让我们来看看这些选项一次一个。

09	19	29	39	49	59	69	79	89	99
08	18	28	38	48	58	68	78	88	98
07	17	27	37	47	57	67	77	87	97
06	16	26	36	46	56	66	76	86	96
05	15	25	35	45	55	65	75	85	95
04	14	24	34	44	54	64	74	84	94
03	13	23	33	43	53	63	73	83	93
02	12	22	32	42	52	62	72	82	92
01	11	21	31	41	51	61	71	81	91
00	10	20	30	40	50	60	70	80	90

图11.4. 测试图像。



如果投影方向大致沿整体法线选择，则Planarprojection可为开始接近平坦的对象或对象的部分提供有用参数化。

11.2.1 几何确定的坐标

几何确定的纹理坐标用于简单形状或特殊情况，作为快速解决方案，或作为设计手动调整纹理坐标图的起点。

我们将通过将图11.4中的测试图像映射到表面来说明各种纹理坐标函数。图像中的数字可以让您从渲染的图像中读取近似 (u, v) 坐标，网格可以让您看到映射的扭曲程度。

$$\phi(x, y, z) = (u, v) \quad \text{where} \quad \begin{bmatrix} u \\ v \\ * \\ 1 \end{bmatrix} = M_t \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

其中纹理化矩阵 M_t 表示仿射变换，星号表示我们不关心在第三个坐标中结束的是什么。

这对于大部分是平坦的表面非常有效，表面法线没有太大的变化，并且可以通过取平均法线找到良好的投影方向。然而，对于任何一种封闭的形状，平面投影将

not be injective: points on the front and back will map to the same point in texture space (Figure 11.6).

By simply substituting perspective projection for orthographic, we get *projective* texture coordinates (Figure 11.7):

$$\phi(x, y, z) = (\tilde{u}/w, \tilde{v}/w) \quad \text{where} \quad \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ * \\ w \end{bmatrix} = P_t \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Now the 4×4 matrix P_t represents a projective (not necessarily affine) transformation—that is, the last row may not be $[0, 0, 0, 1]$.

Projective texture coordinates are important in the technique of shadow mapping, discussed in Section 11.4.4.

Spherical Coordinates

For spheres, the latitude/longitude parameterization is familiar and widely used. It has a lot of distortion near the poles, which can lead to difficulties, but it does cover the whole sphere with discontinuities only along one line of latitude.

Surfaces that are roughly spherical in shape can be parameterized using a texture coordinate function that maps a point on the surface to a point on a sphere using radial projection: take a line from the center of the sphere through the point on the surface, and find the intersection with the sphere. The spherical coordinates of this intersection point are the texture coordinates of the point you started with on the surface.

Another way to say this is that you express the surface point in spherical coordinates (ρ, θ, ϕ) and then discard the ρ coordinate and map θ and ϕ each to the range $[0, 1]$. The formula depends on the spherical coordinates convention; using the convention of Section 2.5.8,

$$\phi(x, y, z) = ([\pi + \text{atan}2(y, x)]/2\pi, [\pi - \text{acos}(z/\|x\|)]/\pi).$$

A spherical coordinates map will be bijective everywhere except at the poles if the whole surface is visible from the center point. It inherits the same distortion near the poles as the latitude-longitude map on the sphere. Figure 11.8 shows an object for which spherical coordinates provide a suitable texture coordinate function.

Cylindrical Coordinates

For objects that are more columnar than spherical, projection outward from an axis onto a cylinder may work better than projection from a point onto a sphere

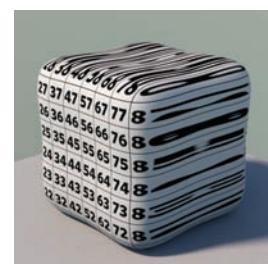


Figure 11.6. Using planar projection on a closed object will always result in a non-injective, one-to-many mapping, and extreme distortion near points where the projection direction is tangent to the surface.

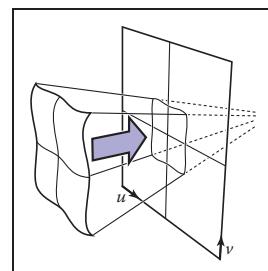


Figure 11.7. A projective texture transformation uses a viewing-like transformation that projects toward a point.

This and other texture coordinate functions in this chapter for objects that are in the box $[-1, 1]^3$ and centered at the origin.

不要注入：正面和背面的点将映射到纹理空间中的同一点（图11.6）。

通过简单地将透视投影替换为正投影，我们得到了射影纹理坐标（图11.7）：

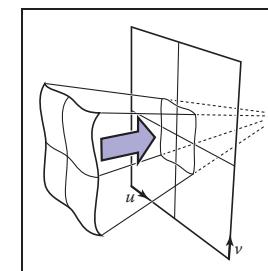
$$\phi(x, y, z) = (\tilde{u}/w, \tilde{v}/w) \quad \text{where} \quad \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ * \\ w \end{bmatrix} = P_t \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

现在 4×4 矩阵 P_t 表示射影（不一定是仿射）变换—也就是说，最后一行可能不是 $[0 \ 0 \ 0 \ 1]$ 。

投影纹理坐标在阴影映射技术中很重要 在第11.4.4节中讨论。



在封闭对象上使用平面投影将始终导致在投影方向与曲面相切的点附近产生非注入性、一对多的映射ping和极端失真。



一个专业的纹理转换使用一个类似观看的变形，投射到一个点。

球面坐标

对于球体，纬度经度参数化是熟悉和广泛使用的。它在两极附近有很多扭曲，这可能会导致困难，但它确实只沿着一条纬度线复盖了整个球体的不连续性。

形状大致为球形的曲面可以使用纹理坐标函数参数化，该函数使用径向投影将曲面上的点映射到球体上的点：从球体中心通过曲面上的点取一条线，并这个交点的球面坐标是你开始在表面上的点的纹理坐标。

另一种说法是，你用球面坐标 (ρ, θ, ϕ) 表达表面点，然后丢弃 ρ 坐标并将 θ 和 ϕ 各自映射到范围 $[0, 1]$ 。该公式取决于球坐标约定；使用第2.5.8节的约定

$$\phi(x, y, z) = ([\pi + \text{atan}2(y, x)]/2\pi, [\pi - \text{acos}(z/\|x\|)]/\pi).$$

球面坐标图将是双射的，除了在两极，如果整个表面是可见的中心点。它继承了与球体上的经纬度地图相同的极点附近的失真。图11.8显示了球坐标为其提供合适的纹理坐标函数的对象。

圆柱坐标

对于柱状多于球形的对象，从轴向外投影到圆柱体上可能比从点投影到球体上效果更好

本章中的此和其他纹理坐标函数适用于框[11]3中并以原点为中心的对象。



Figure 11.8. For this vaguely sphere-like object, projecting each point onto a sphere centered at the center of the object provides an injective mapping, which here is used to place the same map texture as was used for the globe images. Note that areas become magnified (surface points are crowded together in texture space) where the surface is far from the center, and areas shrink where the surface is closer to the center.



Figure 11.9. A far-from-spherical vase for which spherical projection produces a lot of distortion (left) and cylindrical projection produces a very good result on the outer surface.

(Figure 11.9). Analogously to spherical projection, this amounts to converting to cylindrical coordinates and discarding the radius:

$$\phi(x, y, z) = \left(\frac{1}{2\pi}[\pi + \text{atan}2(y, x)]/2\pi, \frac{1}{2}[1 + z] \right).$$

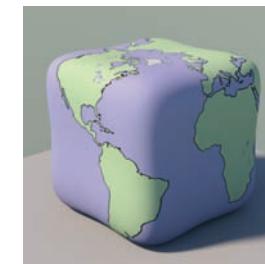
Cubemaps

Using spherical coordinates to parameterize a spherical or sphere-like shape leads to high distortion of shape and area near the poles, which often leads to visible artifacts that reveal that there are two special points where something is going wrong with the texture. A popular alternative is much more uniform at the cost of having more discontinuities. The idea is to project onto a cube, rather than a sphere, and then use six separate square textures for the six faces of the cube. The collection of six square textures is called a *cubemap*. This introduces discontinuities along all the cube edges, but it keeps distortion of shape and area low.

Computing cubemap texture coordinates is also cheaper than for spherical coordinates, because projecting onto a plane just requires a division—essentially the same as perspective projection for viewing. For instance, for a point that projects onto the $+z$ face of the cube:

$$(x, y, z) \mapsto \left(\frac{x}{z}, \frac{y}{z} \right).$$

A confusing aspect of cubemaps is establishing the convention for how the u and v directions are defined on the six faces. Any convention is fine, but the convention chosen affects the contents of textures, so standardization is important. Because



对于这个模糊的球体状object，将每个点投影到以对象中心为中心的球体上提供了一个注入映射，这里用于放置与地球图像相同的地图纹理。请注意，在表面远离中心的地方，区域会被放大（表面点在纹理空间中挤在一起），而在表面更靠近中心的地方，区域会缩小。



一个远非球形的花瓶，球形凸起产生大量的扭曲（左）和圆柱形凸起在外表面产生非常好的结果。

(图11.9)。类似于球面投影，这相当于转换为圆柱坐标并丢弃半径：

$$\phi(x, y, z) = \left(\frac{1}{2\pi}[\pi + \text{atan}2(y, x)]/2\pi, \frac{1}{2}[1 + z] \right).$$

Cubemaps

使用球面坐标来参数化球形或类球体形状会导致极附近的形状和面积的高度失真，这通常会导致可见的伪像，这些伪像会显示有两个特殊点，其中纹理一个流行的替代方案是以更多不连续性为代价的更加统一。这个想法是投影到一个立方体上，而不是一个球体，然后为立方体的六个面使用六个单独的正方形纹理。六个正方形纹理的集合称为立方体贴图。这引入了沿所有立方体边缘的不连续性，但它保持了形状和面积的低失真。

计算立方体贴图纹理坐标也比球面坐标便宜，因为投影到平面上只需要分割—基本上与透视投影相同。例如，对于投影到立方体的 $+z$ 面上的点：

$$(x, y, z) \mapsto \left(\frac{x}{z}, \frac{y}{z} \right).$$

立方体贴图的一个令人困惑的方面是建立如何在六个面上定义 u 和 v 方向的惯例。任何约定都可以，但选择的约定会影响纹理的内容，因此标准化很重要。因为

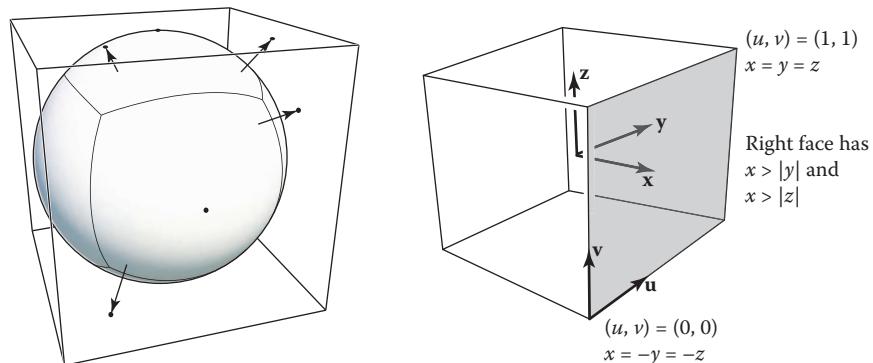


Figure 11.10. A surface being projected into a cubemap. Points on the surface project outward from the center, each mapping to a point on one of the six faces.

cubemaps are very often used for textures that are viewed from the inside of the cube (see environment mapping in Section 11.4.5), the usual conventions have the u and v axes oriented so that u is clockwise from v as viewed from inside. The convention used by OpenGL is

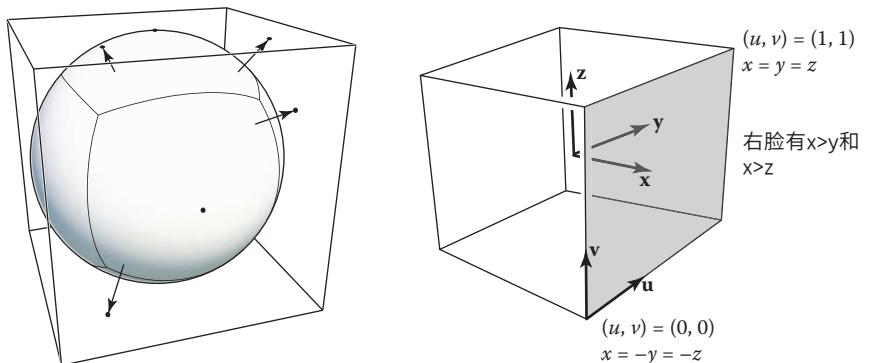
$$\begin{aligned}\phi_{-x}(x, y, z) &= \frac{1}{2} [1 + (+z, -y) / |x|], \\ \phi_{+x}(x, y, z) &= \frac{1}{2} [1 + (-z, -y) / |x|], \\ \phi_{-y}(x, y, z) &= \frac{1}{2} [1 + (+x, -z) / |y|], \\ \phi_{+y}(x, y, z) &= \frac{1}{2} [1 + (+x, +z) / |y|], \\ \phi_{-z}(x, y, z) &= \frac{1}{2} [1 + (-x, -y) / |z|], \\ \phi_{+z}(x, y, z) &= \frac{1}{2} [1 + (+x, -y) / |z|].\end{aligned}$$

The subscripts indicate which face of the cube each projection corresponds to. For example, ϕ_{-x} is used for points that project to the face of the cube at $x = +1$. You can tell which face a point projects to by looking at the coordinate with the largest absolute value: for example, if $|x| > |y|$ and $|x| > |z|$, the point projects to the $+x$ or $-x$ face, depending on the sign of x .

A texture to be used with a cube map has six square pieces. (See Figure 11.10.) Often they are packed together in a single image for storage, arranged as if the cube was unwrapped.

11.2.2 Interpolated Texture Coordinates

For more fine-grained control over the texture coordinate function on a triangle mesh surface, you can explicitly store the texture coordinates at each vertex,



投影到立方体贴图中的表面。表面上的点从中心向外投影，每个映射到六个面之一上的一个点。

立方体贴图通常用于从立方体内部查看的纹理（请参阅第11.4.5节中的环境映射），通常的约定有 u 和 v 轴的方向，以便 u 从 v 顺时针从 v 从内部查看。OpenGL使用的约定是

$$\begin{aligned}\phi_{-x}(x, y, z) &= \frac{1}{2} [1 + (+z, -y) / |x|], \\ \phi_{+x}(x, y, z) &= \frac{1}{2} [1 + (-z, -y) / |x|], \\ \phi_{-y}(x, y, z) &= \frac{1}{2} [1 + (+x, -z) / |y|], \\ \phi_{+y}(x, y, z) &= \frac{1}{2} [1 + (+x, +z) / |y|], \\ \phi_{-z}(x, y, z) &= \frac{1}{2} [1 + (-x, -y) / |z|], \\ \phi_{+z}(x, y, z) &= \frac{1}{2} [1 + (+x, -y) / |z|].\end{aligned}$$

下标表示每个投影对应于立方体的哪个面。例如， ϕ_{-x} 用于在 $x=+1$ 处投影到立方体表面的点。您可以通过查看绝对值最大的坐标来判断一个点投影到哪个面：例如，如果 $x>y$ 和 $x>z$ ，则该点投影到 $+x$ 或 $-x$ 面，具体取决于 x 的符号。

要与立方体贴图一起使用的纹理有六个正方形块。（见图11.10。）通常，它们被打包在一个单一的图像中进行存储，就像立方体被解开一样排列。

11.2.2 插值纹理坐标

要对三角形网格曲面上的纹理坐标函数进行更细粒度的控制，可以显式存储每个顶点处的纹理坐标

09	19	29	39	49	59	69	79	89	99
08	18	28	38	48	58	68	78	88	98
07	17	27	37	47	57	67	77	87	97
06	16	26	36	46	56	66	76	86	96
05	15	25	35	45	55	65	75	85	95
04	14	24	34	44	54	64	74	84	94
03	13	23	33	43	53	63	73	83	93
02	12	22	32	42	52	62	72	82	92
01	11	21	31	41	51	61	71	81	91
00	10	20	30	40	50	60	70	80	90

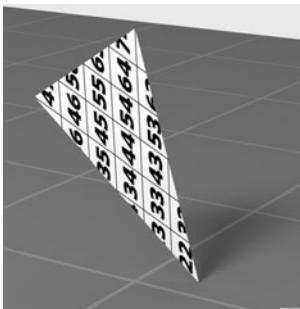


Figure 11.11. A single triangle using linearly interpolated texture coordinates. Left: the triangle drawn in texture space; right: the triangle rendered in a 3D scene.

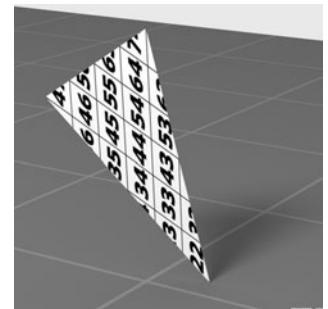
and interpolate them across the triangles using barycentric interpolation (Section 8.1.2). It works in exactly the same way as any other smoothly varying quantities you might define over a mesh: colors, normals, even the 3D position itself.

The idea of interpolated texture coordinates is very simple—but it can be a bit confusing at first.

Let's look at an example with a single triangle. Figure 11.11 shows a triangle texture mapped with part of the by now familiar test pattern. By looking at the pattern that appears on the rendered triangle, you can deduce that the texture coordinates of the three vertices are $(0.2, 0.2)$, $(0.8, 0.2)$, and $(0.2, 0.8)$, because those are the points in the texture that appear at the three corners of the triangle. Just as with the geometrically determined mappings in the previous section, we control where the texture goes on the surface by giving the mapping from the surface to the texture domain, in this case by specifying where each vertex should go in texture space. Once you position the vertices, linear (barycentric) interpolation across triangles takes care of the rest.

In Figure 11.12 we show a common way to visualize texture coordinates on a whole mesh: simply draw triangles in texture space with the vertices positioned at

09	19	29	39	49	59	69	79	89	99
08	18	28	38	48	58	68	78	88	98
07	17	27	37	47	57	67	77	87	97
06	16	26	36	46	56	66	76	86	96
05	15	25	35	45	55	65	75	85	95
04	14	24	34	44	54	64	74	84	94
03	13	23	33	43	53	63	73	83	93
02	12	22	32	42	52	62	72	82	92
01	11	21	31	41	51	61	71	81	91
00	10	20	30	40	50	60	70	80	90



使用线性插值纹理坐标的单个三角形。左：在纹理空间中绘制的三角形；右：在3D场景中渲染的三角形。

并使用重心插值（第8.1.2节）在三角形上插值它们。它的工作方式与您可能在网格上定义的任何其他平滑变化的量完全相同：颜色，法线，甚至3D位置本身。

让我们来看一个单三角形的例子。图11.11显示了与现在熟悉的测试模式的一部分映射的triangle纹理。通过查看呈现的三角形上出现的图案，可以推断出三个顶点的纹理坐标为 $(0.2, 0.2)$ $(0.8, 0.2)$ 和 $(0.2, 0.8)$ ，因为这些是纹理中出现在三角形的三个角处的点。就像上一节中几何确定的映射一样，我们通过给出从surface到纹理域的映射来控制纹理在表面上的位置，在这种情况下，通过指定每个顶点在纹理空间中的位置来控制纹理在表面上的位置。定位顶点后，跨三角形的线性（重心）插值将处理其余部分。

09	19	29	39	49	59	69	79	89	99
08	18	28	38	48	58	68	78	88	98
07	17	27	37	47	57	67	77	87	97
06	16	26	36	46	56	66	76	86	96
05	15	25	35	45	55	65	75	85	95
04	14	24	34	44	54	64	74	84	94
03	13	23	33	43	53	63	73	83	93
02	12	22	32	42	52	62	72	82	92
01	11	21	31	41	51	61	71	81	91
00	10	20	30	40	50	60	70	80	90



Figure 11.12. An icosahedron with its triangles laid out in texture space to provide zero distortion but with many seams.

09	19	29	39	49	59	69	79	89	99
08	18	28	38	48	58	68	78	88	98
07	17	27	37	47	57	67	77	87	97
06	16	26	36	46	56	66	76	86	96
05	15	25	35	45	55	65	75	85	95
04	14	24	34	44	54	64	74	84	94
03	13	23	33	43	53	63	73	83	93
02	12	22	32	42	52	62	72	82	92
01	11	21	31	41	51	61	71	81	91
00	10	20	30	40	50	60	70	80	90



一个二十面体，其三角形布置在纹理空间中，提供零失真，但有许多接缝。

their texture coordinates. This visualization shows you what parts of the texture are being used by which triangles, and it is a handy tool for evaluating texture coordinates and for debugging all sorts of texture-mapping code.

The quality of a texture coordinate mapping that is defined by vertex texture coordinates depends on what coordinates are assigned to the vertices—that is, how the mesh is laid out in texture space. No matter what coordinates are assigned, as long as the triangles in the mesh share vertices (Section 12.1), the texture coordinate mapping is always continuous, because neighboring triangles will agree on the texture coordinate at points on their shared edge. But the other desirable qualities described above are not so automatic. Injectivity means the triangles don't overlap in texture space—if they do, it means there's some point in the texture that will show up at more than one place on the surface.

Size distortion is low when the areas of triangles in texture space are in proportion to their areas in 3D. For instance, if a character's face is mapped with a continuous texture coordinate function, one often ends up with the nose squeezed into a relatively small area in texture space, as shown in Figure 11.13. Although triangles on the nose are smaller than on the cheek, the ratio of sizes is more extreme in texture space. The result is that the texture is enlarged on the nose, because a small area of texture has to cover a large area of surface. Similarly, comparing the forehead to the temple, the triangles are similar in size in 3D, but the triangles around the temple are larger in texture space, causing the texture to appear smaller there.

Similarly, shape distortion is low when the shapes of triangles are similar in 3D and in texture space. The face example has fairly low shape distortion, but, for example, the sphere in Figure 11.17 has very large shape distortion near the poles.

11.2.3 Tiling, Wrapping Modes, and Texture Transformations

It's often useful to allow texture coordinates to go outside the bounds of the texture image. Sometimes this is a detail: rounding error in a texture coordinate calculation might cause a vertex that lands exactly on the texture boundary to be slightly outside, and the texture mapping machinery should not fail in that case. But it can also be a modeling tool.

If a texture is only supposed to cover part of the surface, but texture coordinates are already set up to map the whole surface to the unit square, one option is to prepare a texture image that is mostly blank with the content in a small area. But that might require a very high resolution texture image to get enough detail in the relevant area. Another alternative is to scale up all the texture coordinates so

它们的纹理坐标。此可视化显示了哪些三角形正在使用纹理的哪些部分，它是一个方便的工具，用于评估纹理坐标和调试各种纹理映射代码。

由顶点纹理坐标定义的纹理坐标映射的质量取决于分配给顶点的坐标—即网格在纹理空间中的布局方式。无论分配什么坐标，只要网格中的三角形共享顶点（第12.1节），纹理坐标映射始终是连续的，因为相邻三角形将在其共享边上的点上就纹理坐标达成一致。但上面描述的其他理想品质并不是那么自动。注入性意味着三角形在纹理空间中不重叠—如果它们重叠，则意味着纹理中的某些点将显示在表面上的多个位置。

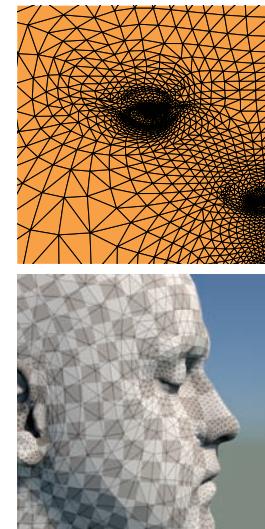
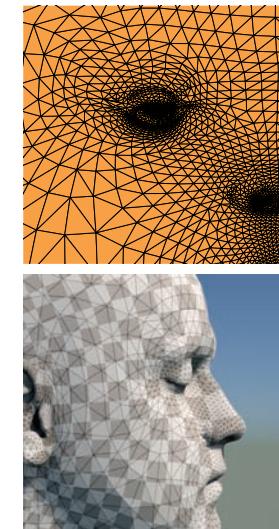


Figure 11.13. A face model, with texture coordinates assigned so as to achieve reasonably low shape distortion, but still showing moderate area distortion.

例如，如果一个角色的脸部用一个连续的纹理坐标函数映射，通常会被压缩到纹理空间中一个相对较小的区域，如图11.13所示。虽然鼻子上的三角形比脸颊上的小，但尺寸的比例在纹理空间中更为极端。结果是纹理在鼻子上被放大，因为小面积的纹理必须覆盖大面积的表面。同样，将额头与太阳穴进行比较，三角形在3D中的大小相似，但太阳穴周围的三角形在纹理空间中较大，导致纹理在那里显得较小。

同样，当三角形的形状在3d和纹理空间中相似时，形状失真也很低。面部示例具有相当低的形状失真，但是，例如，图11.17中的球体在极点附近具有非常大的形状失真。



一个人脸模型，具有纹理协调性，以实现合理的低形状失真，但仍然显示较小的area区域失真。

11.2.3 平铺、包装模式和纹理转换

允许纹理坐标超出texture图像的边界通常很有用。有时这是一个细节：纹理坐标计算中的舍入错误可能会导致恰好落在纹理边界上的顶点稍微外侧，并且纹理映射机制不应该在这种情况下失败。但它也可以是一个建模工具。

如果一个纹理只应该复盖部分表面，但是已经设置了纹理坐标以将整个表面映射到单位正方形，则一种选择是准备一个纹理图像，该图像大部分是空白的，内容在一个小区域中。但这可能需要非常高分辨率的纹理图像才能在相关区域获得足够的细节。另一种选择是放大所有纹理坐标，以便

that they cover a larger range— $[-4.5, 5.5] \times [-4.5, 5.5]$ for instance, to position the unit square at one-tenth size in the center of the surface.

For a case like this, texture lookups outside the unit-square area that's covered by the texture image should return a constant background color. One way to do this is to set a background color to be returned by texture lookups outside the unit square. If the texture image already has a constant background color (for instance, a logo on a white background), another way to extend this background automatically over the plane is to arrange for lookups outside the unit square to return the color of the texture image at the closest point on the edge, achieved by *clamping* the u and v coordinates to the range from the first pixel to the last pixel in the image.

Sometimes we want a repeating pattern, such as a checkerboard, a tile floor, or a brick wall. If the pattern repeats on a rectangular grid, it would be wasteful to create an image with many copies of the same data. Instead we can handle texture lookups outside the texture image using wraparound indexing—when the lookup point exits the right edge of the texture image, it wraps around to the left edge. This is handled very simply using the integer remainder operation on the pixel coordinates.

```
Color texture_lookup_wrap(Texture t, float u, float v) {
    int i = round(u * t.width() - 0.5)
    int j = round(v * t.height() - 0.5)
    return t.get_pixel(i % t.width(), j % t.height())
}
Color texture_lookup_wrap(Texture t, float u, float v) {
    int i = round(u * t.width() - 0.5)
    int j = round(v * t.height() - 0.5)
    return t.get_pixel(max(0, min(i, t.width()-1)),
                      max(0, min(j, t.height()-1)))
}
```

The choice between these two ways of handling out-of-bounds lookups is specified by selecting a *wrapping mode* from a list that includes tiling, clamping, and often combinations or variants of the two. With wrapping modes, we can freely think of a texture as a function that returns a color for any point in the infinite 2D plane (Figure 11.14). When we specify a texture using an image, these modes describe how the finite image data is supposed to be used to define this function. In Section 11.5, we'll see that procedural textures can naturally extend across an infinite plane, since they are not limited by finite image data. Since both are logically infinite in extent, the two types of textures are interchangeable.

When adjusting the scale and placement of textures, it's convenient to avoid actually changing the functions that generate texture coordinates, or the texture coordinate values stored at vertices of meshes, by instead applying a matrix trans-

它们覆盖的范围更大 $[-4.5, 5.5] \times [-4.5, 5.5]$ 。例如，将单位正方形定位在表面中心的十分之一大小。

对于这样的情况，纹理图像覆盖的单位正方形区域之外的纹理查找应该返回一个恒定的背景颜色。这样做的一种方法是设置一个背景颜色，以通过在单位正方形之外的纹理查找返回。如果纹理图像已经具有恒定的背景颜色（例如，白色背景上的徽标），则在平面上自动扩展此背景的另一种方法是安排在单位正方形之外的查找，以返回

有时我们想要一个重复的图案，如棋盘，瓷砖地板或砖墙。如果图案在矩形网格上重复，则创建具有许多相同数据副本的图像将是浪费的。相反，我们可以使用环绕索引来处理纹理图像外部的纹理查找—当查找点退出纹理图像的右边缘时，它会环绕到左边缘。这非常简单地使用像素坐标上的整数余数操作来处理。

```
Colortexture_lookup_wrap(Texture t, float u, float v) {
    int i = round(u * t.width() - 0.5)
    int j = round(v * t.height() - 0.5)
    return t.get_pixel(i % t.width(), j % t.height())
}
Colortexture_lookup_wrap(Texture t, float u, float v) {
    int i = round(u * t.width() - 0.5)
    int j = round(v * t.height() - 0.5)
    return t.get_pixel(max(0, min(i, t.width() - 1)), max(0, min(j, t.height() - 1)))
}
```

这两种处理越界查找的方法之间的选择是通过从包含平铺、夹紧以及通常两者的组合或变体的列表中选择包装模式来指定的。使用包装模式，我们可以自由地将纹理视为一个函数，它为无限二维平面中的任何点返回颜色（图11.14）。当我们使用图像指定纹理时，这些模式描述了如何使用有限图像数据来定义此函数。在第11.5节中，我们将看到过程纹理可以自然地延伸到无限平面，因为它们不受有限图像数据的限制。由于两者在逻辑上是无限的，所以这两种类型的纹理是可以互换的。

在调整纹理的比例和位置时，通过改为应用矩阵来避免实际更改生成纹理坐标的函数或存储在网格顶点处的纹理坐标值

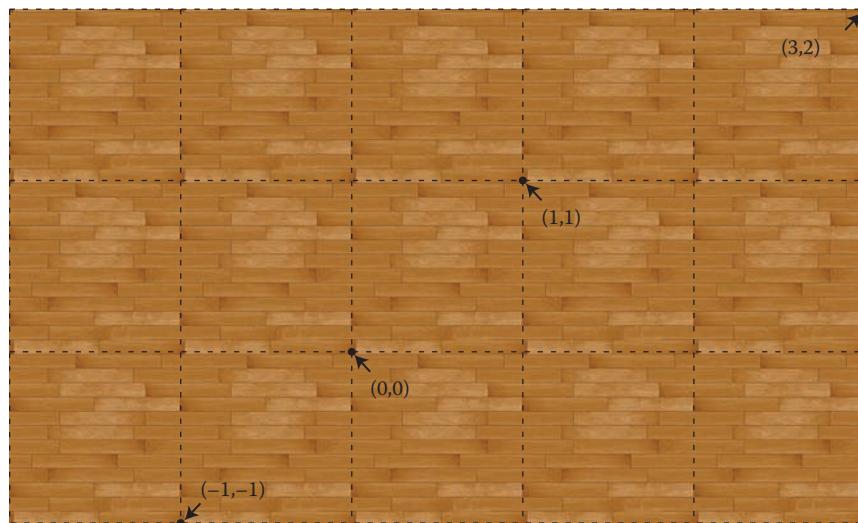


Figure 11.14. A wood floor texture tiled over texture space by wrapping texel coordinates.

formation to the texture coordinates before using them to sample the texture:

$$\phi(\mathbf{x}) = \mathbf{M}_T \phi_{\text{model}}(\mathbf{x}),$$

where ϕ_{model} is the texture coordinate function provided with the model, and \mathbf{M}_T is a 3 by 3 matrix representing an affine or projective transformation of the 2D texture coordinates using homogeneous coordinates. Such a transformation, sometimes limited just to scaling and/or translation, is supported by most renderers that use texture mapping.

11.2.4 Perspective Correct Interpolation

There are some subtleties in achieving correct-looking perspective by interpolating texture coordinates across triangles, but we can address this at the rasterization stage. The reason things are not straightforward is that just interpolating texture coordinates in screen space results in incorrect images, as shown for the grid texture in Figure 11.15. Because things in perspective get smaller as the distance to the viewer increases, the lines that are evenly spaced in 3D should compress in 2D image space. More careful interpolation of texture coordinates is needed to accomplish this.

We can implement texture mapping on triangles by interpolating the (u, v) coordinates, modifying the rasterization method of Section 8.1.2, but this results

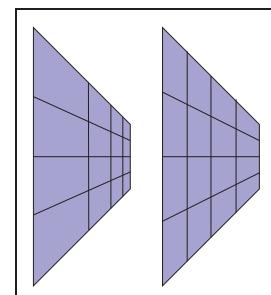


Figure 11.15. Left: correct perspective. Right: interpolation in screen space.

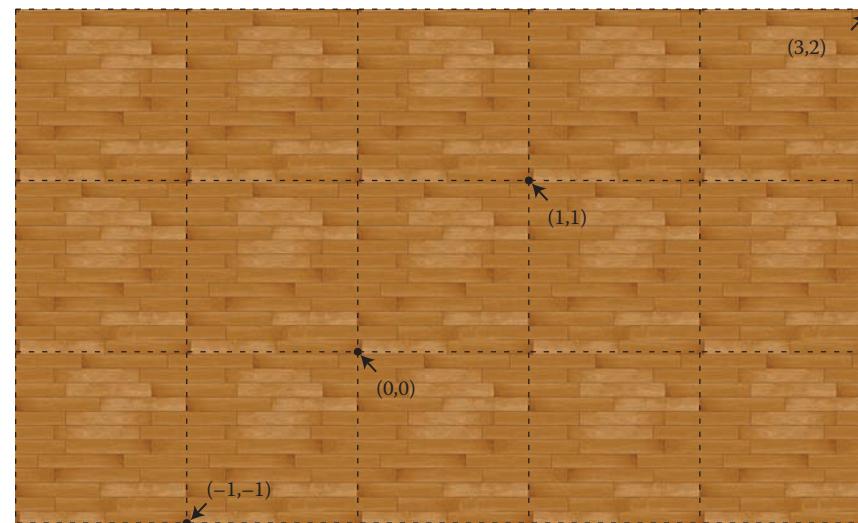


图11.14。通过包裹texel坐标在纹理空间上平铺的木地板纹理。

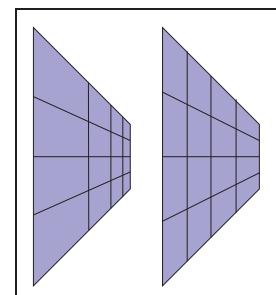
在使用它们对纹理进行采样之前，对纹理坐标进行形成：

$$\phi(\mathbf{x}) = \mathbf{M}_T \phi_{\text{model}}(\mathbf{x}),$$

其中 ϕ 模型是随模型提供的纹理坐标函数， \mathbf{M}_T 是表示使用齐次坐标的2d纹理坐标的仿射或射影变换的3乘3矩阵。大多数使用纹理映射的渲染器都支持这种转换，有时仅限于缩放和/或平移。

11.2.4 透视正确插值

通过跨三角形插值纹理坐标来实现正确透视有一些微妙之处，但我们可以解决这个问题。事情并不简单的原因是，仅仅在屏幕空间中插值纹理坐标会导致不正确的图像，如图11.15中的网格纹理所示。由于透视中的事物随着到观看者的距离的增加而变小，因此在3D中均匀间隔的线条应该在2D图像空间中压缩。需要对纹理坐标进行更仔细的插值来实现这一点。



左：正确的视角。右：屏幕空间中的interpolation。

我们可以通过插值 (u, v) 坐标，修改第8.1.2节的光栅化方法，在三角形上实现纹理映射，但这会导致

in the problem shown at the right of Figure 11.15. A similar problem occurs for triangles if screen space barycentric coordinates are used as in the following rasterization code:

```

for all  $x$  do
  for all  $y$  do
    compute  $(\alpha, \beta, \gamma)$  for  $(x, y)$ 
    if  $\alpha \in (0, 1)$  and  $\beta \in (0, 1)$  and  $\gamma \in (0, 1)$  then
       $t = \alpha t_0 + \beta t_1 + \gamma t_2$ 
      drawpixel  $(x, y)$  with color texture( $t$ ) for a solid texture
      or with texture( $\beta, \gamma$ ) for a 2D texture.
  
```

This code will generate images, but there is a problem. To unravel the basic problem, let's consider the progression from world space q to homogeneous point r to homogenized point s :

$$\begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix} \xrightarrow{\text{transform}} \begin{bmatrix} x_r \\ y_r \\ z_r \\ h_r \end{bmatrix} \xrightarrow{\text{homogenize}} \begin{bmatrix} x_r/h_r \\ y_r/h_r \\ z_r/h_r \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix}.$$

The simplest form of the texture coordinate interpolation problem is when we have texture coordinates (u, v) associated with two points, q and Q , and we need to generate texture coordinates in the image along the line between s and S . If the world-space point q' that is on the line between q and Q projects to the screen-space point s' on the line between s and S , then the two points should have the same texture coordinates.

The naïve screen-space approach, embodied by the algorithm above, says that at the point $s' = s + \alpha(S - s)$ we should use texture coordinates $u_s + \alpha(u_S - u_s)$ and $v_s + \alpha(v_S - v_s)$. This doesn't work correctly because the world-space point q' that transforms to s' is *not* $q + \alpha(Q - q)$.

However, we know from Section 7.4 that the points on the line segment between q and Q do end up somewhere on the line segment between s and S ; in fact, in that section we showed that

$$q + t(Q - q) \mapsto s + \alpha(S - s).$$

The interpolation parameters t and α are not the same, but we can compute one from the other.¹

$$t(\alpha) = \frac{w_r \alpha}{w_R + \alpha(w_r - w_R)} \quad \text{and} \quad \alpha(t) = \frac{w_R t}{w_r + t(w_R - w_r)}. \quad (11.1)$$

¹It is worthwhile to derive these functions yourself from Equation (7.6); in that chapter's notation, $\alpha = f(t)$.

在图11.15右侧所示的问题中。如果像以下光栅化代码中那样使用屏幕空间中心坐标，则三角形也会出现类似的问题：

为所有x做为
所有y做

计算 $(\alpha \beta \gamma)$ for $(x y)$ 如果 $\alpha \in (0, 1)$ 和 $\beta \in (0, 1)$ 和 $\gamma \in (0, 1)$
则 $t=\alpha t_0 + \beta t_1 + \gamma t_2$ drawpixel $(x y)$ 具有颜色纹理(t)用于
实体纹理或具有纹理($\beta \gamma$)用于2D纹理。

此代码将生成图像，但有一个问题。为了解开基本的问题，让我们考虑从世界空间 q 到同质点 r 到同质点 s 的进展：

$$\begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix} \xrightarrow{\text{transform}} \begin{bmatrix} x_r \\ y_r \\ z_r \\ h_r \end{bmatrix} \xrightarrow{\text{homogenize}} \begin{bmatrix} x_r/h_r \\ y_r/h_r \\ z_r/h_r \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix}.$$

纹理坐标插值问题的最简单形式是当我们有纹理坐标 (u, v) 与两个点 q 和 Q 相关联时，我们需要在图像中沿着 s 和 S 之间的线生成纹理坐标。如果位于 q 和 Q 之间线上的世界空间点 q' 投影到 s 和 S 之间线上的屏幕空间点 s' ，那么这两个点应该具有相同的纹理坐标。

由上述算法体现的naïve屏幕空间方法说，在 $s' = s + \alpha(S - s)$ 点，我们应该使用纹理坐标 $u_s + \alpha(u_S - u_s)$ 和 $v_s + \alpha(v_S - v_s)$ 。这不能正常工作，因为转换为 s' 的世界空间点 q' 不是 $q + \alpha(Q - q)$ 。

然而，我们从第7.4节中知道，线段上的点是补间的 q 和 Q 确实最终在 s 和 S 之间的线段上的某个地方；事实上，在该节中我们展示了

$$q + t(Q - q) \mapsto s + \alpha(S - s).$$

插值参数 t 和 α 不相同，但我们可以通过另一个计算一个：1

$$t(\alpha) = \frac{w_r \alpha}{w_R + \alpha(w_r - w_R)} \quad \text{and} \quad \alpha(t) = \frac{w_R t}{w_r + t(w_R - w_r)}. \quad (11.1)$$

¹从等式 (7.6) 中自己推导这些函数是值得的；在该章的表示法中， $\alpha=f(t)$ 。

These equations provide one possible fix to the screen-space interpolation idea. To get texture coordinates for the screen-space point $s' = s + \alpha(S - s)$, compute $u'_s = u_s + t(\alpha)(u_S - u_s)$ and $v'_s = v_s + t(\alpha)(v_S - v_s)$. These are the coordinates of the point q' that maps to s' , so this will work. However, it is slow to evaluate $t(\alpha)$ for each fragment, and there is a simpler way.

The key observation is that because, as we know, the perspective transform preserves lines and planes, it is safe to linearly interpolate any attributes we want across triangles, but only as long as they go through the perspective transformation along with the points. To get a geometric intuition for this, reduce the dimension so that we have homogeneous points (x_r, y_r, w_r) and a single attribute u being interpolated. The attribute u is supposed to be a linear function of x_r and y_r , so if we plot u as a height field over (x_r, y_r) the result is a plane. Now, if we think of u as a third spatial coordinate (call it u_r to emphasize that it's treated the same as the others) and send the whole 3D homogeneous point (x_r, y_r, u_r, w_r) through the perspective transformation, the result (x_s, y_s, u_s) still generates points that lie on a plane. There will be some warping within the plane, but the plane stays flat. This means that u_s is a linear function of (x_s, y_s) —which is to say, we can compute u_s anywhere by using linear interpolation based on the coordinates (x_s, y_s) .

Returning to the full problem, we need to interpolate texture coordinates (u, v) that are linear functions of the world space coordinates (x_q, y_q, z_q) . After transforming the points to screen space, and adding the texture coordinates as if they were additional coordinates, we have

$$\begin{bmatrix} u \\ v \\ 1 \\ x_r \\ y_r \\ z_r \\ w_r \end{bmatrix} \xrightarrow{\text{homogenize}} \begin{bmatrix} u/w_r \\ v/w_r \\ 1/w_r \\ x_r/w_r = x_s \\ y_r/w_r = y_s \\ z_r/w_r = z_s \\ 1 \end{bmatrix}. \quad (11.2)$$

The practical implication of the previous paragraph is that we *can* go ahead and interpolate all of these quantities based on the values of (x_s, y_s) —including the value z_s , used in the z-buffer. The problem with the naïve approach is simply that we are interpolating components selected inconsistently—as long as the quantities involved are from before or all from after the perspective divide, all will be well.

The one remaining problem is that $(u/w_r, v/w_r)$ is not directly useful for looking up texture data; we need (u, v) . This explains the purpose of the extra parameter we slipped into (11.2), whose value is always 1: once we have u/w_r , v/w_r , and $1/w_r$, we can easily recover (u, v) by dividing.

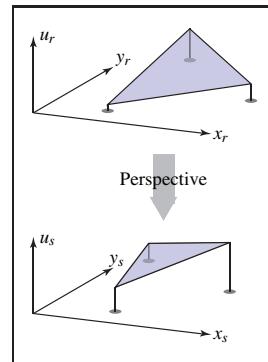
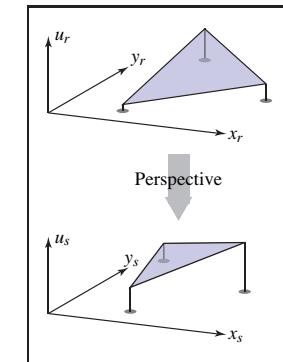


Figure 11.16. Geometric reasoning for screen-space interpolation. Top: u_r is to be interpolated as a linear function of (x_r, y_r) . Bottom: after a perspective transformation from (x_r, y_r, u_r, w_r) to $(x_s, y_s, u_s, 1)$, u_s is a linear function of (x_s, y_s) .

这些方程为屏幕空间插值思想提供了一个可能的解决方案。要获得屏幕空间点 $s' = s + \alpha(S - s)$ 的纹理坐标, 请计算 $u' = u + t(\alpha)(u_S - u_s)$ 和 $v' = v + t(\alpha)(v_S - v_s)$ 。这些是映射到 s' 的点 q' 的坐标, 所以这将起作用。然而, 评估每个片段的 $t(\alpha)$ 是缓慢的, 并且有一种更简单的方法。

关键的观察是, 因为, 正如我们所知, 透视变换保留了线和平面, 所以线性插值我们想要跨越三角形的任何属性是安全的, 但前提是它们与点一起通过透视变换。为了得到一个几何直觉, 减少 dimension, 使我们有同质点 (x_r, y_r, w_r) 和一个单一的属性 u 被插值。属性 u 应该是 x_r 和 y_r 的线性函数, 所以如果我们将 u 绘制为 (x_r, y_r) 上的高度场, 结果就是一个平面。现在, 如果我们将 u 视为第三个空间坐标 (称之为 u_r 以强调它与其他坐标相同), 并通过透视变换发送整个 3D 同质点 (x_r, y_r, u_r, w_r) , 结果 (x_s, y_s, u_s) 飞机内会有一些翘曲, 但飞机保持平坦。这意味着 u_s 是 (x_s, y_s) 的线性函数—也就是说, 我们可以通过使用基于坐标 (x_s, y_s) 的线性插值来将 u_s 放在任何地方。回到完整的问题, 我们需要插值纹理坐标 (u, v) , 它们是世界空间坐标 (x_q, y_q, z_q) 的线性函数。在将点反式形成到屏幕空间, 并添加纹理坐标, 就好像它们是额外的坐标一样, 我们有



Terpolation中屏幕空间的几何推理。上图: u_r 被插值为 (x_r, y_r) 的线性函数.底部:after a perspective transformation from (x_r, y_r, u_r, w_r) to $(x_s, y_s, u_s, 1)$, u_s 是 (x_s, y_s) 的线性函数。

$$\begin{bmatrix} u \\ v \\ 1 \\ x_r \\ y_r \\ z_r \\ w_r \end{bmatrix} \xrightarrow{\text{homogenize}} \begin{bmatrix} u/w_r \\ v/w_r \\ 1/w_r \\ x_r/w_r = x_s \\ y_r/w_r = y_s \\ z_r/w_r = z_s \\ 1 \end{bmatrix}. \quad (11.2)$$

上一段的实际含义是, 我们可以继续并根据 (x_s, y_s) 的值插入所有这些量—包括在 z 缓冲区中使用的值 z_s 。Naïve 方法的问题在于我们对不一致选择的组件进行插值—只要所涉及的量化是从透视划分之前或全部之后, 一切都会好起来。

剩下的一个问题是在 $(u/w_r, v/w_r)$ 对于查找纹理数据不直接有用;我们需要 (u, v) 。这解释了我们滑入 (11.2) 的额外参数的目的, 其值始终为 1:一旦我们有 u/w_r , v/w_r 和 $1/w_r$, 我们可以通过除法轻松恢复 (u, v) 。

To verify that this is all correct, let's check that interpolating the quantity $1/w_r$ in screen space indeed produces the reciprocal of the interpolated w_r in world space. To see this is true, confirm (Exercise 2):

$$\frac{1}{w_r} + \alpha(t) \left(\frac{1}{w_R} - \frac{1}{w_r} \right) = \frac{1}{w'_r} = \frac{1}{w_r + t(w_R - w_r)} \quad (11.3)$$

remembering that $\alpha(t)$ and t are related by Equation 11.1.

This ability to interpolate $1/w_r$ linearly with no error in the transformed space allows us to correctly texture triangles. We can use these facts to modify our scan-conversion code for three points $\mathbf{t}_i = (x_i, y_i, z_i, w_i)$ that have been passed through the viewing matrices, but have not been homogenized, complete with texture coordinates $\mathbf{t}_i = (u_i, v_i)$:

```

for all  $x_s$  do
  for all  $y_s$  do
    compute  $(\alpha, \beta, \gamma)$  for  $(x_s, y_s)$ 
    if  $(\alpha \in [0, 1] \text{ and } \beta \in [0, 1] \text{ and } \gamma \in [0, 1])$  then
       $u_s = \alpha(u_0/w_0) + \beta(u_1/w_1) + \gamma(u_2/w_2)$ 
       $v_s = \alpha(v_0/w_0) + \beta(v_1/w_1) + \gamma(v_2/w_2)$ 
       $l_s = \alpha(1/w_0) + \beta(1/w_1) + \gamma(1/w_2)$ 
       $u = u_s/l_s$ 
       $v = v_s/l_s$ 
      drawpixel  $(x_s, y_s)$  with color texture( $u, v$ )
  
```

Of course, many of the expressions appearing in this pseudocode would be pre-computed outside the loop for speed. For solid textures, it's simple enough to include the original world space coordinates x_q, y_q, z_q in the list of attributes, treated the same as u and v , and correct interpolated world space coordinates will be obtained, which can be passed to the solid texture function.

11.2.5 Continuity and Seams

Although low distortion and continuity are nice properties to have in a texture coordinate function, discontinuities are often unavoidable. For any closed 3D surface, it's a basic result of topology that there is no continuous, bijective function that maps the whole surface into a texture image. Something has to give, and by introducing seams—curves on the surface where the texture coordinates change suddenly—we can have low distortion everywhere else. Many of the geometrically determined mappings discussed above already contain seams: in spherical and cylindrical coordinates, the seams are where the angle computed by atan2

为了验证这一切都是正确的，让我们检查在屏幕空间内插数量 $1/w_r$ 确实会产生世界空间内插 w_r 的倒数。看到这是真的，确认（练习2）：

$$\frac{1}{w_r} + \alpha(t) \left(\frac{1}{w_R} - \frac{1}{w_r} \right) = \frac{1}{w'_r} = \frac{1}{w_r + t(w_R - w_r)} \quad (11.3)$$

记住 $\alpha(t)$ 和 t 通过公式11.1相关。

这种在变换空间中线性插值 $1/w_r$ 而没有错误的能力使我们能够正确地纹理三角形。我们可以使用这些事实来修改三个点 $\mathbf{t}_i = (x_i, y_i, z_i, w_i)$ 的扫描转换代码，这些点已经通过查看矩阵，但没有被均匀化，完成纹理坐标 $\mathbf{t}_i = (u_i, v_i)$ ：

对于所有 x_s 对于所有 y_s 计算 (α, β, γ) for (x_s, y_s) if $(\alpha \in [0, 1] \text{ and } \beta \in [0, 1] \text{ and } \gamma \in [0, 1])$ then $u_s = \alpha(u_0/w_0) + \beta(u_1/w_1) + \gamma(u_2/w_2)$ $v_s = \alpha(v_0/w_0) + \beta(v_1/w_1) + \gamma(v_2/w_2)$ $l_s = \alpha(1/w_0) + \beta(1/w_1) + \gamma(1/w_2)$ $u = u_s/l_s$ $v = v_s/l_s$ drawpixel (x_s, y_s) with color texture(u, v)	对于所有 x_s 对于所有 y_s 计算 (α, β, γ) for (x_s, y_s) if $(\alpha \in [0, 1] \text{ and } \beta \in [0, 1] \text{ and } \gamma \in [0, 1])$ then $ndy \in [0, 1]$ then $u = \alpha(u_0/w_0) + \beta(u_1/w_1) + \gamma(u_2/w_2)$ $vs = \alpha(v_0/w_0) + \beta(v_1/w_1) + \gamma(v_2/w_2)$ $ls = vs/1$ $sdrawpixel(xs ys)$ with color texture(u, v)
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

当然，在这个伪代码中出现的许多表达式将在循环之外预先计算速度。对于实体纹理，将原始世界空间坐标 x_q, y_q, z_q 包含在属性列表中，与 u 和 v 处理相同，将获得正确的插值世界空间坐标，可以传递给实体纹理函数。

11.2.5 连续性和接缝

虽然低失真和连续性是纹理坐标函数中很好的属性，但不连续性通常是不可避免的。对于任何封闭的3d表面，拓扑的基本结果是没有连续的双射函数将整个表面映射到纹理图像中。必须给出一些东西，通过在纹理坐标突然变化的表面上引入接缝—曲线—我们可以在其他地方实现低失真。上面讨论的许多几何确定的映射已经包含接缝：在球形和圆柱形坐标中，接缝是由atan2计算的角度

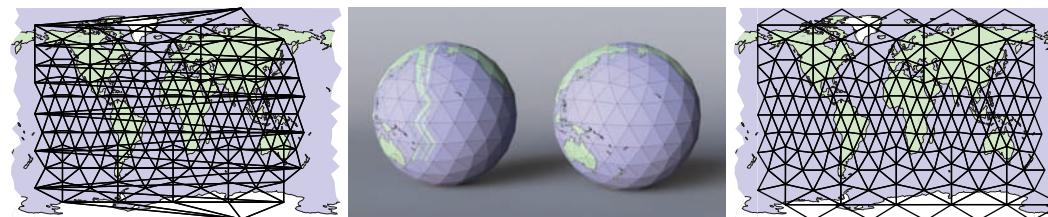
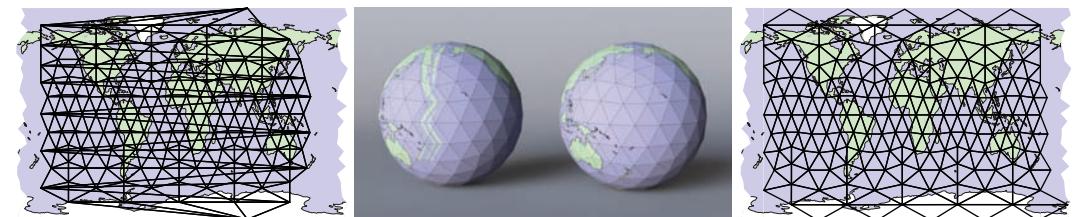


Figure 11.17. Polygonal globes: on the left, with all shared vertices, the texture coordinate function is continuous, but necessarily has problems with triangles that cross the 180th meridian, because texture coordinates are interpolated from longitudes near 180 to longitudes near -180 . On the right, some vertices are duplicated, with identical 3D positions but texture coordinates differing by exactly 360 degrees in longitude, so that texture coordinates are interpolated across the meridian rather than all the way across the map.

wraps around from π to $-\pi$, and in the cubemap, the seams are along the cube edges, where the mapping switches between the six square textures.

With interpolated texture coordinates, seams require special consideration, because they don't happen naturally. We observed earlier that interpolated texture coordinates are automatically continuous on shared-vertex meshes—the sharing of texture coordinates guarantees it. But this means that if a triangle spans a seam, with some vertices on one side and some on the other, the interpolation machinery will cheerfully provide a continuous mapping, but it will likely be highly distorted or fold over so that it's not injective. Figure 11.17 illustrates this problem on a globe mapped with spherical coordinates. For example, there is a triangle near the bottom of the globe that has one vertex at the tip of New Zealand's South Island, and another vertex in the Pacific about 400 km northeast of the North Island. A sensible pilot flying between these points would fly over New Zealand, but the path starts at longitude 167° E ($+167$) and ends at 179° W (that is, longitude -179), so linear interpolation chooses a route that crosses South America on the way. This causes a backward copy of the entire map to be compressed into the strip of triangles that crosses the 180th meridian! The solution is to label the second vertex with the equivalent longitude of 181° E, but this just pushes the problem to the next triangle.

The only way to create a clean transition is to avoid sharing texture coordinates at the seam: the triangle crossing New Zealand needs to interpolate to longitude $+181$, and the next triangle in the Pacific needs to continue starting from longitude -179 . To do this, we duplicate the vertices at the seam: for each vertex we add a second vertex with an equivalent longitude, differing by 360° s, and the triangles on opposite sides of the seam use different vertices. This solution is shown in the right half of Figure 11.17, in which the vertices at the far left and right of the texture space are duplicates, with the same 3D positions.



多边形地球仪：在左侧，具有所有共享顶点，纹理坐标函数是连续的，但必然具有跨越第180子午线的三角形的问题，因为纹理坐标从180附近的经度插值到180在右侧，一些顶点是重复的，具有相同的3D位置，但纹理坐标在经度上恰好相差360度，因此纹理坐标在子午线上插值，而不是在地图上插值。

从 π 到 $-\pi$ 环绕，在立方体贴图中，接缝沿着立方体边缘，其中映射在六个正方形纹理之间切换。

对于插值纹理坐标，接缝需要特别考虑，因为它们不会自然发生。我们之前观察到，插值纹理坐标在共享顶点网格上是自动连续的—纹理坐标的共享保证了这一点。但这意味着，如果一个三角形跨越一个接缝，一边有一些顶点，另一边有一些顶点，插值机制将愉快地提供一个连续的映射，但它可能会高度扭曲或折叠，以图11.17在用球坐标映射的地球仪上说明了这个问题。例如，在地球底部附近有一个三角形，在新西兰南岛的尖端有一个顶点，而在北部东北约400公里的太平洋上的另一个顶点是陆地。在这些点之间飞行的明智飞行员将飞越新西兰，但路径从经度 167° SE (+167)开始，在 179° SW (即longitude 179)结束，因此线性插值选择了在途中穿越南美洲的路线。这会导致整个地图的向后副本被压缩到穿过第180子午线的三角形条中！解决方案是用 181° SE的等效经度标记第二个顶点，但这只是将问题推向下一个三角形。

创建干净过渡的唯一方法是避免在接缝处共享纹理坐标：穿越新西兰的三角形需要插值到经度 $+181$ ，太平洋的下一个三角形需要继续从longitude 179开始。要做到这一点，我们复制接缝处的顶点：对于每个顶点，我们添加一个具有等效经度的第二个顶点，相差 360° s，并且接缝相对两侧的三角使用不同的顶点。该解决方案如图11.17的右半部分所示，其中纹理空间最左侧和右侧的顶点是重复的，具有相同的3D位置。

11.3 Antialiasing Texture Lookups

It's a good idea to review the first half of Chapter 9 now.

The second fundamental problem of texture mapping is antialiasing. Rendering a texture mapped image is a sampling process: mapping the texture onto the surface and then projecting the surface into the image produces a 2D function across the image plane, and we are sampling it at pixels. As we saw in Chapter 9, doing this using point samples will produce aliasing artifacts when the image contains detail or sharp edges—and since the whole point of textures is to introduce detail, they become a prime source of aliasing problems like the ones we saw in Figure 11.3.

Just as with antialiased rasterization of lines or triangles, antialiased ray tracing (Section 13.4), or downsampling images (Section 9.4), the solution is to make each pixel not a point sample but an area average of the image, over an area similar in size to the pixel. Using the same supersampling approach used for antialiased rasterization and ray tracing, with enough samples, excellent results can be obtained with no changes to the texture mapping machinery: many samples within a pixel's area will land at different places in the texture map, and averaging the shading results computed using the different texture lookups is an accurate way to approximate the average color of the image over the pixel. However, with detailed textures it takes very many samples to get good results, which is slow. Computing this area average *efficiently* in the presence of textures on the surface is the first key topic in texture antialiasing.

Texture images are usually defined by raster images, so there is also a reconstruction problem to be considered, just as with upsampling images (Section 9.4). The solution is the same for textures: use a reconstruction filter to interpolate between texels.

We expand on each of these topics in the following sections.

11.3.1 The Footprint of a Pixel

What makes antialiasing textures more complex than other kinds of antialiasing is that the relationship between the rendered image and the texture is constantly changing. Every pixel value should be computed as an average color over the area belonging to the pixel in the image, and in the common case that the pixel is looking at a single surface, this corresponds to averaging over an area on the surface. If the surface color comes from a texture, this in turn amounts to averaging over a corresponding part of the texture, known as the *texture space footprint* of the pixel. Figure 11.18 illustrates how the footprints of square areas (which could be pixel areas in a lower-resolution image) map to very different sized and shaped areas in the floor's texture space.

11.3 Antialiasing Texture Lookups

现在回顾第9章的前半部分是个好主意。

纹理映射的第二个基本问题是抗锯齿。渲染纹理映射图像是一个采样过程：将纹理映射到表面上，然后将表面投影到图像中会产生一个跨越图像平面的2D函数，我们在像素处对其进行采样。正如我们在第9章中所看到的，当图像包含细节或锐边时，使用点样本这样做会产生混叠伪像——而且由于纹理的全部要点是引入细节，它们成为混叠问题的主要来源，就像我们在图11.3中看到的那样。就像线或三角形的抗锯齿光栅化、抗锯齿射线tracing（第13.4节）或下采样图像（第9.4节）一样，解决方案是使每个像素不是点样本，而是图像的面积平均值，在与像素大小相似的区域上。使用用于抗锯齿光栅化和光线跟踪的相同超采样方法，如果有足够的样本，可以在不改变纹理映射机制的情况下获得出色的结果：像素区域内的许多样。然而，对于详细的纹理，需要很多样本才能获得良好的结果，这是缓慢的。在表面存在纹理的情况下有效地计算该面积平均值是纹理抗锯齿的第一个关键主题。

纹理图像通常由光栅图像定义，因此还需要考虑recon结构问题，就像上采样图像一样（第9.4节）。对于纹理，解决方案是相同的：使用重建过滤器在纹理之间进行插值。

我们在以下各节中对每个主题进行了扩展。

11.3.1 像素的足迹

使抗锯齿纹理比其他类型的抗锯齿更复杂的是渲染图像和纹理之间的关系不断变化。每个像素值都应该计算为图像中属于该像素的区域上的平均颜色，并且在该像素正在查看单个表面的常见情况下，这对应于在该表面上的一个区域上如果表面颜色来自纹理，则这反过来相当于对纹理的相应部分进行平均，即像素的纹理空间足迹。图11.18说明了正方形区域（可能是低分辨率图像中的像素区域）的足迹如何映射到地板纹理空间中大小和形状非常不同的区域。

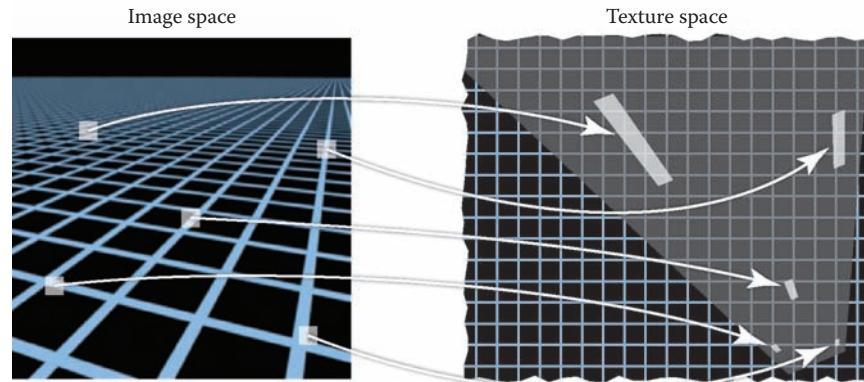


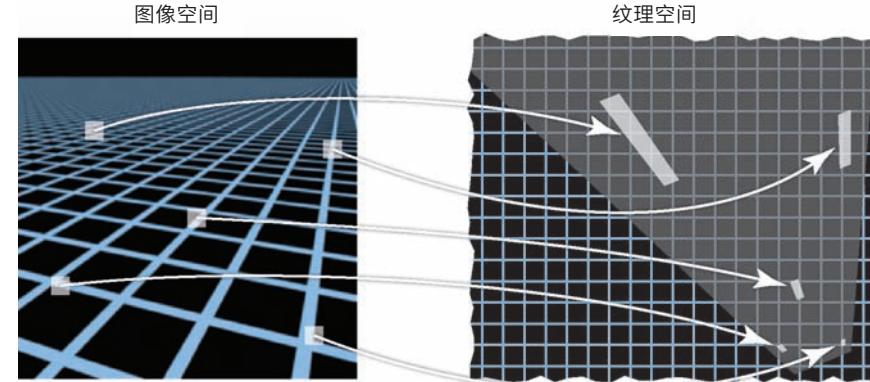
Figure 11.18. The footprints in texture space of identically sized square areas in the image vary in size and shape across the image.

Recall the three spaces involved in rendering with textures: the projection π that maps 3D points into the image and the texture coordinate function ϕ that maps 3D points into texture space. To work with pixel footprints we need to understand the composition of these two mappings: first follow π backwards to get from the image to the surface, then follow ϕ forwards. This composition $\psi = \phi \circ \pi^{-1}$ is what determines pixel footprints: the footprint of a pixel is the image of that pixel's square area of the image under the mapping ψ .

The core problem in texture antialiasing is computing an average value of the texture over the footprint of a pixel. To do this exactly in general could be a pretty complicated job: for a faraway object with a complicated surface shape, the footprint could be a complicated shape covering a large area, or possibly several disconnected areas, in texture space. But in the typical case, a pixel lands in a smooth area of surface that is mapped to a single area in the texture.

Because ψ contains both the mapping from image to surface and the mapping from surface to texture, the size and shape of the footprint depend on both the viewing situation and the texture coordinate function. When a surface is closer to the camera, pixel footprints will be smaller; when the same surface moves farther away, the footprint gets bigger. When surfaces are viewed at an oblique angle, the footprint of a pixel on the surface is elongated, which usually means it will be elongated in texture space also. Even with a fixed view, the texture coordinate function can cause variations in the footprint: if it distorts area, the size of footprints will vary, and if it distorts shape, they can be elongated even for head-on views of the surface.

However, to find an efficient algorithm for computing antialiased lookups, some substantial approximations will be needed. When a function is smooth,



图像中大小相同的正方形区域在纹理空间中的轮廓线在图像的大小和形状上有所不同。

回想一下用纹理渲染涉及的三个空间：将3D点映射到图像中的投影 π 和将3D点映射到纹理空间中的纹理坐标函数 ϕ 。要处理像素封装，我们需要了解这两个映射的组成：首先向后跟随 π 从图像到表面，然后向前跟随 ϕ 。此组成 $\Delta = \phi \Delta \pi^{-1}$

是什么决定像素足迹：一个像素的足迹是该像素的正方形区域的图像在映射关系下的图像。

纹理抗锯齿的核心问题是计算纹理在像素足迹上的平均值。一般来说，要做到这一点可能是一项非常复杂的工作：对于具有复杂表面形状的遥远物体，足迹可能是一个复杂的形状，覆盖纹理空间中的大面积，或可能但在典型情况下，像素会降落在表面的平滑区域中，该区域映射到纹理中的单个区域。

因为 ψ 既包含从图像到表面的映射，也包含从表面到纹理的映射，所以足迹的大小和形状取决于观看情况和纹理坐标函数。当一个表面离相机更近时，像素足迹会更小；当同一个表面移动得更远时，足迹会变大。当以斜角观察曲面时，像素在曲面上的足迹是拉长的，这通常意味着它在纹理空间中也会拉长。即使使用固定视图，纹理坐标函数也会导致足迹的变化：如果它扭曲了区域，足迹的大小也会变化，如果它扭曲了形状，即使对于表面的正面视图，它们也可以被拉长。

然而，要找到计算抗锯齿查找的有效算法，需要进行一些实质性的近似。当一个函数是平滑的

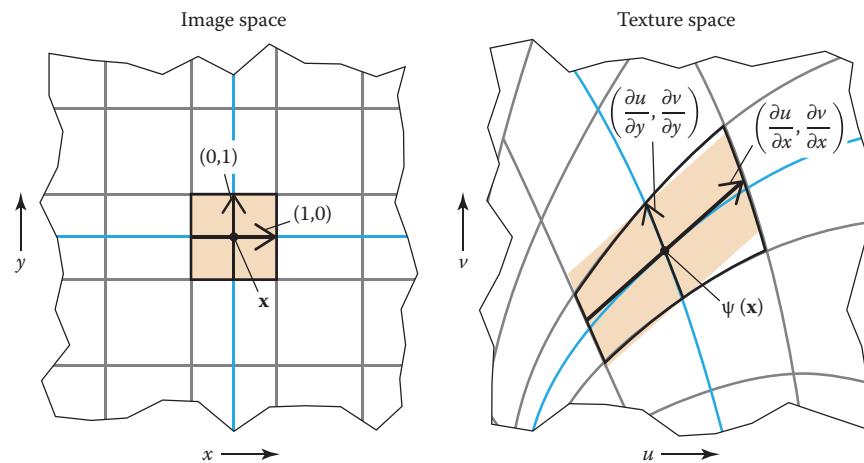


Figure 11.19. An approximation of the texture-space footprint of a pixel can be made using the derivative of the mapping from (x, y) to (u, v) . The partial derivatives with respect to x and y are parallel to the images of the x and y isolines (blue) and span a parallelogram (shaded in orange) that approximates the curved shape of the exact footprint (outlined in black).

a linear approximation is often useful. In the case of texture antialiasing, this means approximating the mapping ψ from image space to texture space as a linear mapping from 2D to 2D:

$$\psi(\mathbf{x}) = \psi(\mathbf{x}_0) + \mathbf{J}(\mathbf{x} - \mathbf{x}_0),$$

where the 2-by-2 matrix \mathbf{J} is some approximation to the derivative of ψ . It has four entries, and if we denote the image-space position as $\mathbf{x} = (x, y)$ and the texture-space position as $\mathbf{u} = (u, v)$ then

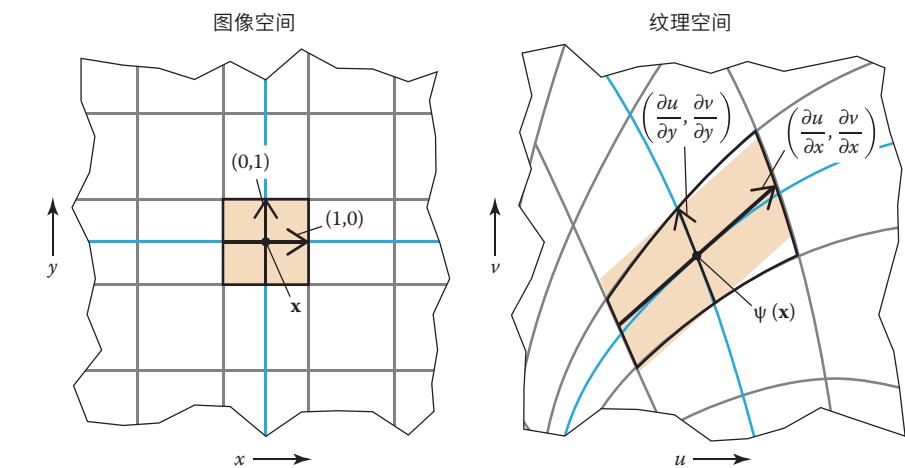
$$\mathbf{M} = \begin{bmatrix} \frac{du}{dx} & \frac{du}{dy} \\ \frac{dv}{dx} & \frac{dv}{dy} \end{bmatrix},$$

where the four derivatives describe how the texture point (u, v) that is seen at a point (x, y) in the image changes when we change x and y .

A geometric interpretation of this approximation is that it says a unit-sized square pixel area centered at \mathbf{x} in the image will map approximately to a parallelogram in texture space, centered at $\psi(\mathbf{x})$ and with its edges parallel to the vectors $\mathbf{u}_x = (du/dx, dv/dx)$ and $\mathbf{u}_y = (du/dy, dv/dy)$.

The derivative matrix \mathbf{J} is useful because it tells the whole story of variation in the (approximated) texture-space footprint across the image. Derivatives that are larger in magnitude indicate larger texture-space footprints, and the relationship between the derivative vectors \mathbf{u}_x and \mathbf{u}_y indicates the shape. When they

In mathematicians' terms, we have made a one-term Taylor series approximation to the function ψ .



可以使用从 (x, y) 到 (u, v) 的映射的导数来近似像素的纹理空间足迹。相对于 x 和 y 的偏导数与 x 和 y 等值线（蓝色）的图像平行，并跨越一个平行四边形（橙色阴影），该平行四边形近似于精确足迹的曲线形状（黑色轮廓）。

线性近似通常是有用的。在纹理抗锯齿的情况下，这意味着将从图像空间到纹理空间的映射关系近似为从2D到2D的线性映射：

$$\psi(\mathbf{x}) = \psi(\mathbf{x}_0) + \mathbf{J}(\mathbf{x} - \mathbf{x}_0),$$

其中2乘2矩阵J是对 ψ 导数的一些近似。它有四个条目，如果我们将图像空间位置表示为 $\mathbf{x}=(x\ y)$ ，纹理空间位置表示为 $\mathbf{u}=(u\ v)$ ，那么

$$\mathbf{M} = \begin{bmatrix} \frac{du}{dx} & \text{杜} \\ \frac{dv}{dx} & \frac{dv}{dy} \end{bmatrix},$$

其中四个导数描述了当我们改变 x 和 y 时，在图像中的点 (x, y) 处看到的纹理点 (u, v) 如何变化。

这种近似的几何解释是，它说图像中以 x 为中心的单位大小的正方形像素区域将近似映射到纹理空间中的平行ogram，以 $\psi(x)$ 为中心，其边缘平行于向量 $u_x = (dudx, dvdx)$ 和 $u_y = (dudy, dvdy)$ 。

导数矩阵J很有用，因为它讲述了图像中（近似的）纹理空间足迹变化的整个故事。量值较大的导数表示较大的纹理空间足迹，而导数向量 u_x 和 u_y 之间的关系能表示形状。当他们



are orthogonal and the same length, the footprint is square, and as they become skewed and/or very different in length, the footprint becomes elongated.

We've now reached the form of the problem that's usually thought of as the "right answer": a *filtered texture sample* at a particular image-space position should be the average value of the texture map over the parallelogram-shaped footprint defined by the texture coordinate derivatives at that point. This already has some assumptions baked into it—namely, that the mapping from image to texture is smooth—but it is sufficiently accurate for excellent image quality. However, this parallelogram area average is already too expensive to compute exactly, so various approximations are used. Approaches to texture antialiasing differ in the speed/quality tradeoffs they make in approximating this lookup. We discuss these in the following sections.

11.3.2 Reconstruction

When the footprint is smaller than a texel, we are magnifying the texture as it is mapped into the image. This case is analogous to upsampling an image, and the main consideration is interpolating between texels to produce a smooth image in which the texel grid is not obvious. Just as in image upsampling, this smoothing process is defined by a reconstruction filter that is used to compute texture samples at arbitrary locations in texture space. (See Figure 11.20.)

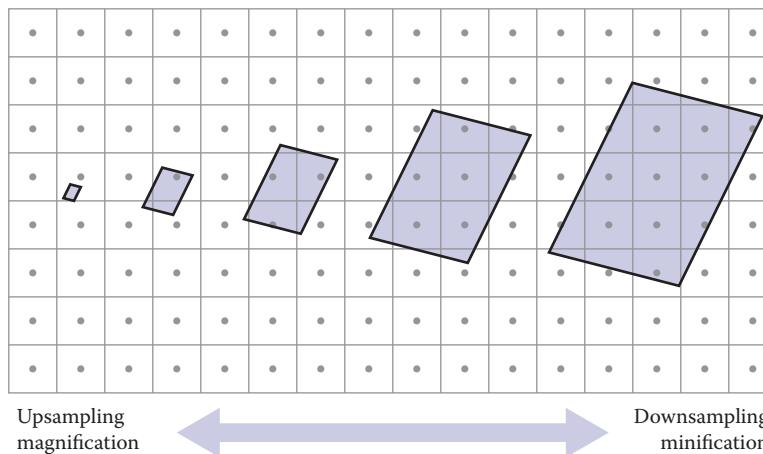


Figure 11.20. The dominant issues in texture filtering change with the footprint size. For small footprints (left) interpolating between pixels is needed to avoid blocky artifacts; for large footprints, the challenge is to efficiently find the average of many pixels.



是正交的和相同的长度，足迹是正方形的，并且随着它们变得偏斜和或长度非常不同，足迹变得细长。

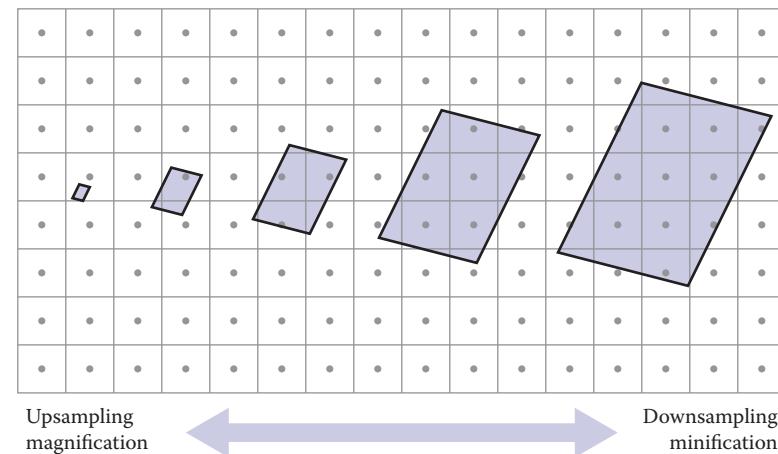
我们现在已经达到了通常被认为是"正确答案"的问题的形式：特定图像空间位置的过滤纹理样本应该是纹理映射在该点由纹理坐标导数定义的平行四边形足迹上的平均值。这已经

有一些假设—即从图像到纹理的映射是平滑的—但对于出色的图像质量来说，它足够准确。然而，这个平行四边形面积平均值已经太昂贵而无法精确计算，因此使用了各种近似值。纹理抗锯齿的方法不同于它们在近似查找时所做的速度质量权衡。我们将在下面的章节中讨论这些问题。

这里的方法使用框过滤器来采样im年龄。
Stead中的一些系统使用高斯像素滤波器，其在texture空间中变为椭圆高斯；
这是埃利普
tical weighted averaging (EWA).

11.3.2 Reconstruction

当足迹小于纹素时，我们正在放大纹理，因为它被映射到图像中。这种情况类似于对图像进行上采样，主要考虑的是在纹素之间进行插值以产生其中纹素网格不明显的平滑图像。正如在图像上采样中一样，此平滑过程由重建滤波器定义，该滤波器用于计算纹理空间中任意位置的纹理样本。（见图11.20。）



纹理过滤中的主要问题会随着足迹大小而变化。对于小脚印（左），需要在像素之间插值以避免块状伪影；对于大脚印，挑战在于有效地找到许多像素的平均值。

The considerations are pretty much the same as in image resampling, with one important difference. In image resampling, the task is to compute output samples on a regular grid, and that regularity enabled an important optimization in the case of a separable reconstruction filter. In texture filtering, the pattern of lookups is not regular, and the samples have to be computed separately. This means large, high-quality reconstruction filters are very expensive to use, and for this reason the highest-quality filter normally used for textures is bilinear interpolation.

The calculation of a bilinearly interpolated texture sample is the same as computing one pixel in an image being upsampled with bilinear interpolation. First we express the texture-space sample point in terms of (real-valued) texel coordinates, then we read the values of the four neighboring texels and average them. Textures are usually parameterized over the unit square, and the texels are located in the same way as pixels in any image, spaced a distance $1/n_u$ apart in the u direction and $1/n_v$ in v , with texel (0,0) positioned half a texel in from the edge for symmetry. (See Chapter 9 for the full explanation.)

```
Color tex_sample_bilinear(Texture t, float u, float v) {
    u_p = u * t.width - 0.5
    v_p = v * t.height - 0.5
    iu0 = floor(u_p); iu1 = iu0 + 1
    iv0 = floor(v_p); iv1 = iv0 + 1
    a_u = (iu1 - u_p); b_u = 1 - a_u
    a_v = (iv1 - v_p); b_v = 1 - a_v
    return a_u * a_v * t[iu0][iv0] + a_u * b_v * t[iu0][iv1] +
           b_u * a_v * t[iu1][iv0] + b_u * b_v * t[iu1][iv1]
}
```

In many systems, this operation becomes an important performance bottleneck, mainly because of the memory latency involved in fetching the four texel values from the texture data. The pattern of sample points for textures is irregular, because the mapping from image to texture space is arbitrary, but often coherent, since nearby image points tend to map to nearby texture points that may read the same texels. For this reason, high-performance systems have special hardware devoted to texture sampling that handles interpolation and manages caches of recently used texture data to minimize the number of slow data fetches from the memory where the texture data is stored.

After reading Chapter 9 you may complain that linear interpolation may not be a smooth enough reconstruction for some demanding applications. However, it can always be made good enough by resampling the texture to a somewhat higher resolution using a better filter, so that the texture is smooth enough that bilinear interpolation works well.

考虑因素与图像重采样几乎相同，但有一个重要区别。在图像重采样中，任务是计算规则网格上的输出样本，而在可分离重建滤波器的情况下，正则性是一个重要的优化。在纹理过滤中，查找的pattern不是规则的，样本必须单独计算。这意味着大的、高质量的重建滤波器使用起来非常昂贵，并且由于这个原因，通常用于纹理的最高质量滤波器是双线性插值。

双线性插值纹理样本的计算与使用双线性插值上采样的图像中的一个像素的计算相同。首先，我们用（实值）纹素坐标表示纹理空间样本点，然后读取四个相邻纹素的值并对它们进行平均。纹理通常在单位正方形上被参数化，并且纹素与任何图像中的像素位于相同的位置，在 u 方向上间隔 $1/n_u$ ，在 v 中间隔 $1/n_v$ ，纹素(0,0)从边缘定位半（完整解释见第9章。）

```
Colortex_sample_bilinear(Texture t, float u, float v) {
    u_p = u * t.width - 0.5
    v_p = v * t.height - 0.5
    iu0 = floor(u_p); iu1 = iu0 + 1
    iv0 = floor(v_p); iv1 = iv0 + 1
    a_u = (iu1 - u_p); b_u = 1 - a_u
    a_v = (iv1 - v_p); b_v = 1 - a_v
    return a_u * a_v * t[iu0][iv0] + a_u * b_v * t[iu0][iv1] +
           b_u * a_v * t[iu1][iv0] + b_u * b_v * t[iu1][iv1]
}
```

在许多系统中，此操作成为一个重要的性能瓶颈，主要是因为从纹理数据中获取四个纹素值所涉及的内存延迟。纹理的样本点的模式是不规则的，因为从图像到纹理空间的映射是任意的，但通常是连贯的，因为附近的图像点倾向于映射到可能读取相同纹理的附近。由于这个原因，高性能系统有专门用于纹理采样的特殊硬件，它处理插值并管理重新使用的纹理数据的缓存，以最大限度地减少从存储纹理数据的内存中提取缓慢数据的次数。

在阅读第9章之后，您可能会抱怨线性插值对于某些要求苛刻的应用来说可能不够平滑。但是，通过使用更好的滤波器将纹理重新采样到更高的分辨率，总是可以使其足够好，以便纹理足够平滑，以便双线性插值工作良好。



11.3.3 Mipmapping

Doing a good job of interpolation only suffices in situations where the texture is being magnified: where the pixel footprint is small compared to the spacing of texels. When a pixel footprint covers many texels, good antialiasing requires computing the average of many texels to smooth out the signal so that it can be sampled safely.

One very accurate way to compute the average texture value over the footprint would be to find all the texels within the footprint and add them up. However, this is potentially very expensive when the footprint is large—it could require reading many thousands of texel just for a single lookup. A better approach is to precompute and store the averages of the texture over various areas of different size and position.

A very popular version of this idea is known as “MIP mapping” or just mipmapping. A mipmap is a sequence of textures that all contain the same image but at lower and lower resolution. The original, full-resolution texture image is called the *base level*, or level 0, of the mipmap, and level 1 is generated by taking that image and downsampling it by a factor of 2 in each dimension, resulting in an image with one-fourth as many texels. The texels in this image are, roughly speaking, averages of square areas 2 by 2 texels in size in the level-0 image.

This process can be continued to define as many mipmap levels as desired: the image at level k is computed by downsampling the image at level $k - 1$ by two. A texel at level k corresponds to a square area measuring 2^k by 2^k texels in the original texture. For instance, starting with a 1024×1024 texture image, we could generate a mipmap with 11 levels: level 0 is 1024×1024 ; level 1 is 512×512 , and so on until level 10, which has just a single texel. This kind of structure, with images that represent the same content at a series of lower and lower sampling rates, is called an *image pyramid*, based on the visual metaphor of stacking all the smaller images on top of the original.

The name “mip” stands for the Latin phrase *multim in parvo* meaning “much in a small space.”

11.3.4 Basic Texture Filtering with Mipmaps

With the mipmap, or image pyramid, in hand, texture filtering can be done much more efficiently than by accessing many texels individually. When we need a texture value averaged over a large area, we simply use values from higher levels of the mipmap, which are already averages over large areas of the image. The simplest and fastest way to do this is to look up a single value from the mipmap, choosing the level so that the size covered by the texels at that level is roughly the same as the overall size of the pixel footprint. Of course, the pixel footprint might



11.3.3 Mipmapping

只有在纹理被放大的情况下，做好插值就足够了：与纹理的间距相比，像素足迹很小。当一个像素足迹复盖许多纹素时，良好的抗锯齿需要计算许多纹素的平均值来平滑信号，以便可以安全地对其进行采样。

计算足迹平均纹理值的一个非常准确的方法是找到足迹内的所有纹素并将它们相加。但是，当占用空间很大时，这可能非常昂贵—只需一次查找就可能需要读取数千个texel。更好的方法是预先计算和存储纹理在不同大小和位置的不同区域上的平均值。

这个想法的一个非常流行的版本被称为“MIP映射”或只是mip映射。Mipmap是一系列纹理，它们都包含相同的图像，但分辨率越来越低。原始的全分辨率纹理图像称为mipmap的基本级别或级别0，级别1是通过获取该图像并在每个维度上对其进行2因子的下采样生成的，从而生成具有四分之粗略地说，此图像中的纹素是0级图像中大小为 2×2 纹素的正方形区域的平均值。

这个过程可以根据需要继续定义尽可能多的mipmap级别：通过将级别 $k - 1$ 的图像降采样2来计算级别 k 的图像。级别 k 的纹素对应于原始纹理中 $2^k \times 2^k$ 纹素的正方形区域。例如，从 1024×1024 的纹理图像开始，我们可以生成一个具有11个级别的mipmap：级别0是 1024×1024 ；级别1是 512×512 ，依此类推，直到级别10，只有一个texel。这种结构，以一系列较低和较低的采样率表示相同内容的图像，称为图像金字塔，基于将所有较小的图像堆叠在原始之上的视觉隐喻。

“Mip”这个名字代表parvo中的拉丁短语，意思是“在一个小空间里很多。”

11.3.4 使用Mipmap进行基本纹理过滤

有了mipmap或图像金字塔，纹理过滤可以比单独访问许多纹素更有效地完成。当我们需要在大面积上平均纹理值时，我们只需使用来自mipmap更高级别的值，这些值已经是图像大面积上的平均值。最简单和最快的方法是从mipmap中查找单个值，选择级别，以便该级别的纹素所复盖的大小与像素足迹的整体大小大致相同。当然，像素足迹可能

be quite different in shape from the (always square) area represented by the texel, and we can expect that to produce some artifacts.

Setting aside for a moment the question of what to do when the pixel footprint has an elongated shape, suppose the footprint is a square of width D , measured in terms of texels in the full-resolution texture. What level of the mipmap is it appropriate to sample? Since the texels at level k cover squares of width 2^k , it seems appropriate to choose k so that

$$2^k \approx D$$

so we let $k = \log_2 D$. Of course this will give non-integer values of k most of the time, and we only have stored mipmap images for integer levels. Two possible solutions are to look up a value only for the integer nearest to k (efficient but produces seams at the abrupt transitions between levels) or to look up values for the two nearest integers to k and linearly interpolate the values (twice the work, but smoother).

Before we can actually write down the algorithm for sampling a mipmap, we have to decide how we will choose the “width” D when footprints are not square. Some possibilities might be to use the square root of the area or to find the longest axis of the footprint and call that the width. A practical compromise that is easy to compute is to use the length of the longest edge:

$$D = \max \{\|\mathbf{u}_x\|, \|\mathbf{u}_y\|\}.$$

```
Color mipmap_sample_trilinear(Texture mip[], float u, float v,
    matrix J) {
    D = max_column_norm(J)
    k = log2(D)
    k0 = floor(k); k1 = k0 + 1
    a = k1 - k; b = 1 - a
    c0 = tex_sample_bilinear(mip[k0], u, v)
    c1 = tex_sample_bilinear(mip[k1], u, v)
    return a * c0 + b * c1
}
```

Basic mipmapping does a good job of removing aliasing, but because it's unable to handle elongated, or *anisotropic* pixel footprints, it doesn't perform well when surfaces are viewed at grazing angles. This is most commonly seen on large planes that represent a surface the viewer is standing on. Points on the floor that are far away are viewed at very steep angles, resulting in very anisotropic footprints that mipmapping approximates with much larger square areas. The resulting image will appear blurred in the horizontal direction.

在形状上与纹素表示的（总是正方形）区域有很大的不同，我们可以预期会产生一些伪像。

暂且撇开当像素足迹具有细长形状时该怎么做的问题，假设足迹是宽度D的正方形，以全分辨率纹理中的纹素衡量。什么级别的mipmap适合采样？由于级别k的纹素覆盖宽度为 2^k 的正方形，因此选择k似乎是合适的，以便

$$2^k \approx D$$

所以我们让 $k=\log_2 D$ 。当然，这将在大多数情况下给出k的非整数值，并且我们只存储了整数级别的mipmap图像。两种可能的解决方案是只查找离k最近的整数的值（效率高，但在水平之间的突然转换时产生接缝），或者查找离k最近的两个整数的值，并线性插值这些值（两倍的工作，但更平滑）。

在我们实际写下采样mipmap的算法之前，我们必须决定当足迹不是正方形时如何选择“宽度”D。一些可能性可能是使用面积的平方根或找到足迹的最长轴并将其称为宽度。一个易于计算的实用折衷方案是使用最长边的长度：

$$D = \max \{\|\mathbf{u}_x\|, \|\mathbf{u}_y\|\}.$$

```
Color mipmap_sample_trilinear(TextureMip[] floatu floatv matrixJ){D=max_column_norm(J)k=log2(D)k0=floor(k);k1=k0+1;a=k1-k;b=1-ac0=tex_sample_bilinear(mip[k0] u v)c1=tex_sample_bilinear(mip[k1] u v)返回a*c0+b*c1}
```

基本的mipmapping在消除混叠方面做得很好，但是由于它无法处理细长的或各向异性的像素足迹，因此在以掠角观察表面时表现不佳。这是最常见的大平面，代表一个表面的观众站在。在地板上较远的点以非常陡峭的角度观察，从而产生非常各向异性的足迹，mipmapping近似于更大的正方形区域。生成的图像在水平方向上会显得模糊。

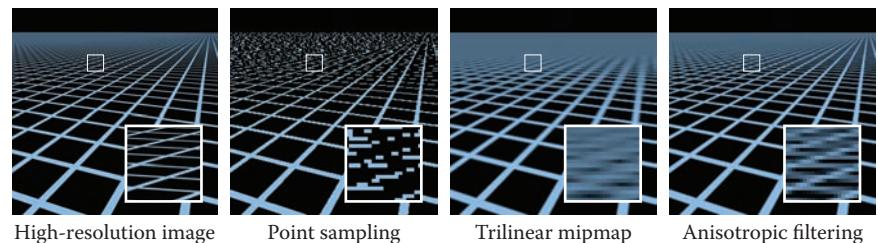


Figure 11.21. The results of antialiasing a challenging test scene (reference images showing detailed structure, at left) using three different strategies: simply taking a single point sample with nearest-neighbor interpolation; using a mipmap pyramid to average a square area in the texture for each pixel; using several samples from a mipmap to average an anisotropic region in the texture.

11.3.5 Anisotropic Filtering

A mipmap can be used with multiple lookups to approximate an elongated footprint better. The idea is to select the mipmap level based on the *shortest* axis of the footprint rather than the largest, then average together several lookups spaced along the long axis. (See Figure 11.21.)

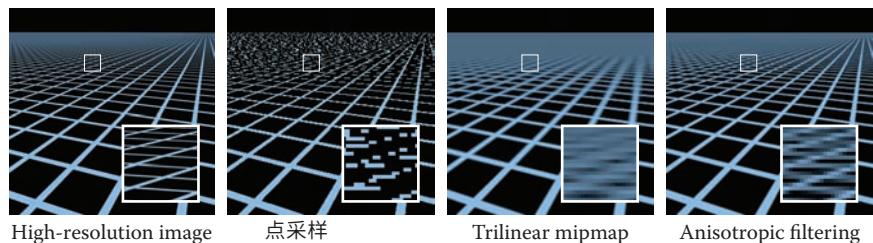
11.4 Applications of Texture Mapping

Once you understand the idea of defining texture coordinates for a surface and the machinery of looking up texture values, this machinery has many uses. In this section we survey a few of the most important techniques in texture mapping, but textures are a very general tool with applications limited only by what the programmer can think up.

11.4.1 Controlling Shading Parameters

The most basic use of texture mapping is to introduce variation in color by making the diffuse color that is used in shading computations—whether in a ray tracer or in a fragment shader—dependent on a value looked up from a texture. A textured diffuse component can be used to paste decals, paint decorations, or print text on a surface, and it can also simulate the variation in material color, for example for wood or stone.

Nothing limits us to varying only the diffuse color, though. Any other parameters, such as the specular reflectance or specular roughness, can also be textured. For instance, a cardboard box with transparent packing tape stuck to it may have the same diffuse color everywhere but be shinier, with higher specular reflectance



使用三种不同的策略对具有挑战性的测试场景（左侧显示详细结构的参考图像）进行抗锯齿的结果：简单地使用最近邻插值的单点样本；使用mipmap金字塔为每个像素平均纹理中的正方形区域；使用来自mipmap的多个样本平均纹理中的各向异性区域。

11.3.5 Anisotropic Filtering

一个mipmap可以与多个查找一起使用，以更好地近似一个细长的足迹。这个想法是根据足迹的最短轴而不是最大轴来选择mipmap级别，然后将沿着长轴间隔的几个查找平均在一起。（见图11.21。）

11.4 纹理映射的应用

一旦您理解了为曲面定义纹理坐标的想法和查找纹理值的机制，这种机制就有很多用途。在本节中，我们调查了纹理映射中一些最重要的技术，但是纹理是一个非常通用的工具，应用程序只受程序员所能想到的限制。

11.4.1 控制着色参数

纹理映射的最基本用途是通过使着色计算中使用的漫反射颜色（无论是在光线跟踪器中还是在片段着色器中）依赖于从纹理中查找的值来引入颜色变纹理漫反射组件可用于在表面上粘贴贴花、绘制装饰或打印文本，还可以模拟材料颜色的变化，例如木材或石头。

不过，没有什么能限制我们只改变漫反射颜色。任何其他参数，如镜面反射率或镜面粗糙度，也可以被纹理化。例如，贴有透明包装带的纸板箱可能到处都有相同的漫射颜色，但更闪亮，镜面反射率更高



Figure 11.22. A ceramic mug with specular roughness controlled by an inverted copy of the diffuse color texture.

11.4.2 Normal Maps and Bump Maps

Another quantity that is important for shading is the surface normal. With interpolated normals (Section 8.2), we know that the shading normal does not have to be the same as the geometric normal of the underlying surface. *Normal mapping* takes advantage of this fact by making the shading normal depend on values read from a texture map. The simplest way to do this is just to store the normals in a texture, with three numbers stored at every texel that are interpreted, instead of as the three components of a color, as the 3D coordinates of the normal vector.

Before a normal map can be used, though, we need to know what coordinate system the normals read from the map are represented in. Storing normals directly in object space, in the same coordinate system used for representing the surface geometry itself, is simplest: the normal read from the map can be used in exactly the same way as the normal reported by the surface itself: in most cases it will need to be transformed into world space for lighting calculations, just like a normal that came with the geometry.

However, normal maps that are stored in object space are inherently tied to the surface geometry—even for the normal map to have no effect, to reproduce the result with the geometric normals, the contents of the normal map have to track the orientation of the surface. Furthermore, if the surface is going to deform, so that the geometric normal changes, the object-space normal map can no longer be used, since it would keep providing the same shading normals.

The solution is to define a coordinate system for the normals that is attached to the surface. Such a coordinate system can be defined based on the tangent space of the surface (see Section 2.5): select a pair of tangent vectors and use them to define an orthonormal basis (Section 2.4.5). The texture coordinate function itself provides a useful way to select a pair of tangent vectors: use the directions tangent to lines of constant u and v . These tangents are not generally orthogonal, but we can use the procedure from Section 2.4.7 to “square up” the orthonormal basis, or it can be defined using the surface normal and just one tangent vector.

When normals are expressed in this basis they vary a lot less; since they are mostly pointing near the direction of the normal to the smooth surface, they will be near the vector $(0, 0, 1)^T$ in the normal map.



具有镜面粗糙度的陶瓷杯，由漫反射颜色纹理的倒置副本控制。

和较低的粗糙度，其中胶带比其他地方。在许多情况下，不同参数的地图是相关的：例如，一个印有标志的光滑的白色陶瓷杯在印刷的地方可能会更粗糙和更暗（图11.22），而一本用金属墨水印刷的书名可能会同时改变漫射颜色、镜面颜色和粗糙度。

11.4.2 法线贴图和凹凸贴图

另一个对阴影很重要的量是表面法线。对于间极化法线（第8.2节），我们知道阴影法线不必与基础表面的几何法线相同。法线贴图通过使着色法线依赖于从纹理贴图读取的值来利用这一事实。最简单的方法是将法线存储在纹理中，在每个纹素中存储三个数字，而不是作为颜色的三个分量，作为法向量的3D坐标。

但是，在使用法线贴图之前，我们需要知道从地图读取的法线在哪个坐标系中表示。将法线直接存储在对象空间中，在用于表示表面几何本身的同一坐标系中，是最简单的：从地图读取的法线可以与表面本身报告的法线完全相同的方式使用：在大多数情况下，它需要转换到世界空间进行照明计算，就像几何附带的法线一样。

但是，存储在对象空间中的法线贴图固有地与表面几何相关联—即使法线贴图没有效果，为了用几何法线重现结果，法线贴图的内容必须跟踪表面的方向。此外，如果表面发生变形，从而导致几何法线发生变化，则不能再使用对象空间法线贴图，因为它将继续提供相同的着色法线。

解决方案是为附加到曲面的法线定义一个坐标系。可以基于曲面的切线空间来定义这样的坐标系（参见第2.5节）：选择一对切向量并使用它们来定义正交基（第2.4.5节）。纹理坐标函数本身提供了一种选择一对切向量的有用方法：使用与常数 u 和 v 的线相切的方向。这些切线通常不是正交的，但是我们可以使用第2.4.7节中的过程来“平方”正交基，或者可以使用表面法线和一个切向矢量来定义它。

当法线在此基础上表示时，它们的变化要小得多；由于它们主要指向法线到光滑表面的方向附近，因此它们将靠近法线贴图中的矢量 $(0\ 0\ 1)^T$ 。

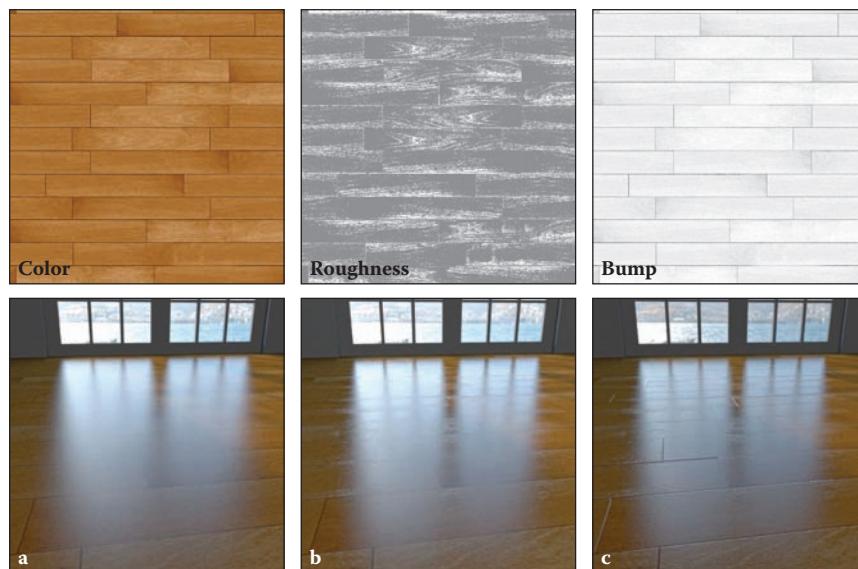


Figure 11.23. A wood floor rendered using texture maps to control the shading. (a) Only the diffuse color is modulated by a texture map. (b) The specular roughness is also modulated by a second texture map. (c) The surface normal is modified by a bump map.

Where do normal maps come from? Often they are computed from a more detailed model to which the smooth surface is an approximation; other times they can be measured directly from real surfaces. They can also be authored as part of the modeling process; in this case it's often nice to use a *bump map* to specify the normals indirectly. The idea is that a bump map is a height field: a function that give the local height of the detailed surface above the smooth surface. Where the values are high (where the map looks bright, if you display it as an image) the surface is protruding outside the smooth surface; where the values are low (where the map looks dark) the surface is receding below it. For instance, a narrow dark line in the bump map is a scratch, or a small white dot is a bump.

Deriving a normal map from a bump map is simple: the normal map (expressed in the tangent frame) is the derivative of the bump map.

Figure 11.23 shows texture maps being used to create woodgrain color and to simulate increased surface roughness due to finish soaking into the more porous parts of the wood, together with a bump map to create an imperfect finish and gaps between boards, to make a realistic wood floor.

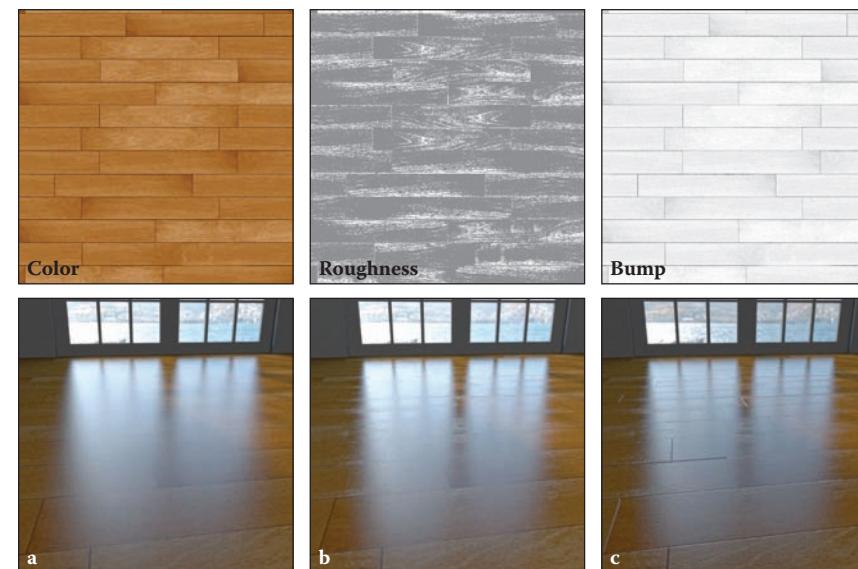


图11.23。 使用纹理贴图来控制阴影的木地板。(a)只有漫反射颜色由纹理贴图调制。(b)镜面粗糙度也由第二纹理图调制。(c)通过凹凸贴图修改表面法线。

法线贴图从何而来？通常，它们是从一个更详细的模型计算出来的，光滑表面是一个近似值；其他时候，它们可以直接从真实表面测量。它们也可以作为建模过程的一部分编写；在这种情况下，使用凹凸贴图间接指定法线通常是很好的。这个想法是，凹凸贴图是一个高度字段：一个函数，给出光滑表面上方的详细表面的局部高度。如果值高（如果将地图显示为图像，则地图看起来很亮），则表面突出到光滑表面之外；如果值低（地图看起来很暗），则表面在其下方后退。例如，凹凸贴图中的窄暗线是划痕，或小白点是凹凸。

从凹凸贴图导出法线贴图很简单：法线贴图（ex按在切线框中）是凹凸贴图的导数。

图11.23显示了用于创建木纹颜色的纹理图，并模拟由于木材的多孔部分浸透而增加的表面粗糙度，以及一个凹凸图，以创建一个不完美的表面和板之间的间隙，以制作一个真实的木地板。

11.4.3 Displacement Maps

A problem with normal maps is that they don't actually change the surface at all; they are just a shading trick. This becomes obvious when the geometry implied by the normal map should cause noticeable effects in 3D. In still images, the first problem to be noticed is usually that the silhouettes of objects remain smooth despite the appearance of bumps in the interior. In animations, the lack of parallax gives away that the bumps, however convincing, are really just "painted" on the surface.

Textures can be used for more than just shading, though: they can be used to alter geometry. A displacement map is one of the simplest versions of this idea. The concept is the same as a bump map: a scalar (one-channel) map that gives the height above the "average terrain." But the effect is different. Rather than deriving a shading normal from the height map while using the smooth geometry, a displacement map actually changes the surface, moving each point along the normal of the smooth surface to a new location. The normals are roughly the same in each case, but the surface is different.

The most common way to implement displacement maps is to tessellate the smooth surface with a large number of small triangles, and then displace the vertices of the resulting mesh using the displacement map. In the graphics pipeline, this can be done using a texture lookup at the vertex stage, and is particularly handy for terrain.

11.4.4 Shadow Maps

Shadows are an important cue to object relationships in a scene, and as we have seen, they are simple to include in ray-traced images. However, it's not obvious how to get shadows in rasterized renderings, because surfaces are considered one at a time, in isolation. Shadow maps are a technique for using the machinery of texture mapping to get shadows from point light sources.

The idea of a shadow map is to represent the volume of space that is illuminated by a point light source. Think of a source like a spotlight or video projector, which emits light from a point into a limited range of directions. The volume that is illuminated—the set of points where you would see light on your hand if you held it there—is the union of line segments joining the light source to the closest surface point along every ray leaving that point.

Interestingly, this volume is the same as the volume that is visible to a perspective camera located at the same point as the light source: a point is illuminated by a source if and only if it is visible from the light source location. In both

法线贴图的一个问题是它们实际上根本不会改变表面;它们只是一个着色技巧。在静态图像中，首先要注意的问题通常是，尽管内部出现了凹凸，但物体的轮廓仍然是平滑的。在动画中，视差的缺乏使得颠簸，无论多么令人信服，实际上只是"画"在表面上。

纹理可以用于不仅仅是阴影，虽然：他们可以用来改变几何。位移图是这个想法的最简单版本之一。这个概念与凹凸图相同：标量（单通道）地图，给出高于"平均地形"的高度。"但效果不同。在使用平滑几何体时，位移图实际上改变了曲面，将每个点沿着平滑曲面的法线移动到一个新的位置，而不是从高度图派生着色法线。法线在每种情况下大致相同，但表面不同。

实现位移贴图的最常用方法是用大量小三角形对光滑表面进行细分，然后使用位移贴图对所得网格的vertices进行置换。在图形管道中，这可以在顶点阶段使用纹理查找来完成，对于地形尤其方便。

11.4.4 阴影地图

阴影是场景中对象关系的重要提示，正如我们所看到的，它们很容易包含在光线追踪图像中。但是，如何在光栅化渲染中获取阴影并不明显，因为表面被孤立地视为一个一个。阴影贴图是一种利用纹理映射机制从点光源获取阴影的技术。

阴影图的概念是表示由点光源照亮的空间体积。想象一个像聚光灯或视频投影仪这样的光源，它从一个点向有限的方向发射光。被照亮的体积—如果你把它放在那里，你会看到你手上的光的一组点—是沿着离开该点的每条光线将光源连接到最近的表面点的线段的联合。

有趣的是，这个体积与位于与光源相同点的透视相机可见的体积相同：当且仅当从光源位置可见时，一个点才被光源照亮。在两者中

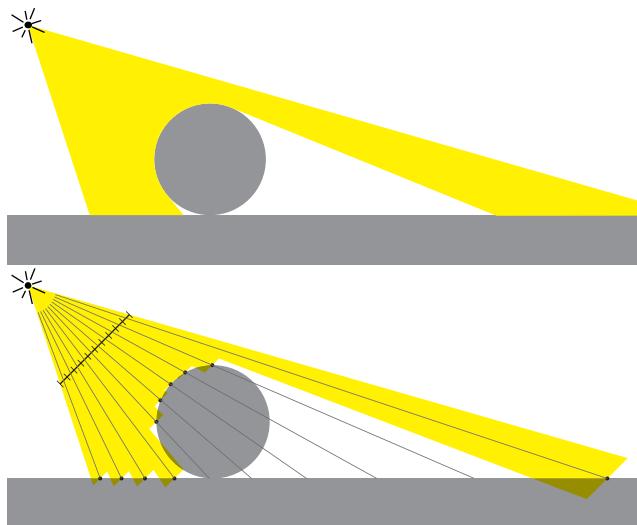
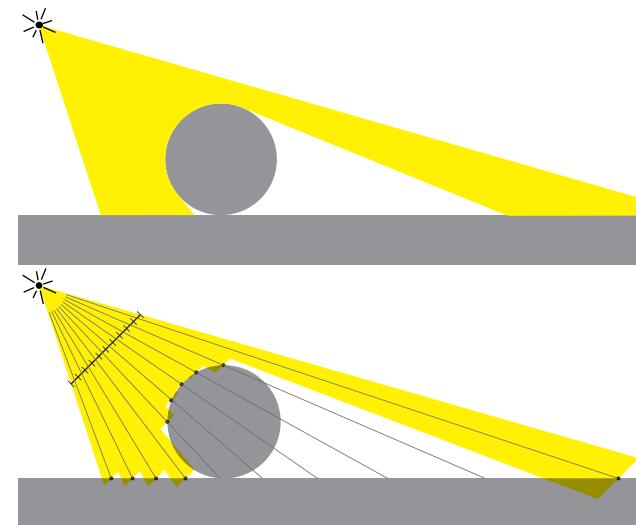


Figure 11.24. Top: the region of space illuminated by a point light. Bottom: that region as approximated by a 10-pixel-wide shadow map.

cases, there's a need to evaluate visibility for points in the scene: for visibility, we needed to know whether a fragment was visible to the camera, to know whether to draw it in the image; and for shadowing, we need to know whether a fragment is visible to the light source, to know whether it's illuminated by that source or not. (See Figure 11.24.)

In both cases, the solution is the same: a depth map that tells the distance to the closest surface along a bunch of rays. In the visibility case, this is the *z-buffer* (Section 8.2.3), and for the shadowing case, it is called a *shadow map*. In both cases, visibility is evaluated by comparing the depth of a new fragment to the depth stored in the map, and the surface is hidden from the projection point (occluded or shadowed) if its depth is greater than the depth of the closest visible surface. A difference is that the *z* buffer is used to keep track of the closest surface seen *so far* and is updated during rendering, whereas a shadow map tells the distance to the closest surface in the whole scene.

A shadow map is calculated in a separate rendering pass ahead of time: simply rasterize the whole scene as usual, and retain the resulting depth map (there is no need to bother with calculating pixel values). Then, with the shadow map in hand, you perform an ordinary rendering pass, and when you need to know whether a fragment is visible to the source, you project its location in the shadow map (using the same perspective projection that was used to render the shadow map in the first place) and compare the looked-up value d_{map} with the actual distance d to



顶部：由点光源照亮的空间区域。底部：该区域由10像素宽的阴影贴图近似。

在某些情况下，我们需要评估场景中点的可见性：对于可见性，我们需要知道一个片段是否对摄像机可见，知道是否在图像中绘制它；对于阴影，我们需要知道一个片段是否对光源可见，知道它是否被光源照亮。（见图11.24。）

在这两种情况下，解决方案都是相同的：深度图告诉沿着一堆射线到最近表面的距离。在可见性情况下，这是 z 缓冲区（第8.2.3节），对于阴影情况，它被称为阴影图。在这两种情况下，通过将新片段的深度与存储在地图中的深度进行比较来评估可见性，并且如果其深度大于最接近的可见表面的深度，则从投影点隐藏（不同之处在于， z 缓冲区用于跟踪迄今为止看到的最接近的表面，并在渲染过程中更新，而阴影贴图告诉整个场景中到最接近的表面的距离）。

阴影图是在一个单独的渲染过程中提前计算的：简单地像往常一样栅格化整个场景，并保留生成的深度图（不需要费心计算像素值）。然后，使用阴影贴图执行普通渲染过程，当需要知道片段对源是否可见时，将其在阴影贴图中的位置投影（使用最初用于渲染阴影贴图的相同透视投影），并将查找值 d_{map} 与实际距离 d 进行比较。

the source. If the distances are the same, the fragment's point is illuminated; if the $d > d_{\text{map}}$, that implies there is a different surface closer to the source, so it is shadowed.

The phrase “if the distances are the same” should raise some red flags in your mind: since all the quantities involved are approximations with limited precision, we can't expect them to be exactly the same. For visible points, the $d \approx d_{\text{map}}$ but sometimes d will be a bit larger and sometimes a bit smaller. For this reason, a tolerance is required: a point is considered illuminated if $d - d_{\text{map}} < \epsilon$. This tolerance ϵ is known as *shadow bias*.

When looking up in shadow maps it doesn't make a lot of sense to interpolate between the depth values recorded in the map. This might lead to more accurate depths (requiring less shadow bias) in smooth areas, but will cause bigger problems near shadow boundaries, where the depth value changes suddenly. Therefore, texture lookups in shadow maps are done using nearest-neighbor reconstruction. To reduce aliasing, multiple samples can be used, with the 1-or-0 shadow results (rather than the depths) averaged; this is known as *percentage closer filtering*.

11.4.5 Environment Maps

Just as a texture is handy for introducing detail into the shading on a surface without having to add more detail to the model, a texture can also be used to introduce detail into the illumination without having to model complicated light source geometry. When light comes from far away compared to the size of objects in view, the illumination changes very little from point to point in the scene. It is handy to make the assumption that the illumination depends only on the direction you look, and is the same for all points in the scene, and then to express this dependence of illumination on direction using an *environment map*.

The idea of an environment map is that a function defined over directions in 3D is a function on the unit sphere, so it can be represented using a texture map in exactly the same way as we might represent color variation on a spherical object. Instead of computing texture coordinates from the 3D coordinates of a surface point, we use exactly the same formulas to compute texture coordinates from the 3D coordinates of the unit vector that represents the direction from which we want to know the illumination.

The simplest application of an environment map is to give colors to rays in a ray tracer that don't hit any objects:

```
trace_ray(ray, scene) {
    if (surface = scene.intersect(ray)) {
```

源。如果距离相同，片段的点将被照亮；如果 $d>d_{\text{map}}$ ，这意味着有一个不同的表面更接近源，因此它被阴影。

“如果距离相同”这句话应该会在你的脑海中引发一些危险信号：由于所涉及的所有数量都是精度有限的近似值，我们不能期望它们完全相同。对于可见点， $d \in d_{\text{map}}$ ，但有时 d 会大一点，有时会小一点。出于这个原因，需要一个公差：如果 $d - d_{\text{map}} < \epsilon$ ，则认为一个点被照亮。这种公差偏差称为阴影偏差。

在阴影贴图中查找时，在地图中记录的深度值之间进行交互并不是很有意义。这可能会在平滑区域中导致更精确的深度（需要更少的阴影偏置），但会在阴影边界附近引起大的ger问题，其中深度值突然变化。因此，阴影贴图中的纹理查找是使用最近邻重构完成的。为了减少混叠，可以使用多个样本，将1或0阴影结果（而不是深度）平均；这称为接近百分比滤波。

11.4.5 环境图

就像纹理可以方便地将细节引入曲面上的阴影而无需向模型添加更多细节一样，纹理也可以用于将细节引入照明而无需对复杂的光源几何进行建模。当光线来自远处时，与视图中物体的大小相比，照明在场景中从点到点的变化很小。假设照明只取决于你看的方向，并且场景中的所有点都是一样的，然后用环境图来表达照明对方向的依赖是很方便的。

环境贴图的概念是，在3d中定义的方向上的函数是单位球体上的函数，因此可以使用纹理贴图来表示它，就像我们在球形对象上表示颜色变化一样。我们不是从表面点的3D坐标计算纹理坐标，而是使用完全相同的公式从表示我们想要知道照明方向的单位向量的3d坐标计算纹理坐标。

环境贴图最简单的应用是为光线追踪器中的光线赋予颜色，这些光线不会击中任何物体：

```
trace_ray(ray, scene) {
    if (surface = scene.intersect(ray)) {
```

```

        return surface.shade(ray)
    } else {
        u, v = spheremap_coords(r.direction)
        return texture_lookup(scene.env_map, u, v)
    }
}

```

With this change to the ray tracer, shiny objects that reflect other scene objects will now also reflect the background environment.

A similar effect can be achieved in the rasterization context by adding a mirror reflection to the shading computation, which is computed in the same way as in a ray tracer, but simply looks up directly in the environment map with no regard for other objects in the scene:

```

shade_fragment(view_dir, normal) {
    out_color = diffuse_shading(k_d, normal)
    out_color += specular_shading(k_s, view_dir, normal)
    u, v = spheremap_coords(reflect(view_dir, normal))
    out_color += k_m * texture_lookup(environment_map, u, v)
}

```

This technique is known as *reflection mapping*.

A more advanced use of environment maps computes all the illumination from the environment map, not just the mirror reflection. This is *environment lighting*, and can be computed in a ray tracer using Monte Carlo integration or in rasterization by approximating the environment with a collection of point sources and computing many shadow maps.

Environment maps can be stored in any coordinates that could be used for mapping a sphere. Spherical (longitude–latitude) coordinates are one popular option, though the compression of texture at the poles wastes texture resolution and can create artifacts at the poles. Cubemaps are a more efficient choice, widely used in interactive applications (Figure 11.25).

11.5 Procedural 3D Textures

In previous chapters, we used c_r as the diffuse reflectance at a point on an object. For an object that does not have a solid color, we can replace this with a function $c_r(p)$ which maps 3D points to RGB colors (Peachey, 1985; Perlin, 1985). This function might just return the reflectance of the object that contains p . But for objects with *texture*, we should expect $c_r(p)$ to vary as p moves across a surface.

An alternative to defining texture mapping functions that map from a 3D surface to a 2D texture domain is to create a 3D texture that defines an RGB value at

```

返回表面。阴影(射线){
    u, v = spheremap_coords(r.direction)
    return texture_lookup(scene.env_map, u, v)
}

```

通过对光线追踪器的这种更改，反射其他场景对象的闪亮对象现在也将反映背景环境。

在光栅化环境中，也可以通过在阴影计算中添加镜像反射来实现类似的效果，阴影计算的计算方式与光线追踪器的计算方式相同，但只是直接在环境图中查找，而不考虑场景中的其他对象：

```

shade_fragment(view_dir, normal) {
    out_color = diffuse_shading(k_d, normal)
    out_color += specular_shading(k_s, view_dir, normal)
    u, v = spheremap_coords(reflect(view_dir, normal))
    out_color += k_m * texture_lookup(environment_map, u, v)
}

```

这种技术被称为反射映射。

更高级的环境图使用计算来自环境图的所有照明，而不仅仅是镜面反射。这是环境照明，可以使用蒙特卡罗积分在光线追踪器中计算，也可以通过使用点源集合近似环境并计算许多阴影图来计算光栅化。

环境贴图可以存储在可用于绘制球体的任何坐标中。球形（经度 纬度）坐标是一种流行的选择，尽管在两极压缩纹理会浪费纹理分辨率，并且会在两极产生伪像。立方体贴图是一种更有效的选择，广泛用于交互式应用程序（图11.25）。

11.5 程序化3d纹理

在前面的章节中，我们使用 c_r 作为物体上点的漫反射率。对于没有纯色的对象，我们可以用函数 $c_r(p)$ 替换它，该函数将3D点映射到RGB颜色 (Peachey, 1985;Perlin, 1985)。此函数可能只是返回包含 p 的对象的反射率。但是对于具有纹理的对象，我们应该期望 $c_r(p)$ 随着 p 在表面上移动而变化。定义从3D表面映射到2D纹理域的纹理映射函数的替代方法是创建一个定义RGB值的3D纹理

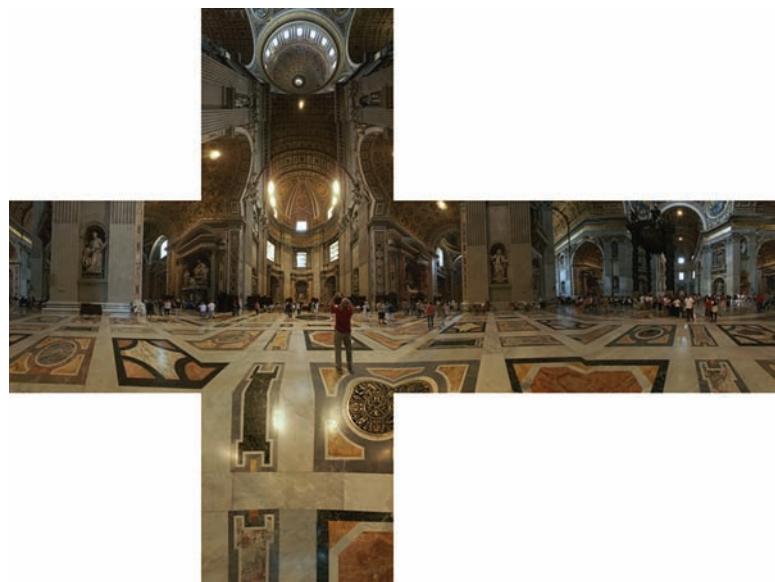


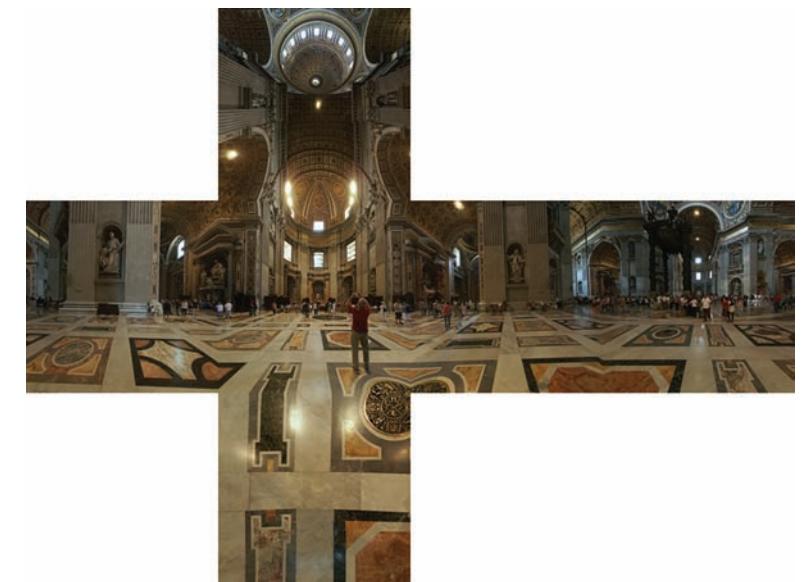
Figure 11.25. A cube map of St. Peter’s Basilica, with the six faces stored in one image in the unwrapped “horizontal cross” arrangement. (texture: Emil Persson)

every point in 3D space. We will only call it for points p on the surface, but it is usually easier to define it for all 3D points than a potentially strange 2D subset of points that are on an arbitrary surface. The good thing about 3D texture mapping is that it is easy to define the mapping function, because the surface is already embedded in 3D space, and there is no distortion in the mapping from 3D to texture space. Such a strategy is clearly suitable for surfaces that are “carved” from a solid medium, such as a marble sculpture.

The downside to 3D textures is that storing them as 3D raster images or *volumes* consumes a great deal of memory. For this reason, 3D texture coordinates are most commonly used with *procedural textures* in which the texture values are computed using a mathematical procedure rather than by looking them up from a texture image. In this section, we look at a couple of the fundamental tools used to define procedural textures. These could also be used to define 2D procedural textures, though in 2D it is more common to use raster texture images.

11.5.1 3D Stripe Textures

There are a surprising number of ways to make a striped texture. Let’s assume we have two colors c_0 and c_1 that we want to use to make the stripe color. We need



圣彼得大教堂的立方体地图，六个面以展开的“水平十字架”排列存储在图像中。（纹理：EmilPersson）

3d空间中的每个点。我们将只为表面上的点 p 调用它，但通常为所有3D点定义它比在任意表面上的点的潜在奇怪的2d子集更容易。3D纹理映射的好处是很容易定义映射函数，因为表面已经嵌入到3D空间中，并且在从3d到texture空间的映射中没有失真。这样的策略显然适用于从固体介质“雕刻”的表面，例如大理石雕塑。

3d纹理的缺点是将它们存储为3D光栅图像或volumes会消耗大量内存。因此，3d纹理坐标最常用于过程纹理，其中纹理值是使用数学过程计算的，而不是通过从纹理图像中查找它们。在本节中，我们将介绍用于定义过程纹理的几个基本工具。这些也可用于定义2D过程纹理，但在2D中更常见的是使用光栅纹理图像。

11.5.1 3d条纹纹理

有很多方法可以制作条纹纹理。假设我们有两种颜色 c_0 和 c_1 ，我们想用它来制作条纹颜色。我们需要



some oscillating function to switch between the two colors. An easy one is a sine:

```
RGB stripe( point p )
if (sin( $x_p$ ) > 0) then
    return  $c_0$ 
else
    return  $c_1$ 
```

We can also make the stripe's width w controllable:

```
RGB stripe( point p, real w )
if ( $\sin(\pi x_p / w)$  > 0) then
    return  $c_0$ 
else
    return  $c_1$ 
```

If we want to interpolate smoothly between the stripe colors, we can use a parameter t to vary the color linearly:

```
RGB stripe( point p, real w )
 $t = (1 + \sin(\pi p_x / w)) / 2$ 
return  $(1 - t)c_0 + tc_1$ 
```

These three possibilities are shown in Figure 11.26.

11.5.2 Solid Noise

Although regular textures such as stripes are often useful, we would like to be able to make “mottled” textures such as we see on birds’ eggs. This is usually done by using a sort of “solid noise,” usually called *Perlin noise* after its inventor, who received a technical Academy Award for its impact in the film industry (Perlin, 1985).

Getting a noisy appearance by calling a random number for every point would not be appropriate, because it would just be like “white noise” in TV static. We would like to make it smoother without losing the random quality. One possibility is to blur white noise, but there is no practical implementation of this. Another possibility is to make a large lattice with a random number at every lattice point, and then interpolate these random points for new points between lattice nodes; this is just a 3D texture array as described in the last section with random numbers in the array. This technique makes the lattice too obvious. Perlin used a variety of tricks to improve this basic lattice technique so the lattice was not so obvious. This results in a rather baroque-looking set of steps, but essentially there are just

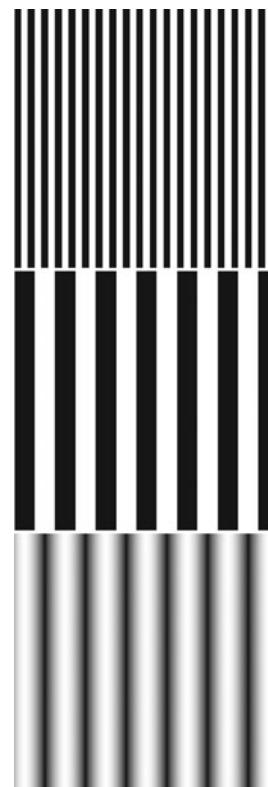


Figure 11.26. Various stripe textures result from drawing a regular array of xy points while keeping z constant.



一些振荡功能在两种颜色之间切换。一个简单的是一个正弦:

```
RGB条纹 (点p) if
(sin (xp) >0) 然
后返回c0 else
return c1
```

我们也可以使条纹的宽度w可控:

```
RGB条纹(点p 实w)if(sin(
npxw)>0)thenreturnc0
lse
return c1
```

如果我们想在条纹颜色之间进行平滑插值，我们可以使用参数t来线性地改变颜色:

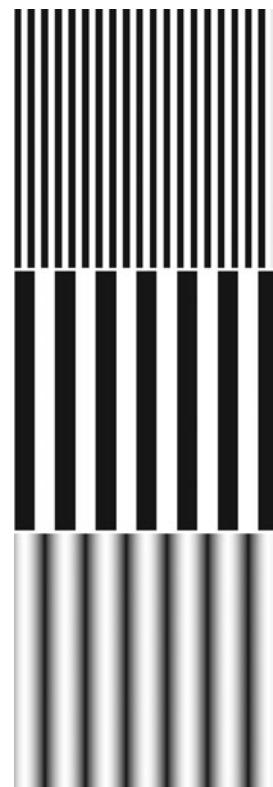
```
RGB条纹 (点p, 实w) t=
(1+sin (npwx) ) 2返回
(1-t) c0+tc1
```

这三种可能性如图11.26所示。

11.5.2 固体噪音

虽然像条纹这样的规则纹理通常很有用，但我们希望能够制作“斑驳”的纹理，就像我们在鸟蛋上看到的那样。这通常是通过使用一种“固体噪音”来完成的，通常以其发明者的名字称为Perlin噪音，他因其在电影行业的影响而获得技术学院奖（Perlin, 1985）。

通过为每个点调用随机数来获得嘈杂的外观是不合适的，因为它就像电视静态中的“白噪声”一样。我们希望在不丢失随机质量的情况下使其更平滑。一种可能性是模糊白噪声，但没有实际实现这一点。另一种可能性是在每个晶格点上制作一个带有随机数的大晶格，然后在晶格节点之间为新点插值这些随机点；这只是一个3d纹理阵列，如上一节所述，这种技术使晶格过于明显。Perlin使用了各种技巧来改进这种基本的晶格技术，因此晶格不是那么明显。这导致了一组相当巴洛克式的步骤，但基本上只有



Various条纹纹理是通过绘制xy点的常规数组而产生的，同时保持zconstant。

three changes from linearly interpolating a 3D array of random values. The first change is to use Hermite interpolation to avoid mach bands, just as can be done with regular textures. The second change is the use of random vectors rather than values, with a dot product to derive a random number; this makes the underlying grid structure less visually obvious by moving the local minima and maxima off the grid vertices. The third change is to use a 1D array and hashing to create a virtual 3D array of random vectors. This adds computation to lower memory use. Here is his basic method:

$$n(x, y, z) = \sum_{i=\lfloor x \rfloor}^{\lfloor x \rfloor+1} \sum_{j=\lfloor y \rfloor}^{\lfloor y \rfloor+1} \sum_{k=\lfloor z \rfloor}^{\lfloor z \rfloor+1} \Omega_{ijk}(x - i, y - j, z - k),$$

where (x, y, z) are the Cartesian coordinates of \mathbf{x} , and

$$\Omega_{ijk}(u, v, w) = \omega(u)\omega(v)\omega(w)(\Gamma_{ijk} \cdot (u, v, w)),$$

and $\omega(t)$ is the cubic weighting function:

$$\omega(t) = \begin{cases} 2|t|^3 - 3|t|^2 + 1 & \text{if } |t| < 1, \\ 0 & \text{otherwise.} \end{cases}$$

The final piece is that Γ_{ijk} is a random unit vector for the lattice point $(x, y, z) = (i, j, k)$. Since we want any potential ijk , we use a pseudorandom table:

$$\Gamma_{ijk} = \mathbf{G}(\phi(i + \phi(j + \phi(k)))),$$

where \mathbf{G} is a precomputed array of n random unit vectors, and $\phi(i) = P[i \bmod n]$ where P is an array of length n containing a permutation of the integers 0 through $n - 1$. In practice, Perlin reports $n = 256$ works well. To choose a random unit vector (v_x, v_y, v_z) first set

$$\begin{aligned} v_x &= 2\xi - 1, \\ v_y &= 2\xi' - 1, \\ v_z &= 2\xi'' - 1, \end{aligned}$$

where ξ, ξ', ξ'' are canonical random numbers (uniform in the interval $[0, 1]$). Then, if $(v_x^2 + v_y^2 + v_z^2) < 1$, make the vector a unit vector. Otherwise keep setting it randomly until its length is less than one, and then make it a unit vector. This is an example of a *rejection method*, which will be discussed more in Chapter 14. Essentially, the “less than” test gets a random point in the unit sphere, and the vector for the origin to that point is uniformly random. That would not be true of random points in the cube, so we “get rid” of the corners with the test.

从线性插值随机值的3d数组的三个变化。第一个变化是使用Hermite插值来避免马赫带，就像使用常规纹理一样。第二个变化是使用随机向量而不是值，用点积来推导一个随机数；这使得底层网格结构在视觉上不那么明显，因为将局部最小值和最大值从网格顶点移出。第三个变化是使用1D数组和散列来创建随机向量的虚拟3d数组。这增加了计算以减少内存使用。这是他的基本方法：

$$n(x, y, z) = \sum_{i=\lfloor x \rfloor}^{\lfloor x \rfloor+1} \sum_{j=\lfloor y \rfloor}^{\lfloor y \rfloor+1} \sum_{k=\lfloor z \rfloor}^{\lfloor z \rfloor+1} \Omega_{ijk}(x - i, y - j, z - k),$$

其中 (x, y, z) 是 x 的笛卡尔坐标，以及

$$\Omega_{ijk}(u, v, w) = \omega(u)\omega(v)\omega(w)(\Gamma_{ijk} \cdot (u, v, w)),$$

和 $\omega(t)$ 是三次加权函数：

$$\omega(t) = \begin{cases} 2|t|^3 - 3|t|^2 + 1 & \text{if } |t| < 1, \\ 0 & \text{otherwise.} \end{cases}$$

最后一块是 Γ_{ijk} 是晶格点 $(x, y, z) = (i, j, k)$ 的随机单位向量。由于我们想要任何潜在的 ijk ，我们使用伪随机表：

$$\Gamma_{ijk} = \mathbf{G}(\phi(i + \phi(j + \phi(k)))),$$

其中 \mathbf{G} 是 n 个随机单位向量的预计算数组， $\phi(i) = p[i \bmod n]$ 其中 P 是长度为 n 的数组，包含整数 0 到 $n - 1$ 的排列。在实践中，Perlin 报告 $n = 256$ 效果很好。要选择随机单位向量 (vx, vy, vz) 第一组

$$\begin{aligned} vx &= 2\xi - 1, \\ vy &= 2\xi' - 1, \\ vz &= 2\xi'' - 1, \end{aligned}$$

其中 ξ, ξ', ξ'' 是规范随机数（在区间 $[0, 1]$ 中均匀）。然后，如果 $(v_x^2 + v_y^2 + v_z^2) < 1$ ，使向量为单位向量。否则保持随机设置，直到其长度小于一，然后使其成为单位向量。这是拒绝方法的一个例子，将在第 14 章中更多地讨论。从本质上讲，“小于”测试在单位球体中得到一个随机点，并且原点到该点的向量是均匀随机的。对于立方体中的随机点来说，这不是真的，所以我们通过测试“摆脱”了角落。

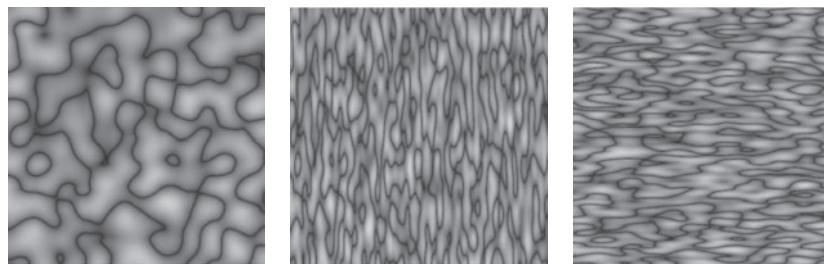


Figure 11.27. Absolute value of solid noise, and noise for scaled x and y values.

Because solid noise can be positive or negative, it must be transformed before being converted to a color. The absolute value of noise over a 10×10 square is shown in Figure 11.27, along with stretched versions. These versions are stretched by scaling the points input to the noise function.

The dark curves are where the original noise function changed from positive to negative. Since noise varies from -1 to 1 , a smoother image can be achieved by using $(\text{noise} + 1)/2$ for color. However, since noise values close to 1 or -1 are rare, this will be a fairly smooth image. Larger scaling can increase the contrast (Figure 11.28).

11.5.3 Turbulence

Many natural textures contain a variety of feature sizes in the same texture. Perlin uses a pseudofractal “turbulence” function:

$$n_t(\mathbf{x}) = \sum_i \frac{|n(2^i \mathbf{x})|}{2^i}$$

This effectively repeatedly adds scaled copies of the noise function on top of itself as shown in Figure 11.29.

The turbulence can be used to distort the stripe function:

```
RGB turbstripe( point p, double w )
double t = (1 + sin(k1 z_p + turbulence(k2 p))/w)/2
return t * s0 + (1 - t) * s1
```

Various values for k_1 and k_2 were used to generate Figure 11.30.

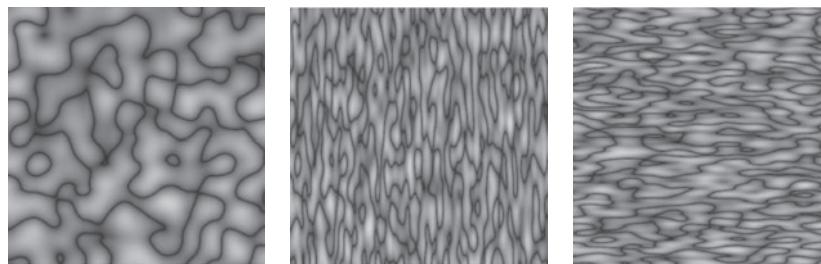


图11.27。 固体噪声的绝对值，以及缩放x和y值的噪声。

因为固体噪声可以是正的或负的，所以在转换为颜色之前必须对其进行转换。噪声在 10×10 平方上的绝对值如图11.27所示，以及拉伸版本。通过缩放输入到噪声函数的点来拉伸这些版本。

暗曲线是原始噪声函数从正变为负的地方。由于噪声在 1 到 1 之间变化，因此可以通过使用 $(\text{noise} + 1)/2$ 进行颜色来实现更平滑的图像。然而，由于接近 1 或 -1 的噪声值是罕见的，这将是一个相当平滑的图像。较大的缩放可以增加对比度（图11.28）。



Figure 11.28. Using 0.5(noise+1) (top) and 0.8(noise+1) (bottom) for intensity.

11.5.3 Turbulence

许多自然纹理在同一纹理中包含多种特征尺寸。Perlin使用pseudofractal“湍流”功能：

$$n_t(\mathbf{x}) = \sum_i \frac{|n(2^i \mathbf{x})|}{2^i}$$

这有效地在自身之上重复添加噪声函数的缩放副本，如图11.29所示。

湍流可以用来扭曲条纹函数：

```
RGBturbstripe (点p, 双w) 双t= (1+sin (k1zp
+湍流 (k2p) ) w) 2返回t s0+ (1-t) s1
```

K1和K2的各种值用于生成图11.30。

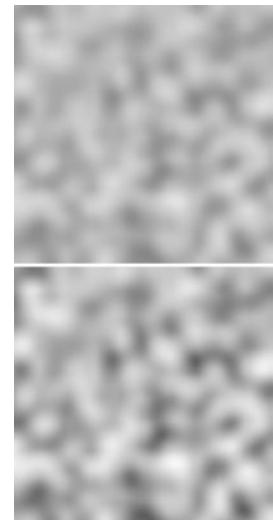


Figure 11.28. Using 0.5(noise+1) (top) and 0.8(noise+1) (bottom) for intensity.

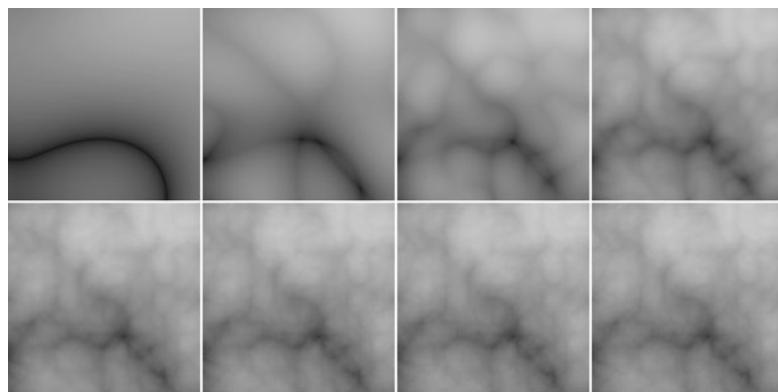


Figure 11.29. Turbulence function with (from top left to bottom right) one through eight terms in the summation.

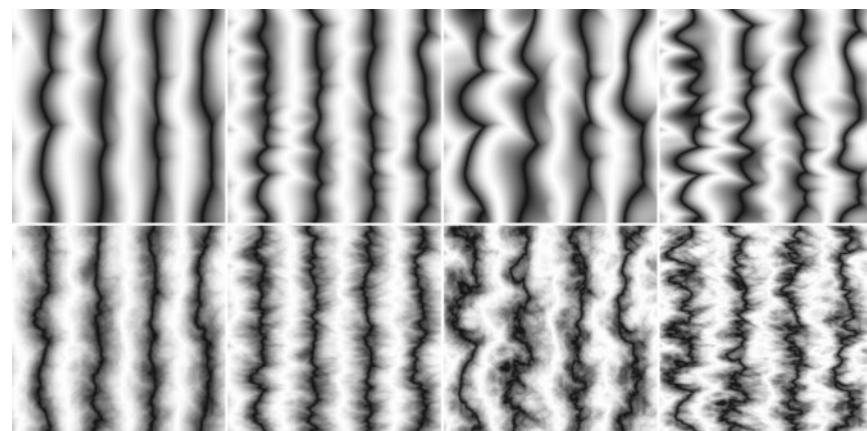


Figure 11.30. Various turbulent stripe textures with different k_1, k_2 . The top row has only the first term of the turbulence series.

Frequently Asked Questions

- How do I implement displacement mapping in ray tracing?

There is no ideal way to do it. Generating all the triangles and caching the geometry when necessary will prevent memory overload (Pharr & Hanrahan, 1996; Pharr, Kolb, Gershbein, & Hanrahan, 1997). Trying to intersect the displaced surface directly is possible when the displacement function is restricted (Patterson, Hoggar, & Logie, 1991; Heidrich & Seidel, 1998; Smits, Shirley, & Stark, 2000).

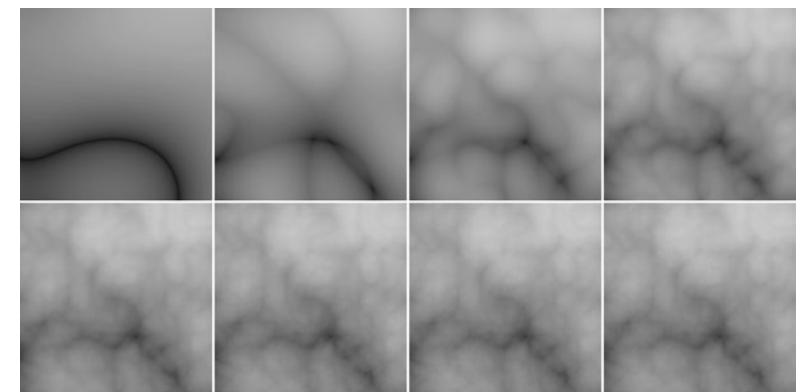


Figure 11.29. 湍流函数（从左上到右下）一到八个项的总和。

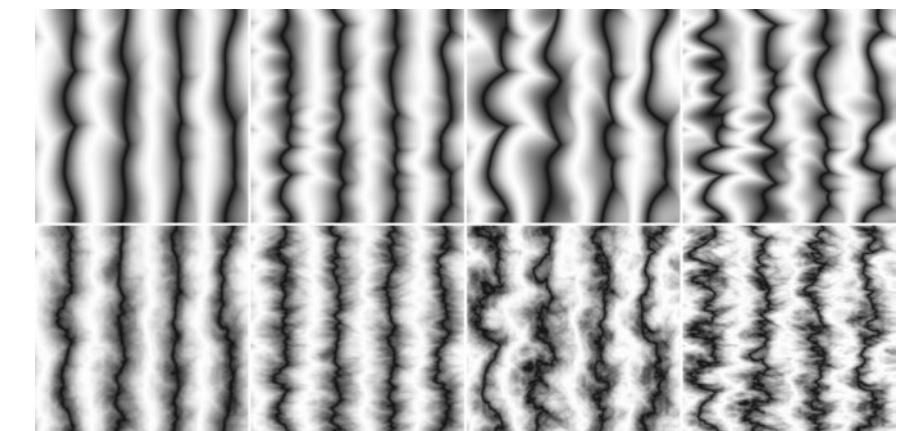


图11.30。 具有不同 k_1, k_2 的各种湍流条纹纹理。顶排只有湍流系列的第一个术语。

常见问题

- *如何在光线追踪中实现位移映射？

没有理想的方法来做到这一点。生成所有三角形并在必要时缓存几何将防止内存过载 (Pharr & Hanrahan, 1996; Pharr, Kolb, Gershbein, & Hanrahan, 1997)。当位移函数受到限制时，试图直接与位移表面相交是可能的 (Patterson, Hoggar, & Logie, 1991; Heidrich & Seidel, 1998; Smits, Shirley, & Stark, 2000)。



- Why don't my images with textures look realistic?

Humans are good at seeing small imperfections in surfaces. Geometric imperfections are typically absent in computer-generated images that use texture maps for details, so they look "too smooth."

Notes

The discussion of perspective-correct textures is based on *Fast Shadows and Lighting Effects Using Texture Mapping* (Segal, Korobkin, van Widenfelt, Foran, & Haeberli, 1992) and on *3D Game Engine Design* (Eberly, 2000).

Exercises

1. Find several ways to implement an infinite 2D checkerboard using surface and solid techniques. Which is best?
2. Verify that Equation (11.3) is a valid equality using brute-force algebra.
3. How could you implement solid texturing by using the z-buffer depth and a matrix transform?
4. Expand the function `mipmap_sample_trilinear` into a single function.



- 为什么我的纹理图像看起来不现实?

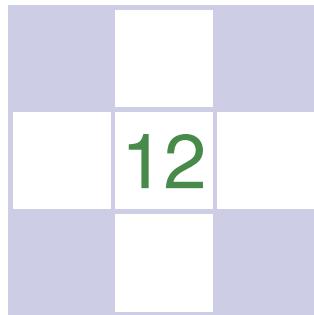
人类善于看到表面上的小瑕疵。几何缺陷通常在使用纹理贴图获取细节的计算机生成图像中不存在，因此它们看起来“太光滑了。”

Notes

透视正确纹理的讨论基于使用纹理映射的快速阴影和照明效果 (Segal, K orobkin, vanWidenfelt, Foran, & Haeberli, 1992) 和3D游戏引擎设计 (Eberly, 2000)。

Exercises

- 1.找到使用曲面和实体技术实现无限2D棋盘的几种方法。哪个最好?
- 2.使用蛮力代数验证方程 (11.3) 是有效的相等性。
- 3.如何使用z缓冲区深度和矩阵变换来实现实体纹理化?
- 4.将函数mipmap示例三线性展开为单个函数。



Data Structures for Graphics

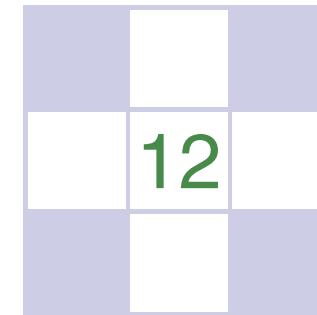
Certain data structures seem to pop up repeatedly in graphics applications, perhaps because they address fundamental underlying ideas like surfaces, space, and scene structure. This chapter talks about several basic and unrelated categories of data structures that are among the most common and useful: mesh structures, spatial data structures, scene graphs, and tiled multidimensional arrays.

For meshes, we discuss the basic storage schemes used for storing static meshes and for transferring meshes to graphics APIs. We also discuss the winged-edge data structure (Baumgart, 1974) and the related half-edge structure, which are useful for managing models where the tessellation changes, such as in subdivision or model simplification. Although these methods generalize to arbitrary polygon meshes, we focus on the simpler case of triangle meshes here.

Next, the scene-graph data structure is presented. Various forms of this data structure are ubiquitous in graphics applications because they are so useful in managing objects and transformations. All new graphics APIs are designed to support scene graphs well.

For spatial data structures, we discuss three approaches to organizing models in 3D space—bounding volume hierarchies, hierarchical space subdivision, and uniform space subdivision—and the use of hierarchical space subdivision (BSP trees) for hidden surface removal. The same methods are also used for other purposes, including geometry culling and collision detection.

Finally, the tiled multidimensional array is presented. Originally developed to help paging performance in applications where graphics data needed to be swapped in from disk, such structures are now crucial for memory locality on machines regardless of whether the array fits in main memory.



图形的数据结构

某些数据结构似乎在图形应用程序中反复出现，因为它们涉及基本的底层思想，如表面、空间和场景结构。本章讨论了最常见和最有用的几种基本和不相关的数据结构类别：网格结构、空间数据结构、场景图和平铺多维数组。

对于网格，我们讨论了用于存储静态网格和将网格传输到图形API的基本存储方案。我们还讨论了翼边数据结构（Baumgart, 1974）和相关的半边结构，这些结构对于管理曲面细分变化的模型非常有用，例如在子划分或模型简化中。虽然这些方法概括为任意多边形网格，但我们在这里关注三角形网格的更简单情况。

接下来，给出了场景图数据结构。这种数据结构的各种形式在图形应用程序中无处不在，因为它们在管理对象和转换方面非常有用。所有新的图形API都旨在很好地支持场景图。

对于空间数据结构，我们讨论了三种在3D空间中组织模型的方法—边界体积层次结构，分层空间细分和均匀空间细分—以及使用分层空间细分（BSP树）进行同样的方法也用于其他目的，包括几何剔除和碰撞检测。

最后，给出了平铺的多维数组。这些结构最初是为了帮助在需要从磁盘交换图形数据的应用程序中分页性能而开发的，现在无论阵列是否适合主内存，这些结构对于机器上的内存局部性都是至关重要的。

12.1 Triangle Meshes

Most real-world models are composed of complexes of triangles with shared vertices. These are usually known as *triangular meshes*, *triangle meshes*, or *triangular irregular networks* (TINs), and handling them efficiently is crucial to the performance of many graphics programs. The kind of efficiency that is important depends on the application. Meshes are stored on disk and in memory, and we'd like to minimize the amount of storage consumed. When meshes are transmitted across networks or from the CPU to the graphics system, they consume bandwidth, which is often even more precious than storage. In applications that perform operations on meshes, besides simply storing and drawing them—such as subdivision, mesh editing, mesh compression, or other operations—efficient access to adjacency information is crucial.

Triangle meshes are generally used to represent surfaces, so a mesh is not just a collection of unrelated triangles, but rather a network of triangles that connect to one another through shared vertices and edges to form a single continuous surface. This is a key insight about meshes: a mesh can be handled more efficiently than a collection of the same number of unrelated triangles.

The minimum information required for a triangle mesh is a set of triangles (triples of vertices) and the positions (in 3D space) of their vertices. But many, if not most, programs require the ability to store additional data at the vertices, edges, or faces to support texture mapping, shading, animation, and other operations. Vertex data is the most common: each vertex can have material parameters, texture coordinates, irradiances—any parameters whose values change across the surface. These parameters are then linearly interpolated across each triangle to define a continuous function over the whole surface of the mesh. However, it is also occasionally important to be able to store data per edge or per face.

12.1.1 Mesh Topology

The idea that meshes are surface-like can be formalized as constraints on the *mesh topology*—the way the triangles connect together, without regard for the vertex positions. Many algorithms will only work, or are much easier to implement, on a mesh with predictable connectivity. The simplest and most restrictive requirement on the topology of a mesh is for the surface to be a *manifold*. A manifold mesh is “watertight”—it has no gaps and separates the space on the inside of the surface from the space outside. It also looks like a surface everywhere on the mesh.

The term *manifold* comes from the mathematical field of topology: roughly speaking, a manifold (specifically a two-dimensional manifold, or 2-manifold) is

We'll leave the precise definitions to the mathematicians; see the chapter notes.

12.1 三角形网格

大多数真实世界的模型都是由具有共享vertices的三角形复合体组成的。这些通常被称为三角网格、三角网格或三角不规则网络（Tin），有效地处理它们对许多图形程序的性能至关重要。影响效率的种类取决于应用。网格存储在磁盘和内存中，我们希望最大限度地减少所消耗的存储量。当网格跨网络传输或从CPU传输到图形系统时，它们会消耗带宽，这通常比存储更宝贵。在对网格执行操作的应用程序中，除了简单地存储和绘制它们（如细分、网格编辑、网格压缩或其他操作）之外，对邻接信息的高效访问至关重要。

三角形网格通常用于表示曲面，因此网格不仅仅是不相关三角形的集合，而是三角形的网络，这些三角形通过共享顶点和边相互连接以形成单个连续。这是关于网格的一个关键见解：网格可以比相同数量的不相关三角形的集合更有效地处理。

三角形网格所需的最小信息是一组三角形（顶点的三元组）及其顶点的位置（在3D空间中）。但许多（如果不是大多数）程序需要在顶点、边或面存储额外数据的能力，以支持纹理映射、阴影、动画和其他效果。顶点数据是最常见的：每个顶点都可以有材质参数，纹理坐标，辐照度—任何值在整个表面上变化的参数。然后在每个三角形上线性插值这些参数，以便在网格的整个表面上定义一个连续函数。但是，能够存储每条边或每张脸的数据有时也很重要。

12.1.1 网格拓扑

网格是表面状的想法可以形式化为网格拓扑的约束—三角形连接在一起的方式，而不考虑顶点位置。许多算法只能在具有可预测连通性的网格上工作，或者更容易实现。对网格拓扑最简单和最严格的要求是表面是一个流形。流形网格是“水密的”—它没有间隙，并将表面内侧的空间与外部的空间分开。它看起来也像网格上到处都是一个表面。

我们将把精确的解释留给数学学者；见章节笔记。

术语流形来自拓扑学的数学领域：粗略地说，流形（具体地说是二维流形，或2-流形）是



a surface in which a small neighborhood around any point could be smoothed out into a bit of flat surface. This idea is most clearly explained by counterexample: if an edge on a mesh has three triangles connected to it, the neighborhood of a point on the edge is different from the neighborhood of one of the points in the interior of one of the triangles, because it has an extra “fin” sticking out of it (Figure 12.1). If the edge has exactly two triangles attached to it, points on the edge have neighborhoods just like points in the interior, only with a crease down the middle. Similarly, if the triangles sharing a vertex are in a configuration like the left one in Figure 12.2, the neighborhood is like two pieces of surface glued together at the center, which can’t be flattened without doubling it up. The vertex with the simpler neighborhood shown at right is just fine.

Many algorithms assume that meshes are manifold, and it’s always a good idea to verify this property to prevent crashes or infinite loops if you are handed a malformed mesh as input. This verification boils down to checking that all edges are manifold and checking that all vertices are manifold by verifying the following conditions:

- Every edge is shared by exactly two triangles.
- Every vertex has a single, complete loop of triangles around it.

Figure 12.1 illustrates how an edge can fail the first test by having too many triangles, and Figure 12.2 illustrates how a vertex can fail the second test by having two separate loops of triangles attached to it.

Manifold meshes are convenient, but sometimes it’s necessary to allow meshes to have edges, or *boundaries*. Such meshes are not manifolds—a point on the boundary has a neighborhood that is cut off on one side. They are not necessarily watertight. However, we can relax the requirements of a manifold mesh to those for a *manifold with boundary* without causing problems for most mesh processing algorithms. The relaxed conditions are:

- Every edge is used by either one or two triangles.
- Every vertex connects to a single edge-connected set of triangles.

Figure 12.3 illustrates these conditions: from left to right, there is an edge with one triangle, a vertex whose neighboring triangles are in a single edge-connected set, and a vertex with two disconnected sets of triangles attached to it.

Finally, in many applications it’s important to be able to distinguish the “front” or “outside” of a surface from the “back” or “inside”—this is known as the *orientation* of the surface. For a single triangle, we define orientation based on the order in which the vertices are listed: the front is the side from which the triangle’s three vertices are arranged in counterclockwise order. A connected mesh is

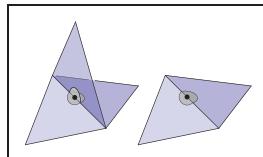


Figure 12.1. Non-manifold (left) and manifold (right) interior edges.

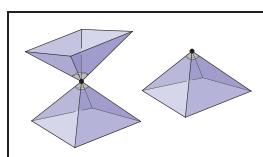


Figure 12.2. Non-manifold (left) and manifold (right) interior vertices.

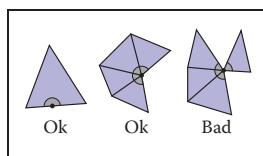


Figure 12.3. Conditions at the edge of a manifold with boundary.

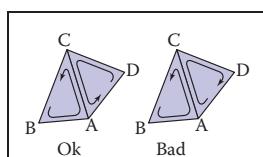


Figure 12.4. Triangles (B,A,C) and (D,C,A) are consistently oriented, whereas (B,A,C) and (A,C,D) are inconsistently oriented.

一个表面，其中任何点周围的一个小邻域都可以平滑成一个平坦的表面。反例最清楚地解释了这个想法：如果网格上的一条边有三个三角形连接到它，那么边缘上的一个点的邻域不同于其中一个三角形内部的一个点的邻域，因为它有一个额外的“鳍”伸出它（图12.1）。如果边缘正好有两个三角形连接到它，则指向

边缘有邻域，就像内部的点一样，只有中间有一个折痕。类似地，如果共享一个顶点的三角形处于类似图12.2中左侧的配置中，则邻域就像两块表面胶合

在中心一起，不把它加倍就不能被压扁。右侧显示的具有更简单邻域的顶点就好了。

许多算法都假设网格是流形的，如果将格式错误的网格作为输入，最好验证此属性以防止崩溃或无限循环。此验证归结为检查所有边都是流形，并通过验证以下条件来检查所有顶点都是流形：

*每条边都由两个三角形共享。

•每个顶点周围都有一个完整的三角形循环。

图12.1说明了一个边如何通过有太多的三角形来失败第一个测试，图12.2说明了一个顶点如何通过附加两个单独的三角形循环来失败第二个测试。

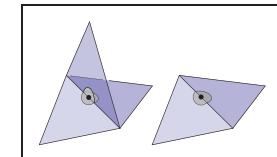
流形网格很方便，但有时需要允许网格具有边或边界。这样的网格不是流形—边界上的一个点有一个在一侧被切断的邻域。它们不一定是水密的。但是，我们可以将流形网格的要求放宽到具有边界的流形的要求，而不会对大多数网格处理算法造成问题。宽松的条件是：

*每条边都由一个或两个三角形使用。

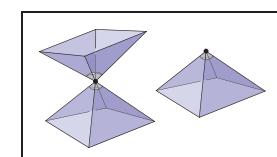
•每个顶点连接到单个边连接的三角形集合。

图12.3说明了这些条件：从左到右，有一个带有一个三角形的边，一个相邻三角形在一个边连接集合中的顶点，以及一个带有两个断开的三角形集合的顶点。

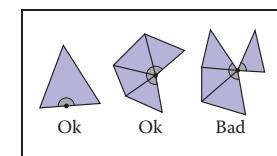
最后，在许多应用中，能够区分表面的“正面”或“外部”与“背面”或“内部”是很重要的一这被称为表面的定向。对于单个三角形，我们根据顶点列出的顺序定义方向：正面是三角形的三个顶点按逆时针顺序排列的侧面。一个连通的网格是



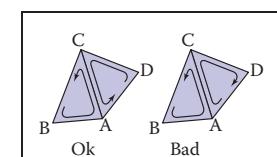
在terior边缘的非流形（左）和流形（右）。



Terior顶点中的非流形（左）和流形（右）。



具有边界的流形边缘的条件。



Triangles (B, A, C) 和 (D, C, A) 一致定向，而 (B, A, C) 和 (A, C, D) 一致定向。

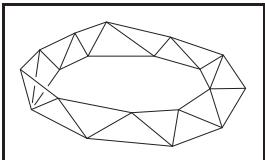


Figure 12.5. A triangulated Möbius band, which is not orientable.

consistently oriented if its triangles all agree on which side is the front—and this is true if and only if every pair of adjacent triangles is consistently oriented.

In a consistently oriented pair of triangles, the two shared vertices appear in opposite orders in the two triangles' vertex lists (Figure 12.4). What's important is consistency of orientation—some systems define the front using clockwise rather than counterclockwise order.

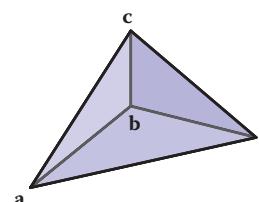
Any mesh that has non-manifold edges can't be oriented consistently. But it's also possible for a mesh to be a valid manifold with boundary (or even a manifold), and yet have no consistent way to orient the triangles—they are not *orientable* surfaces. An example is the Möbius band shown in Figure 12.5. This is rarely an issue in practice, however.

12.1.2 Indexed Mesh Storage

A simple triangular mesh is shown in Figure 12.6. You could store these three triangles as independent entities, each of this form:

```
Triangle {
    vector3 vertexPosition[3]
}
```

This would result in storing vertex **b** three times and the other vertices twice each for a total of nine stored points (three vertices for each of three triangles). Or you could instead arrange to share the common vertices and store only four, resulting



Separate triangles:

#	Vertex 0	Vertex 1	Vertex 2
0	(a_x, a_y, a_z)	(b_x, b_y, b_z)	(c_x, c_y, c_z)
1	(b_x, b_y, b_z)	(d_x, d_y, d_z)	(c_x, c_y, c_z)
2	(a_x, a_y, a_z)	(d_x, d_y, d_z)	(b_x, b_y, b_z)

Shared vertices:

Triangles		Vertices	
#	Vertices	#	Position
0	(0, 1, 2)	0	(a_x, a_y, a_z)
1	(1, 3, 2)	1	(b_x, b_y, b_z)
2	(0, 3, 1)	2	(c_x, c_y, c_z)
		3	(d_x, d_y, d_z)

Figure 12.6. A three-triangle mesh with four vertices, represented with separate triangles (left) and with shared vertices (right).

一致地定向，如果它的三角形都同意哪一侧是正面—当且仅当每对相邻三角形一致地定向时才是如此。

在一致定向的一对三角形中，两个共享顶点在两个三角形的顶点列表中以相反的顺序出现（图12.4）。重要的是方向的一致性—一些系统使用顺时针而不是逆时针顺序定义前部。

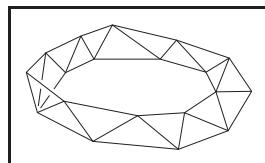


Figure 12.5. A triangulated Möbius band, which is not orientable.

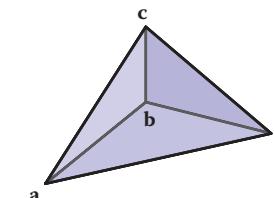
任何具有非流形边缘的网格都不能一致地定向。但是网格也可能是一个有边界的有效流形（甚至是流形），但没有一致的方法来定位三角形—它们不是可定向的表面。一个例子是图12.5所示的Möbius波段。然而，这在实践中很少是一个问题。

12.1.2 索引网格存储

一个简单的三角形网格如图12.6所示。您可以将这三个三角形存储为独立的实体，每个都是这种形式：

```
Triangle {
    vector3 vertexPosition[3]
}
```

这将导致将顶点**b**存储三次，其他顶点每个存储两次，总共存储九个点（三个三角形中的每一个都有三个顶点）。或者你可以安排共享公共顶点，只存储四个顶点，结果



Separate triangles:

#	Vertex 0	Vertex 1	Vertex 2
0	(a_x, a_y, a_z)	(b_x, b_y, b_z)	(c_x, c_y, c_z)
1	(b_x, b_y, b_z)	(d_x, d_y, d_z)	(c_x, c_y, c_z)
2	(a_x, a_y, a_z)	(d_x, d_y, d_z)	(b_x, b_y, b_z)

Shared vertices:

Triangles		Vertices	
#	Vertices	#	Position
0	(0, 1, 2)	0	(a_x, a_y, a_z)
1	(1, 3, 2)	1	(b_x, b_y, b_z)
2	(0, 3, 1)	2	(c_x, c_y, c_z)
		3	(d_x, d_y, d_z)

图12.6。具有四个顶点的三三角形网格，用单独的三角形（左）和共享顶点（右）表示。



in a *shared-vertex mesh*. Logically, this data structure has triangles which point to vertices which contain the vertex data:

```
Triangle {
    Vertex v[3]
}

Vertex {
    vector3 position // or other vertex data
}
```

Note that the entries in the *v* array are references, or pointers, to *Vertex* objects; the vertices are not contained in the triangle.

In implementation, the vertices and triangles are normally stored in arrays, with the triangle-to-vertex references handled by storing array indices:

```
IndexedMesh {
    int tInd[nt][3]
    vector3 verts[nv]
}
```

The index of the *k*th vertex of the *i*th triangle is found in *tInd[i][k]*, and the position of that vertex is stored in the corresponding row of the *verts* array; see Figure 12.8 for an example. This way of storing a shared-vertex mesh is an *indexed triangle mesh*.

Separate triangles or shared vertices will both work well. Is there a space advantage for sharing vertices? If our mesh has *n_v* vertices and *n_t* triangles, and if we assume that the data for floats, pointers, and ints all require the same storage (a dubious assumption), the space requirements are as follows:

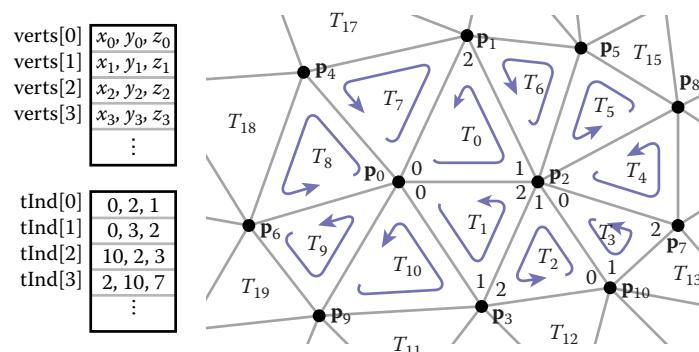


Figure 12.8. A larger triangle mesh, with part of its representation as an indexed triangle mesh.

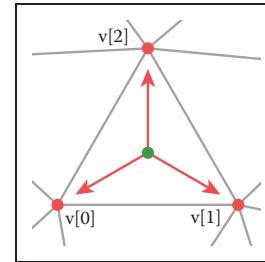


Figure 12.7. The triangle-to-vertex references in a shared-vertex mesh.

在共享顶点网格中。从逻辑上讲，此数据结构具有三角形，这些三角形指向包含顶点数据的顶点：

```
Triangle {
    Vertex v[3]
}

Vertex {
    vector3 position // 或其他顶点数据
}
```

请注意，*v*数组中的条目是顶点对象的引用或指针；顶点不包含在三角形中。

在实现中，顶点和三角形通常存储在数组中，三角形到顶点的引用通过存储数组索引来处理：

```
IndexedMesh {
    int tInd[nt][3]
    vector3 verts[nv]
}
```

在*tInd[i][k]*中找到第*i*个三角形的第*k*个顶点的索引，并且该顶点的位置存储在*verts*数组的相应行中；参见图12.8的示例。这种存储共享顶点网格的方式是索引三角形网格。

单独的三角形或共享顶点都可以很好地工作。共享顶点是否有空间优势？如果我们的网格有*n_v*个顶点和*n_t*个三角形，并且如果我们假设浮点数，指针和int的数据都需要相同的存储（一个可疑的假设），空间要求如下：

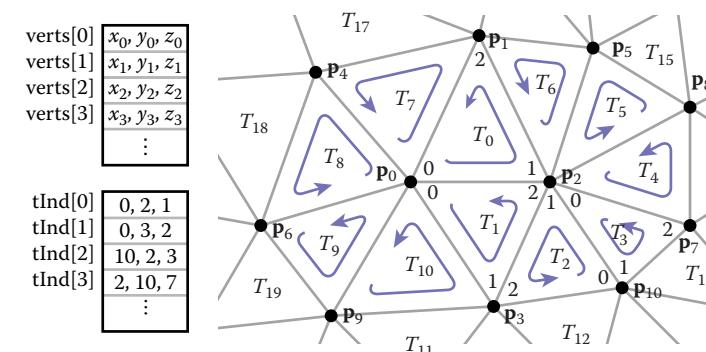


图12.8。一个更大的三角形网格，其部分表示为索引三角形网格。

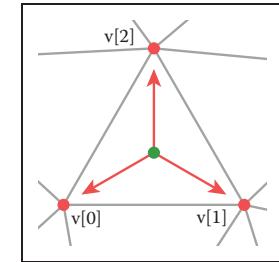


Figure 12.7. 三角形 to-vertex references in a shared-vertex mesh.

- **Triangle.** Three vectors per triangle, for $9n_t$ units of storage;
- **IndexedMesh.** One vector per vertex and three ints per triangle, for $3n_v + 3n_t$ units of storage.

The relative storage requirements depend on the ratio of n_t to n_v .

As a rule of thumb, a large mesh has each vertex connected to about six triangles (although there can be any number for extreme cases). Since each triangle connects to three vertices, this means that there are generally twice as many triangles as vertices in a large mesh: $n_t \approx 2n_v$. Making this substitution, we can conclude that the storage requirements are $18n_v$ for the Triangle structure and $9n_v$ for IndexedMesh. Using shared vertices reduces storage requirements by about a factor of two; and this seems to hold in practice for most implementations.

Is this factor of two worth the complication? I think the answer is yes, and it becomes an even bigger win as soon as you start adding “properties” to the vertices.

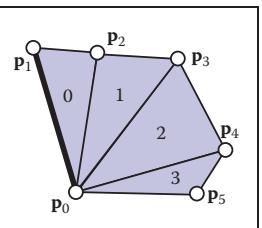


Figure 12.9. A triangle fan.

Indexed meshes are the most common in-memory representation of triangle meshes, because they achieve a good balance of simplicity, convenience, and compactness. They are also commonly used to transfer meshes over networks and between the application and graphics pipeline. In applications where even more compactness is desirable, the triangle vertex indices (which take up two-thirds of the space in an indexed mesh with only positions at the vertices) can be expressed more efficiently using *triangle strips* and *triangle fans*.

A triangle fan is shown in Figure 12.9. In an indexed mesh, the triangles array would contain $[(0, 1, 2), (0, 2, 3), (0, 3, 4), (0, 4, 5)]$. We are storing 12 vertex indices, although there are only six distinct vertices. In a triangle fan, all the triangles share one common vertex, and the other vertices generate a set of triangles like the vanes of a collapsible fan. The fan in the figure could be specified with the sequence $[0, 1, 2, 3, 4, 5]$: the first vertex establishes the center, and subsequently each pair of adjacent vertices (1-2, 2-3, etc.) creates a triangle.

The triangle strip is a similar concept, but it is useful for a wider range of meshes. Here, vertices are added alternating top and bottom in a linear strip as shown in Figure 12.10. The triangle strip in the figure could be specified by the sequence $[0 1 2 3 4 5 6 7]$, and every subsequence of three adjacent vertices (0-1-2, 1-2-3, etc.) creates a triangle. For consistent orientation, every other triangle needs to have its order reversed. In the example, this results in the triangles $(0, 1, 2), (2, 1, 3), (2, 3, 4), (4, 3, 5)$, etc. For each new vertex that comes in, the oldest vertex is forgotten and the order of the two remaining vertices is swapped. See Figure 12.11 for a larger example.

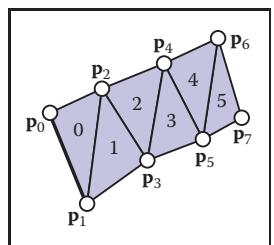


Figure 12.10. A triangle strip.

*三角形。每个三角形三个向量，用于9个nt存储单位；

*IndexedMesh。每个顶点一个向量，每个三角形三个整数，用于3nv+3nt存储单位。

相对存储要求取决于nt与nv的比率。

根据经验，大型网格的每个顶点都连接到大约六个三角形（尽管对于极端情况可以有任何数量）。由于每个三角形连接到三个顶点，这意味着在大型网格中通常有两倍于顶点的三角形： $nt \approx 2nv$ 。通过这种替换，我们可以得出结论，对于三角形结构，存储要求为 $18nv$ ，对于IndexedMesh，存储要求为 $9nv$ 。使用共享顶点将存储需求减少了大约两倍；这似乎在实践中适用于大多数实现。

这两个因素值得复杂吗？
我认为答案是肯定的，
一旦你开始向顶点添加“属性”，它就会带来更大的胜利。

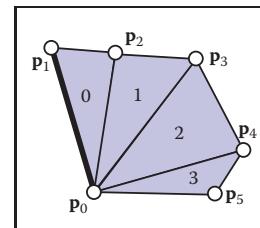


图12.9. 一个三角扇。

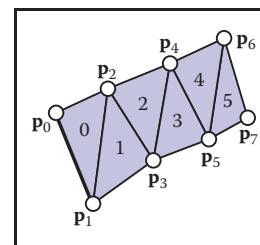


Figure 12.10. A triangle strip.

12.1.3 三角带和风扇

索引网格是三角形网格最常见的内存中表示形式，因为它们在简单性、方便性和紧凑性之间取得了很好的平衡。它们也常用于通过网络以及在应用程序和图形管道之间传输网格。在需要更紧凑的应用中，三角形顶点索引（在只有顶点位置的索引网格中占用两个空间）可以使用三角形条带和三角形风扇更有效地表达。

三角形风扇如图12.9所示。在索引网格中，三角形数组将包含 $[(0 1 2) (0 2 3) (0 3 4) (0 4 5)]$ 。我们正在存储12个顶点索引，尽管只有六个不同的顶点。在三角形风扇中，所有三角形共享一个公共顶点，其他顶点生成一组三角形，就像可折叠风扇的叶片一样。图中的风扇可以用序列指定 $[0 1 2 3 4 5]$ ：第一个顶点建立中心，随后每对相邻顶点（1-2, 2-3等）创建一个三角形。三角形条带是一个类似的概念，但它对于更广泛的网格很有用。在这里，顶点在一个线性条中交替添加顶部和底部，如图12.10所示。图中的三角形条可以由序列指定 $[01234567]$ 和三个相邻顶点的每个子序列（01-2, 1-2-3等）创建一个三角形。为了保持一致的方向，每个其他三角形都需要颠倒其顺序。在示例中，这会导致三角形 $(0, 1, 2) (2, 1, 3) (2, 3, 4) (4, 3, 5)$ 等。对于每个进来的新顶点，最旧的顶点被遗忘，其余两个顶点的顺序被交换。有关更大的示例，请参阅图12.11。

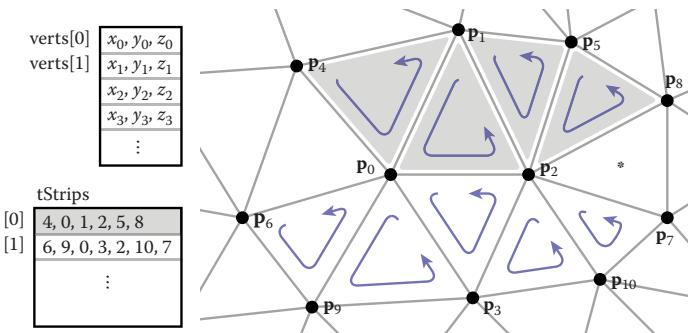


Figure 12.11. Two triangle strips in the context of a larger mesh. Note that neither strip can be extended to include the triangle marked with an asterisk.

In both strips and fans, $n + 2$ vertices suffice to describe n triangles—a substantial savings over the $3n$ vertices required by a standard indexed mesh. Long triangle strips will save approximately a factor of three if the program is vertex-bound.

It might seem that triangle strips are only useful if the strips are very long, but even relatively short strips already gain most of the benefits. The savings in storage space (for only the vertex indices) are as follows:

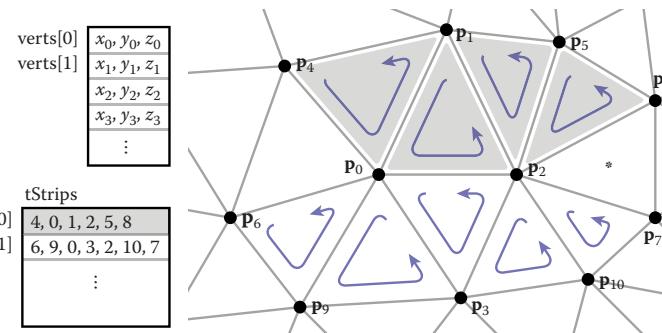
strip length	1	2	3	4	5	6	7	8	16	100	∞
relative size	1.00	0.67	0.56	0.50	0.47	0.44	0.43	0.42	0.38	0.34	0.33

So, in fact, there is a rather rapid diminishing return as the strips grow longer. Thus, even for an unstructured mesh, it is worthwhile to use some greedy algorithm to gather them into short strips.

12.1.4 Data Structures for Mesh Connectivity

Indexed meshes, strips, and fans are all good, compact representations for static meshes. However, they do not readily allow for meshes to be modified. In order to efficiently edit meshes, more complicated data structures are needed to efficiently answer queries such as:

- Given a triangle, what are the three adjacent triangles?
- Given an edge, which two triangles share it?



较大网格背景下的两个三角形条带。请注意，两个条带都不能扩展到包括标有星号的三角形。

在条带和风扇中， $n+2$ 个顶点足以描述 n 个三角形—在标准索引网格所需的 $3n$ 个顶点上的亚稳态节省。如果程序是顶点绑定的长的三角形条将节省大约三个因子。

看起来三角形条带只有在条带很长的情况下才有用，但即使是相对较短的条带也已经获得了大部分好处。存储空间的节省（仅用于顶点索引）如下所示：

带材长度	1	2	3	4	5	6	7	8	16	100	∞
相对尺寸	1.00	0.67	0.56	0.50	0.47	0.44	0.43	0.42	0.38	0.34	0.33

因此，事实上，随着条带变长，回报率会迅速下降。

因此，即使对于非结构化网格，也值得使用一些贪婪的算法将它们聚集成短条。

12.1.4 网格连接的数据结构

索引网格、条带和风扇都是静态网格的良好紧凑表示形式。但是，它们不容易允许修改网格。为了有效地编辑网格，需要更复杂的数据结构来有效地回答查询，例如：

- *给定一个三角形，相邻的三个三角形是什么？
- *给定一条边，哪两个三角形共享它？

- Given a vertex, which faces share it?
- Given a vertex, which edges share it?

There are many data structures for triangle meshes, polygonal meshes, and polygonal meshes with holes (see the notes at the end of the chapter for references). In many applications the meshes are very large, so an efficient representation can be crucial.

The most straightforward, though bloated, implementation would be to have three types, `Vertex`, `Edge`, and `Triangle`, and to just store all the relationships directly:

```
Triangle {
    Vertex v[3]
    Edge e[3]
}

Edge {
    Vertex v[2]
    Triangle t[2]
}

Vertex {
    Triangle t[]
    Edge e[]
}
```

This lets us directly look up answers to the connectivity questions above, but because this information is all inter-related, it stores more than is really needed. Also, storing connectivity in vertices makes for variable-length data structures (since vertices can have arbitrary numbers of neighbors), which are generally less efficient to implement. Rather than committing to store all these relationships explicitly, it is best to define a class interface to answer these questions, behind which a more efficient data structure can hide. It turns out we can store only some of the connectivity and efficiently recover the other information when needed.

The fixed-size arrays in the `Edge` and `Triangle` classes suggest that it will be more efficient to store the connectivity information there. In fact, for polygon meshes, in which polygons have arbitrary numbers of edges and vertices, only edges have fixed-size connectivity information, which leads to many traditional mesh data structures being based on edges. But for triangle-only meshes, storing connectivity in the (less numerous) faces is appealing.

A good mesh data structure should be reasonably compact and allow efficient answers to all adjacency queries. Efficient means constant-time: the time to find neighbors should not depend on the size of the mesh. We'll look at three data structures for meshes, one based on triangles and two based on edges.

- *给定一个顶点，哪些面共享它？
- *给定一个顶点，哪些边共享它？

三角形网格体、多边形网格体和带孔的多边形网格体的数据结构很多（参考参见章节末尾的注释）。在许多应用中，网格非常大，因此有效的表示至关重要。

最简单的，虽然臃肿，实现将有三种类型，顶点，边和三角形，并直接存储所有的关系船：

```
Triangle {
    Vertex v[3]
    Edge e[3]
}

Edge {
    Vertex v[2]
    Triangle t[2]
}

Vertex {
    Triangle t[]
    Edge e[]
}
```

这使我们可以直接查找上述连接问题的答案，但由于这些信息都是相互关联的，因此它存储的内容比实际需要的要多。此外，将连通性存储在顶点中会产生可变长度的数据结构（因为顶点可以具有任意数量的邻居），而这些结构的实现效率通常较低。与其承诺显式存储所有这些关系，最好定义一个类接口来回答这些问题，在这些问题后面可以隐藏更有效的数据结构。事实证明，我们只能存储一些连接，并在需要时有效地恢复其他信息。

`Edge`和`Triangle`类中的固定大小数组表明将连通性信息存储在那里会更有效。实际上，对于多边形网格，其中多边形具有任意数量的边和顶点，只有边具有固定大小的连通性信息，这导致许多传统的网格数据结构基于边。但对于仅限三角形的网格体，将连通性存储在（较少数量的）面中是有吸引力的。

一个好的网格数据结构应该合理紧凑，并允许有效地回答所有邻接查询。高效意味着恒定时间：找到邻居的时间不应该取决于网格的大小。我们将看到网格的三个数据结构，一个基于三角形，两个基于边。



The Triangle-Neighbor Structure

We can create a compact mesh data structure based on triangles by augmenting the basic shared-vertex mesh with pointers from the triangles to the three neighboring triangles, and a pointer from each vertex to one of the adjacent triangles (it doesn't matter which one); see Figure 12.12:

```
Triangle {
    Triangle nbr[3];
    Vertex v[3];
}

Vertex {
    // ... per-vertex data ...
    Triangle t; // any adjacent tri
}
```

In the array `Triangle.nbr`, the k th entry points to the neighboring triangle that shares vertices k and $k + 1$. We call this structure the *triangle-neighbor structure*. Starting from standard indexed mesh arrays, it can be implemented with two additional arrays: one that stores the three neighbors of each triangle, and one that stores a single neighboring triangle for each vertex (see Figure 12.13 for an example):

```
Mesh {
    // ... per-vertex data ...
    int tInd[nt][3]; // vertex indices
    int tNbr[nt][3]; // indices of neighbor triangles
    int vTri[nv]; // index of any adjacent triangle
}
```

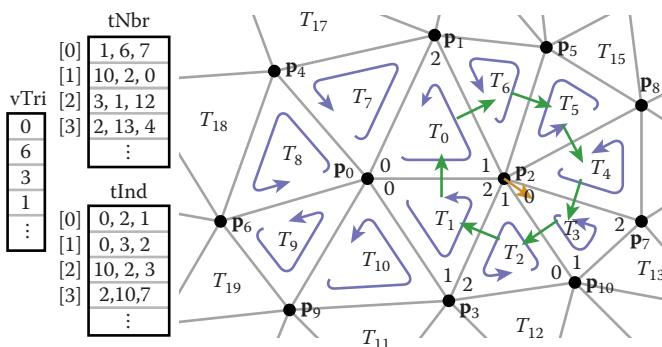


Figure 12.13. The triangle-neighbor structure as encoded in arrays, and the sequence that is followed in traversing the neighboring triangles of vertex 2.

三角形-邻居结构

我们可以创建一个基于三角形的紧凑网格数据结构，方法是用从三角形到三个相邻三角形的指针，以及从每个顶点到一个相邻三角形的指针（不管哪一个）来增加基本的共享顶点网格；见图12.12：

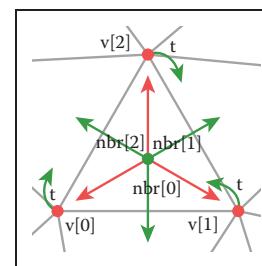
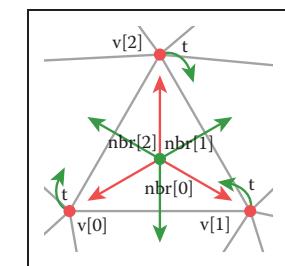


Figure 12.12. The references between triangles and vertices in the triangle-neighbor structure.

```
Triangle {
    Triangle nbr[3];
    Vertex v[3];
}

Vertex {
    //每顶点数据。..三角形t;任何相邻的三
    }
}
```

阵三角形中。`nbr`, 第 k 个入口指向共享顶点 k 和 $k+1$ 的相邻三角形。我们将这种结构称为三角形-邻居结构。从标准索引网格数组开始，它可以用两个额外的数组来实现：一个存储每个三角形的三个邻居，一个存储每个顶点的单个相邻三角形（参见图12.13的示例）：



三角形邻居结构中三角形和顶点之间的引用。

```
Mesh {
    //每顶点数据。..inttInd[nt][3];顶点索引inttNbr[nt][3];相邻三角形的索引intvTri[nv];任何相邻三角形的索引
    }
}
```

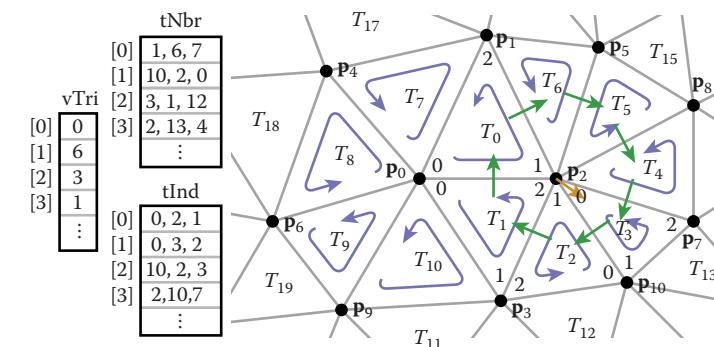


Figure 12.13. 在数组中编码的三角形邻居结构，以及接着遍历顶点2的相邻三角形。

Clearly the neighboring triangles and vertices of a triangle can be found directly in the data structure, but by using this triangle adjacency information carefully it is also possible to answer connectivity queries about vertices in constant time. The idea is to move from triangle to triangle, visiting only the triangles adjacent to the relevant vertex. If triangle t has vertex v as its k th vertex, then the triangle $t.nbr[k]$ is the next triangle around v in the clockwise direction. This observation leads to the following algorithm to traverse all the triangles adjacent to a given vertex:

```
TrianglesOfVertex(v) {
    t = v.t
    do {
        find i such that (t.v[i] == v)
        t = t.nbr[i]
    } while (t != v.t)
}
```

Of course, a real program would *do* something with the triangles as it found them.

This operation finds each subsequent triangle in constant time—even though a search is required to find the position of the central vertex in each triangle’s vertex list, the vertex lists have constant size so the search takes constant time. However, that search is awkward and requires extra branching.

A small refinement can avoid these searches. The problem is that once we follow a pointer from one triangle to the next, we don’t know from which way we came: we have to search the triangle’s vertices to find the vertex that connects back to the previous triangle. To solve this, instead of storing pointers to neighboring triangles, we can store pointers to specific edges of those triangles by storing an index with the pointer:

```
Triangle {
    Edge nbr[3];
    Vertex v[3];
}

Edge { // the i-th edge of triangle t
    Triangle t;
    int i; // in {0,1,2}
}

Vertex {
    // ... per-vertex data ...
    Edge e; // any edge leaving vertex
}
```

In practice the `Edge` is stored by borrowing two bits of storage from the triangle index t to store the edge index i , so that the total storage requirements remain the same.

显然，三角形的相邻三角形和顶点可以直接在数据结构中找到，但通过仔细使用此三角形邻接信息，也可以在恒定时间内回答有关顶点的连通性查询。这个想法是从三角形移动到三角形，只访问与相关顶点相邻的三角形。如果三角形 t 具有顶点 v 作为其第 k 个顶点，则三角形 $t.nbr[k]$ 是沿顺时针方向围绕 v 的下一个三角形。该观察结果导致以下算法遍历与给定顶点相邻的所有三角形：

```
TrianglesOfVertex(v) {
    t = v.t
    do {
        找到使得(t.v[i]==v)t=t.nbr[i] while(t!=v.t)
    }
}
```

当然，一个真正的程序会对三角形做一些事情，因为它发现了它们。

此操作以恒定的时间查找每个后续三角形—即使需要搜索才能在每个三角形的顶点列表中找到中心顶点的位置，但顶点列表具有恒定的大小，因此搜索然而，这种搜索很尴尬，需要额外的分支。

一个小的细化可以避免这些搜索。问题是，一旦我们跟随一个从一个三角形到下一个三角形的指针，我们就不知道我们是从哪条路来的：我们必须搜索三角形的顶点来找到与前一个三角形相交的顶点。为了解决这个问题，我们可以通过使用指针存储索引来存储指向相邻三角形的指针，而不是存储指向这些三角形的特定边的指针：

```
Triangle {
    边nbr[3];顶点v[
    3];
}

边{三角形t的第i条边
    三角形t:inti;
    in {0,1,2}
}

Vertex {
    ...每顶点数据。边缘e;
    离开顶点的任何边
}
```

在实践中通过从三角形索引 t 借用两位存储来存储边索引 i 来存储边，使得总存储需求保持不变。

In this structure the neighbor array for a triangle tells *which* of the neighboring triangles' edges are shared with the three edges of that triangle. With this extra information, we always know where to find the original triangle, which leads to an invariant of the data structure: for any j th edge of any triangle t ,

$$t.\text{nbr}[j].t.\text{nbr}[t.\text{nbr}[j].i].t == t.$$

Knowing which edge we came in through lets us know immediately which edge to leave through in order to continue traversing around a vertex, leading to a streamlined algorithm:

```
TrianglesOfVertex(v) {
    {t, i} = v.e;
    do {
        {t, i} = t.nbr[i];
        i = (i+1) mod 3;
    } while (t != v.e.t);
}
```

The triangle-neighbor structure is quite compact. For a mesh with only vertex positions, we are storing four numbers (three coordinates and an edge) per vertex and six (three vertex indices and three edges) per face, for a total of $4n_v + 6n_t \approx 16n_v$ units of storage per vertex, compared with $9n_v$ for the basic indexed mesh.

The triangle neighbor structure as presented here works only for manifold meshes, because it depends on returning to the starting triangle to terminate the traversal of a vertex's neighbors, which will not happen at a boundary vertex that doesn't have a full cycle of triangles. However, it is not difficult to generalize it to manifolds with boundary, by introducing a suitable sentinel value (such as -1) for the neighbors of boundary triangles and taking care that the boundary vertices point to the most counterclockwise neighboring triangle, rather than to any arbitrary triangle.

The Winged-Edge Structure

One widely used mesh data structure that stores connectivity information at the edges instead of the faces is the *winged-edge* data structure. This data structure makes edges the first-class citizen of the data structure, as illustrated in Figures 12.14 and 12.15.

In a winged-edge mesh, each edge stores pointers to the two vertices it connects (the *head* and *tail* vertices), the two faces it is part of (the *left* and *right* faces), and, most importantly, the next and previous edges in the counterclockwise traversal of its left and right faces (Figure 12.16). Each vertex and face also stores a pointer to a single, arbitrary edge that connects to it:

在这个结构中，三角形的邻居数组告诉相邻三角形的哪些边与该三角形的三条边共享。有了这些额外的信息，我们总是知道在哪里可以找到原始三角形，这导致了数据结构的不变量：对于任何三角形 t 的任何第 j 条边

$$t.\text{nbr}[j].t.\text{nbr}[t.\text{nbr}[j].i].t == t.$$

知道我们通过哪条边可以让我们立即知道要通过哪条边，以便继续遍历一个顶点，从而产生一个流线型的算法：

```
TrianglesOfVertex(v) {
    {t, i} = v.e;
    do {
        {t, i} = t.nbr[i];
        if (t.i = t.nbr[i]; i = (i+1) mod 3; while (t != v.e.t));
    }
}
```

三角邻结构相当紧凑。对于只有顶点位置的网格，我们每个顶点存储四个数字（三个坐标和一条边），每个面存储六个数字（三个顶点索引和三条边），每个顶点总共存储 $4nv + 6nt / 16nv$ 单位，而基本索引网格为 $9nv$ 。这里介绍的三角形邻居结构仅适用于流形网格，因为它依赖于返回到起始三角形来终止顶点邻居的遍历，这将不会发生在没有完整周期三角形的边界顶点。然而，将其推广到具有边界的流形并不困难，方法是为边界三角形的邻居引入合适的前哨值（如 1 ），并注意边界顶点指向最逆时针的相邻三角形，而不是任

翼缘结构

翼边数据结构是一种广泛使用的网格数据结构，它将连通性信息存储在边而不是面上。这个数据结构使边成为数据结构的一级公民，如图12.14和12.15所示。

在有翼边网格中，每条边都存储指向它连接的两个顶点（头和尾顶点）的指针，它所属的两个面（左面和右面），以及最重要的是，在counterclockwise遍历其左面和右面中的下一条和前一条边（图12.16）。每个顶点和面还存储一个指向连接到它的单个任意边的指针：

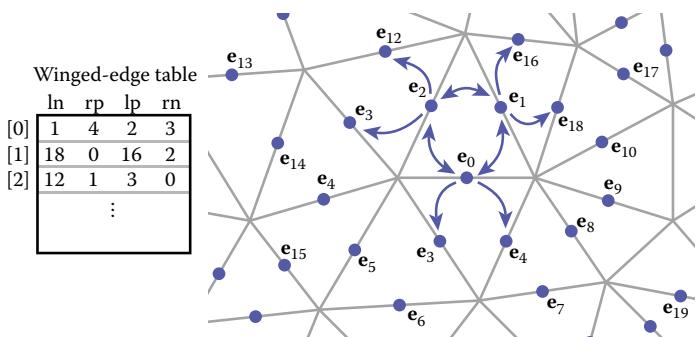


Figure 12.14. An example of a winged-edge mesh structure, stored in arrays.

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	e	b	d

Vertex	Edge
A	a
B	d
C	d
D	e

Face	Edge
0	a
1	c
2	d
3	a

Figure 12.15. A tetrahedron and the associated elements for a winged-edge data structure. The two small tables are not unique; each vertex and face stores any one of the edges with which it is associated.

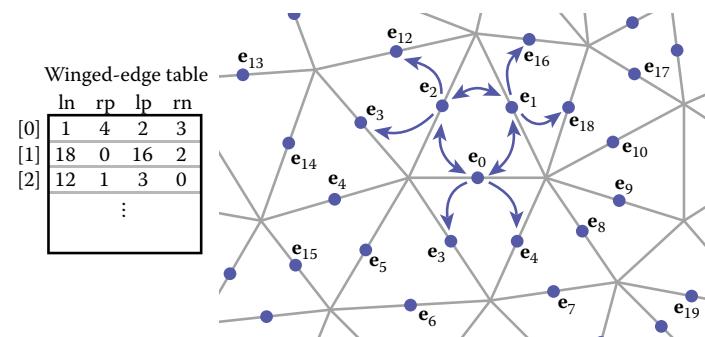


图12.14。翼边网格结构的一个例子，存储在数组中。

Edge	Vertex 1	Vertex 2	面向左	面向右	Pre d左	Succ left	Pred 权利	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	e	b	d

Vertex	Edge
A	a
B	d
C	d
D	e

Face	Edge
------	------

四面体和翼边数据结构的相关元素。两个小表不是唯一的;每个顶点和面部存储与其关联的任何一个边。



```

Edge {
    Edge lprev, lnext, rprev, rnxt;
    Vertex head, tail;
    Face left, right;
}

Face {
    // ... per-face data ...
    Edge e; // any adjacent edge
}

Vertex {
    // ... per-vertex data ...
    Edge e; // any incident edge
}

```

The winged-edge data structure supports constant-time access to the edges of a face or of a vertex, and from those edges the adjoining vertices or faces can be found:

```

EdgesOfVertex(v) {
    e = v.e;
    do {
        if (e.tail == v)
            e = e.lprev;
        else
            e = e.rprev;
    } while (e != v.e);
}

EdgesOfFace(f) {
    e = f.e;
    do {
        if (e.left == f)
            e = e.lnext;
        else
            e = e.rnxt;
    } while (e != f.e);
}

```

These same algorithms and data structures will work equally well in a polygon mesh that isn't limited to triangles; this is one important advantage of edge-based structures.

As with any data structure, the winged-edge data structure makes a variety of time/space tradeoffs. For example, we can eliminate the prev references. This makes it more difficult to traverse clockwise around faces or counterclockwise around vertices, but when we need to know the previous edge, we can always follow the successor edges in a circle until we get back to the original edge. This saves space, but it makes some operations slower. (See the chapter notes for more information on these tradeoffs).

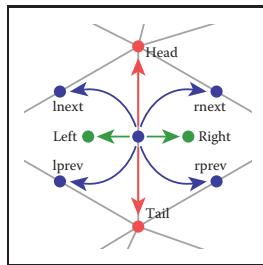


Figure 12.16. The references from an edge to the neighboring edges, faces, and vertices in the winged-edge structure.



```

Edge {
    Edge lprev, lnext, rprev, rnxt;
    顶点头, 尾; 面左, 右;
}

```

```

Face {
    ...人脸数据。..边缘e;
    任何相邻边
}

```

```

Vertex {
    ...每顶点数据。..边缘e;
    任何入射边缘
}

```

翼边数据结构支持对面或顶点的边进行恒定时间访问，从这些边可以找到相邻的顶点或面：

```

EdgesOfVertex(v) {
    e = v.e;
    do {
        if(e.tail==v)e=e.lprev;
        else

            e = e.rprev;
    } while (e != v.e);
}

```

```

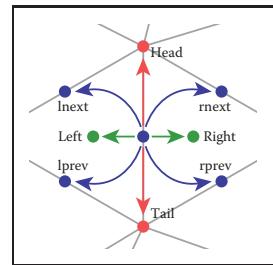
EdgesOfFace(f) {
    e = f.e;
    do {
        if(e.left==f)e=e.lnext;
        else

            e = e.rnxt;
    } while (e != f.e);
}

```

这些相同的算法和数据结构在不限于三角形的多边形网格中同样有效；这是基于边的结构的一个重要优势。

与任何数据结构一样，翼缘数据结构进行各种时间空间权衡。例如，我们可以消除prev引用。这使得顺时针绕面或逆时针绕顶点遍历更加困难，但是当我们需要知道前一条边时，我们总是可以在一个圆圈中跟随后继边，直到我们回到原始边。这样可以节省空间，但会使某些操作变慢。（有关这些权衡的更多信息，请参阅章节注释）。



Ref从边到翼边结构中的相邻边、面和顶点。

The Half-Edge Structure

The winged-edge structure is quite elegant, but it has one remaining awkwardness—the need to constantly check which way the edge is oriented before moving to the next edge. This check is directly analogous to the search we saw in the basic version of the triangle neighbor structure: we are looking to find out whether we entered the present edge from the head or from the tail. The solution is also almost indistinguishable: rather than storing data for each edge, we store data for each *half-edge*. There is one half-edge for each of the two triangles that share an edge, and the two half-edges are oriented oppositely, each oriented consistently with its own triangle.

The data normally stored in an edge is split between the two half-edges. Each half-edge points to the face on its side of the edge and to the vertex at its head, and each contains the edge pointers for its face. It also points to its neighbor on the other side of the edge, from which the other half of the information can be found. Like the winged-edge, a half-edge can contain pointers to both the previous and next half-edges around its face, or only to the next half-edge. We'll show the example that uses a single pointer.

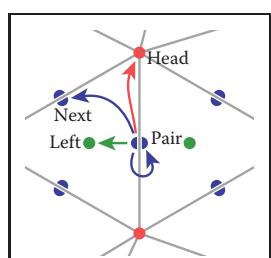


Figure 12.17. The references from a half-edge to its neighboring mesh components.

```

HEdge {
    HEdge pair, next;
    Vertex v;
    Face f;
}

Face {
    // ... per-face data ...
    HEdge h; // any h-edge of this face
}

Vertex {
    // ... per-vertex data ...
    HEdge h; // any h-edge pointing toward this vertex
}

```

Traversing a half-edge structure is just like traversing a winged-edge structure except that we no longer need to check orientation, and we follow the pair pointer to access the edges in the opposite face.

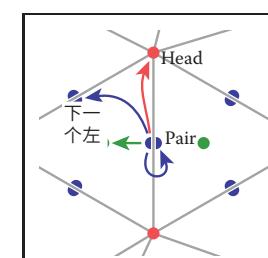
```

EdgesOfVertex(v) {
    h = v.h;
    do {
        h = h.pair.next;
    } while (h != v.h);
}

```

半边结构

翼缘结构非常优雅，但它还有一个尴尬的问题——在移动到下一个边缘之前，需要不断检查边缘的方向。这个检查直接类似于我们在三角形邻居结构的基本版本中看到的搜索：我们正在寻找我们是从头部还是从尾部输入当前边缘。解决方案也几乎无法区分：我们不是为每个边存储数据，而是为每个半边存储数据。对于共享一条边的两个三角形中的每一个都有一个半边，并且两个半边相反地定向，每个都与其自己的三角形一致地定向。



Ref从一个半边向它的相邻网格复合。

```

HEdge {
    树篱对, 下一个;
    Vertex v;
    Face f;
}

Face {
    ...人脸数据。...对冲h;
    这张脸的任何h边
}

Vertex {
    ...每顶点数据。...对冲h;
    指向此顶点的任何h边
}

```

遍历一个半边结构就像遍历一个有翼的边结构一样，除了我们不再需要检查方向，并且我们遵循pair指针来访问对面的边。

```

EdgesOfVertex(v) {
    h = v.h;
    do {
        h = h.pair.next;
    } while (h != v.h);
}

```

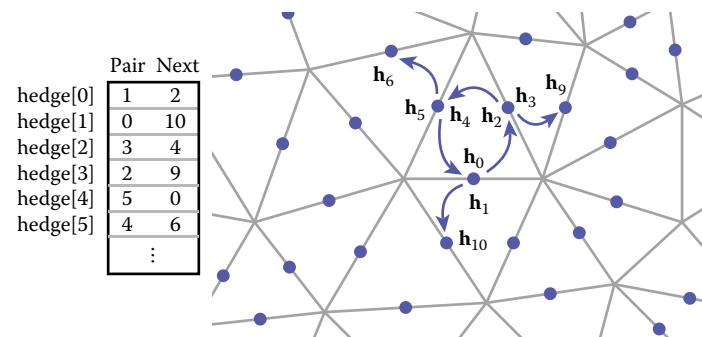


Figure 12.18. An example of a half-edge mesh structure, stored in arrays.

```
EdgesOfFace(f) {
    h = f.h;
    do {
        h = h.next;
    } while (h != f.h);
}
```

The vertex traversal here is clockwise, which is necessary because of omitting the `prev` pointer from the structure.

Because half-edges are generally allocated in pairs (at least in a mesh with no boundaries), many implementations can do away with the `pair` pointers. For instance, in an implementation based on array indexing (such as shown in Figure 12.18), the array can be arranged so that an even-numbered edge i always pairs with edge $i + 1$ and an odd-numbered edge j always pairs with edge $j - 1$.

In addition to the simple traversal algorithms shown in this chapter, all three of these mesh topology structures can support “mesh surgery” operations of various sorts, such as splitting or collapsing vertices, swapping edges, adding or removing triangles, etc.

12.2 Scene Graphs

A triangle mesh manages a collection of triangles that constitute an object in a scene, but another universal problem in graphics applications is arranging the objects in the desired positions. As we saw in Chapter 6, this is done using transformations, but complex scenes can contain a great many transformations and organizing them well makes the scene much easier to manipulate. Most scenes admit to a hierarchical organization, and the transformations can be managed according to this hierarchy using a *scene graph*.

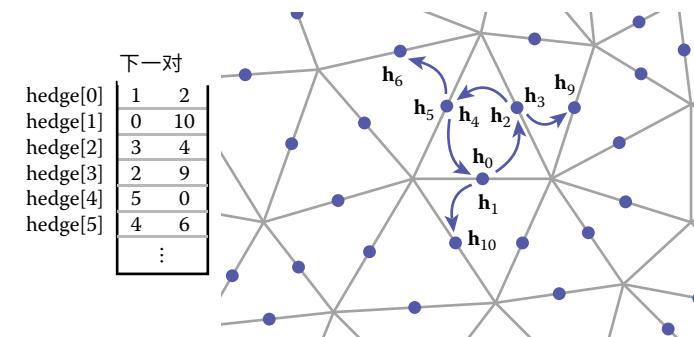


图12.18。一个半边网格结构的例子，存储在数组中。

```
EdgesOfFace(f) {
    h = f.h;
    do {
        h = h.next;
    } while (h != f.h);
}
```

这里的顶点遍历是顺时针的，这是必要的，因为从结构中省略了`prev`指针。

因为半边通常是成对分配的（至少在没有边界的网格中），许多实现可以取消对指针。例如，在基于阵列索引的实现中（如图12.18所示），阵列可以被布置成使得偶数编号的边 i 总是与边 $i+1$ 配对并且奇数编号的边 j 总是与边 $j-1$ 配对。除了本章所示的简单遍历算法之外，所有这三种网格拓扑结构都可以支持各种类型的“网格手术”操作，例如分割或折叠顶点，交换边，添加或删除三角形等。

12.2 场景图

三角形网格体管理构成场景中对象的三角形集合，但图形应用程序中的另一个普遍问题是将对象排列在所需的位置。正如我们在第6章中所看到的，这是使用转换来完成的，但是复杂的场景可以包含大量的转换，并且很好地组织它们使得场景更容易操作。大多数场景都接受分层组织，并且可以使用场景图根据此层次结构管理转换。

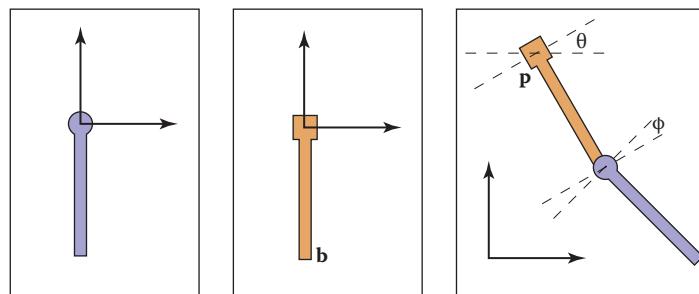


Figure 12.19. A hinged pendulum. On the left are the two pieces in their “local” coordinate systems. The hinge of the bottom piece is at point b and the attachment for the bottom piece is at its local origin. The degrees of freedom for the assembled object are the angles (θ, ϕ) and the location p of the top hinge.

To motivate the scene-graph data structure, we will use the hinged pendulum shown in Figure 12.19. Consider how we would draw the top part of the pendulum:

$$M_1 = \text{rotate}(\theta)$$

$$M_2 = \text{translate}(p)$$

$$M_3 = M_2 M_1$$

Apply M_3 to all points in upper pendulum

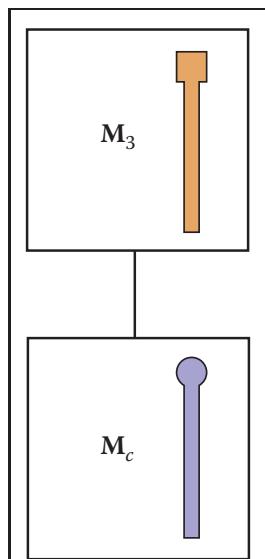


Figure 12.20. The scene graph for the hinged pendulum of Figure 12.19.

The bottom is more complicated, but we can take advantage of the fact that it is attached to the bottom of the upper pendulum at point b in the local coordinate system. First, we rotate the lower pendulum so that it is at an angle ϕ relative to its initial position. Then, we move it so that its top hinge is at point b . Now it is at the appropriate position in the local coordinates of the upper pendulum, and it can then be moved along with that coordinate system. The composite transform for the lower pendulum is:

$$M_a = \text{rotate}(\phi)$$

$$M_b = \text{translate}(b)$$

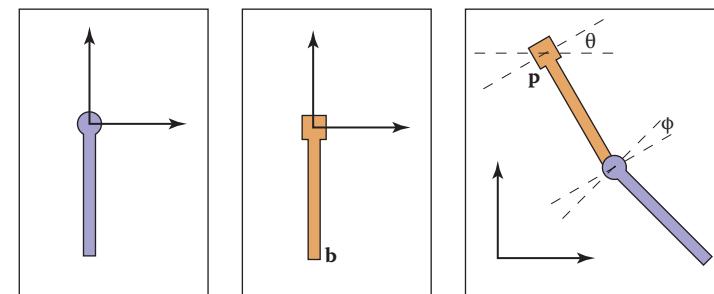
$$M_c = M_b M_a$$

$$M_d = M_3 M_c$$

Apply M_d to all points in lower pendulum

Thus, we see that the lower pendulum not only lives in its own local coordinate system, but also that coordinate system itself is moved along with that of the upper pendulum.

We can encode the pendulum in a data structure that makes management of these coordinate system issues easier, as shown in Figure 12.20. The appropriate matrix to apply to an object is just the product of all the matrices in the chain from



一个铰接的摆锤。左边是它们“局部”坐标系中的两个部分。底部部件的铰链在b点，底部部件的附件在其局部原点。装配对象的自由度是角度 (θ, ϕ) 和顶部铰链的位置 p 。

为了激励场景图数据结构，我们将使用图12.19所示的铰接摆。考虑一下我们如何绘制钟摆的顶部：

$$M_1 = \text{rotate}(\theta)$$

$$M_2 = \text{translate}(p)$$

$$M_3 = M_2 M_1$$

将 M_3 应用于上摆中的所有点

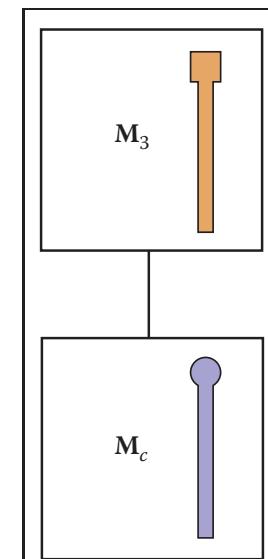


图12.19的铰接摆的场景图。

底部更复杂，但我们可以利用它在局部坐标系中的b点处附着在上摆的底部这一事实。首先，我们旋转下摆，使其相对于其初始位置成角度 ϕ 。然后，我们移动它，使其顶部铰链位于 b 点。现在是

在上摆的局部坐标中的适当位置处，并且其随后可随该坐标系一起移动。下摆的复合变换为：

$$M_a = \text{rotate}(\phi)$$

$$M_b = \text{translate}(b)$$

$$M_c = M_b M_a$$

$$M_d = M_3 M_c$$

将 M_d 应用于下摆中的所有点

因此，我们看到下摆不仅存在于自己的局部坐标系中，而且坐标系本身也与上摆的坐标系一起移动。

我们可以在数据结构中编码摆锤，使这些坐标系问题的管理更容易，如图 12.20 所示。应用于对象的适当矩阵只是链中所有矩阵的乘积。

the object to the root of the data structure. For example, consider the model of a ferry that has a car that can move freely on the deck of the ferry, and wheels that each move relative to the car as shown in Figure 12.21.

As with the pendulum, each object should be transformed by the product of the matrices in the path from the root to the object:

- **ferry** transform using M_0 ;
- **car body** transform using M_0M_1 ;
- **left wheel** transform using $M_0M_1M_2$;
- **right wheel** transform using $M_0M_1M_3$.

An efficient implementation can be achieved using a *matrix stack*, a data structure supported by many APIs. A matrix stack is manipulated using *push* and *pop* operations that add and delete matrices from the right-hand side of a matrix product.

For example, calling:

```
push(M0)
push(M1)
push(M2)
```

creates the active matrix $\mathbf{M} = M_0M_1M_2$. A subsequent call to *pop()* strips the last matrix added so that the active matrix becomes $\mathbf{M} = M_0M_1$. Combining the matrix stack with a recursive traversal of a scene graph gives us:

```
function traverse(node)
push(Mlocal)
draw object using composite matrix from stack
traverse(left child)
traverse(right child)
pop()
```

There are many variations on scene graphs but all follow the basic idea above.

12.3 Spatial Data Structures

In many, if not all, graphics applications, the ability to quickly locate geometric objects in particular regions of space is important. Ray tracers need to find objects that intersect rays; interactive applications navigating an environment need to find the objects visible from any given viewpoint; games and physical simulations require detecting when and where objects collide. All these needs can be supported

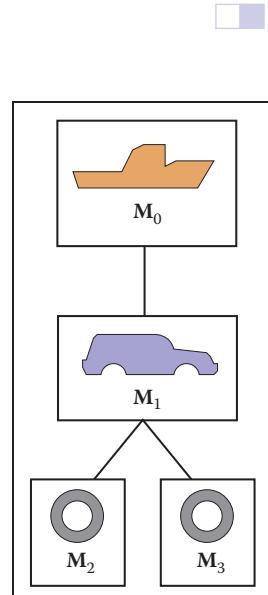


Figure 12.21. A ferry, a car on the ferry, and the wheels of the car (only two shown) are stored in a scene-graph.

到数据结构根的对象。例如，考虑渡轮的模型，该模型具有可以在渡轮甲板上自由移动的汽车，以及每个相对于汽车移动的车轮，如图12.21所示。

与钟摆一样，每个对象都应该由从根到对象的路径中的矩阵的乘积进行转换：

- * 使用 M_0 的渡轮变换；
- * 使用 M_0M_1 进行车身变换；
- * 使用 $M_0M_1M_2$ 进行左轮变换；
- * 使用 $M_0M_1M_3$ 进行右轮变换。

使用矩阵堆栈可以实现有效的实现，矩阵堆栈是许多 API 支持的数据结构。使用 *push* 和 *pop* 操作来操作矩阵堆栈，这些操作从矩阵产品的右侧添加和删除矩阵。例如，调用：

```
push(M0)
push(M1)
push(M2)
```

创建有源矩阵 $\mathbf{M} = M_0M_1M_2$ 。对 *pop()* 的后续调用会剥离添加的最后一个矩阵，以便活动矩阵变为 $\mathbf{M} = M_0M_1$ 。将矩阵堆栈与场景图的递归遍历相结合给出了我们：

```
function traverse(node)
push(mlocal)
draw object using composite matrix from stack
traverse(left child)
traverse(right child)
pop()
```

场景图有许多变化，但都遵循上面的基本思想。

12.3 空间数据结构

在许多（如果不是全部的话）图形应用中，在空间的特定区域中快速定位几何对象的能力是重要的。光线追踪器需要找到与光线相交的物体；导航环境的交互式应用程序需要找到从任何给定视点可见的物体；游戏和物理模拟需要检测物体碰撞的时间和所有这些需求都可以得到支持。

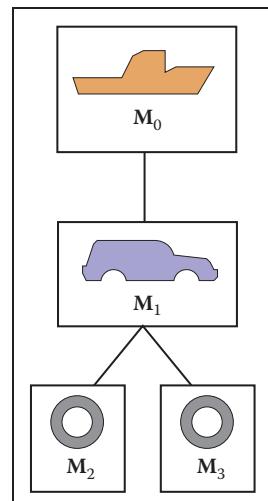


图12.21。渡轮、渡轮上的汽车和汽车的车轮（仅示出两个）存储在场景图中。

by various *spatial data structures* designed to organize objects in space so they can be looked up efficiently.

In this section we will discuss examples of three general classes of spatial data structures. Structures that group objects together into a hierarchy are *object partitioning* schemes: objects are divided into disjoint groups, but the groups may end up overlapping in space. Structures that divide space into disjoint regions are *space partitioning* schemes: space is divided into separate partitions, but one object may have to intersect more than one partition. Space partitioning schemes can be regular, in which space is divided into uniformly shaped pieces, or irregular, in which space is divided adaptively into irregular pieces, with smaller pieces where there are more and smaller objects.

We will use ray tracing as the primary motivation while discussing these structures, though they can all also be used for view culling or collision detection. In Chapter 4, all objects were looped over while checking for intersections. For N objects, this is an $O(N)$ linear search and is thus slow for large scenes. Like most search problems, the ray-object intersection can be computed in sub-linear time using “divide and conquer” techniques, provided we can create an ordered data structure as a preprocess. There are many techniques to do this.

This section discusses three of these techniques in detail: bounding volume hierarchies (Rubin & Whitted, 1980; Whitted, 1980; Goldsmith & Salmon, 1987), uniform spatial subdivision (Cleary, Wyvill, Birtwistle, & Vatti, 1983; Fujimoto, Tanaka, & Iwata, 1986; Amanatides & Woo, 1987), and binary space partitioning (Glassner, 1984; Jansen, 1986; Havran, 2000). An example of the first two strategies is shown in Figure 12.22.

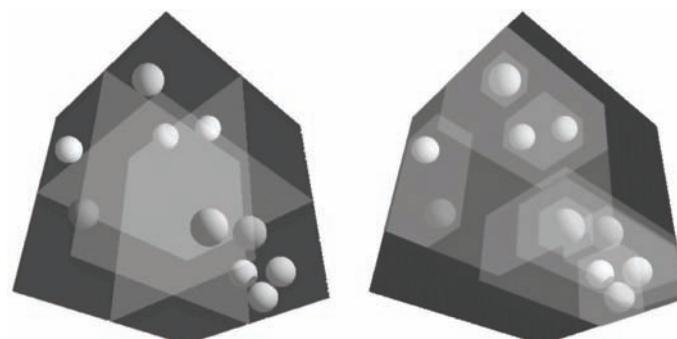


Figure 12.22. Left: a uniform partitioning of space. Right: adaptive bounding-box hierarchy.
Image courtesy David DeMarle.

通过各种空间数据结构设计来组织空间中的对象，以便有效地查找它们。

在本节中，我们将讨论空间数据结构的三个一般类的示例。将对象组合到层次结构中的结构是对象分割方案：对象被分成不相交的组，但这些组最终可能在空间中重叠。将空间划分为不相交区域的结构是空间分区方案：空间被划分为单独的分区，但一个对象可能必须与多个分区相交。空间划分方案可以是规则的，其中空间被划分为均匀形状的块，或者不规则的，其中空间被自适应地划分为不规则的块，其中有更多和更小的物体的较小

在讨论这些结构时，我们将使用光线追踪作为主要动机，尽管它们也可用于视图剔除或碰撞检测。在第4章中，在检查交叉点时，所有对象都被循环。对于 N 个对象，这是一个 $O(N)$ 线性搜索，因此对于大场景来说很慢。像大多数搜索问题一样，射线-物体相交可以使用“分而治之”技术在亚线性时间内计算，前提是我们可以创建有序的数据结构作为预处理。有很多技巧可以做到这一点。

本节详细讨论了其中的三种技术：边界体积层次结构（Rubin & Whitted, 1980; Whitted, 1980; Goldsmith & Salmon, 1987），均匀空间细分（Cleary, Wyvill, Birtwistle, & Vatti, 1983; Fujimoto, Tanaka, & Iwata, 1986; Amanatides & Woo, 1987）和二元空间分区（Glassner, 1984; Jansen, 1986; Havran, 2000年）。前两种策略的示例如图12.22所示。

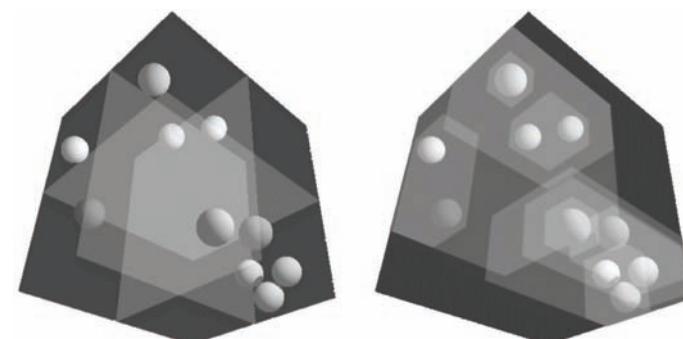


Figure 12.22. 左图：空间的统一划分。右：自适应边界框层次结构。
图片由DavidDeMarle提供。



12.3.1 Bounding Boxes

A key operation in most intersection-acceleration schemes is computing the intersection of a ray with a bounding box (Figure 12.23). This differs from conventional intersection tests in that we do not need to know where the ray hits the box; we only need to know whether it hits the box.

To build an algorithm for ray-box intersection, we begin by considering a 2D ray whose direction vector has positive x and y components. We can generalize this to arbitrary 3D rays later. The 2D bounding box is defined by two horizontal and two vertical lines:

$$\begin{aligned}x &= x_{\min}, \\x &= x_{\max}, \\y &= y_{\min}, \\y &= y_{\max}.\end{aligned}$$

The points bounded by these lines can be described in interval notation:

$$(x, y) \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}].$$

As shown in Figure 12.24, the intersection test can be phrased in terms of these intervals. First, we compute the ray parameter where the ray hits the line $x = x_{\min}$:

$$t_{x\min} = \frac{x_{\min} - x_e}{x_d}.$$

We then make similar computations for $t_{x\max}$, $t_{y\min}$, and $t_{y\max}$. The ray hits the box if and only if the intervals $[t_{x\min}, t_{x\max}]$ and $[t_{y\min}, t_{y\max}]$ overlap, i.e., their intersection is nonempty. In pseudocode this algorithm is:

```

 $t_{x\min} = (x_{\min} - x_e)/x_d$ 
 $t_{x\max} = (x_{\max} - x_e)/x_d$ 
 $t_{y\min} = (y_{\min} - y_e)/y_d$ 
 $t_{y\max} = (y_{\max} - y_e)/y_d$ 
if ( $t_{x\min} > t_{y\max}$ ) or ( $t_{y\min} > t_{x\max}$ ) then
    return false
else
    return true

```

The if statement may seem non-obvious. To see the logic of it, note that there is no overlap if the first interval is either entirely to the right or entirely to the left of the second interval.

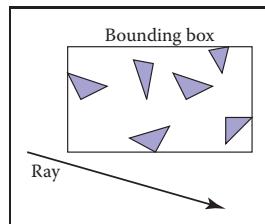


Figure 12.23. The ray is only tested for intersection with the surfaces if it hits the bounding box.

12.3.1 边界框

在大多数交叉加速方案中，一个关键的操作是用边界框计算射线的透射率（图12.23）。这与conventionalintersectiontests的不同之处在于我们不需要知道光线击中盒子的位置；我们只需要知道它是否击中盒子。

为了构建射线盒相交的算法，我们首先考虑方向矢量具有正 x 和 y 分量的二维射线。我们可以稍后将其推广到任意的3d射线。二维边界框由两条水平线和两条垂直线定义：

$$\begin{aligned}x &= x_{\min}, \\x &= x_{\max}, \\y &= y_{\min}, \\y &= y_{\max}.\end{aligned}$$

以这些线为界的点可以用区间表示法来描述：

$$(x, y) \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}].$$

如图12.24所示，相交测试可以用这些间隔来表述。首先，我们计算射线击中线 $x=x_{\min}$ 的射线参数：

$$t_{x\min} = \frac{\min(x_e - x_{\min})}{x_d}.$$

然后我们对 x_{\max} 、 y_{\min} 和 y_{\max} 进行类似的计算。当且仅当间隔 $[t_{x\min}, t_{x\max}]$ 和 $[t_{y\min}, t_{y\max}]$ 重叠时，射线才会击中框，即它们的交集是非空的。在伪代码中，该算法是：

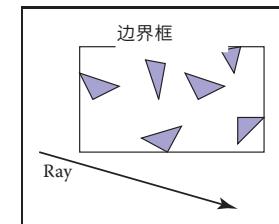
```

 $t_{x\min} = (x_{\min} - x_e)/x_d$ 
 $t_{x\max} = (x_{\max} - x_e)/x_d$ 
 $t_{y\min} = (y_{\min} - y_e)/y_d$ 
 $t_{y\max} = (y_{\max} - y_e)/y_d$ 
if ( $t_{x\min} > t_{y\max}$ ) or ( $t_{y\min} > t_{x\max}$ ) then
    return false
else
    return true

```

返回真实

If语句可能看起来不明显。要查看它的逻辑，请注意，如果第一个间隔完全位于右侧或完全位于第二个间隔的左侧，则没有重叠。



只有当光线击中边界框时，才会测试光线与曲面的相交。

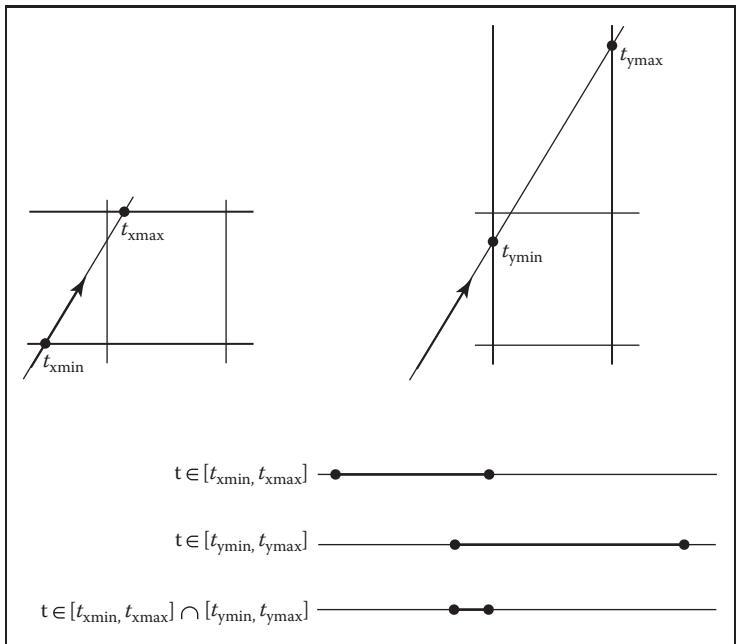


Figure 12.24. The ray will be inside the interval $x \in [x_{\text{min}}, x_{\text{max}}]$ for some interval in its parameter space $t \in [t_{x\text{min}}, t_{x\text{max}}]$. A similar interval exists for the y interval. The ray intersects the box if it is in both the x interval and y interval at the same time, i.e., the intersection of the two one-dimensional intervals is not empty.

The first thing we must address is the case when x_d or y_d is negative. If x_d is negative, then the ray will hit x_{max} before it hits x_{min} . Thus the code for computing $t_{x\text{min}}$ and $t_{x\text{max}}$ expands to:

```

if ( $x_d \geq 0$ ) then
     $t_{x\text{min}} = (x_{\text{min}} - x_e) / x_d$ 
     $t_{x\text{max}} = (x_{\text{max}} - x_e) / x_d$ 
else
     $t_{x\text{min}} = (x_{\text{max}} - x_e) / x_d$ 
     $t_{x\text{max}} = (x_{\text{min}} - x_e) / x_d$ 

```

A similar code expansion must be made for the y cases. A major concern is that horizontal and vertical rays have a zero value for y_d and x_d , respectively. This will cause divide by zero which may be a problem. However, before addressing this directly, we check whether IEEE floating point computation handles these cases gracefully for us. Recall from Section 1.5 the rules for divide by zero: for

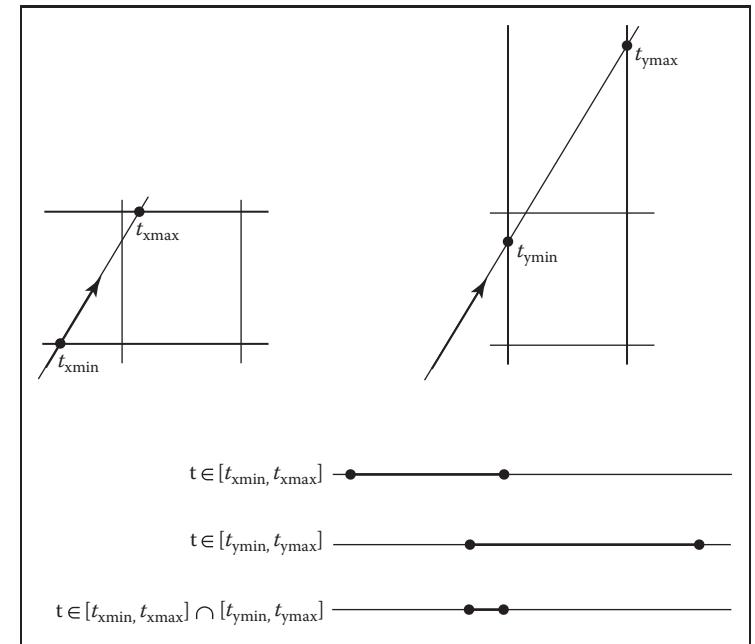


图12.24。射线将在其参数空间 $t \in [t_{x\text{min}}, t_{x\text{max}}]$ 中的某个区间 $x \in [x_{\text{min}}, x_{\text{max}}]$ 内。对于 y 区间存在类似的区间。射线在同一时刻处于 x 区间和 y 区间的情况下与框相交，即两个一维区间的交点不为空。

我们必须解决的第一件事是 x_d 或 y_d 为负的情况。如果 x_d 为负数，那么射线将在击中 x_{min} 之前击中 x_{max} 。因此计算 $t_{x\text{min}}$ 和 $t_{x\text{max}}$ 的代码扩展为：

如果($x_d \geq 0$)则
 $t_{x\text{min}}=(x_{\text{min}}-x_e)x_d$
 $t_{x\text{max}}=(x_{\text{max}}-x_e)x_d$
else
 $t_{x\text{min}}=(x_{\text{max}}-x_e)x_d$
 $t_{x\text{max}}=(x_{\text{min}}-x_e)x_d$

对于 y 情况，必须进行类似的代码扩展。一个主要的问题是水平和垂直射线对 y_d 和 x_d 分别具有零值。这将导致除以零，这可能是一个问题。但是，在直接解决这个问题之前，我们检查IEEE浮点运算是为我们优雅地处理这些情况。回顾第1.5节除以零的规则：



any positive real number a ,

$$\begin{aligned} +a/0 &= +\infty; \\ -a/0 &= -\infty. \end{aligned}$$

Consider the case of a vertical ray where $x_d = 0$ and $y_d > 0$. We can then calculate

$$\begin{aligned} t_{x\min} &= \frac{x_{\min} - x_e}{0}; \\ t_{x\max} &= \frac{x_{\max} - x_e}{0}. \end{aligned}$$

There are three possibilities of interest:

1. $x_e \leq x_{\min}$ (no hit);
2. $x_{\min} < x_e < x_{\max}$ (hit);
3. $x_{\max} \leq x_e$ (no hit).

For the first case we have

$$\begin{aligned} t_{x\min} &= \frac{\text{positive number}}{0}; \\ t_{x\max} &= \frac{\text{positive number}}{0}. \end{aligned}$$

This yields the interval $(t_{x\min}, t_{x\max}) = (\infty, \infty)$. That interval will not overlap with any interval, so there will be no hit, as desired. For the second case, we have

$$\begin{aligned} t_{x\min} &= \frac{\text{negative number}}{0}; \\ t_{x\max} &= \frac{\text{positive number}}{0}. \end{aligned}$$

This yields the interval $(t_{x\min}, t_{x\max}) = (-\infty, \infty)$ which will overlap with all intervals and thus will yield a hit as desired. The third case results in the interval $(-\infty, -\infty)$ which yields no hit, as desired. Because these cases work as desired, we need no special checks for them. As is often the case, IEEE floating point conventions are our ally. However, there is still a problem with this approach.

Consider the code segment:

```
if ( $x_d \geq 0$ ) then
     $t_{\min} = (x_{\min} - x_e)/x_d$ 
     $t_{\max} = (x_{\max} - x_e)/x_d$ 
```



任何正实数a

$$\begin{aligned} +a/0 &= +\infty; \\ -a/0 &= -\infty. \end{aligned}$$

考虑 $x_d=0$ 且 $y_d>0$ 的垂直射线的情况。然后我们可以计算

$$\begin{aligned} t_{x\min} &= \frac{x_{\min} - x_e}{0}; \\ t_{x\max} &= \frac{x_{\max} - x_e}{0}. \end{aligned}$$

感兴趣的可能有三种:

1. $x_e \leq x_{\min}$ (no hit);
2. $x_{\min} < x_e < x_{\max}$ (命中);
3. $x_{\max} \leq x_e$ (no hit).

对于第一个案例，我们有

$$\begin{aligned} t_{x\min} &= \text{正数} \\ ; & \\ t_{x\max} &= \text{正数} \\ . & \end{aligned}$$

这产生间隔 $(t_{x\min}, t_{x\max}) = (\infty, \infty)$ 。该间隔不会与任何间隔重叠，因此不会有命中，如所需。对于第二种情况，我们有

$$\begin{aligned} t_{x\min} &= \text{负数} \\ ; & \\ t_{x\max} &= \text{正数} \\ . & \end{aligned}$$

这将产生间隔 $(t_{x\min}, t_{x\max}) = (-\infty, \infty)$ ，它将与所有间隔重叠，因此将产生所需的命中。第三种情况导致间隔 $(-\infty, -\infty)$ 不产生命中，如所需的。由于这些情况如愿以偿，我们不需要对它们进行特殊检查。通常情况下，IEEE浮点约定是我们的盟友。但是，这种方法仍然存在问题。考虑代码段:

若($x_d \geq 0$)则
 $t_{\min} = (x_{\min} - x_e)/x_d$
 $ax = (x_{\max} - x_e)x_d$

```

else
     $t_{\min} = (x_{\max} - x_e)/x_d$ 
     $t_{\max} = (x_{\min} - x_e)/x_d$ 

```

This code breaks down when $x_d = -0$. This can be overcome by testing on the reciprocal of x_d (A. Williams, Barrus, Morley, & Shirley, 2005):

```

 $a = 1/x_d$ 
if ( $a \geq 0$ ) then
     $t_{\min} = a(x_{\min} - x_e)$ 
     $t_{\max} = a(x_{\max} - x_e)$ 
else
     $t_{\min} = a(x_{\max} - x_e)$ 
     $t_{\max} = a(x_{\min} - x_e)$ 

```

12.3.2 Hierarchical Bounding Boxes

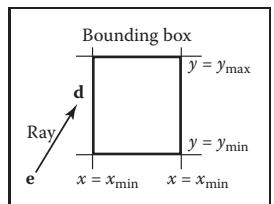


Figure 12.25. A 2D ray $e + t d$ is tested against a 2D bounding box.

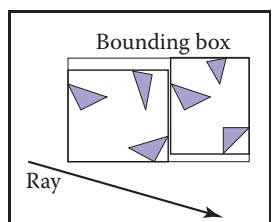


Figure 12.26. The bounding boxes can be nested by creating boxes around subsets of the model.

```

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 intersection is returned from each subtree) then
        return the closest of the two hits
    else if (an intersection is returned from exactly one subtree) then
        return that intersection
    else
        return false
else
    return false

```

```

else
     $t_{\min} = (x_{\max} - x_e)x_d$ 
     $t_{\max} = (x_{\min} - x_e)x_d$ 

```

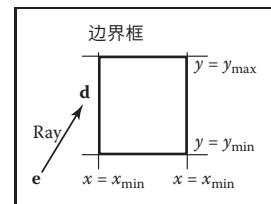
此代码在 $x_d = 0$ 时分解。这可以通过对 x_d 的倒数进行测试来克服 (A. Williams, Barrus, Morley, & Shirley, 2005) :

```

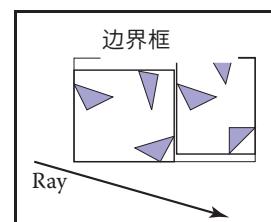
 $a = 1/x_d$  如果 ( $a \geq 0$ ) 则
 $t_{\min} = a(x_{\min} - x_e)x_d$ 
 $x = a(x_{\max} - x_e)$ 
 $t_{\min} = a(x_{\max} - x_e)x_d$ 
 $t_{\max} = a(x_{\min} - x_e)x_d$ 

```

12.3.2 分层边界框



对二维边界框测试二维射线
 $e+td$ 。



可以通过在模型子集周围创建框来嵌套绑定的ing框。

分层边界框的基本思想可以通过在所有对象周围放置一个轴对齐的3D边界框的常见策略来看出, 如图12.25所示。击中边界框的光线实际上比在蛮力搜索中计算更昂贵, 因为测试与框的交集不是免费的。但是, 错过盒子的光线比蛮力搜索便宜。这样的边界框可以通过对

对象放在一个盒子里 并在每个分区周围放置一个盒子 如图12.26所示.图12.27所示层次结构的数据结构可能是一棵树, 根部有一个大的边界框, 两个较小的边界框作为左和右子树。这些将反过来每个点到三个三角形的列表。射线与这种特殊的硬编码树的交集将是:

```

如果 (射线击中根框), 那么
    如果 (射线击中左子树框), 那么
        检查三个三角形是否相交 if (射线与
        右子树框相交) 然后
        检查其他三个三角形是否有交集 if (从每个子树返回的交
        叉点) 然后返回两个命中中最接近的 elseif (从正好一个
        子树返回交叉点) 然后返回该交叉点 else

```

返回错误的其他
返回错误

Some observations related to this algorithm are that there is no geometric ordering between the two subtrees, and there is no reason a ray might not hit both subtrees. Indeed, there is no reason that the two subtrees might not overlap.

A key point of such data hierarchies is that a box is guaranteed to bound all objects that are below it in the hierarchy, but they are *not* guaranteed to contain all objects that overlap it spatially, as shown in Figure 12.27. This makes this geometric search somewhat more complicated than a traditional binary search on strictly ordered one-dimensional data. The reader may note that several possible optimizations present themselves. We defer optimizations until we have a full hierarchical algorithm.

If we restrict the tree to be binary and require that each node in the tree have a bounding box, then this traversal code extends naturally. Further, assume that all nodes are either leaves in the tree and contain a primitive, or that they contain one or two subtrees.

The bvh-node class should be of type surface, so it should implement `surface::hit`. The data it contains should be simple:

```
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
```

The traversal code can then be called recursively in an object-oriented style:

```
function 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
```

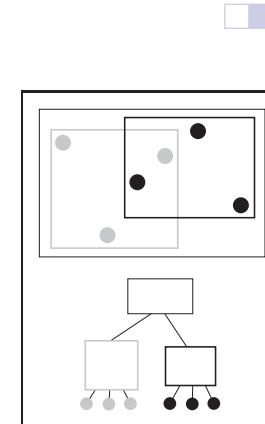


Figure 12.27. The gray box is a tree node that points to the three gray spheres, and the thick black box points to the three black spheres. Note that not all spheres enclosed by the box are guaranteed to be pointed to by the corresponding tree node.

与此算法相关的一些观察结果是，两个子树之间没有几何排序，并且没有理由光线可能不会击中两个子树。事实上，没有理由两个子树可能不重叠。

这种数据层次结构的一个关键点是，一个框保证绑定了层次结构中它下面的所有对象，但它们不能保证包含空间上与其重叠的所有对象，如图12.27所示。这使得

几何搜索比严格有序的一维数据上的传统二进制搜索要复杂得多。读者可能会注意到有几种可能的优化。我们推迟优化，直到我们有一个完整的分层算法。

如果我们限制树是二进制的，并且要求树中的每个节点都有一个边界框，那么这个遍历代码自然扩展。此外，假设所有节点都是树中的叶子并包含一个基元，或者它们包含一个或两个子树。

Bvh-node类应该是surface类型，因此它应该实现`surface::hit`。它包含的数据应该很简单：

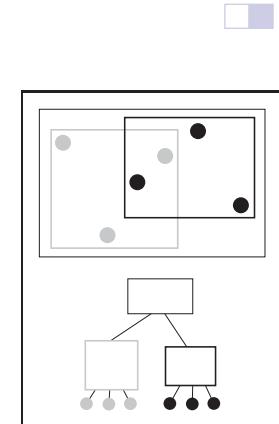
类bvh-表面的节点子类

```
virtual bool hit (ray e + td, real t0, real t1, hit-record rec)
virtual box bounding-box () surface-pointer
leftsurface-pointer rightboxbbox
```

然后可以以面向对象的样式递归地调用遍历代码：

```
函数boolbvh-node::hit(ray a + tb realt0 realt1 hit-recordrec)if(
bbox.hitbox (a+tb, t0, t1) )然后hit-recordlrec, rrecleft-
hit= (left=NULL) and (left→hit (a+tb, t0, t1, lrec) ) rig
ht-hit= (right=NULL) and (right→hit (a+tb, t0, t1, rrec)
) ) if (left-hitandright-hit) thenif (lrec. t<rrec. t)则rec=lr
ecelse
```

```
rec=rrecreturntrue
lseif(left-hit)thenre
c=lrecreturntrueels
eif(right-hit)then
```



灰色框是指向三个灰色球体的树节点，粗黑盒指向三个黑色球体。请注意，并非所有由框包围的球体都保证由corresponding树节点指向。

```

rec = rrec
return true
else
    return false
else
    return false

```

Note that because `left` and `right` point to surfaces rather than bvh-nodes specifically, we can let the virtual functions take care of distinguishing between internal and leaf nodes; the appropriate hit function will be called. Note that if the tree is built properly, we can eliminate the check for `left` being `NULL`. If we want to eliminate the check for `right` being `NULL`, we can replace `NULL` right pointers with a redundant pointer to `left`. This will end up checking `left` twice, but will eliminate the check throughout the tree. Whether that is worth it will depend on the details of tree construction.

There are many ways to build a tree for a bounding volume hierarchy. It is convenient to make the tree binary, roughly balanced, and to have the boxes of sibling subtrees not overlap too much. A heuristic to accomplish this is to sort the surfaces along an axis before dividing them into two sublists. If the axes are defined by an integer with $x = 0$, $y = 1$, and $z = 2$ we have:

```

function bvh-node::create(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 → bbox, right → bbox)

```

The quality of the tree can be improved by carefully choosing `AXIS` each time. One way to do this is to choose the axis such that the sum of the volumes of the bounding boxes of the two subtrees is minimized. This change compared to rotating through the axes will make little difference for scenes composed of isotopically distributed small objects, but it may help significantly in less well-behaved

`rec=rrec返
回trueelse`

**返回错误的其
他**
返回错误

请注意，因为`left`和`right`指向曲面而不是bvh-nodes，所以我们可以让虚函数来区分内部节点和叶节点;将调用适当的hit函数。请注意，如果树构建正确，我们可以消除对左为`NULL`的检查。如果我们想消除对右为`NULL`的检查，我们可以用向左的冗余指针替换空右指针。这将最终检查左两次，但将消除整个树的检查。这是否值得将取决于树木建造的细节。

有许多方法可以为边界卷层次结构构建树。使树二进制，大致平衡，并使兄弟子树的框不重叠太多是很方便的。实现此目的的启发式方法是在将表面划分为两个子列表之前沿轴对其进行排序。如果轴由 $x=0$, $y=1$ 和 $z=2$ 的整数定义，我们有：`function bvh-node::create(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]))`其他

按对象中心沿AXIS
left=newbvh-node(A[0..N2-1])(
轴+1)mod3)right=newbvh-node(A[N2..N-1], (轴
+1) mod3) bbox=组合 (左→bbox, 右→bbox)

通过每次仔细选择轴，可以提高树的质量。这样做的一种方法是选择轴，使得两个子树的边界框的体积之和最小化。与通过轴旋转相比，这种变化对于由同位素分布的小物体组成的场景几乎没有区别，但它可能会在表现不佳的情况下显着帮助

scenes. This code can also be made more efficient by doing just a partition rather than a full sort.

Another, and probably better, way to build the tree is to have the subtrees contain about the same amount of space rather than the same number of objects. To do this we partition the list based on space:

```
function bvh-node::create(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 bvh-node(A[0..k], (AXIS +1) mod 3)
        right = new bvh-node(A[k + 1..N - 1], (AXIS +1) mod 3)
        bbox = combine(left → bbox, right → bbox)
```

Although this results in an unbalanced tree, it allows for easy traversal of empty space and is cheaper to build because partitioning is cheaper than sorting.

12.3.3 Uniform Spatial Subdivision

Another strategy to reduce intersection tests is to divide space. This is fundamentally different from dividing objects as was done with hierarchical bounding volumes:

- In hierarchical bounding volumes, each object belongs to one of two sibling nodes, whereas a point in space may be inside both sibling nodes.
- In spatial subdivision, each point in space belongs to exactly one node, whereas objects may belong to many nodes.

In uniform spatial subdivision, the scene is partitioned into axis-aligned boxes. These boxes are all the same size, although they are not necessarily cubes. The ray traverses these boxes as shown in Figure 12.28. When an object is hit, the traversal ends.

场景。这段代码也可以通过只做一个分区而不是一个完整的排序来提高效率。

构建树的另一种可能更好的方法是让子树包含大约相同数量的空间，而不是相同数量的对象。为此，我们根据空间对列表进行分区：

```
函数bvh-node::create(object-array A int AXIS)n=a.lengthif(
N=1)thenleft=A[0]right=NULLbbox=bounding-box(A[0])els
eif(N=2)thenleft=A[0]right=a[1]bbox=combine(bounding-
box(A[0]) bounding-box(A[1]))else
```

沿着轴分区A找到a的边界框的中点m到长度为k和 (N - k) 的列表中，围绕m left=new bvh-node (A[0..k], (轴+1) mod3) 右=新bvh-节点 (A[k+1..N-1], (轴+1) mod3) bbox=组合 (左→bbox, 右→bbox)

虽然这会导致不平衡的树，但它允许轻松遍历空间，并且构建起来更便宜，因为分区比排序便宜。

12.3.3 均匀空间细分

减少交叉测试的另一种策略是划分空间。这与使用分层边界卷所做的分割对象有根本的不同：

- *在分层边界卷中，每个对象属于两个同级节点中的一个，而空间中的一个点可能位于两个同级节点中。
- 在空间细分中，空间中的每个点只属于一个节点，而对象可能属于许多节点。

在均匀空间细分中，场景被划分为轴对齐的框。这些盒子都是相同的大小，尽管它们不一定是立方体。射线遍历这些框，如图12.28所示。当一个对象被命中时，遍历结束。

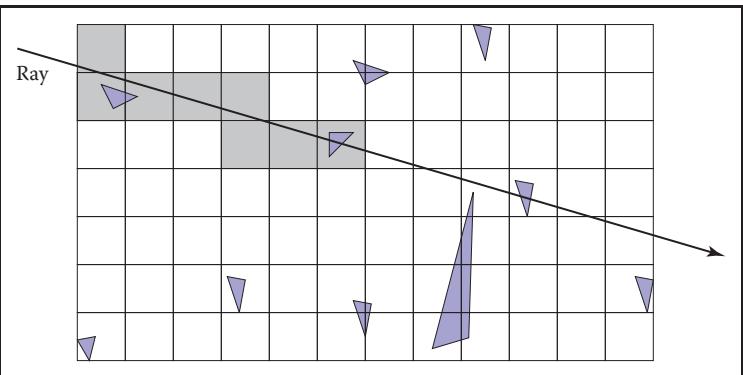


Figure 12.28. In uniform spatial subdivision, the ray is tracked forward through cells until an object in one of those cells is hit. In this example, only objects in the shaded cells are checked.

The grid itself should be a subclass of surface and should be implemented as a 3D array of pointers to surface. For empty cells these pointers are NULL. For cells with one object, the pointer points to that object. For cells with more than one object, the pointer can point to a list, another grid, or another data structure, such as a bounding volume hierarchy.

This traversal is done in an incremental fashion. The regularity comes from the way that a ray hits each set of parallel planes, as shown in Figure 12.29. To see how this traversal works, first consider the 2D case where the ray direction has positive x and y components and starts outside the grid. Assume the grid is bounded by points (x_{\min}, y_{\min}) and (x_{\max}, y_{\max}) . The grid has $n_x \times n_y$ cells.

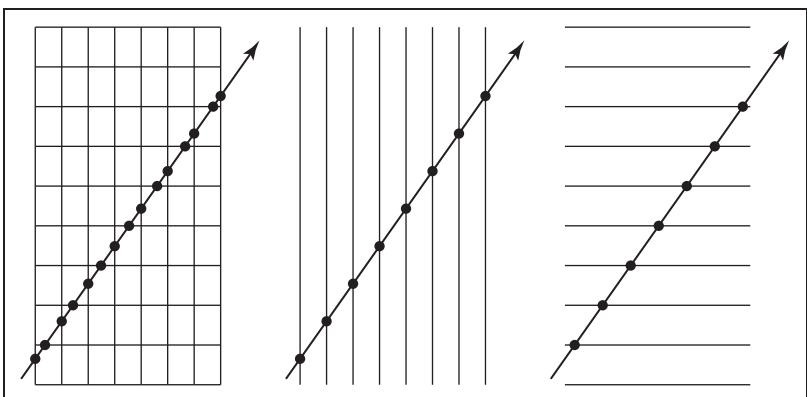


Figure 12.29. Although the pattern of cell hits seems irregular (left), the hits on sets of parallel planes are very even.

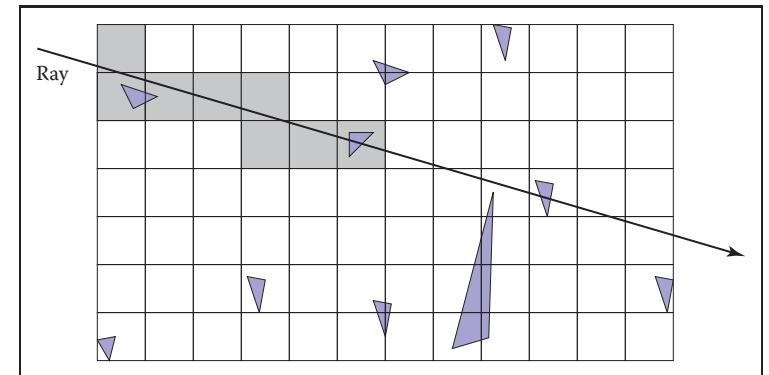
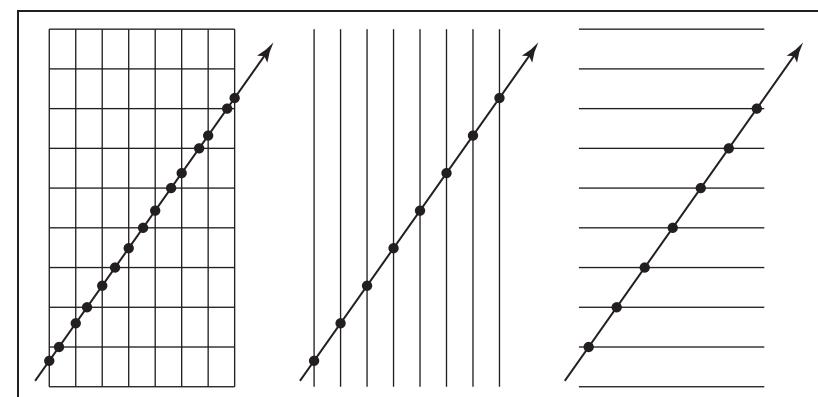


图12.28。在均匀空间细分中，射线通过单元格向前跟踪，直到其中一个单元格中的对象被命中。在此示例中，只检查阴影单元格中的对象。

网格本身应该是surface的子类，应该实现为指向surface的3d指针数组。对于空单元格，这些指针为NULL。对于具有一个对象的单元格，指针指向该对象。对于具有多个对象的单元格，指针可以指向列表、另一个网格或另一个数据结构，例如边界卷层次结构。

此遍历以增量方式完成。规律性来自射线撞击每组平行平面的方式，如图12.29所示。要了解这种遍历是如何工作的，首先考虑射线方向具有正 x 和 y 分量并从网格外部开始的二维情况。假设网格以点 (x_{\min}, y_{\min}) 和 (x_{\max}, y_{\max}) 为界。网格有 n 个 $\times n$ 个 y 单元。



虽然细胞命中的模式看起来不规则（左），但在平行平面上的命中是非常均匀的。

Our first order of business is to find the index (i, j) of the first cell hit by the ray $e + td$. Then, we need to traverse the cells in an appropriate order. The key parts to this algorithm are finding the initial cell (i, j) and deciding whether to increment i or j (Figure 12.30). Note that when we check for an intersection with objects in a cell, we restrict the range of t to be within the cell (Figure 12.31). Most implementations make the 3D array of type “pointer to surface.” To improve the locality of the traversal, the array can be tiled as discussed in Section 12.5.

12.3.4 Axis-Aligned Binary Space Partitioning

We can also partition space in a hierarchical data structure such as a *binary space partitioning tree* (BSP tree). This is similar to the BSP tree used for visibility sorting in Section 12.4, but it's most common to use axis-aligned, rather than polygon-aligned, cutting planes for ray intersection.

A node in this structure contains a single cutting plane and a left and right subtree. Each subtree contains all the objects on one side of the cutting plane. Objects that pass through the plane are stored in both subtrees. If we assume the cutting plane is parallel to the yz plane at $x = D$, then the node class is:

```
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
```

We generalize this to y and z cutting planes later. The intersection code can then be called recursively in an object-oriented style. The code considers the four cases shown in Figure 12.32. For our purposes, the origin of these rays is a point at parameter t_0 :

$$\mathbf{p} = \mathbf{a} + t_0 \mathbf{b}.$$

The four cases are:

1. The ray only interacts with the left subtree, and we need not test it for intersection with the cutting plane. It occurs for $x_p < D$ and $x_b < 0$.
2. The ray is tested against the left subtree, and if there are no hits, it is then tested against the right subtree. We need to find the ray parameter at $x = D$, so we can make sure we only test for intersections within the subtree. This case occurs for $x_p < D$ and $x_b > 0$.

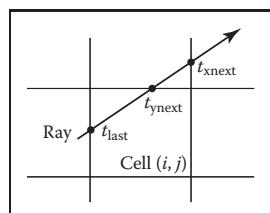


Figure 12.30. To decide whether we advance right or upward, we keep track of the intersections with the next vertical and horizontal boundary of the cell.

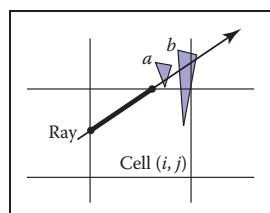


Figure 12.31. Only hits within the cell should be reported. Otherwise the case above would cause us to report hitting object *b* rather than object *a*.

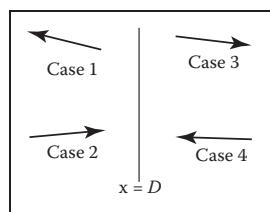


Figure 12.32. The four cases of how a ray relates to the BSP cutting plane $x = D$.

我们的第一个业务顺序是找到射线 $e+td$ 击中的第一个单元格的索引 (i, j) 。然后，我们需要以适当的顺序遍历单元格。该算法的关键部分是找到初始单元格 (i, j) 并决定是否递增 i 或 j (图12.30)。请注意，当我们检查一个路口

对于单元格中的对象，我们将 t 的范围限制在单元格内 (图12.31)。大多数实现使3d数组类型为“指向表面的指针。”为了提高遍历的局部性，可以像第12.5节中所讨论的那样平铺数组。

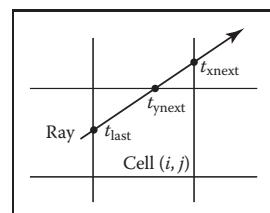
12.3.4 轴对齐二进制空间分区

我们还可以在分层数据结构中对空间进行分区，例如二进制空间分区树 (BSP树)。这类似于第12.4节中用于可见性排序的bsp树，但最常见的是使用轴对齐而不是多边形对齐的切割平面进行光线相交。

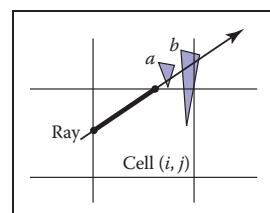
此结构中的节点包含单个切割平面和左右子树。每个子树包含切割平面一侧的所有对象。通过平面的对象存储在两个子树中。如果我们假设切割平面在 $x=D$ 处平行于 yz 平面，那么节点类是：

类bsp-表面的节点子类

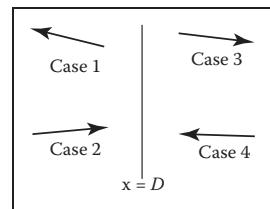
```
virtual bool hit (ray e + td, real t0, real t1, hit-recor drec)
virtual box bounding-box () surface-pointer
leftsurface-pointer rightrealD
```



为了决定我们是向右还是向前进，我们跟踪与细胞的下一个垂直和水平边界的交叉点。



只应重新移植单元格内的命中。
否则，上面的情况将导致我们报告击中对象b而不是objecta。



射线如何重新定位到BSP切割平面 $x=D$ 的四种情况。

$$\mathbf{p} = \mathbf{a} + t_0 \mathbf{b}.$$

这四种情况是：

1. 射线只与左子树相互作用，我们不需要测试它与切割平面的相交。它发生在 $x_p < D$ 和 $x_b < 0$ 。
2. 射线针对左子树进行测试，如果没有命中，则针对右子树进行测试。我们需要在 $x=D$ 处找到 ray 参数，因此我们可以确保我们只测试子树内的交叉点。这种情况发生在 $x_p < D$ 和 $x_b > 0$ 。

3. This case is analogous to case 1 and occurs for $x_p > D$ and $x_b > 0$.
4. This case is analogous to case 2 and occurs for $x_p > D$ and $x_b < 0$.

The resulting traversal code handling these cases in order is:

```
function bool bsp-node::hit(ray a + tb, real t0, real t1,
                           hit-record rec)
     $x_p = x_a + t_0 x_b$ 
    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 ( $t > t_1$ ) then
            return (left ≠ NULL) and (left→hit(a + tb, t0, t1, rec))
        if (left ≠ NULL) and (left→hit(a + tb, t0, t, rec)) then
            return true
        return (right ≠ NULL) and (right→hit(a + tb, t, t1, rec))
    else
        analogous code for cases 3 and 4
```

This is very clean code. However, to get it started, we need to hit some root object that includes a bounding box so we can initialize the traversal, t_0 and t_1 . An issue we have to address is that the cutting plane may be along any axis. We can add an integer index *axis* to the *bsp-node* class. If we allow an indexing operator for points, this will result in some simple modifications to the code above, for example,

$$x_p = x_a + t_0 x_b$$

would become

$$u_p = a[\text{axis}] + t_0 b[\text{axis}]$$

which will result in some additional array indexing, but will not generate more branches.

While the processing of a single bsp-node is faster than processing a bvh-node, the fact that a single surface may exist in more than one subtree means there are more nodes and, potentially, a higher memory use. How “well” the trees are built determines which is faster. Building the tree is similar to building the BVH tree. We can pick axes to split in a cycle, and we can split in half each time, or we can try to be more sophisticated in how we divide.

3. 这种情况类似于情况1，发生在 $x_p > D$ 和 $x_b > 0$ 。
4. 这种情况类似于情况2，发生在 $x_p > D$ 和 $x_b < 0$ 。

按顺序处理这些情况的结果遍历代码是：

```
函数 bool bsp-node::hit(ray a + tb, real t0, real t1, hit-record rec)
     $x_p = x_a + t_0 x_b$ 
    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 ( $t > t_1$ ) then
            return (left = NULL) and (left→hit(a + tb, t0, t1, rec))
        if (left = NULL) and (left→hit(a + tb, t0, t, rec)) then
            return true
        return (right = NULL) and (right→hit(a + tb, t, t1, rec))
    else
        analogous code for cases 3 and 4
```

案例3和案例4的类似代码

这是非常干净的代码。但是，要开始，我们需要击中一些包含边界框的根对象，以便我们可以初始化遍历， t_0 和 t_1 。我们必须解决的一个问题是切割平面可以沿着任何轴。我们可以在`bsp-node`类中添加一个整数索引轴。如果我们允许点的索引运算符，这将导致对上面的代码进行一些简单的修改，例如

$$x_p = x_a + t_0 x_b$$

将成为

$$u_p = a[\text{axis}] + t_0 b[\text{axis}]$$

这将导致一些额外的数组索引，但不会生成更多分支。

虽然单个bsp节点的处理比处理bvh节点快，但单个表面可能存在于多个子树中这一事实意味着有更多的节点，并且可能会有更高的内存使用率。树木建造得如何“好”决定了哪个更快。构建树类似于构建BVH树。我们可以选择轴在一个周期中分割，我们可以每次分割成两半，或者我们可以尝试更复杂的分割方式。



12.4 BSP Trees for Visibility

Another geometric problem in which spatial data structures can be used is determining the visibility ordering of objects in a scene with changing viewpoint.

If we are making many images of a fixed scene composed of planar polygons, from different viewpoints—as is often the case for applications such as games—we can use a *binary space partitioning* scheme closely related to the method for ray intersection discussed in the previous section. The difference is that for visibility sorting we use non-axis-aligned splitting planes, so that the planes can be made coincident with the polygons. This leads to an elegant algorithm known as the BSP tree algorithm to order the surfaces from front to back. The key aspect of the BSP tree is that it uses a preprocess to create a data structure that is useful for any viewpoint. So, as the viewpoint changes, the same data structure is used without change.

12.4.1 Overview of BSP Tree Algorithm

The BSP tree algorithm is an example of a *painter’s algorithm*. A painter’s algorithm draws every object from back-to-front, with each new polygon potentially overrawing previous polygons, as is shown in Figure 12.33. It can be implemented as follows:

```
sort objects back to front relative to viewpoint
for each object do
    draw object on screen
```

The problem with the first step (the sort) is that the relative order of multiple objects is not always well defined, even if the order of every pair of objects is. This problem is illustrated in Figure 12.34 where the three triangles form a *cycle*.

The BSP tree algorithm works on any scene composed of polygons where no polygon crosses the plane defined by any other polygon. This restriction is then relaxed by a preprocessing step. For the rest of this discussion, triangles are assumed to be the only primitive, but the ideas extend to arbitrary polygons.

The basic idea of the BSP tree can be illustrated with two triangles, T_1 and T_2 . We first recall (see Section 2.5.3) the implicit plane equation of the plane containing T_1 : $f_1(\mathbf{p}) = 0$. The key property of implicit planes that we wish to take advantage of is that for all points \mathbf{p}^+ on one side of the plane, $f_1(\mathbf{p}^+) > 0$; and for all points \mathbf{p}^- on the other side of the plane, $f_1(\mathbf{p}^-) < 0$. Using this property, we can find out on which side of the plane T_2 lies. Again, this assumes all three vertices of T_2 are on the same side of the plane. For discussion, assume



12.4 能见度的BSP树

可以使用空间数据结构的另一个几何问题是在具有变化视点的场景中挖掘对象的可见性排序。

如果我们从不同的视点制作由平面多边形组成的固定场景的许多图像—就像游戏等应用程序经常出现的情况一样—我们可以使用与上一节讨论的光线交不同之处在于，对于可见性排序，我们使用非轴对齐的分割平面，这样可以使平面与多边形重合。这导致了一个优雅的算法，称为BSP树算法，以排序表面从前到后。BSP树的关键方面是它使用预处理来创建对任何视点都有用的数据结构。因此，随着视点改变，相同的数据结构被使用而不改变。

12.4.1 BSP树算法概述

BSP树算法是画家算法的一个例子。画家的算法从后到前绘制每个对象，每个新的多边形都可能复盖以前的多边形，如图12.33所示。它可以实现如下：

相对于每个对象的视点，将对象向后排序
在屏幕上绘制对象

第一步（排序）的问题是，多个对象的相对顺序并不总是很好地定义，即使每对对象的顺序都是如此。这个问题如图12.34所示，其中三个三角形形成一个循环。BSP树算法适用于任何由多边形组成的场景，其中没有多边形穿过由任何其他多边形定义的平面。然后通过预处理步骤放宽此限制。在本讨论的其余部分中，三角形被假定为唯一的基元，但这些想法扩展到任意多边形。

BSP树的基本思想可以用两个三角形来说明， T_1 和 T_2 。我们首先回顾（见2.5.3节）包含 T_1 的平面的隐式平面方程： $f_1(\mathbf{p}) = 0$ 。我们希望利用的隐式平面的关键属性是，对于平面一侧的所有点 \mathbf{p}^+ ， $f_1(\mathbf{p}^+) > 0$ ；而对于平面另一侧的所有点 \mathbf{p}^- ， $f_1(\mathbf{p}^-) < 0$ 。使用此属性，我们可以找出 T_2 位于平面的哪一侧。再次，这假设 T_2 的所有三个顶点都在平面的同一侧。为了讨论，假设

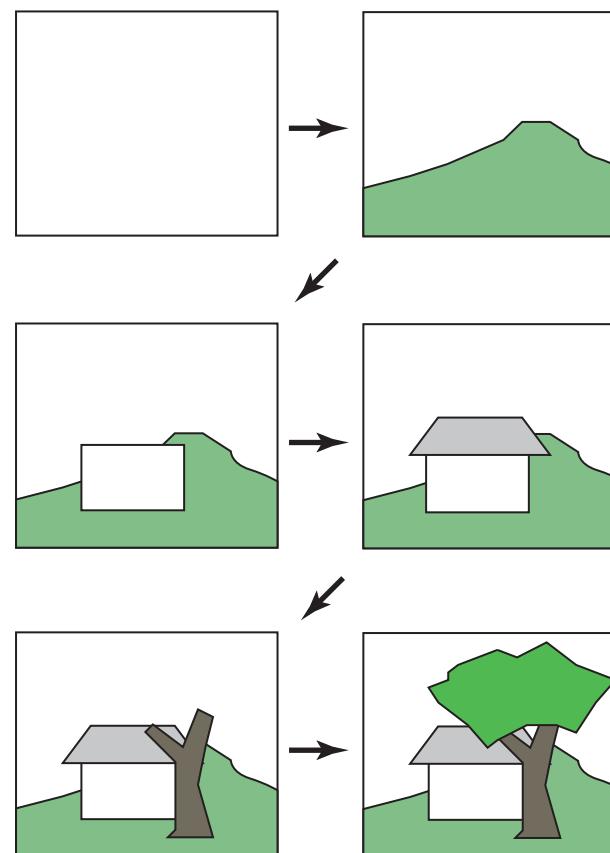


Figure 12.33. A painter's algorithm starts with a blank image and then draws the scene one object at a time from back-to-front, overdrawing whatever is already there. This automatically eliminates hidden surfaces.

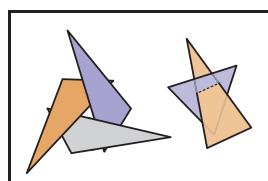


Figure 12.34. A cycle occurs if a global back-to-front ordering is not possible for a particular eye position.

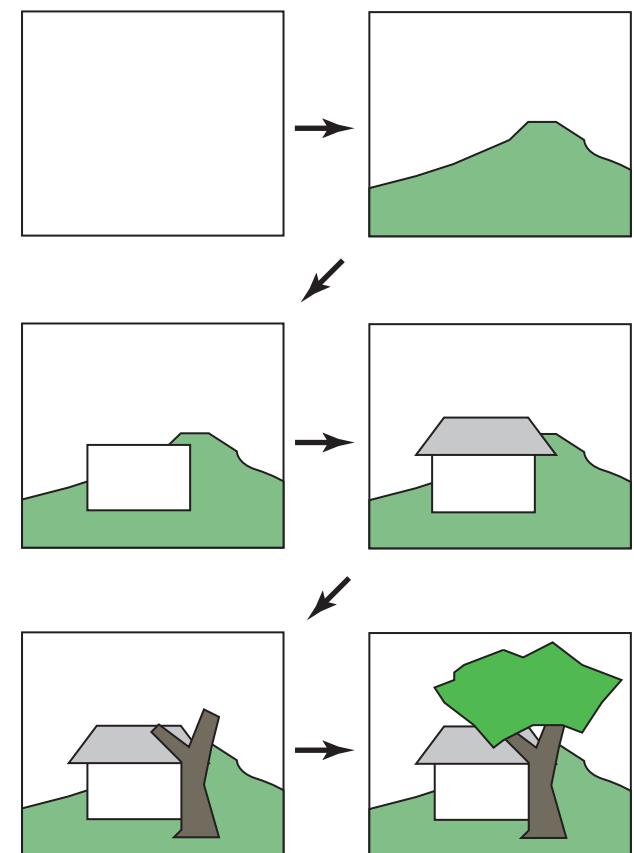
that T_2 is on the $f_1(p) < 0$ side of the plane. Then, we can draw T_1 and T_2 in the right order for any eyepoint e :

```

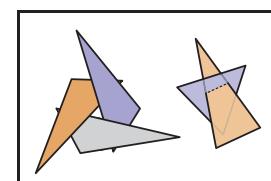
if ( $f_1(e) < 0$ ) then
    draw  $T_1$ 
    draw  $T_2$ 
else
    draw  $T_2$ 
    draw  $T_1$ 

```

The reason this works is that if T_2 and e are on the same side of the plane containing T_1 , there is no way for T_2 to be fully or partially blocked by T_1 as seen



画家的算法从一个空白图像开始，然后从后到前依次绘制一个对象的场景，透绘已经存在的任何东西。这会自动消除隐藏的表面。



一个周期occurs，如果一个全局的后到前的顺序是不可能的一个特定的眼睛位置。

t_2 在平面的 $f_1(p) < 0$ 侧。然后，我们可以为任何eyepointe以正确的顺序绘制 T_1 和 T_2 :

```

if ( $f_1(e) < 0$ ) then
    rawT1 drawT2 els
    e
    draw  $T_2$ 
    draw  $T_1$ 

```

这起作用的原因是，如果 T_2 和 e 在包含 T_1 的平面的同一侧，则 T_2 没有办法完全或部分地被 T_1 阻挡

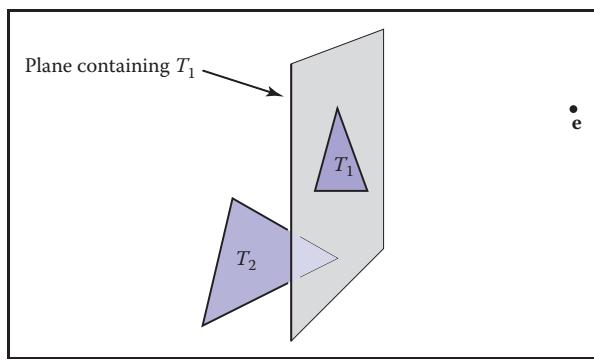


Figure 12.35. When e and T_2 are on opposite sides of the plane containing T_1 , then it is safe to draw T_2 first and T_1 second. If e and T_2 are on the same side of the plane, then T_1 should be drawn before T_2 . This is the core idea of the BSP tree algorithm.

from e , so it is safe to draw T_1 first. If e and T_2 are on opposite sides of the plane containing T_1 , then T_2 cannot fully or partially block T_1 , and the opposite drawing order is safe (Figure 12.35).

This observation can be generalized to many objects provided none of them span the plane defined by T_1 . If we use a binary tree data structure with T_1 as root, the *negative* branch of the tree contains all the triangles whose vertices have $f_i(p) < 0$, and the *positive* branch of the tree contains all the triangles whose vertices have $f_i(p) > 0$. We can draw in proper order as follows:

```

function draw(bsptree tree, point e)
  if (tree.empty) then
    return
  if ( $f_{\text{tree.root}}(e) < 0$ ) then
    draw(tree.plus, e)
    rasterize tree.triangle
    draw(tree.minus, e)
  else
    draw(tree.minus, e)
    rasterize tree.triangle
    draw(tree.plus, e)
  
```

The nice thing about that code is that it will work for any viewpoint e , so the tree can be precomputed. Note that, if each subtree is itself a tree, where the root triangle divides the other triangles into two groups relative to the plane containing it, the code will work as is. It can be made slightly more efficient by terminating the recursive calls one level higher, but the code will still be simple. A tree

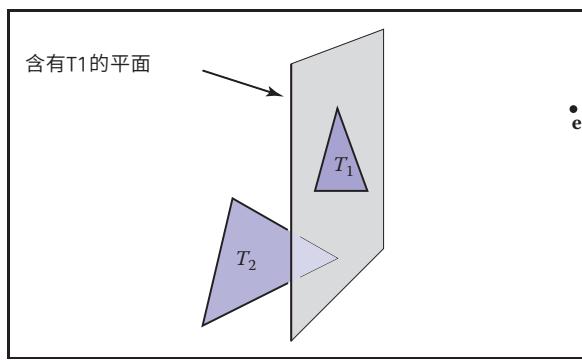


图12.35。 当 e 和 T_2 在包含 T_1 的平面的相对两侧时，那么首先绘制 T_2 和 T_1 第二是安全的。如果 e 和 T_2 在平面的同一侧，那么 T_1 应该在 T_2 之前绘制。这就是BSP树算法的核心思想。

从 e ，所以首先绘制 T_1 是安全的。如果 e 和 T_2 在包含 T_1 的平面的相对两侧，那么 T_2 不能完全或部分阻挡 T_1 ，并且相反的绘制顺序是安全的（图12.35）。

这种观察可以推广到许多物体，前提是它们都没有跨越 T_1 定义的平面。如果我们使用以 T_1 为根的二叉树数据结构，则树的负分支包含顶点具有 $f_i(p) < 0$ 的所有三角形，并且树的正分支包含顶点具有 $f_i(p) > 0$ 的所有三角形

我们可以按适当的顺序画画
as follows:

```

函数绘制 (bsptree树, 点e) if
  (树。空) 然后返回if(ftree。根
  ( $e) < 0$ ) 然后绘制(树。加, e
  ) 光栅化树。三角形绘制(树。
  减, e) 其他
  
```

```

draw(tree.minus, e)
rasterize tree.triangle
draw(tree.plus, e)
  
```

该代码的好处是它适用于任何视点 e ，因此可以预算树。请注意，如果每个子树本身是一棵树，其中根三角形相对于包含它的平面将其他三角形分成两组，则代码将按原样工作。通过将递归调用终止一个级别，可以稍微提高效率，但代码仍然很简单。一棵树

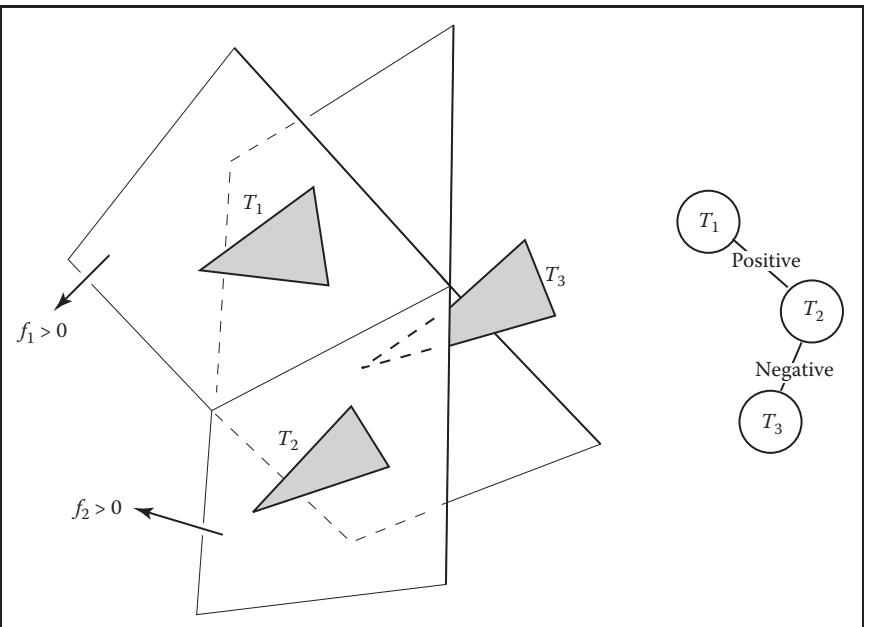


Figure 12.36. Three triangles and a BSP tree that is valid for them. The “positive” and “negative” are encoded by right and left subtree position, respectively.

illustrating this code is shown in Figure 12.36. As discussed in Section 2.5.5, the implicit equation for a point \mathbf{p} on a plane containing three non-colinear points \mathbf{a} , \mathbf{b} , and \mathbf{c} is

$$f(\mathbf{p}) = ((\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})) \cdot (\mathbf{p} - \mathbf{a}) = 0. \quad (12.1)$$

It can be faster to store the (A, B, C, D) of the implicit equation of the form

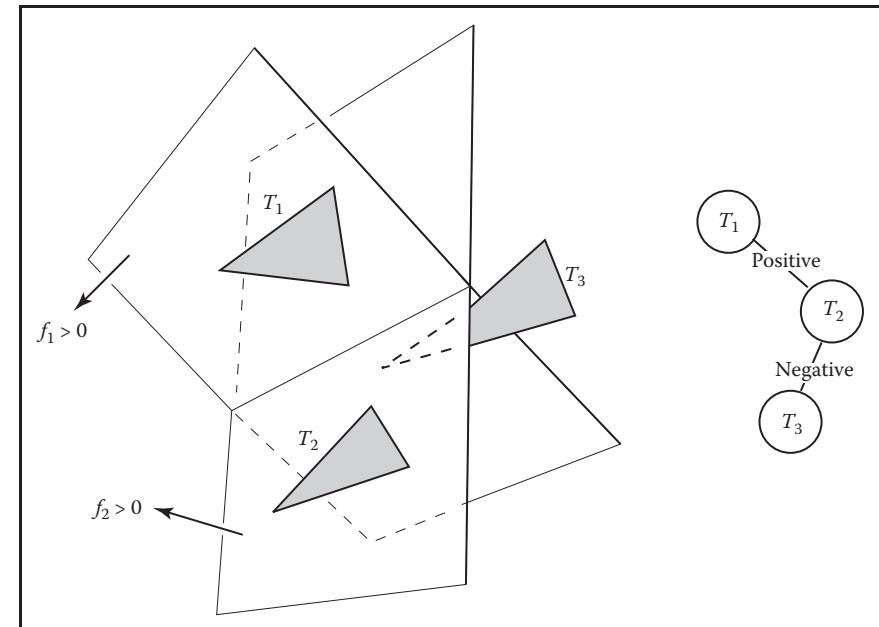
$$f(x, y, z) = Ax + By + Cz + D = 0. \quad (12.2)$$

Equations (12.1) and (12.2) are equivalent, as is clear when you recall that the gradient of the implicit equation is the normal to the triangle. The gradient of Equation (12.2) is $\mathbf{n} = (A, B, C)$ which is just the normal vector

$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}).$$

We can solve for D by plugging in any point on the plane, e.g., \mathbf{a} :

$$\begin{aligned} D &= -Ax_a - By_a - Cz_a \\ &= -\mathbf{n} \cdot \mathbf{a}. \end{aligned}$$



三个三角形和一个对它们有效的BSP树。“正”和“负”分别由右和左子树位置编码。

说明此代码如图12.36所示。如第2.5.5节所述，包含三个非共线点a, b和c的平面上的点p的隐式方程为

$$f(\mathbf{p}) = ((\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})) \cdot (\mathbf{p} - \mathbf{a}) = 0. \quad (12.1)$$

它可以更快地存储形式的隐式方程的 (A, B, C, D)

$$f(x, y, z) = Ax + By + Cz + D = 0. \quad (12.2)$$

方程 (12.1) 和 (12.2) 是等价的，当你回想起隐式方程的梯度是三角形的法线时，这是清楚的。方程 (12.2) 的梯度是 $\mathbf{n} = (A, B, C)$ 这只是法向量

$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}).$$

我们可以通过插入平面上的任何点来求解D，例如：

$$\begin{aligned} D &= -Ax_a - By_a - Cz_a = -\mathbf{n} \cdot \mathbf{a} \\ &= \dots \end{aligned}$$



This suggests the form:

$$\begin{aligned}f(\mathbf{p}) &= \mathbf{n} \cdot \mathbf{p} - \mathbf{n} \cdot \mathbf{a} \\&= \mathbf{n} \cdot (\mathbf{p} - \mathbf{a}) \\&= 0,\end{aligned}$$

which is the same as Equation (12.1) once you recall that \mathbf{n} is computed using the cross product. Which form of the plane equation you use and whether you store only the vertices, \mathbf{n} and the vertices, or \mathbf{n}, D , and the vertices, is probably a matter of taste—a classic time-storage tradeoff that will be settled best by profiling. For debugging, using Equation (12.1) is probably the best.

The only issue that prevents the code above from working in general is that one cannot guarantee that a triangle can be uniquely classified on one side of a plane or the other. It can have two vertices on one side of the plane and the third on the other. Or it can have vertices on the plane. This is handled by splitting the triangle into smaller triangles using the plane to “cut” them.

12.4.2 Building the Tree

If none of the triangles in the dataset cross each other’s planes, so that all triangles are on one side of all other triangles, a BSP tree that can be traversed using the code above can be built using the following algorithm:

```
tree-root = node( $T_1$ )
for  $i \in \{2, \dots, N\}$  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
```



这表明形式:

$$\begin{aligned}f(\mathbf{p}) &= \mathbf{n} \cdot \mathbf{p} - \mathbf{n} \cdot \mathbf{a} \\&= \mathbf{n} \cdot (\mathbf{p} - \mathbf{a}) \\&= 0,\end{aligned}$$

这与公式 (12.1) 相同，一旦你记得 \mathbf{n} 是用叉积计算的。您使用哪种形式的平面方程，以及您是否只存储顶点， \mathbf{n} 和顶点，或 \mathbf{n}, D 和顶点，可能是一个品味问题—一个经典的时间存储权衡，将通过分析解决最好。对于调试，使用公式 (12.1) 可能是最好的。

阻止上述代码一般工作的唯一问题是，不能保证三角形可以在平面的一侧或另一侧唯一分类。它可以在平面的一侧有两个顶点，在另一侧有第三个顶点。或者它可以在平面上具有顶点。这是通过使用平面将三角形分割成更小的三角形来“切割”它们来处理的。

12.4.2 建造树

如果数据集中没有一个三角形交叉彼此的平面，使得所有三角形都在所有其他三角形的一侧，则可以使用以下算法构建可以使用上面代码遍历的BSP树:

```
tree-root=node( $T_1$ )
for  $i \in \{2 \dots N\}$  做树根。add( $T_i$ )
函数add(triangle $T$ )
    if( $f(a) < 0$  and  $f(b) < 0$  and  $f(c) < 0$ )thenif(negative-subtree is empty)thennegative-subtree=node( $T$ )else
        if( $f(a) > 0$  and  $f(b) > 0$  and  $f(c) > 0$ )thenif(positive-subtree is empty)thenpositive-subtree=node( $T$ )else
            positive-subtree.add ( $T$ )
    else
       我们认为这种情况是不可能的
```

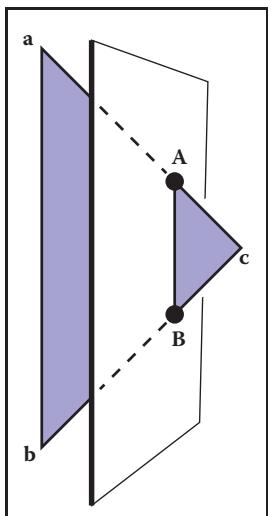


Figure 12.37. When a triangle spans a plane, there will be one vertex on one side and two on the other.

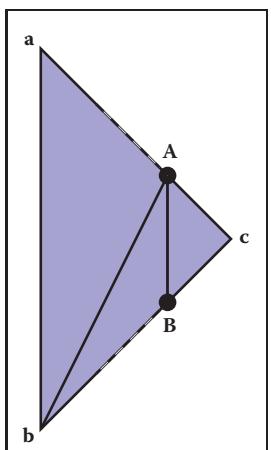


Figure 12.38. When a triangle is cut, we break it into three triangles, none of which span the cutting plane.

The only thing we need to fix is the case where the triangle crosses the dividing plane, as shown in Figure 12.37. Assume, for simplicity, that the triangle has vertices **a** and **b** on one side of the plane, and vertex **c** is on the other side. In this case, we can find the intersection points **A** and **B** and cut the triangle into three new triangles with vertices

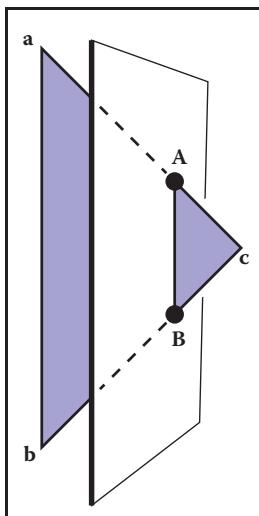
$$\begin{aligned}T_1 &= (\mathbf{a}, \mathbf{b}, \mathbf{A}), \\T_2 &= (\mathbf{b}, \mathbf{B}, \mathbf{A}), \\T_3 &= (\mathbf{A}, \mathbf{B}, \mathbf{c}),\end{aligned}$$

as shown in Figure 12.38. This order of vertices is important so that the direction of the normal remains the same as for the original triangle. If we assume that $f(\mathbf{c}) < 0$, the following code could add these three triangles to the tree assuming the positive and negative subtrees are not empty:

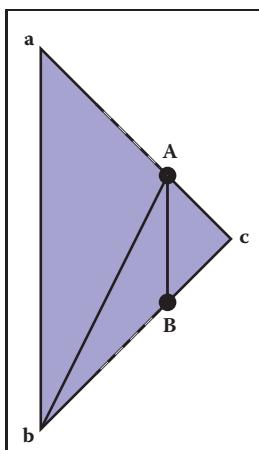
```
positive-subtree = node ( $T_1$ )
positive-subtree = node ( $T_2$ )
negative-subtree = node ( $T_3$ )
```

A precision problem that will plague a naive implementation occurs when a vertex is very near the splitting plane. For example, if we have two vertices on one side of the splitting plane and the other vertex is only an extremely small distance on the other side, we will create a new triangle almost the same as the old one, a triangle that is a sliver, and a triangle of almost zero size. It would be better to detect this as a special case and not split into three new triangles. One might expect this case to be rare, but because many models have tessellated planes and triangles with shared vertices, it occurs frequently, and thus must be handled carefully. Some simple manipulations that accomplish this are:

```
function add( triangle  $T$  )
    fa =  $f(\mathbf{a})$ 
    fb =  $f(\mathbf{b})$ 
    fc =  $f(\mathbf{c})$ 
    if ( $abs(fa) < \epsilon$ ) then
        fa = 0
    if ( $abs(fb) < \epsilon$ ) then
        fb = 0
    if ( $abs(fc) < \epsilon$ ) then
        fc = 0
    if ( $fa \leq 0$  and  $fb \leq 0$  and  $fc \leq 0$ ) then
        if (negative subtree is empty) then
            negative-subtree = node( $T$ )
```



当三角跨越一个平面时，一侧会有一个顶点，另一侧会有两个顶点。



当一个三角被切割时，我们将它分成三个三角形，其中没有一个跨越切割平面。

我们唯一需要修复的是三角形穿过分割平面的情况，如图12.37所示。为了简单起见，假设三角形在平面的一侧具有顶点a和b，顶点c在另一侧。在这种情况下，我们可以找到交点A和B，并将三角形切割成三个具有顶点的新三角形

$$\begin{aligned}T_1 &= (\mathbf{a}, \mathbf{b}, \mathbf{A}), \\T_2 &= (\mathbf{b}, \mathbf{B}, \mathbf{A}), \\T_3 &= (\mathbf{A}, \mathbf{B}, \mathbf{c}),\end{aligned}$$

如图12.38所示。顶点的这个顺序是重要的，以便方向法线与原始三角形保持相同。如果我们假设 $f(c) < 0$ ，下面的代码可以将这三个三角形添加到树中，假设正负子树不为空：

```
positive-subtree = node ( $T_1$ )
positive-subtree = node ( $T_2$ )
negative-subtree = node ( $T_3$ )
```

当顶点非常靠近分裂平面时，会出现一个困扰天真实现的精度问题。例如，如果我们在分割平面的一侧有两个顶点，而另一个顶点在分割平面上只有一个非常小的距离。

另一边，我们将创建一个几乎与旧三角形相同的新三角形，一个三角形是条子，一个几乎为零大小的三角形。这将是更好地检测到这是一个特殊情况，而不是分裂成三个新的三角形。人们可能会期望这种情况很少见，但由于许多模型具有具有共享顶点的tessellated平面和三角形，因此它经常发生，因此必须小心处理。完成此操作的一些简单操作是：

```
函数add(三角形T)
fa=f(a)fb=f(b)fc=f(c)
f(abs(fa)<∞)thenfa=0if(abs(fb)<∞)the
nfb=0if(abs(fc)<∞)thenfc=0if(fa≤0and
fb≤0andfc≤0)thenif(负子树为空)then
negative-subtree=node(T)
```

```

else
    negative-subtree.add( $T$ )
else if ( $fa \geq 0$  and  $fb \geq 0$  and  $fc \geq 0$ ) 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

```

This takes any vertex whose f value is within ϵ of the plane and counts it as positive or negative. The constant ϵ is a small positive real chosen by the user. The technique above is a rare instance where testing for floating-point equality is useful and works because the zero value is set rather than being computed. Comparing for equality with a computed floating-point value is almost never advisable, but we are not doing that.

12.4.3 Cutting Triangles

Filling out the details of the last case “cut triangle into three triangles and add to each side” is straightforward, but tedious. We should take advantage of the BSP tree construction as a preprocess where highest efficiency is not key. Instead, we should attempt to have a clean compact code. A nice trick is to force many of the cases into one by ensuring that c is on one side of the plane and the other two vertices are on the other. This is easily done with swaps. Filling out the details in the final else statement (assuming the subtrees are nonempty for simplicity) gives:

```

if ( $fa * fc \geq 0$ ) then
    swap( $fb, fc$ )
    swap( $b, c$ )
    swap( $fa, fb$ )
    swap( $a, b$ )
else if ( $fb * fc \geq 0$ ) then
    swap( $fa, fc$ )
    swap( $a, c$ )
    swap( $fa, fb$ )
    swap( $a, b$ )
compute A
compute B
 $T_1 = (a, b, A)$ 
 $T_2 = (b, B, A)$ 

```

```

else
    负-子树。add( $T$ )elseif( $fa \geq 0$ and $fb \geq 0$ and $f
c \geq 0$ )thenif(positivesubtreeisempty)then
    positive-subtree=node( $T$ )else
        positive-subtree.add( $T$ )
    else
        将三角形切成三个三角形，并添加到每边

```

这取f值在平面内的任何顶点，并将其计为正或负。常数△是用户选择的小正实。上面的技术是一个罕见的实例，其中浮点相等性测试是有用的，并且有效，因为零值被设置而不是被计算。比较计算浮点值的相等性几乎是不可取的，但我们没有这样做。

12.4.3 切割三角形

填写最后一个案例的细节“将三角形切成三个三角形并添加到每一边”很简单，但很乏味。我们应该利用BSP树结构作为一个预处理，其中最高效率不是关键。相反，我们应该尝试有一个干净的紧凑代码。一个很好的技巧是通过确保 c 在平面的一侧，而另外两个顶点在另一侧，将许多情况强制为一个。这很容易用掉期完成。在最终的else语句中填写详细信息（为了简单起见，假设子树是非空的）给出：

```

if( $fa \in fc \geq 0$ )thenswap(f
b fc)swap(b c)swap(fa f
b)swap(a b)elseif( $fb \in f
c \geq 0$ )thenswap(fa fc)sw
ap(a c)swap(fa fb)swap
(a b)computeacompute
b T1=(a b A)T2=(b B A)

```

```

 $T_3 = (\mathbf{A}, \mathbf{B}, \mathbf{c})$ 
if ( $fc \geq 0$ ) then
    negative-subtree.add( $T_1$ )
    negative-subtree.add( $T_2$ )
    positive-subtree.add( $T_3$ )
else
    positive-subtree.add( $T_1$ )
    positive-subtree.add( $T_2$ )
    negative-subtree.add( $T_3$ )

```

This code takes advantage of the fact that the product of a and b are positive if they have the same sign—thus, the first if statement. If vertices are swapped, we must do two swaps to keep the vertices ordered counterclockwise. Note that exactly one of the vertices may lie exactly on the plane, in which case the code above will work, but one of the generated triangles will have zero area. This can be handled by ignoring the possibility, which is not that risky, because the rasterization code must handle zero-area triangles in screen space (i.e., edge-on triangles). You can also add a check that does not add zero-area triangles to the tree. Finally, you can put in a special case for when exactly one of fa , fb , and fc is zero which cuts the triangle into two triangles.

To compute \mathbf{A} and \mathbf{B} , a line segment and implicit plane intersection is needed. For example, the parametric line connecting \mathbf{a} and \mathbf{c} is

$$\mathbf{p}(t) = \mathbf{a} + t(\mathbf{c} - \mathbf{a}).$$

The point of intersection with the plane $\mathbf{n} \cdot \mathbf{p} + D = 0$ is found by plugging $\mathbf{p}(t)$ into the plane equation:

$$\mathbf{n} \cdot (\mathbf{a} + t(\mathbf{c} - \mathbf{a})) + D = 0,$$

and solving for t :

$$t = -\frac{\mathbf{n} \cdot \mathbf{a} + D}{\mathbf{n} \cdot (\mathbf{c} - \mathbf{a})}.$$

Calling this solution t_A , we can write the expression for \mathbf{A} :

$$\mathbf{A} = \mathbf{a} + t_A(\mathbf{c} - \mathbf{a}).$$

A similar computation will give \mathbf{B} .

```

T3=(A B c)若(
fc≥0)则
    ee.add( $T_1$ )
    negative-subtree.add( $T_2$ )
    positive-subtree.add( $T_3$ )
else
    positive-subtree.add( $T_1$ )
    positive-subtree.add( $T_2$ )
    negative-subtree.add( $T_3$ )

```

这段代码利用了一个事实，即 a 和 b 的乘积是正的，如果它们具有相同的符号—因此，第一个if语句。如果顶点被交换，我们必须做两次交换以保持顶点逆时针顺序。请注意，其中一个顶点可能正好位于平面上，在这种情况下，上面的代码将起作用，但其中一个生成的三角形将具有零面积。这可以通过忽略可能性来处理，这并不是那么危险，因为光栅化代码必须处理屏幕空间中的零区域三角形（即边缘三角形）。您还可以添加不向树中添加零区域三角形的检查。最后，你可以在一个特殊的情况下，当 fa , fb 和 fc 中的一个是零时，将三角形切割成两个三角形。

要计算 A 和 B ，需要线段和隐式平面相交。
例如，连接 a 和 c 的参数线是

$$\mathbf{p}(t) = \mathbf{a} + t(\mathbf{c} - \mathbf{a}).$$

与平面 $n \cdot p + D = 0$ 的交点通过将 $p(t)$ 插入平面方程中找到：

$$\mathbf{n} \cdot (\mathbf{a} + t(\mathbf{c} - \mathbf{a})) + D = 0,$$

并求解 t :

$$t = -\frac{\mathbf{n} \cdot \mathbf{a} + D}{\mathbf{n} \cdot (\mathbf{c} - \mathbf{a})}.$$

调用此解决方案 t_A ，我们可以为 A 编写表达式：

$$\mathbf{A} = \mathbf{a} + t_A(\mathbf{c} - \mathbf{a}).$$

类似的计算将给出 B 。



12.4.4 Optimizing the Tree

The efficiency of tree creation is much less of a concern than tree traversal because it is a preprocess. The traversal of the BSP tree takes time proportional to the number of nodes in the tree. (How well balanced the tree is does not matter.) There will be one node for each triangle, including the triangles that are created as a result of splitting. This number can depend on the order in which triangles are added to the tree. For example, in Figure 12.39, if T_1 is the root, there will be two nodes in the tree, but if T_2 is the root, there will be more nodes, because T_1 will be split.

It is difficult to find the “best” order of triangles to add to the tree. For N triangles, there are $N!$ orderings that are possible. So trying all orderings is not usually feasible. Alternatively, some predetermined number of orderings can be tried from a random collection of permutations, and the best one can be kept for the final tree.

The splitting algorithm described above splits one triangle into three triangles. It could be more efficient to split a triangle into a triangle and a convex quadrilateral. This is probably not worth it if all input models have only triangles, but would be easy to support for implementations that accommodate arbitrary polygons.

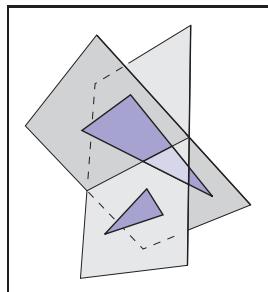


Figure 12.39. Using T_1 as the root of a BSP tree will result in a tree with two nodes. Using T_2 as the root will require a cut and thus make a larger tree.

12.5 Tiling Multidimensional Arrays

Effectively utilizing the memory hierarchy is a crucial task in designing algorithms for modern architectures. Making sure that multidimensional arrays have data in a “nice” arrangement is accomplished by *tiling*, sometimes also called *bricking*. A traditional 2D array is stored as a 1D array together with an indexing mechanism; for example, an N_x by N_y array is stored in a 1D array of length $N_x N_y$ and the 2D index (x, y) (which runs from $(0, 0)$ to $(N_x - 1, N_y - 1)$) maps to the 1D index (running from 0 to $N_x N_y - 1$) using the formula

$$\text{index} = x + N_x y.$$

An example of how that memory lays out is shown in Figure 12.40. A problem with this layout is that although two adjacent array elements that are in the same row are next to each other in memory, two adjacent elements in the same column will be separated by N_x elements in memory. This can cause poor memory locality for large N_x . The standard solution to this is to use *tiles* to make memory

$j = 2$	8	9	10	11
$j = 1$	4	5	6	7
$j = 0$	0	1	2	3
$i = 0$	$i = 1$	$i = 2$	$i = 3$	

Figure 12.40. The memory layout for an untilled 2D array with $N_x = 4$ and $N_y = 3$.



12.4.4 优化树

树创建的效率比树遍历要少得多，因为它是一个预处理。Bsp树的遍历需要与树中的节点数成比例的时间。（树的平衡程度如何并不重要。）每个三角形将有一个节点，包括由于分割而创建的三角形。这个数字可以取决于三角形被添加到树中的顺序。例如，在图12.39中，如果 T_1 是根，则树中将有两个节点，但如果 T_2 是根，则会有更多节点，因为 T_1 将被拆分。

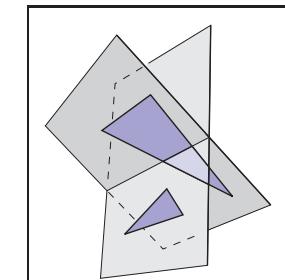


图12.39。使用T1作为BSP树的根将重新出现在具有两个节点的树中。使用T2作为根将重新获得一个切割从而使一个更大的树。

很难找到要添加到树中的三角形的“最佳”顺序。对于 N 个三角形，有 $N!$ 可能的命令。所以尝试所有的顺序通常是不可行的。或者，一些预定数量的顺序可以是

从一个随机的排列集合中尝试，最好的一个可以保留给最终的树。

上面描述的分割算法将一个三角形分割成三个三角形。

将三角形拆分为三角形和convex四边形可能会更有效。如果所有输入模型都只有三角形，那么这可能是不值得的，但是很容易支持容纳任意多边形的实现。

12.5 平铺多维数组

有效利用内存层次结构是设计现代体系结构算法的关键任务。确保多维数组具有“漂亮”排列的数据是通过平铺（有时也称为砖砌）来完成的。传统的2D数组与索引机制一起存储为1D数组；例如， nxbyNy 数组存储在长度为 NxNy 的1D数组中，并且2D索引 (x, y) （从 $(0, 0)$ 到 $(\text{Nx}-1, \text{Ny}-1)$ ）

$$\text{index} = x + N_x y.$$

图12.40显示了该内存如何布局的一个例子。这种布局的一个问题是，尽管在同一行中的两个相邻阵列元素在内存中彼此相邻，但在同一列中的两个相邻元素将在内存中被 N_x 个 x 元素分开。这可能会导致大 Nx 的内存局部性差。对此的标准解决方案是使用瓷砖来制作内存

$j = 2$	8	9	10	11
$j = 1$	4	5	6	7
$j = 0$	0	1	2	3
$i = 0$	$i = 1$	$i = 2$	$i = 3$	

Nx=4且Ny=3的未标记2D数组的内存布局。

$j = 2$	8	9	12
$j = 1$	2	3	6
$j = 0$	0	1	4
$i = 0$	$i = 1$	$i = 2$	$i = 3$

Figure 12.41. The memory layout for a tiled 2D array with $N_x = 4$ and $N_y = 3$ and 2×2 tiles. Note that padding on the top of the array is needed because N_y is not a multiple of the tile size two.

A key question is what size to make the tiles. In practice, they should be similar to the memory-unit size on the machine. For example, if we are using 16-bit (2-byte) data values on a machine with 128-byte cache lines, 8×8 tiles fit exactly in a cache line. However, using 32-bit floating-point numbers, which fit 32 elements to a cache line, 5×5 tiles are a bit too small and 6×6 tiles are a bit too large. Because there are also coarser-sized memory units such as pages, hierarchical tiling with similar logic can be useful.

12.5.1 One-Level Tiling for 2D Arrays

If we assume an $N_x \times N_y$ array decomposed into square $n \times n$ tiles (Figure 12.42), then the number of tiles required is

$$B_x = N_x/n, \\ B_y = N_y/n.$$

Here, we assume that n divides N_x and N_y exactly. When this is not true, the array should be *padded*. For example, if $N_x = 15$ and $n = 4$, then N_x should be changed to 16. To work out a formula for indexing such an array, we first find

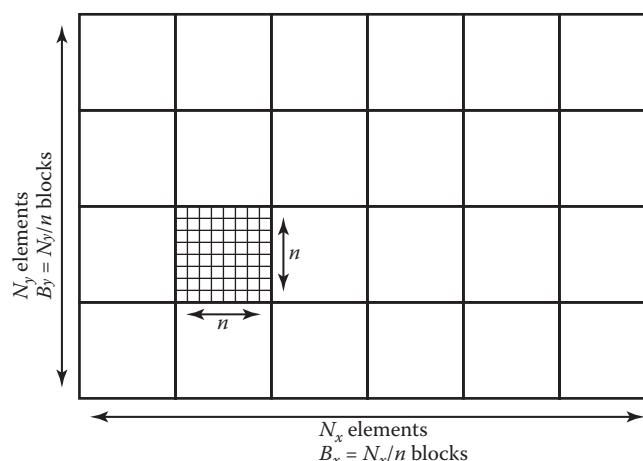


Figure 12.42. A tiled 2D array composed of $B_x \times B_y$ tiles each of size n by n .

$j = 2$	8	9	12
$j = 1$	2	3	6
$j = 0$	0	1	4
$i = 0$	$i = 1$	$i = 2$	$i = 3$

具有 $N_x=4$ 和 $N_y=3$ 和 2×2 瓦片的平铺2D阵列的存储器布局。请注意，需要数组顶部的填充，因为 N_y 不是切片大小2的倍数。

行和列的局部性更相等。一个例子如图12.41所示，其中使用 2×2 瓷砖。索引这样的数组的细节将在下一节中讨论。一个更复杂的例子，在一个3D阵列上有两个层次的平铺，然后复盖。

一个关键问题是制作瓷砖的尺寸。在实践中，它们应该类似于机器上的内存单元大小。例如，如果我们使用

16位（2字节）数据值在具有128字节缓存行的机器上， 8×8 块恰好适合缓存行。但是，使用32位浮点数（将32个元素适合缓存行）， 5×5 块有点太小， 6×6 块有点太大。因为还存在较粗大小的存储器单元，例如页，所以具有类似逻辑的分层平铺可能是有用的。

12.5.1 二维阵列的一级平铺

如果我们假设一个 $n \times n$ 阵列分解成正方形 $n \times n$ 个瓦片（图12.42），那么所需的瓦片数量是

$$B_x = N_x/n, \\ B_y = N_y/n.$$

在这里，我们假设 n 将 N_x 和 N_y 完全分开。当这不是真的时，阵列应该被填充。例如，如果 $N_x=15$ 且 $n=4$ ，则 N_x 应更改为16。为了计算出索引这样一个数组的公式，我们首先找到

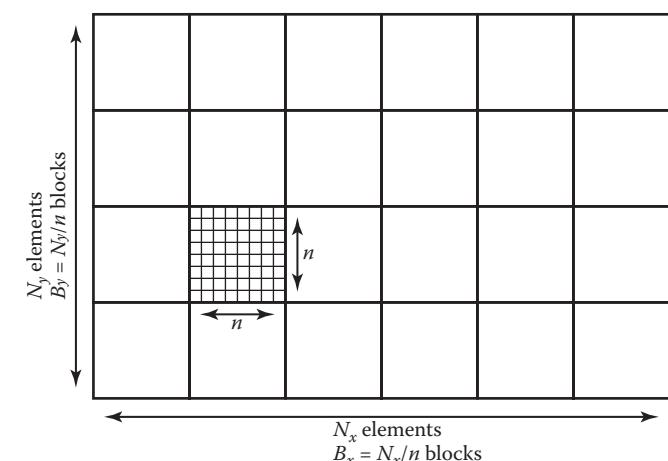


图12.42. 由每个尺寸为n乘n的 $B_x \times B_y$ 瓦片组成的平铺2D阵列。

the tile indices (b_x, b_y) that give the row/column for the tiles (the tiles themselves form a 2D array):

$$\begin{aligned} b_x &= x \div n, \\ b_y &= y \div n, \end{aligned}$$

where \div is integer division, e.g., $12 \div 5 = 2$. If we order the tiles along rows as shown in Figure 12.40, then the index of the first element of the tile (b_x, b_y) is

$$\text{index} = n^2(B_x b_y + b_x).$$

The memory in that tile is arranged like a traditional 2D array as shown in Figure 12.41. The partial offsets (x', y') inside the tile are

$$\begin{aligned} x' &= x \bmod n, \\ y' &= y \bmod n, \end{aligned}$$

where mod is the remainder operator, e.g., $12 \bmod 5 = 2$. Therefore, the offset inside the tile is

$$\text{offset} = y'n + x'.$$

Thus the full formula for finding the 1D index element (x, y) in an $N_x \times N_y$ array with $n \times n$ tiles is

$$\begin{aligned} \text{index} &= n^2(B_x b_y + b_x) + y'n + x', \\ &= n^2((N_x \div n)(y \div n) + x \div n) + (y \bmod n)n + (x \bmod n). \end{aligned}$$

This expression contains many integer multiplication, divide and modulus operations, which are costly on some processors. When n is a power of two, these operations can be converted to bitshifts and bitwise logical operations. However, as noted above, the ideal size is not always a power of two. Some of the multiplications can be converted to shift/add operations, but the divide and modulus operations are more problematic. The indices could be computed incrementally, but this would require tracking counters, with numerous comparisons and poor branch prediction performance.

However, there is a simple solution; note that the index expression can be written as

$$\text{index} = F_x(x) + F_y(y),$$

where

$$\begin{aligned} F_x(x) &= n^2(x \div n) + (x \bmod n), \\ F_y(y) &= n^2(N_x \div n)(y \div n) + (y \bmod n)n. \end{aligned}$$

We tabulate F_x and F_y , and use x and y to find the index into the data array. These tables will consist of N_x and N_y elements, respectively. The total size of the tables will fit in the primary data cache of the processor, even for very large data set sizes.

为瓦片提供行列的瓦片索引 (bx, by) (瓦片本身形成一个2D数组) :

$$\begin{aligned} b_x &= x \div n, \\ b_y &= y \div n, \end{aligned}$$

其中 \div 是整数除法, 例如, $12 \div 5 = 2$ 。如果我们按照图12.40所示的行对瓦片进行排序, 那么瓦片的第一个元素 (bx, by) 的索引是

$$\text{index} = n^2(B_x b_y + b_x).$$

如图12.41所示, 该图块中的内存就像传统的2D阵列一样排列。图块内的部分偏移量 (x', y') 为

$$\begin{aligned} x' &= x \bmod n, \\ y' &= y \bmod n, \end{aligned}$$

其中 mod 是余数运算符, 例如, $12 \bmod 5 = 2$ 。因此, 瓦片内部的偏移量为

$$\text{offset} = y'n + x'.$$

因此, 在具有 $n \times n$ 个瓦片的 $N_x \times N_y$ 数组中查找1D索引元素 (x, y) 的完整公式为

$$\begin{aligned} &x \quad y \quad x \\ &= n^2((N_x \div n)(y \div n) + x \div n) + (y \bmod n)n + (x \bmod n). \end{aligned}$$

该表达式包含许多整数乘法, 除法和模运算, 这些运算在某些处理器上成本很高。当 n 为 2 的幂时, 这些运算可以转换为位移和按位逻辑运算。然而, 如上所述, 理想尺寸并不总是 2 的幂。一些 multiplications 可以转换为 shift/add 操作, 但 divide 和 modulus 操作更成问题。索引可以增量计算, 但这需要跟踪计数器, 有大量的比较和较差的分支预测性能。

但是, 有一个简单的解决方案; 请注意, 索引表达式可以写成

$$\text{index} = F_x(x) + F_y(y),$$

where

$$\begin{aligned} F_x(x) &= n^2(x \div n) + (x \bmod n), \\ F_y(y) &= n^2(N_x \div n)(y \div n) + (y \bmod n)n. \end{aligned}$$

我们将 F_x 和 F_y 制表, 并使用 x 和 y 查找数据数组中的索引。这些表将分别由 N_x 和 N_y 元素组成。表的总大小将适合处理器的主数据缓存, 即使对于非常大的数据集大小也是如此。

12.5.2 Example: Two-Level Tiling for 3D Arrays

Effective TLB utilization is also becoming a crucial factor in algorithm performance. The same technique can be used to improve TLB hit rates in a 3D array by creating $m \times m \times m$ bricks of $n \times n \times n$ cells. For example, a $40 \times 20 \times 19$ volume could be decomposed into $4 \times 2 \times 2$ macrobricks of $2 \times 2 \times 2$ bricks of $5 \times 5 \times 5$ cells. This corresponds to $m = 2$ and $n = 5$. Because 19 cannot be factored by $mn = 10$, one level of padding is needed. Empirically useful sizes are $m = 5$ for 16-bit datasets and $m = 6$ for float datasets.

The resulting index into the data array can be computed for any (x, y, z) triple with the expression

$$\begin{aligned} \text{index} = & ((x \div n) \div m)n^3m^3((N_z \div n) \div m)((N_y \div n) \div m) \\ & +((y \div n) \div m)n^3m^3((N_z \div n) \div m) \\ & +((z \div n) \div m)n^3m^3 \\ & +((x \div n) \bmod m)n^3m^2 \\ & +((y \div n) \bmod m)n^3m \\ & +((z \div n) \bmod m)n^3 \\ & +(x \bmod (n^2))n^2 \\ & +(y \bmod n)n \\ & +(z \bmod n), \end{aligned}$$

where N_x , N_y and N_z are the respective sizes of the dataset.

Note that, as in the simpler 2D one-level case, this expression can be written as

$$\text{index} = F_x(x) + F_y(y) + F_z(z),$$

where

$$\begin{aligned} F_x(x) = & ((x \div n) \div m)n^3m^3((N_z \div n) \div m)((N_y \div n) \div m) \\ & +((x \div n) \bmod m)n^3m^2 \\ & +(x \bmod n)n^2, \\ F_y(y) = & ((y \div n) \div m)n^3m^3((N_z \div n) \div m) \\ & +((y \div n) \bmod m)n^3m \\ & +(y \bmod n)n, \\ F_z(z) = & ((z \div n) \div m)n^3m^3 \\ & +((z \div n) \bmod m)n^3 \\ & +(z \bmod n). \end{aligned}$$

12.5.2 示例：3d数组的两级平铺

有效的TLB利用率也正成为算法性能的关键因素。相同的技术可用于提高3D阵列中的TLB命中率

通过创建 $n \times n \times n$ 细胞的 $m \times m \times m$ 砖。例如，一个 $40 \times 20 \times 19$ 的体积可以分解成 $4 \times 2 \times 2$ 的大砖 $5 \times 5 \times 5$ 的细胞。这对应于 $m=2$ 和 $n=5$ 。因为19不能被 $mn=10$ 分解，所以需要一层填充。根据经验，16位数据集的大小为 $m=5$ ，浮点数据集的大小为 $m=6$ 。

可以使用表达式为任何 $(x y z)$ 三元组计算数据数组的结果索引

$$\begin{aligned} \text{index} = & ((x \div n) \div m)n^3m^3((N_z \div n) \div m)((N_y \div n) \div m) \\ & +((y \div n) \div m)n^3m^3((N_z \div n) \div m) \\ & +((z \div n) \div m)n^3m^3 \\ & +((x \div n) \bmod m)n^3m^2 \\ & +((y \div n) \bmod m)n^3m \\ & +((z \div n) \bmod m)n^3 \\ & +(x \bmod (n^2))n^2 \\ & +(y \bmod n)n \\ & +(z \bmod n), \end{aligned}$$

其中 N_x , N_y 和 N_z 是数据集的各自大小。

请注意，与更简单的2D一级情况一样，此表达式可以写为

$$\text{index} = F_x(x) + F_y(y) + F_z(z),$$

where

$$\begin{aligned} F_x(x) = & ((x \div n) \div m)n^3m^3((N_z \div n) \div m)((N_y \div n) \div m) \\ & +((x \div n) \bmod m)n^3m^2 \\ & +(x \bmod n)n^2, \\ F_y(y) = & ((y \div n) \div m)n^3m^3((N_z \div n) \div m) \\ & +((y \div n) \bmod m)n^3m \\ & +(y \bmod n)n, \\ F_z(z) = & ((z \div n) \div m)n^3m^3 \\ & +((z \div n) \bmod m)n^3 \\ & +(z \bmod n). \end{aligned}$$



Frequently Asked Questions

- Does tiling really make that much difference in performance?

On some volume rendering applications, a two-level tiling strategy made as much as a factor-of-ten performance difference. When the array does not fit in main memory, it can effectively prevent thrashing in some applications such as image editing.

- How do I store the lists in a winged-edge structure?

For most applications, it is feasible to use arrays and indices for the references. However, if many delete operations are to be performed, then it is wise to use linked lists and pointers.

Notes

The discussion of the winged-edge data structure is based on the course notes of *Ching-Kuang Shene* (Shene, 2003). There are smaller mesh data structures than winged-edge. The tradeoffs in using such structures is discussed in *Directed Edges—A Scalable Representation for Triangle Meshes* (Campagna, Kobbelt, & Seidel, 1998). The tiled-array discussion is based on *Interactive Ray Tracing for Volume Visualization* (S. Parker et al., 1999). A structure similar to the triangle neighbor structure is discussed in a technical report by Charles Loop (Loop, 2000). A discussion of manifolds can be found in an introductory topology text (Munkres, 2000).

Exercises

1. What is the memory difference for a simple tetrahedron stored as four independent triangles and one stored in a winged-edge data structure?
2. Diagram a scene graph for a bicycle.
3. How many look-up tables are needed for a single-level tiling of an n -dimensional array?
4. Given N triangles, what is the minimum number of triangles that could be added to a resulting BSP tree? What is the maximum number?



常见问题

- *平铺真的在性能上有那么大的不同吗？

在某些体积渲染应用程序上，两级平铺策略的性能差异高达10倍。当数组不适合主内存时，它可以有效地防止图像编辑等应用程序的抖动。

- *如何将列表存储在翼边结构中？

对于大多数应用程序，使用数组和索引作为引用是可行的。但是，如果要执行许多删除操作，那么使用链表和指针是明智的。

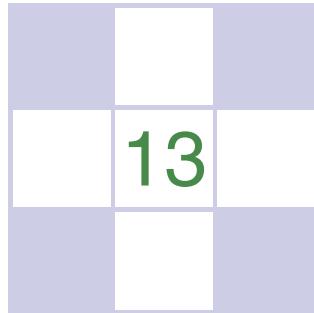
Notes

对翼缘数据结构的讨论是基于清光Shene的课程笔记（Shene, 2003）。有比winged-edge更小的网格数据结构。在有向边-三角形网格的可缩放表示中讨论了使用这种结构的权衡（Campagna, Kobbelt, & Seidel, 1998）。平铺阵列讨论基于用于体积可视化的交互式光线追踪(S.Parkeretal. 1999)。在CharlesLoop (Loop, 2000) 的技术报告中讨论了类似于triangle邻居结构的结构。

关于流形的讨论可以在介绍性拓扑文本中找到（Munkres, 2000）。

Exercises

1. 存储为四个独立三角形的简单四面体和存储在翼边数据结构中的一个的内存差异是什么？
2. 图示自行车的场景图。
3. N维数组的单级平铺需要多少个查找表？
4. 给定N个三角形，可以添加到生成的BSP树的三角形的最小数量是多少？最大数量是多少？



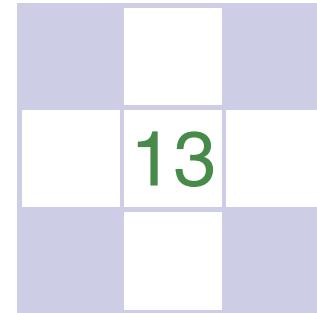
More Ray Tracing

A ray tracer is a great substrate on which to build all kinds of advanced rendering effects. Many effects that take significant work to fit into the object-order rasterization framework, including basics like the shadows and reflections already presented in Chapter 4, are simple and elegant in a ray tracer. In this chapter, we discuss some fancier techniques that can be used to ray-trace a wider variety of scenes and to include a wider variety of effects. Some extensions allow more general geometry: instancing and constructive solid geometry (CSG) are two ways to make models more complex with minimal complexity added to the program. Other extensions add to the range of materials we can handle: refraction through transparent materials, like glass and water, and glossy reflections on a variety of surfaces are essential for realism in many scenes.

This chapter also discusses the general framework of *distribution ray tracing* (Cook, Porter, & Carpenter, 1984), a powerful extension to the basic ray-tracing idea in which multiple random rays are sent through each pixel in an image to produce images with smooth edges and to simply and elegantly (if slowly) produce a wide range of effects from soft shadows to camera depth-of-field.

The price of the elegance of ray tracing is exacted in terms of computer time: most of these extensions will trace a very large number of rays for any nontrivial scene. Because of this, it's crucial to use the methods described in Chapter 12 to accelerate the tracing of rays.

If you start with a brute-force ray intersection loop, you'll have ample time to implement an acceleration structure while you wait for images to render.



更多光线追踪

光线追踪器是构建各种高级渲染效果的绝佳基底。许多需要大量工作才能适应对象顺序栅格化框架的效果，包括第4章中已经介绍的阴影和反射等基础知识，在光线追踪器中是简单而优雅的。在本章中，我们将讨论一些更奇特的技术，这些技术可用于光线追踪更多种类的场景并包含更多种类的效果。一些扩展允许更通用的几何：实例化和构造实体几何（CSG）是使模型更复杂的两种方法，同时将复杂性添加到程序中。其他扩展增加了我们可以处理的材料范围：通过透明材料（如玻璃和水）进行折射，以及在各种表面上的光泽反射对于许多场景的真实感至关重要。

本章还讨论了分布射线tracing (Cook, Porter, & Carpenter, 1984) 的一般框架，这是对基本光线追踪思想的有力扩展，其中多个随机光线在im时代通过每个像素发送，以产生具有平滑边

光线追踪优雅的价格是严格的计算机时间：大多数这些扩展将跟踪大量的光线为任何不寻常的场景。因此，使用第12章中描述的方法来加速光线的追踪至关重要。

如果您从蛮力射线相交循环开始，则在等待图像渲染时，您将有足够的时间实现加速结构。

13.1 Transparency and Refraction

In Chapter 4, we discussed the use of recursive ray tracing to compute specular, or mirror, reflection from surfaces. Another type of specular object is a *dielectric*—a transparent material that refracts light. Diamonds, glass, water, and air are dielectrics. Dielectrics also filter light; some glass filters out more red and blue light than green light, so the glass takes on a green tint. When a ray travels from a medium with refractive index n into one with a refractive index n_t , some of the light is transmitted, and it bends. This is shown for $n_t > n$ in Figure 13.1. Snell's Law tells us that

$$n \sin \theta = n_t \sin \phi.$$

Computing the sine of an angle between two vectors is usually not as convenient as computing the cosine, which is a simple dot product for the unit vectors such as we have here. Using the trigonometric identity $\sin^2 \theta + \cos^2 \theta = 1$, we can derive a refraction relationship for cosines:

$$\cos^2 \phi = 1 - \frac{n^2 (1 - \cos^2 \theta)}{n_t^2}.$$

Example values of n :
air: 1.00;
water: 1.33–1.34;
window glass: 1.51;
optical glass: 1.49–1.92;
diamond: 2.42.

Note that if n and n_t are reversed, then so are θ and ϕ as shown on the right of Figure 13.1.

To convert $\sin \phi$ and $\cos \phi$ into a 3D vector, we can set up a 2D orthonormal basis in the plane of the surface normal, \mathbf{n} , and the ray direction, \mathbf{d} .

From Figure 13.2, we can see that \mathbf{n} and \mathbf{b} form an orthonormal basis for the plane of refraction. By definition, we can describe the direction of the transformed

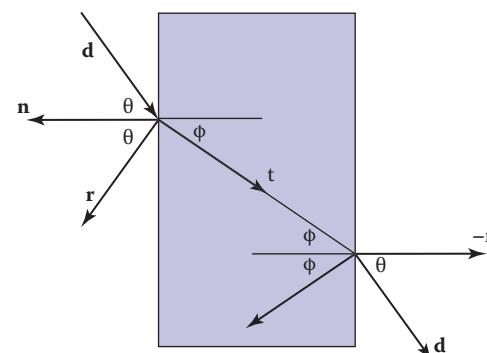


Figure 13.1. Snell's Law describes how the angle ϕ depends on the angle θ and the refractive indices of the object and the surrounding medium.

13.1 透明度和折射

在第4章中，我们讨论了使用递归光线追踪来计算表面的镜面反射或镜面反光。另一种类型的镜面物体是电介质——一种折射光线的透明材料。钻石、玻璃、水和空气都是电介质。电介质也过滤光；一些玻璃过滤掉比绿光更多的红光和蓝光，因此玻璃呈现绿色色调。当一条射线从折射率为 n 的介质进入折射率为 n_t 的介质时，一些光被透射，并且它弯曲。这在图13.1中为 $n_t > n$ 显示。斯内尔定律告诉我们

$$n \sin \theta = n_t \sin \phi.$$

N的示例值：空气: 1.00; 水: 1.33 1.34; 窗玻璃: 1.51; 光学玻璃: 1.49 1.92; 金刚石: 2.42。

计算两个矢量之间的角度的正弦通常不那么方便

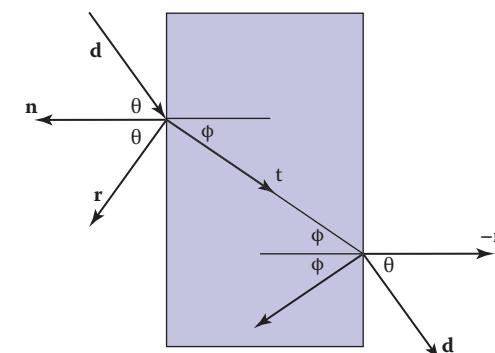
作为计算余弦，这是一个简单的点积的单位矢量，如我们在这里。使用三角恒等式 $\sin^2 \theta + \cos^2 \theta = 1$ ，我们可以推导出余弦的折射关系：

$$\cos^2 \phi = 1 - \frac{n^2 (1 - \cos^2 \theta)}{n_t^2}.$$

请注意，如果 n 和 n_t 反转，那么 θ 和 ϕ 也是如此，如Figure 13.1。

为了将 $\sin \phi$ 和 $\cos \phi$ 转换为3D矢量，我们可以在表面法线 \mathbf{n} 和射线方向 \mathbf{d} 的平面上建立2D正交基。

从图13.2中，我们可以看到 \mathbf{n} 和 \mathbf{b} 形成折射平面的正交基础。根据定义，我们可以描述变换的方向



斯内尔定律描述了角度 ϕ 如何取决于角度 θ 以及物体和周围介质的折射率。



ray, \mathbf{t} , in terms of this basis:

$$\mathbf{t} = \sin \phi \mathbf{b} - \cos \phi \mathbf{n}.$$

Since we can describe \mathbf{d} in the same basis, and \mathbf{d} is known, we can solve for \mathbf{b} :

$$\begin{aligned}\mathbf{d} &= \sin \theta \mathbf{b} - \cos \theta \mathbf{n}, \\ \mathbf{b} &= \frac{\mathbf{d} + \mathbf{n} \cos \theta}{\sin \theta}.\end{aligned}$$

This means that we can solve for \mathbf{t} with known variables:

$$\begin{aligned}\mathbf{t} &= \frac{n(\mathbf{d} + \mathbf{n} \cos \theta)}{n_t} - \mathbf{n} \cos \phi \\ &= \frac{n(\mathbf{d} - \mathbf{n}(\mathbf{d} \cdot \mathbf{n}))}{n_t} - \mathbf{n} \sqrt{1 - \frac{n^2(1 - (\mathbf{d} \cdot \mathbf{n})^2)}{n_t^2}}.\end{aligned}$$

Note that this equation works regardless of which of n and n_t is larger. An immediate question is, "What should you do if the number under the square root is negative?" In this case, there is no refracted ray and all of the energy is reflected. This is known as *total internal reflection*, and it is responsible for much of the rich appearance of glass objects.

The reflectivity of a dielectric varies with the incident angle according to the *Fresnel equations*. A nice way to implement something close to the Fresnel equations is to use the *Schlick approximation* (Schlick, 1994a),

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos \theta)^5,$$

where R_0 is the reflectance at normal incidence:

$$R_0 = \left(\frac{n_t - 1}{n_t + 1} \right)^2.$$

Note that the $\cos \theta$ terms above are always for the angle in air (the larger of the internal and external angles relative to the normal).

For homogeneous impurities, as is found in typical colored glass, a light-carrying ray's intensity will be attenuated according to *Beer's Law*. As the ray travels through the medium it loses intensity according to $dI = -CI dx$, where dx is distance. Thus, $dI/dx = -CI$. We can solve this equation and get the exponential $I = k \exp(-Cx)$. The degree of attenuation is described by the RGB attenuation constant a , which is the amount of attenuation after one unit of distance. Putting in boundary conditions, we know that $I(0) = I_0$, and $I(1) =$

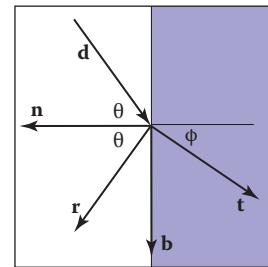


Figure 13.2. The vectors \mathbf{n} and \mathbf{b} form a 2D orthonormal basis that is parallel to the transmission vector \mathbf{t} .

雷, \mathbf{t} , 就这个基础而言:

$$\mathbf{t} = \sin \phi \mathbf{b} - \cos \phi \mathbf{n}.$$

由于我们可以在相同的基础上描述 \mathbf{d} , 并且 \mathbf{d} 是已知的, 因此我们可以求解 \mathbf{b} :

$$\mathbf{d} = \sin \theta \mathbf{b} - \cos \theta \mathbf{n}$$

$$\mathbf{b} = \frac{\mathbf{d} + \mathbf{n} \cos \theta}{\sin \theta}.$$

这意味着我们可以用已知变量求解 \mathbf{t} :

$$\begin{aligned}\mathbf{t} &= \frac{n(\mathbf{d} + \mathbf{n} \cos \theta)}{n_t} - \mathbf{n} \cos \phi \\ &= \frac{n(\mathbf{d} - \mathbf{n}(\mathbf{d} \cdot \mathbf{n}))}{n_t} - \mathbf{n} \sqrt{1 - \frac{n^2(1 - (\mathbf{d} \cdot \mathbf{n})^2)}{n_t^2}}.\end{aligned}$$

请注意, 无论 n 和 n_t 中的哪一个更大, 该方程都有效。一个简单的问题是, "如果平方根下的数字是负数, 你应该怎么做?" 在这种情况下, 没有折射射线, 所有的能量都被反射。这被称为全内反射, 它是玻璃物体丰富外观的主要原因。

电介质的反射率随入射角的变化而变化。
菲涅耳方程。一个很好的方法来实现接近菲涅耳等距的东西是使用Schlick近似 (Schlick, 1994a)

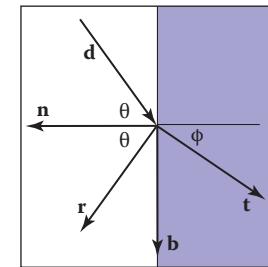
$$R(\theta) = R_0 + (1 - R_0)(1 - \cos \theta)^5,$$

其中 R_0 是法向入射时的反射率:

$$R_0 = \left(\frac{n_t - 1}{n_t + 1} \right)^2.$$

请注意, 上面的 $\cos \theta$ 项总是表示空气中的角度 (相对于法线的内部和外部角度中较大的一个)。

对于同质杂质, 如典型的有色玻璃中所发现的那样, 根据啤酒定律, 携带射线强度的光将被衰减。当射线穿过介质时, 它会根据 $dI = -CI dx$ 失去强度, 其中 dx 是距离。因此, $dI/dx = -CI$ 。我们可以求解这个方程并得到指数 $I = k \exp(-Cx)$ 。衰减程度用RGB衰减常数 a 来描述, 它是一个单位距离后的衰减量。放入边界条件, 我们知道 $I(0) = I_0$, $I(1) =$



矢量 \mathbf{n} 和 \mathbf{b} 形成平行于传输矢量 \mathbf{t} 的2D正交基。

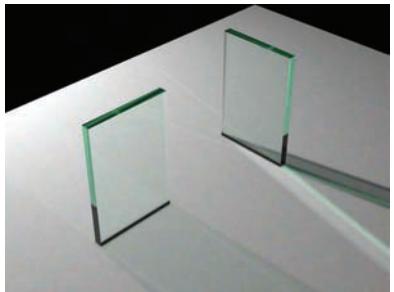


Figure 13.3. The color of the glass is affected by total internal reflection and Beer’s Law. The amount of light transmitted and reflected is determined by the Fresnel equations. The complex lighting on the ground plane was computed using particle tracing as described in Chapter 23.

$aI(0)$. The former implies $I(x) = I_0 \exp(-Cx)$. The latter implies $I_0a = I_0 \exp(-C)$, so $-C = \ln(a)$. Thus, the final formula is

$$I(s) = I(0)e^{\ln(a)s},$$

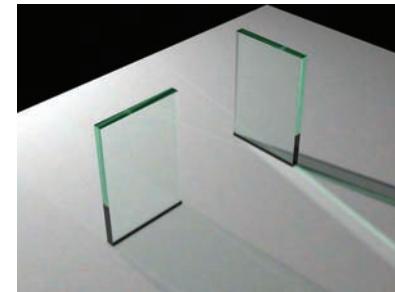
where $I(s)$ is the intensity of the beam at distance s from the interface. In practice, we reverse-engineer a by eye, because such data is rarely easy to find. The effect of Beer’s Law can be seen in Figure 13.3, where the glass takes on a green tint.

To add transparent materials to our code, we need a way to determine when a ray is going “into” an object. The simplest way to do this is to assume that all objects are embedded in air with refractive index very close to 1.0, and that surface normals point “out” (toward the air). The code segment for rays and dielectrics with these assumptions is:

```

if (p is on a dielectric) then
    r = reflect(d, n)
    if (d · n < 0) then
        refract(d, n, n, t)
        c = -d · n
        kr = kg = kb = 1
    else
        kr = exp(-art)
        kg = exp(-agt)
        kb = exp(-abt)
    if refract(d, -n, 1/n, t) then
        c = t · n
    else
        return k * color(p + tr)
    R0 = (n - 1)2/(n + 1)2

```



玻璃的颜色受全内反射和啤酒定律的影响。透射和反射的光量由菲涅耳方程确定。根据第23章所述，使用粒子跟踪计算了地平面上的复杂光照。

$aI(0)$ 。前者暗示 $I(x)=I_0\exp(-Cx)$ 。后者意味着 $I_0a=I_0\exp(-C)$ ，所以 $C=\ln(a)$ 。因此，最终的公式是

$$I(s) = I(0)e^{\ln(a)s},$$

其中 $I(s)$ 是距离界面 s 处光束的强度。在实践中，我们通过眼睛对 a 进行逆向工程，因为这样的数据很少容易找到。啤酒定律的影响可以在图 13.3 中看到，其中玻璃呈现绿色。要在代码中添加透明材质，我们需要一种方法来确定光线何时“进入”对象。最简单的方法是假设所有物体都嵌入空气中，折射率非常接近 1.0，并且表面法线指向“向外”（朝向空气）。具有这些假设的射线和电介质的代码段是：

```

if(p在电介质上)thenr=refl  

ect(d n)if(d·n<0)thenrefra  

ct(d n t)c= d·nr=kg=k  

b=1else  

kr=exp(-art)kg=exp(-agt)k  

b=exp(-abt)ifrefract(d -n  

t)thenc=t·nelse  

返回k∈color(p+tr)R0=(n  

1)2(n+1)2

```



```
R = R₀ + (1 - R₀)(1 - c)⁵
return k(R color(p + tr) + (1 - R) color(p + tt))
```

The code above assumes that the natural log has been folded into the constants (a_r, a_g, a_b) . The *refract* function returns false if there is total internal reflection, and otherwise it fills in the last argument of the argument list.

13.2 Instancing

An elegant property of ray tracing is that it allows very natural *instancing*. The basic idea of instancing is to distort all points on an object by a transformation matrix before the object is displayed. For example, if we transform the unit circle (in 2D) by a scale factor $(2, 1)$ in x and y , respectively, then rotate it by 45° , and move one unit in the x -direction, the result is an ellipse with an eccentricity of 2 and a long axis along the $(x = -y)$ -direction centered at $(0, 1)$ (Figure 13.4). The key thing that makes that entity an “instance” is that we store the circle and the composite transform matrix. Thus, the explicit construction of the ellipse is left as a future operation at render time.

The advantage of instancing in ray tracing is that we can choose the space in which to do intersection. If the base object is composed of a set of points, one of which is p , then the transformed object is composed of that set of points transformed by matrix M , where the example point is transformed to Mp . If we have a ray $a + tb$ that we want to intersect with the transformed object, we can instead intersect an *inverse-transformed ray* with the untransformed object (Figure 13.5). There are two potential advantages to computing in the untransformed space (i.e., the right-hand side of Figure 13.5):

1. The untransformed object may have a simpler intersection routine, e.g., a sphere versus an ellipsoid.
2. Many transformed objects can share the same untransformed object thus reducing storage, e.g., a traffic jam of cars, where individual cars are just transforms of a few base (untransformed) models.

As discussed in Section 6.2.2, surface normal vectors transform differently. With this in mind and using the concepts illustrated in Figure 13.5, we can determine the intersection of a ray and an object transformed by matrix M . If we create an instance class of type *surface*, we need to create a *hit* function:

```
instance::hit(ray a + tb, real t₀, real t₁, hit-record rec)
ray r' = M⁻¹a + tM⁻¹b
```



```
R = R₀ + (1 - R₀)(1 - c)⁵
返回k(R颜色(p+tr)+(1-R)颜色(p+tt))
```

上面的代码假设自然日志已折叠到常量 (ar, ag, ab) 中。如果有全内部反射，*refract*函数返回false，否则它将填充参数列表的最后一个参数。

13.2 Instancing

光线追踪的一个优雅特性是它允许非常自然的实例化。实例化的基本思想是在显示对象之前通过变换矩阵扭曲对象上的所有点。例如，如果我们将单位圆（在2D中）分别在x和y中通过比例因子 $(2, 1)$ 进行变换，然后将其旋转 45° ，并在x方向上移动一个单位，结果是一个椭圆，其偏心度为2，沿 $(x=-y)$ 方向的长轴以 $(0, 1)$ 为重心（图13.4）。使该实体成为“实例”的关键是我们存储圆和复合变换矩阵。因此，椭圆的显式构造留作渲染时的未来操作。

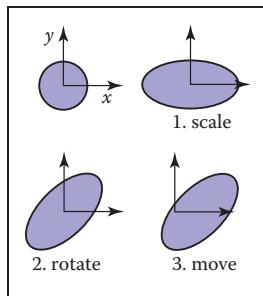


Figure 13.4. An instance of a circle with a series of three transforms is an ellipse.

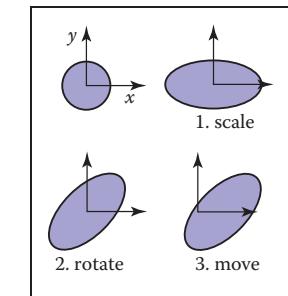


图13.4。具有一系列三个变换的圆的实例是椭圆。

在光线追踪中实例化的优点是，我们可以选择在其中进行交叉的空间。如果基础对象由一组点组成，其中一个点是 p ，那么变换的对象由矩阵 M 变换的那组点组成，其中示例点被变换为 Mp 。如果我们有一条射线 $a + tb$ ，我们希望与变换对象相交，我们可以将一条逆变换的射线与未变换的对象相交（图13.5）。在未转换空间中进行计算有两个潜在的优势（即图13.5的右侧）：

1. 未转换对象可能具有更简单的相交例程，例如，球体与椭球体。
2. 许多转换的对象可以共享相同的未转换的对象，从而减少存储，例如，汽车的交通堵塞，其中单个汽车只是几个基本（未转换）模型的转换。

如第6.2.2节中所讨论的，表面法向量的变换是不同的。

考虑到这一点，并使用图13.5中所示的概念，我们可以对由矩阵 M 变换的射线和物体的交点进行去线性化。如果我们创建一个 *surface* 类型的实例类，我们需要创建一个 *hit* 函数：

```
实例::hit(ray a + tb, real t₀, real t₁, hit-record rec)
ray r' = M⁻¹a + tM⁻¹b
```

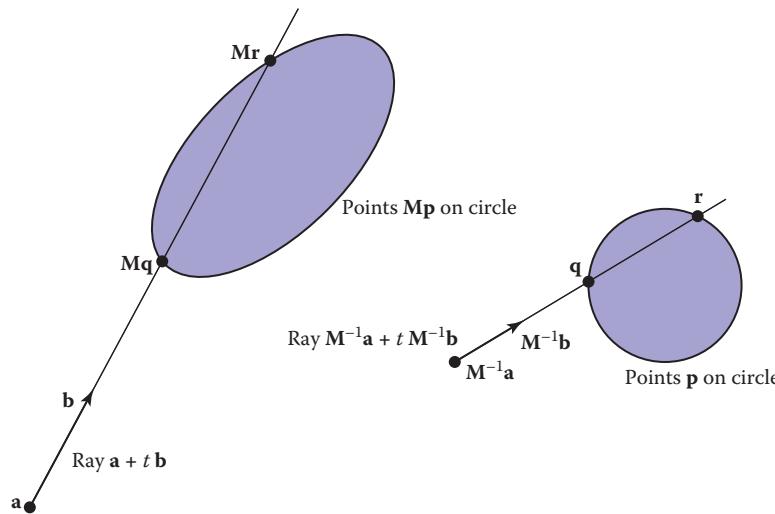


Figure 13.5. The ray intersection problem in the two spaces are just simple transforms of each other. The object is specified as a sphere plus matrix \mathbf{M} . The ray is specified in the transformed (world) space by location \mathbf{a} and direction \mathbf{b} .

```

if (base-object→hit(r', t0, t1, rec)) then
    rec.n = ( $\mathbf{M}^{-1}$ )Trec.n
    return true
else
    return false
  
```

An elegant thing about this function is that the parameter $\text{rec}.t$ does not need to be changed, because it is the same in either space. Also note that we need not compute or store the matrix \mathbf{M} .

This brings up a very important point: the ray direction \mathbf{b} must *not* be restricted to a unit-length vector, or none of the infrastructure above works. For this reason, it is useful not to restrict ray directions to unit vectors.

13.3 Constructive Solid Geometry

One nice thing about ray tracing is that any geometric primitive whose intersection with a 3D line can be computed can be seamlessly added to a ray tracer. It turns out to also be straightforward to add *constructive solid geometry* (CSG) to a ray

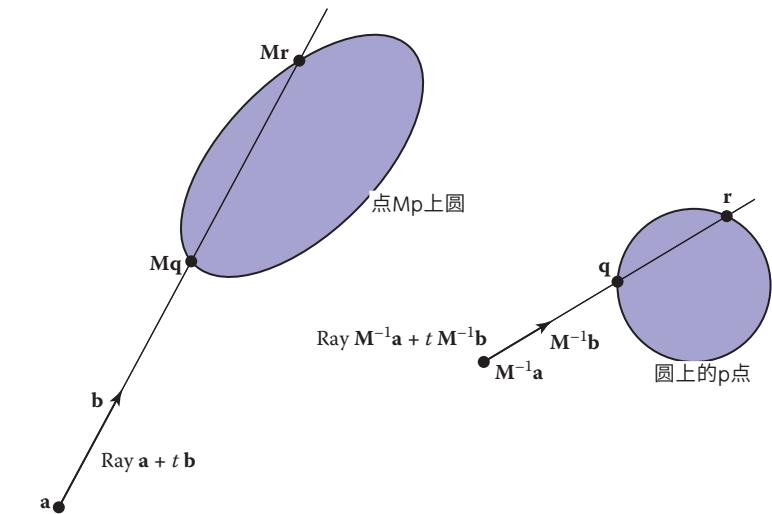


图13.5。两个空间中的射线相交问题只是彼此的简单变换。对象指定为球体加矩阵M。射线通过位置a和方向b在变换（世界）空间中指定。

```

if(base-object→hit(r' t0 t1 rec))th
enrec.n=(M-1)trec.n返回trueelse
  
```

返回错误

关于这个函数的一个优雅的事情是参数rec。t不需要改变，因为它在任一空间中都是相同的。还要注意，我们不需要计算或存储矩阵M。

这带来了一个非常重要的观点：射线方向b不能被重新限制为单位长度矢量，或者上述基础设施都不起作用。因此，不将射线方向限制为单位向量是有用的。

13.3 构造实体几何

光线追踪的一个好处是，可以计算与3D线相交的任何几何图元都可以无缝地添加到光线追踪器中。事实证明，将构造实体几何（CSG）添加到射线中也很简单



tracer (Roth, 1982). The basic idea of CSG is to use set operations to combine solid shapes. These basic operations are shown in Figure 13.6. The operations can be viewed as *set* operations. For example, we can consider C the set of all points in the circle and S the set of all points in the square. The intersection operation $C \cap S$ is the set of all points that are both members of C and S . The other operations are analogous.

Although one can do CSG directly on the model, if all that is desired is an image, we do not need to explicitly change the model. Instead, we perform the set operations directly on the rays as they interact with a model. To make this natural, we find all the intersections of a ray with a model rather than just the closest. For example, a ray $a + tb$ might hit a sphere at $t = 1$ and $t = 2$. In the context of CSG, we think of this as the ray being inside the sphere for $t \in [1, 2]$. We can compute these “inside intervals” for all of the surfaces and do set operations on those intervals (recall Section 2.1.2). This is illustrated in Figure 13.7, where the hit intervals are processed to indicate that there are two intervals inside the difference object. The first hit for $t > 0$ is what the ray actually intersects.

In practice, the CSG intersection routine must maintain a list of intervals. When the first hitpoint is determined, the material property and surface normal is that associated with the hitpoint. In addition, you must pay attention to precision issues because there is nothing to prevent the user from taking two objects that abut and taking an intersection. This can be made robust by eliminating any interval whose thickness is below a certain tolerance.

13.4 Distribution Ray Tracing

For some applications, ray-traced images are just too “clean.” This effect can be mitigated using *distribution ray tracing* (Cook et al., 1984). The conventionally ray-traced images look clean, because everything is crisp; the shadows are perfectly sharp, the reflections have no fuzziness, and everything is in perfect focus. Sometimes we would like to have the shadows be soft (as they are in real life), the reflections be fuzzy as with brushed metal, and the image have variable degrees of focus as in a photograph with a large aperture. While accomplishing these things from first principles is somewhat involved (as is developed in Chapter 23), we can get most of the visual impact with some fairly simple changes to the basic ray tracing algorithm. In addition, the framework gives us a relatively simple way to antialias (recall Section 8.3) the image.

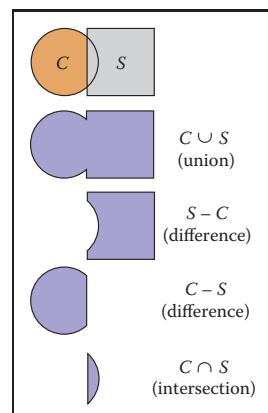


Figure 13.6. The basic CSG operations on a 2D circle and square.

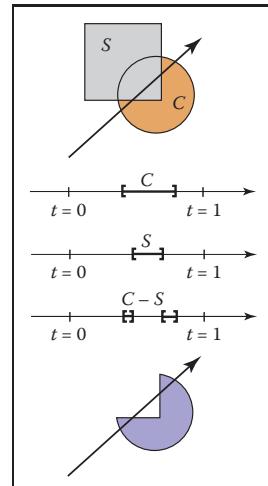


Figure 13.7. Intervals are processed to indicate how the ray hits the composite object.



示踪剂 (Roth, 1982)。CSG的基本思想是使用集合操作来组合实体形状。这些基本操作如图13.6所示。这些操作可以看作是设置操作。例如，我们可以考虑 C 是圆中所有点的集合， S 是正方形中所有点的集合。交叉路口操作 $C \in S$ 是同时是 C 和 S 成员的所有点的集合。其他操作类似。

虽然可以直接在模型上执行CSG，但如果所需的只是图像，我们不需要显示更改模型。相反，当光线与模型交互时，我们直接对光线执行set操作。为了使这变得自然，我们找到了一个模型的射线的所有交叉点，而不仅仅是最近的。例如，光线 $a+tb$ 可能会在 $t=1$ 和 $t=2$ 处击中球体。在CSG的背景下，我们认为这是光线在 $t \in [1, 2]$ 的球体内。我们可以计算所有表面的“内部间隔”，并对这些间隔进行设置操作（回顾第2.1.2节）。这在图13.7中说明，其中命中间隔被处理以指示差异对象内部有两个间隔。 $T > 0$ 的第一个命中是射线实际相交的东西。

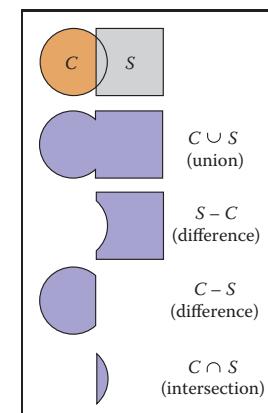
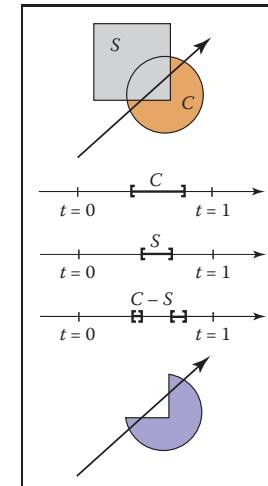


图13.6. 在二维圆和正方形上的基本CSG操作。

在实践中，CSG交集例程必须维护间隔列表。

确定第一个命中点后，材质属性和表面法线将与命中点相关联。此外，您必须注意精度问题，因为没有什么可以阻止用户拍摄两个抵接的对象并采取交叉。这可以通过消除厚度低于某一公差的任何间隔而变得坚固。



间隔被处理以指示射线如何撞击复合对象。

13.4 分布光线追踪

对于某些应用，光线追踪图像太“干净”了。“使用分布光线追踪可以减轻这种影响(Cooketal. 1984)。传统的光线追踪图像看起来很干净，因为一切都是清晰的；阴影非常清晰，反射没有模糊，一切都是完美的焦点。有时我们希望阴影是柔和的（就像在现实生活中一样），反射是模糊的，就像刷过的金属一样，图像的焦点是可变的，就像在大光圈的照片中一样。虽然从第一原则来完成这些事情是有点涉及的（正如第23章所开发的那样），但我们可以通过对基本光线跟踪算法进行一些相当简单的更改来获得大部分视觉影响。此外，该框架为我们提供了一个相对简单的方法antialias（回忆第8.3节）的图像。

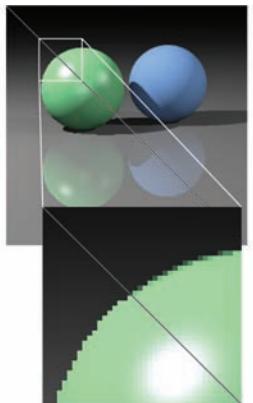


Figure 13.8. A simple scene rendered with one sample per pixel (lower left half) and nine samples per pixel (upper right half).

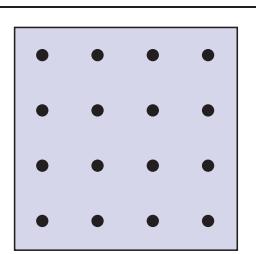


Figure 13.9. Sixteen regular samples for a single pixel.

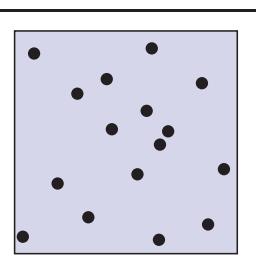


Figure 13.10. Sixteen random samples for a single pixel.

13.4.1 Antialiasing

Recall that a simple way to antialias an image is to compute the average color for the area of the pixel rather than the color at the center point. In ray tracing, our computational primitive is to compute the color at a point on the screen. If we average many of these points across the pixel, we are approximating the true average. If the screen coordinates bounding the pixel are $[i, i + 1] \times [j, j + 1]$, then we can replace the loop:

```
for each pixel  $(i, j)$  do
     $c_{ij} = \text{ray-color}(i + 0.5, j + 0.5)$ 
```

with code that samples on a regular $n \times n$ grid of samples within each pixel:

```
for each pixel  $(i, j)$  do
     $c = 0$ 
    for  $p = 0$  to  $n - 1$  do
        for  $q = 0$  to  $n - 1$  do
             $c = c + \text{ray-color}(i + (p + 0.5)/n, j + (q + 0.5)/n)$ 
     $c_{ij} = c/n^2$ 
```

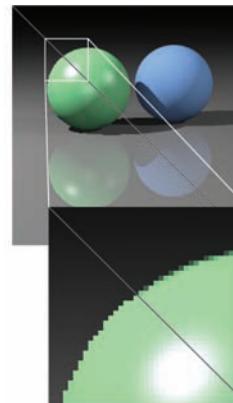
This is usually called *regular sampling*. The 16 sample locations in a pixel for $n = 4$ are shown in Figure 13.9. Note that this produces the same answer as rendering a traditional ray-traced image with one sample per pixel at $n_x n$ by $n_y n$ resolution and then averaging blocks of n by n pixels to get a n_x by n_y image.

One potential problem with taking samples in a regular pattern within a pixel is that regular artifacts such as moiré patterns can arise. These artifacts can be turned into noise by taking samples in a random pattern within each pixel as shown in Figure 13.10. This is usually called *random sampling* and involves just a small change to the code:

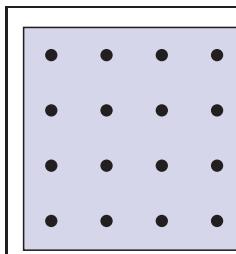
```
for each pixel  $(i, j)$  do
     $c = 0$ 
    for  $p = 1$  to  $n^2$  do
         $c = c + \text{ray-color}(i + \xi, j + \xi)$ 
     $c_{ij} = c/n^2$ 
```

Here ξ is a call that returns a uniform random number in the range $[0, 1]$. Unfortunately, the noise can be quite objectionable unless many samples are taken. A compromise is to make a hybrid strategy that randomly perturbs a regular grid:

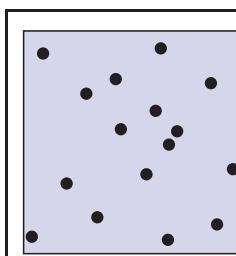
```
for each pixel  $(i, j)$  do
     $c = 0$ 
    for  $p = 0$  to  $n - 1$  do
```



以像素一个采样点（左下半部分）和每像素九个采样点（右上半部分）渲染的简单场景。



单个像素的十六个regular样本。



单个像素的十六个随机样本。

13.4.1 Antialiasing

回想一下，一个简单的反锯齿图像的方法是计算像素区域的平均颜色，而不是中心点的颜色。在光线追踪中，我们的计算原语是计算屏幕上某个点的颜色。如果我们对像素上的许多点进行平均，我们就近似于真实平均值。如果边界像素的屏幕坐标是 $[i:i+1] \times [j:j+1]$ ，那么我们可以替换循环：

对于每个像素 (i, j) 做
 $c_{ij} = \text{ray-color}(i + 0.5, j + 0.5)$

使用在每个像素内的常规 $n \times n$ 网格上采样的代码：

对于每个像素 (i, j) 做
 $c=0$ 对于 $p=0$ 到 $n-1$ 做 对
 $q=0$ 到 $n-1$ 做

$+ (p + 0.5)/n, j + (q + 0.5)/n)$
 $c_{ij} = c/n^2$

这通常称为定期采样。 $N=4$ 的像素中的16个样本位置如图13.9所示。请注意，这会产生与 $n \times n$ 乘 $n \times n$ 分辨率渲染每像素一个样本的传统光线追踪图像相同的答案，然后对 $n \times n$ 像素的块进行平均以获得 $n \times n$ 。

在像素内以规则图案采集样本的一个潜在问题是可能产生诸如莫尔图案的规则伪像。这些工件可以是

如图13.10所示，通过在每个像素内以随机模式采样而变成噪声。这通常称为随机抽样，只涉及对代码的一个小的更改：

对于每个像素 (i, j) 做
 $c=0$ 对于 $p=1$ 到 n^2 做
 $c = c + \text{ray-color}(i + \xi, j + \xi)$
 $c_{ij} = c/n^2$

这里 ξ 是一个调用，它返回范围 $[0, 1]$ 中的统一随机数。不幸的是，除非采取许多样本，否则噪音可能非常令人反感。一个折衷方案是制定一个随机扰动常规网格的混合策略：

对于每个像素 (i, j) 做
 $c=0$ 对于 $p=0$ 到 $n-1$ 做



```

for q = 0 to n - 1 do
    c = c + ray-color(i + (p + ξ)/n, j + (q + ξ)/n)
    cij = c/n2

```

That method is usually called *jittering* or *stratified sampling* (Figure 13.11).

13.4.2 Soft Shadows

The reason shadows are hard to handle in standard ray tracing is that lights are infinitesimal points or directions and are thus either visible or invisible. In real life, lights have nonzero area and can thus be partially visible. This idea is shown in 2D in Figure 13.12. The region where the light is entirely invisible is called the *umbra*. The partially visible region is called the *penumbra*. There is not a commonly used term for the region not in shadow, but it is sometimes called the *anti-umbra*.

The key to implementing soft shadows is to somehow account for the light being an area rather than a point. An easy way to do this is to approximate the light with a distributed set of N point lights each with one N th of the intensity of the base light. This concept is illustrated at the left of Figure 13.13 where nine lights are used. You can do this in a standard ray tracer, and it is a common trick to get soft shadows in an off-the-shelf renderer. There are two potential problems with this technique. First, typically dozens of point lights are needed to achieve visually smooth results, which slows down the program a great deal. The second problem is that the shadows have sharp transitions inside the penumbra.

Distribution ray tracing introduces a small change in the shadowing code. Instead of representing the area light at a discrete number of point sources, we represent it as an infinite number and choose one at random for each viewing ray. This amounts to choosing a random point on the light for any surface point being lit as is shown at the right of Figure 13.13.

If the light is a parallelogram specified by a corner point c and two edge vectors a and b (Figure 13.14), then choosing a random point r is straightforward:

$$\mathbf{r} = \mathbf{c} + \xi_1 \mathbf{a} + \xi_2 \mathbf{b},$$

where ξ_1 and ξ_2 are uniform random numbers in the range $[0, 1]$.

We then send a shadow ray to this point as shown at the right in Figure 13.13. Note that the direction of this ray is not unit length, which may require some modification to your basic ray tracer depending upon its assumptions.

We would really like to jitter points on the light. However, it can be dangerous to implement this without some thought. We would not want to always have the

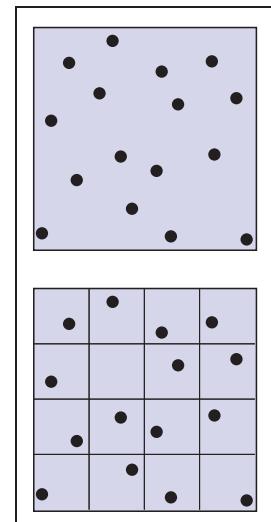


Figure 13.11. Sixteen stratified (jittered) samples for a single pixel shown with and without the bins highlighted. There is exactly one random sample taken within each bin.

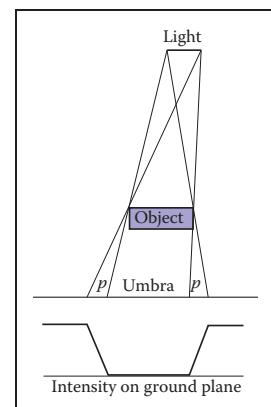


Figure 13.12. A soft shadow has a gradual transition from the unshadowed to shadowed region. The transition zone is the “penumbra” denoted by p in the figure.



```

对于q=0到n-1do
    + (p + ξ)/n, j + (q + ξ)/n)
    cij = c/n2

```

这种方法通常被称为抖动或分层采样（图13.11）。

13.4.2 柔和的阴影

在标准光线追踪中，阴影难以处理的原因是光线是无穷小的点或方向，因此是可见的或不可见的。在现实生活中，灯光具有非零区域，因此可以部分可见。这个想法在图13.12的2D中显示。光线完全不可见的区域称为本影。部分可见的区域称为半影。没有一个常用的术语来表示不在阴影中的区域，但它有时被称为反本影。

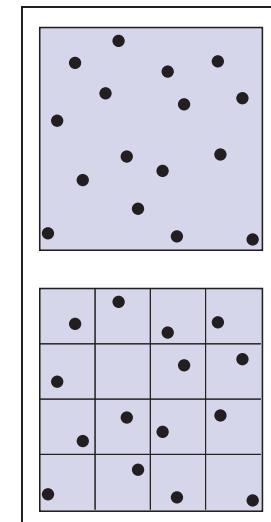
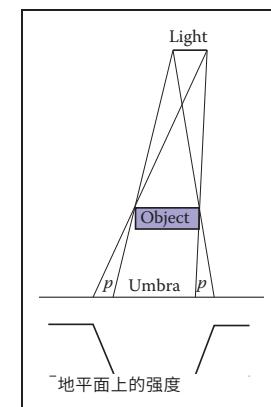


图13.11。16个分层（抖动）样本为一个像素显示有和没有箱突出显示。每个垃圾箱内正好有一个随机样本。



柔和的阴影有一个从未阴影区域到阴影区域的逐渐过渡。

Transition区是图中用p表示的“半影”。

实现软阴影的关键是以某种方式考虑光线是一个区域而不是一个点。一个简单的方法是用一组分布的N个点灯来近似光，每个点灯的强度为基础光的强度的N。这个概念在图13.13的左边说明，其中9

灯使用。您可以在标准光线追踪器中执行此操作，并且在现成的渲染器中获得柔和阴影是一种常见的技巧。这种技术有两个潜在的问题。首先，通常需要数十个点光源来实现视觉平滑的结果，这大大减慢了程序的速度。第二个问题是阴影在半影内部有尖锐的过渡。

分布光线追踪在阴影代码中引入了一个小的变化。

我们不是在离散数量的点光源处表示区域光，而是将其表示为无限数量，并为每个观察光线随机选择一个。这相当于在光线上为任何被点亮的表面点选择一个随机点，如图13.13右侧所示。

如果光线是由一个角点 c 和两个边缘向量 a 和 b 指定的平行四边形（图13.14），那么选择一个随机点 r 很简单：

$$\mathbf{r} = \mathbf{c} + \xi_1 \mathbf{a} + \xi_2 \mathbf{b},$$

其中 ξ_1 和 ξ_2 是范围 $[0, 1]$ 内的均匀随机数。

然后我们发送一个阴影射线到这一点 如图13.13的右边所示。

请注意，此射线的方向不是单位长度，这可能需要对您的基本射线追踪器进行一些修改，具体取决于其假设。

我们真的很想在灯光上抖动点。但是，不经过一些思考就实现这一点可能是危险的。我们不想永远拥有

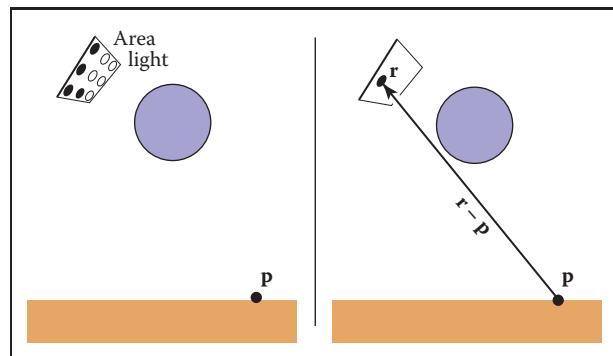


Figure 13.13. Left: an area light can be approximated by some number of point lights; four of the nine points are visible to p so it is in the penumbra. Right: a random point on the light is chosen for the shadow ray, and it has some chance of hitting the light or not.

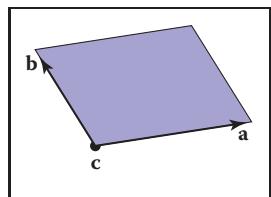


Figure 13.14. The geometry of a parallelogram light specified by a corner point and two edge vectors.

ray in the upper left-hand corner of the pixel generate a shadow ray to the upper left-hand corner of the light. Instead we would like to scramble the samples, such that the pixel samples and the light samples are each themselves jittered, but so that there is no correlation between pixel samples and light samples. A good way to accomplish this is to generate two distinct sets of n^2 jittered samples and pass samples into the light source routine:

```

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
         $c = c + \text{ray-color}(i + r[p].x(), j + r[p].y(), s[p])$ 
         $c_{ij} = c/N$ 
    
```

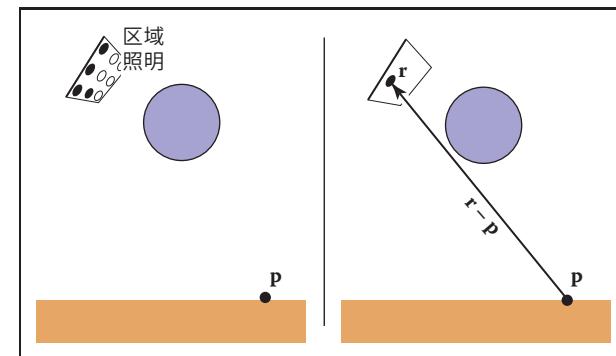
This shuffle routine eliminates any coherence between arrays r and s . The shadow routine will just use the 2D random point stored in $s[p]$ rather than calling the random number generator. A shuffle routine for an array indexed from 0 to $N - 1$ is:

```

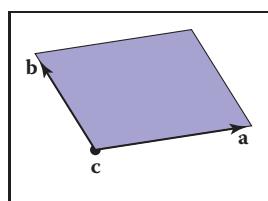
for  $i = N - 1$  downto 1 do
    choose random integer  $j$  between 0 and  $i$  inclusive
    swap array elements  $i$  and  $j$ 
    
```

13.4.3 Depth of Field

The soft focus effects seen in most photos can be simulated by collecting light at a nonzero size “lens” rather than at a point. This is called *depth of field*. The



左图：一个区域光可以用一定数量的点光来近似；九个点中的四个对p可见，所以它在半影中。右图：光源上的一个随机点被选择为阴影光线，它有一定的机会击中光线。



一个角点和两个边缘向量指定的平行四边形光的geometry。

在像素的左上角的射线产生阴影射线到光的左上角。相反，我们希望混淆样本，使得像素样本和光样本各自抖动，但使得像素样本和光样本之间没有相关性。实现此目的的一个好方法是生成两组不同的 n^2 抖动样本，并将样本传递到光源例程中：

对于每个像素 (i, j) $\text{doc}=0$ 生成 $N=n^2$ 个抖动的2D点
并存储在数组 $r[]$ 生成 $N=n^2$ 个抖动的2D点并存储在数组
 $s[]$ 洗牌数组 $s[]$ 中的点对于 $p=0$ 到 $N-1$ do

$$c_{ij} = c/N$$

这个shuffle例程消除了数组 r 和 s 之间的任何相干性。阴影例程将只使用存储在 $s[p]$ 中的2D随机点，而不是调用随机数生成器。从0到 $N-1$ 索引的数组的shuffle例程是：

对于 $i=N-1$ 下降到1做
选择0和i之间的随机整数j，包括交换数组元素
 i 和 j

13.4.3 景深

在大多数照片中看到的柔焦效果可以通过在非零尺寸的“镜头”而不是在一点上收集光线来模拟。这称为景深。该



lens collects light from a cone of directions that has its apex at a distance where everything is in focus (Figure 13.15). We can place the “window” we are sampling on the plane where everything is in focus (rather than at the $z = n$ plane as we did previously) and the lens at the eye. The distance to the plane where everything is in focus we call the *focus plane*, and the distance to it is set by the user, just as the distance to the focus plane in a real camera is set by the user or range finder.

To be most faithful to a real camera, we should make the lens a disk. However, we will get very similar effects with a square lens (Figure 13.16). So we choose the side-length of the lens and take random samples on it. The origin of the view rays will be these perturbed positions rather than the eye position. Again, a shuffling routine is used to prevent correlation with the pixel sample positions. An example using 25 samples per pixel and a large disk lens is shown in Figure 13.17.

13.4.4 Glossy Reflection

Some surfaces, such as brushed metal, are somewhere between an ideal mirror and a diffuse surface. Some discernible image is visible in the reflection, but it is blurred. We can simulate this by randomly perturbing ideal specular reflection rays as shown in Figure 13.18.

Only two details need to be worked out: how to choose the vector r' and what to do when the resulting perturbed ray is below the surface from which the ray is



Figure 13.17. An example of depth of field. The caustic in the shadow of the wine glass is computed using particle tracing as described in Chapter 23.

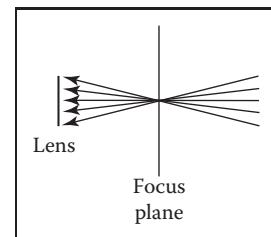


Figure 13.15. The lens averages over a cone of directions that hit the pixel location being sampled.

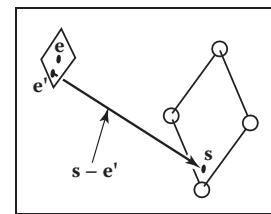


Figure 13.16. To create depth-of-field effects, the eye is randomly selected from a square region.



透镜从一个方向锥体收集光线，该方向锥体的顶点位于所有物体都聚焦的距离（图13.15）。我们可以将我们正在采样的“窗口”放在所有东西都在焦点的平面上（而不是像以前那样在 $z=n$ 平面上）和眼睛的镜头上。到所有东西都在聚焦的平面的距离我们称之为聚焦平面，到它的距离是由用户设置的，就像在真实相机中到聚焦平面的距离是由用户或测距仪设置的。

为了最忠实于真正的相机，我们应该把镜头做成磁盘。但是，我们将获得与方形镜头非常相似的效果（图13.16）。因此，我们选择镜头的边长并对其进行随机采样。视图射线的原点将是这些扰动位置而不是眼睛位置。再次，混洗例程用于防止与像素样本位置相关。使用每像素25个样本和大盘透镜的示例如图13.17所示。

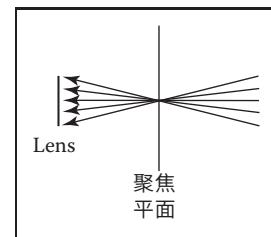
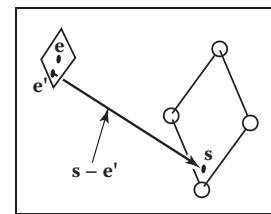


Figure 13.15. 镜头击中正在采样的像素位置的方向锥。



为了创建景深效果，眼睛是从正方形区域随机选择的。

13.4.4 光面反射

某些表面（如拉丝金属）介于理想镜面和漫反射表面之间。一些可辨别的图像在反射中是可见的，但它是模糊的。我们可以通过随机扰动理想镜面反射射线来模拟这一点，如图13.18所示。

只需要计算两个细节：如何选择矢量 r' 以及当产生的扰动射线低于射线所在的表面时该怎么办



图13.17。景深的一个例子。如第23章所述，使用粒子跟踪计算酒杯阴影中的苛性碱度。

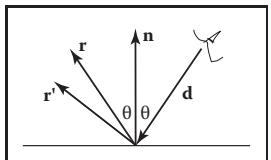


Figure 13.18. The reflection ray is perturbed to a random vector \mathbf{r}' .

reflected. The latter detail is usually settled by returning a zero color when the ray is below the surface.

To choose \mathbf{r}' , we again sample a random square. This square is perpendicular to \mathbf{r} and has width a which controls the degree of blur. We can set up the square's orientation by creating an orthonormal basis with $\mathbf{w} = \mathbf{r}$ using the techniques in Section 2.4.6. Then, we create a random point in the 2D square with side length a centered at the origin. If we have 2D sample points $(\xi, \xi') \in [0, 1]^2$, then the analogous point on the desired square is

$$u = -\frac{a}{2} + \xi a,$$

$$v = -\frac{a}{2} + \xi' a.$$

Because the square over which we will perturb is parallel to both the \mathbf{u} and \mathbf{v} vectors, the ray \mathbf{r}' is just

$$\mathbf{r}' = \mathbf{r} + u\mathbf{u} + v\mathbf{v}.$$

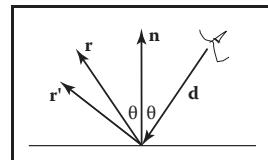
Note that \mathbf{r}' is not necessarily a unit vector and should be normalized if your code requires that for ray directions.

13.4.5 Motion Blur

We can add a blurred appearance to objects as shown in Figure 13.19. This is called *motion blur* and is the result of the image being formed over a nonzero



Figure 13.19. The bottom right sphere is in motion, and a blurred appearance results. Image courtesy Chad Barb.



反射射线被扰动到一个random向量 \mathbf{r}' .

反映出来的。后一个细节通常通过在光线低于表面时返回零颜色来确定。

要选择 \mathbf{r}' , 我们再次采样一个随机正方形。这个正方形垂直于 \mathbf{r} , 宽度 a 控制模糊程度。我们可以通过使用第2.4.6节中的技术创建一个具有 $\mathbf{w}=\mathbf{r}$ 的正交基来设置正方形的方向。然后, 我们在二维正方形中创建一个随机点, 边长 a 以原点为中心。如果我们有2D样本点 $(\xi, \xi') \in [0, 1]^2$ 然后在所需的正方形上的类似点是

$$u = -\frac{a}{2} + \xi a,$$

$$v = -\frac{a}{2} + \xi' a.$$

因为我们将扰动的平方与 \mathbf{u} 和 \mathbf{v} 向量平行, 所以射线 \mathbf{r}' 只是

$$\mathbf{r}' = \mathbf{r} + u\mathbf{u} + v\mathbf{v}.$$

请注意, \mathbf{r}' 不一定是一个单位向量, 如果你的代码要求射线方向, 应该标准化。

13.4.5 运动模糊

我们可以为对象添加模糊的外观, 如图13.19所示。这称为运动模糊, 是在非零上形成图像的结果

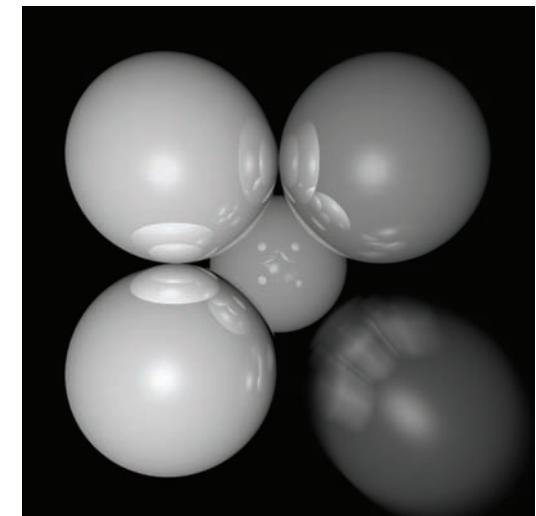


图13.19. 右下方的球体在运动中 结果是模糊的外观.图片由查德·巴布提供。



span of time. In a real camera, the aperture is open for some time interval during which objects move. We can simulate the open aperture by setting a time variable ranging from T_0 to T_1 . For each viewing ray we choose a random time,

$$T = T_0 + \xi(T_1 - T_0).$$

We may also need to create some objects to move with time. For example, we might have a moving sphere whose center travels from c_0 to c_1 during the interval. Given T , we could compute the actual center and do a ray–intersection with that sphere. Because each ray is sent at a different time, each will encounter the sphere at a different position, and the final appearance will be blurred. Note that the bounding box for the moving sphere should bound its entire path so an efficiency structure can be built for the whole time interval (Glassner, 1988).

Notes

There are many, many other advanced methods that can be implemented in the ray-tracing framework. Some resources for further information are Glassner's *An Introduction to Ray Tracing* and *Principles of Digital Image Synthesis*, Shirley's *Realistic Ray Tracing*, and Pharr and Humphreys's *Physically Based Rendering: From Theory to Implementation*.

Frequently Asked Questions

- What is the best ray-intersection efficiency structure?

The most popular structures are binary space partitioning trees (BSP trees), uniform subdivision grids, and bounding volume hierarchies. Most people who use BSP trees make the splitting planes axis-aligned, and such trees are usually called k-d trees. There is no clear-cut answer for which is best, but all are much, much better than brute-force search in practice. If I were to implement only one, it would be the bounding volume hierarchy because of its simplicity and robustness.

- Why do people use bounding boxes rather than spheres or ellipsoids?

Sometimes spheres or ellipsoids are better. However, many models have polygonal elements that are tightly bounded by boxes, but they would be difficult to tightly bind with an ellipsoid.



时间跨度。在真实相机中，光圈在物体移动的一段时间间隔内是打开的。我们可以通过设置从 T_0 到 T_1 的时间变量来模拟开放孔径。对于每个观看光线，我们选择一个随机时间

$$T = T_0 + \xi(T_1 - T_0).$$

我们可能还需要创建一些对象来随时间移动。例如，我们可能有一个移动球体，其中心在间隔期间从 c_0 移动到 c_1 。给定 T ，我们可以计算实际的中心并与该球体进行光线相交。因为每条光线在不同的时间发送，所以每条都会在不同的位置遇到球体，最终的外观会变得模糊。请注意，移动球体的边界框应该约束其整个路径，因此可以在整个时间间隔内构建效率结构 (Glassner, 1988)。

Notes

在光线追踪框架中可以实现许多其他高级方法。有关更多信息的一些资源包括 Glassner 的《光线追踪和数字图像合成原理导论》、Shirley 的《逼真光线追踪》以及 Pharr 和 Humphreys 的基于物理的渲染：

从理论到实施。

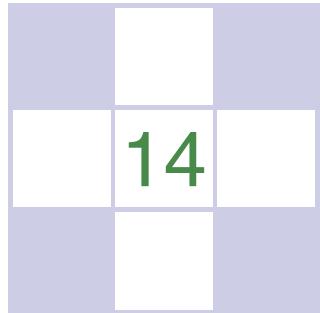
常见问题

- * 什么是最佳的射线交叉效率结构？

最流行的结构是二元空间分区树（BSP树），均匀细分网格和边界卷层次结构。大多数使用BSP树的人使分裂平面轴对齐，这种树通常被称为k-d树。没有明确的答案是最好的，但在实践中，所有的答案都比蛮力搜索好得多。如果我只实现一个，它将是边界卷层次结构，因为它的简单性和鲁棒性。

- 为什么人们使用边界框而不是球体或椭球体？

有时球体或椭球体更好。但是，许多模型具有由框紧密绑定的多边形元素，但它们很难与椭球紧密绑定。



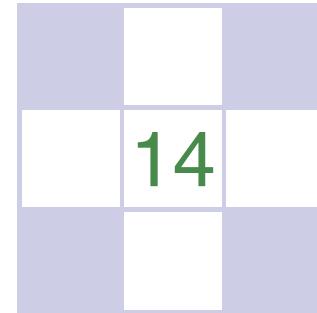
Sampling

Many applications in graphics require “fair” sampling of unusual spaces, such as the space of all possible lines. For example, we might need to generate random edges within a pixel, or random sample points on a pixel that vary in density according to some density function. This chapter provides the machinery for such probability operations. These techniques will also prove useful for numerically evaluating complicated integrals using *Monte Carlo integration*, also covered in this chapter.

14.1 Integration

Although the words “integral” and “measure” often seem intimidating, they relate to some of the most intuitive concepts found in mathematics, and they should not be feared. For our very non-rigorous purposes, a *measure* is just a function that maps subsets to \mathbb{R}^+ in a manner consistent with our intuitive notions of length, area, and volume. For example, on the 2D real plane \mathbb{R}^2 , we have the area measure A which assigns a value to a set of points in the plane. Note that A is just a function that takes pieces of the plane and returns area. This means the domain of A is all possible subsets of \mathbb{R}^2 , which we denote as the *power set* $\mathcal{P}(\mathbb{R}^2)$. Thus, we can characterize A in arrow notation:

$$A : \mathcal{P}(\mathbb{R}^2) \rightarrow \mathbb{R}^+.$$



Sampling

图形中的许多应用需要对不寻常的空间进行“公平”采样，例如所有可能行的空间。例如，我们可能需要在像素内生成随机边缘，或者在像素上生成随机样本点，这些样本点的密度根据某种密度函数而变化。本章提供了这种概率操作的机制。这些技术对于使用蒙特卡罗积分对复杂积分进行数值评估也很有用，本章也将介绍这些技术。

14.1 Integration

虽然“积分”和“度量”这个词经常看起来令人生畏，但它们与数学中发现的一些最直观的概念有关，不应该害怕。对于我们非常不严格的目的，度量只是一个函数，它以符合我们直观的长度，面积和体积概念的方式将子集映射到 \mathbb{R}^+ 。例如，在2D实平面 \mathbb{R}^2 上，我们有面积度量 A ，它为平面中的一组点赋值。请注意， A 只是一个函数，它接受平面的碎片并返回区域。这意味着 A 的域是 \mathbb{R}^2 的所有可能子集，我们将其表示为幂集 $\mathcal{P}(\mathbb{R}^2)$ 。因此，我们可以用箭头表示法来表征 A ：

$$A : \mathcal{P}(\mathbb{R}^2) \rightarrow \mathbb{R}^+.$$

An example of applying the area measure shows that the area of the square with side length one is one:

$$A([a, a+1] \times [b, b+1]) = 1,$$

where (a, b) is just the lower left-hand corner of the square. Note that a single point such as $(3, 7)$ is a valid subset of \mathbb{R}^2 and has zero area: $A((3, 7)) = 0$. The same is true of the set of points S on the x -axis, $S = (x, y)$ such that $(x, y) \in \mathbb{R}^2$ and $y = 0$, i.e., $A(S) = 0$. Such sets are called *zero measure sets*.

To be considered a measure, a function has to obey certain area-like properties. For example, we have a function $\mu : \mathcal{P}(S) \rightarrow \mathbb{R}^+$. For μ to be a measure, the following conditions must be true:

1. The measure of the empty set is zero: $\mu(\emptyset) = 0$,
2. The measure of two distinct sets together is the sum of their measure alone.
This rule with possible intersections is

$$\mu(A \cup B) = \mu(A) + \mu(B) - \mu(A \cap B),$$

where \cup is the set union operator and \cap is the set intersection operator.

When we actually compute measures, we usually use *integration*. We can think of integration as really just notation:

$$A(S) \equiv \int_{x \in S} dA(x).$$

You can informally read the right-hand side as “take all points x in the region S , and sum their associated differential areas.” The integral is often written other ways including

$$\int_S dA, \quad \int_{\mathbf{x} \in S} d\mathbf{x}, \quad \int_{\mathbf{x} \in S} dA_{\mathbf{x}}, \quad \int_{\mathbf{x}} d\mathbf{x}.$$

All of the above formulas represent “the area of region S .” We will stick with the first one we used, because it is so verbose it avoids ambiguity. To evaluate such integrals analytically, we usually need to lay down some coordinate system and use our bag of calculus tricks to solve the equations. But have no fear if those skills have faded, as we usually have to numerically approximate integrals, and that requires only a few simple techniques which are covered later in this chapter.

Given a measure on a set S , we can always create a new measure by weighting with a nonnegative function $w : S \rightarrow \mathbb{R}^+$. This is best expressed in integral

应用面积度量的示例显示边长为1的正方形的面积为1:

$$A([a, a+1] \times [b, b+1]) = 1,$$

其中 (a, b) 只是正方形的左下角。请注意，诸如 $(3, 7)$ 的单个点是 \mathbb{R}^2 的有效子集，并且具有零区域： $A((3, 7))=0$ 。 x 轴上的点集 S 也是如此， $S=(x, y)$ 使得 $(x, y) \in \mathbb{R}^2$ 且 $y=0$ ，即 $A(S)=0$ 。此类集称为零度量集。

要被视为度量，函数必须服从某些类似区域的属性。例如，我们有一个函数 $\mu : \mathcal{P}(S) \rightarrow \mathbb{R}^+$ 。要使 μ 成为度量值，必须满足以下条件：

1. 空集的度量为零： $\mu(\emptyset)=0$

2. 两个不同集合的度量是它们单独度量的总和。这条规则与可能的交叉点是

$$\mu(A \cup B) = \mu(A) + \mu(B) - \mu(A \cap B),$$

其中 \cup 是集并运算符， \cap 是集交集运算符。

当我们实际计算度量时，我们通常使用积分。我们可以把积分看作是真正的符号：

$$A(S) \equiv \int_{x \in S} dA(x).$$

您可以非正式地将右侧读取为“取区域 S 中的所有点 x ，并对其相关联的差分区域求和。”积分通常以其他方式编写，包括

$$\int_S dA, \quad \int_{\mathbf{x} \in S} d\mathbf{x}, \quad \int_{\mathbf{x} \in S} dA_{\mathbf{x}}, \quad \int_{\mathbf{x}} d\mathbf{x}.$$

所有上述式表示“区域 S 的面积。”我们将坚持使用我们使用的第一，因为它非常冗长，避免了歧义。为了分析地评估这些积分，我们通常需要放下一些坐标系并使用我们的微积分技巧来求解方程。但是，如果这些技能已经消失，不要担心，因为我们通常必须在数值上近似积分，这只需要一些简单的技术，这将在本章后面介绍。给定一个集合 S 上的度量，我们总是可以通过用非负函数 $w : S \rightarrow \mathbb{R}^+$ 加权来创建一个新的度量。这最好用积分表示



notation. For example, we can start with the example of the simple area measure on $[0, 1]^2$:

$$\int_{\mathbf{x} \in [0, 1]^2} dA(\mathbf{x}),$$

and we can use a “radially weighted” measure by inserting a weighting function of radius squared:

$$\int_{\mathbf{x} \in [0, 1]^2} \|\mathbf{x}\|^2 dA(\mathbf{x}).$$

To evaluate this analytically, we can expand using a Cartesian coordinate system with $dA \equiv dx dy$:

$$\int_{\mathbf{x} \in [0, 1]^2} \|\mathbf{x}\|^2 dA(\mathbf{x}) = \int_{x=0}^1 \int_{y=0}^1 (x^2 + y^2) \, dx \, dy.$$

The key thing here is that if you think of the $\|\mathbf{x}\|^2$ term as married to the dA term, and that these together form a new measure, we can call that measure ν . This would allow us to write $\nu(S)$ instead of the whole integral. If this strikes you as just a bunch of notation and bookkeeping, you are right. But it does allow us to write down equations that are either compact or expanded depending on our preference.

14.1.1 Measures and Averages

Measures really start paying off when taking averages of a function. You can only take an average with respect to a particular measure, and you would like to select a measure that is “natural” for the application or domain. Once a measure is chosen, the average of a function f over a region S with respect to measure μ is

$$\text{average}(f) \equiv \frac{\int_{x \in S} f(\mathbf{x}) \, d\mu(\mathbf{x})}{\int_{x \in S} d\mu(\mathbf{x})}.$$

For example, the average of the function $f(x, y) = x^2$ over $[0, 2]^2$ with respect to the area measure is

$$\text{average}(f) \equiv \frac{\int_{x=0}^2 \int_{y=0}^2 x^2 \, dx \, dy}{\int_{x=0}^2 \int_{y=0}^2 \, dx \, dy} = \frac{4}{3}.$$

This machinery helps solve seemingly hard problems where choosing the measure is the tricky part. Such problems often arise in *integral geometry*, a field that studies measures on geometric entities, such as lines and planes. For example,



符号。例如，我们可以从 $[0, 1]^2$ 上的简单面积度量的示例开始：

$$\int_{\mathbf{x} \in [0, 1]^2} dA(\mathbf{x}),$$

我们可以通过插入半径平方的加权函数来使用“径向加权”度量：

$$\int_{\mathbf{x} \in [0, 1]^2} \|\mathbf{x}\|^2 dA(\mathbf{x}).$$

为了分析地评估这一点，我们可以使用具有 $dA \equiv dx dy$ 的笛卡尔坐标系进行扩展：

$$\int_{\mathbf{x} \in [0, 1]^2} \|\mathbf{x}\|^2 dA(\mathbf{x}) = \int_{x=0}^1 \int_{y=0}^1 (x^2 + y^2) \, dx \, dy.$$

这里的关键是，如果你认为 $\|\mathbf{x}\|^2$ 术语与 dA 术语结婚，并且这些共同构成了一个新的度量，我们可以称之为度量 ν 。这将允许我们写 $\nu(S)$ 而不是整个积分。如果这是一堆符号和笔记，你是对的。但它确实允许我们根据我们的偏好写下紧凑或扩展的方程。

14.1.1 量度及平均值

当对一个函数取平均值时 度量才真正开始得到回报。您只能对特定度量值取平均值，并且您希望为应用程序或域选择“自然”的度量值。一旦选择了度量，函数 f 在区域 S 上相对于度量 μ 的平均值为

$$\text{average}(f) \equiv \frac{\int_{x \in S} f(\mathbf{x}) \, d\mu(\mathbf{x})}{\int_{x \in S} d\mu(\mathbf{x})}.$$

例如，函数 $f(x, y) = x^2$ over $[0, 2]^2$ 相对于面积度量的平均值为

$$\text{average}(f) \equiv \frac{\int_{x=0}^2 \int_{y=0}^2 x^2 \, dx \, dy}{\int_{x=0}^2 \int_{y=0}^2 \, dx \, dy} = \frac{4}{3}.$$

这种机器有助于解决看似困难的问题，其中选择措施是棘手的部分。这样的问题经常出现在积分几何中，积分几何是一个研究几何实体（如线和平面）度量的领域。例如

one might want to know the average length of a line through $[0, 1]^2$. That is, by definition,

$$\text{average}(\text{length}) = \frac{\int_{\text{lines } L \text{ through } [0, 1]^2} \text{length}(L) d\mu(L)}{\int_{\text{lines } L \text{ through } [0, 1]^2} d\mu(L)}.$$

All that is left, once we know that, is choosing the appropriate μ for the application. This is dealt with for lines in the next section.

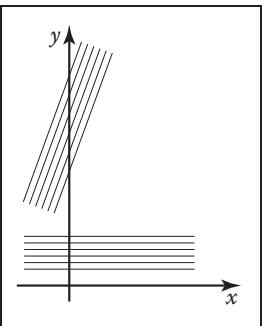


Figure 14.1. These two bundles of lines should have the same measure. They have different intersection lengths with the y-axis so using db would be a poor choice for a differential measure.

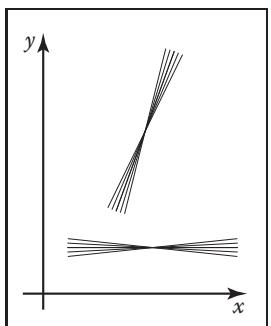


Figure 14.2. These two bundles of lines should have the same measure. Since they have different values for change in slope, using dm would be a poor choice for a differential measure.

$$d\mu = \cos \phi \, d\phi \, db.$$

It can be shown that this measure, up to a constant, is the only one that is invariant with respect to rotation and translation.

This measure can be converted into an appropriate measure for other parameterizations of the line. For example, the appropriate measure for (m, b) space is

$$d\mu = \frac{dm \, db}{(1 + m^2)^{\frac{3}{2}}}.$$

人们可能想知道通过 $[0, 1]^2$ 的线的平均长度。也就是说，根据定义

$$\text{average}(\text{length}) = \frac{\int_{\text{线 } L \text{ 至 } [0, 1]^2} \text{长度}(L) d\mu(L)}{\int_{\text{线 } L \text{ 至 } [0, 1]^2} d\mu(L)}.$$

一旦我们知道这一点，剩下的就是为应用程序选择合适的 μ 。这将在下一节中处理。

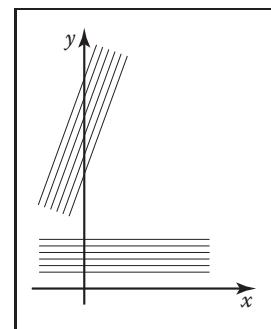
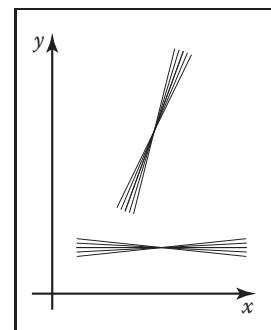


Figure 14.1. 这两个 bundles of lines should have the same measure. They have different intersection lengths with the y-axis so using db 对于y轴来说，使用 db 对于差分测量来说是一个糟糕的选择。



这两束线应该具有相同的度量。由于它们具有不同的斜率变化值，因此使用 dm 对于差分度量来说是一个糟糕的选择。

14.1.2 Example: Measures on the Lines in the 2D Plane

What measure μ is “natural”?

If you parameterize the lines as $y = mx + b$, you might think of a given line as a point (m, b) in “slope-intercept” space. An easy measure to use would be $dm \, db$, but this would not be a “good” measure in that not all equal size “bundles” of lines would have the same measure. More precisely, the measure would not be invariant with respect to change of coordinate system. For example, if you took all lines through the square $[0, 1]^2$, the measure of lines through it would not be the same as the measure through a unit square rotated 45 degrees. What we would really like is a “fair” measure that does not change with rotation or translation of a set of lines. This idea is illustrated in Figures 14.1 and 14.2.

To develop a natural measure on the lines, we should first start thinking of them as points in a dual space. This is a simple concept: the line $y = mx + b$ can be specified as the point (m, b) in a slope-intercept space. This concept is illustrated in Figure 14.3. It is more straightforward to develop a measure in (ϕ, b) space. In that space b is the y -intercept, while ϕ is the angle the line makes with the x -axis, as shown in Figure 14.4. Here, the differential measure $d\phi \, db$ almost works, but it would not be fair due to the effect shown in Figure 14.1. To account for the larger span b that a constant width bundle of lines makes, we must add a cosine factor:

$$d\mu = \cos \phi \, d\phi \, db.$$

14.1.2 示例：2D平面中的线上的度量

什么措施 μ 是“自然”？

如果将线参数化为 $y=mx+b$ ，则可能会将给定线视为“斜率-截距”空间中的点 (m, b) 。一个易于使用的措施是 $dmdb$ ，但这不是一个“好”的措施，因为并非所有大小相等的“束”线都具有相同的措施。更确切地说，这项措施不会是

坐标系变化的不变性。例如，如果您对穿过正方形 $[0, 1]^2$ 的所有线进行了取值，则穿过它的线的度量值将与穿过旋转45度的单位正方形的度量值不同。我们真正想要的是一个“公平”的度量，它不会随着一组线的旋转或平移而改变。这个想法在图14.1和14.2中说明。

为了在线条上发展一个自然的度量，我们应该首先把它们看作是双重空间中的点。这是一个简单的概念：线 $y=mx+b$ 可以指定为斜率截距空间中的点 (m, b) 。这个概念如图14.3所示。在 (ϕ, b) 空间中开发度量更为直接。在该空间中， b 是 y 截距，而 ϕ 是线与 x 轴的角度，如图14.4所示。在这里，差分度量 $d\phi db$ 几乎有效，但由于图14.1所示的效果，它不会公平。到

考虑到一个恒定宽度的线束所产生的较大的跨度 b ，我们必须添加一个余弦因子：

$$d\mu = \cos \phi \, d\phi \, db.$$

可以表明，这个度量，直到一个常数，是唯一一个在旋转和平移方面不变的度量。

此度量可以转换成用于行的其他参数化的适当度量。例如， (m, b) 空间的适当度量为

$$d\mu = \frac{dm \, db}{(1 + m^2)^{\frac{3}{2}}}.$$

For the space of lines parameterized in (u, v) space,

$$ux + vy + 1 = 0,$$

the appropriate measure is

$$d\mu = \frac{du \ dv}{(u^2 + v^2)^{\frac{3}{2}}}.$$

For lines parameterized in terms of (a, b) , the x -intercept and y -intercept, the measure is

$$d\mu = \frac{ab \ da \ db}{(a^2 + b^2)^{\frac{3}{2}}}.$$

Note that any of those spaces are equally valid ways to specify lines, and which is best depends upon the circumstances. However, one might wonder whether there exists a coordinate system where the measure of a set of lines is just an area in the dual space. In fact, there is such a coordinate system, and it is delightfully simple; it is the *normal coordinates* which specify a line in terms of the normal distance from the origin to the line, and the angle the normal of the line makes with respect to the x -axis (Figure 14.5). The implicit equation for such lines is

$$x \cos \theta + y \sin \theta - p = 0.$$

And, indeed, the measure in that space is

$$d\mu = dp \ d\theta.$$

We shall use these measures to choose fair random lines in a later section.

14.1.3 Example: Measure of Lines in 3D

In 3D there are many ways to parameterize lines. Perhaps, the simplest way is to use their intersection with a particular plane along with some specification of their orientation. For example, we could chart the intersection with the xy plane along with the spherical coordinates of its orientation. Thus, each line would be specified as a (x, y, θ, ϕ) quadruple. This shows that lines in 3D are 4D entities, i.e., they can be described as points in a 4D space.

The differential measure of a line should not vary with (x, y) , but bundles of lines with equal cross section should have equal measure. Thus, a fair differential measure is

$$d\mu = dx \ dy \ \sin \theta \ d\theta \ d\phi.$$

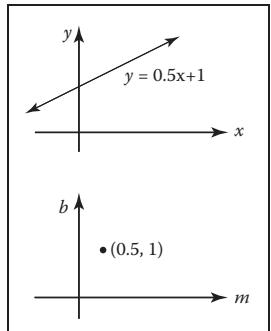


Figure 14.3. The set of points on the line $y = mx + b$ in (x, y) space can also be represented by a single point in (m, b) space so the top line and the bottom point represent the same geometric entity: a 2D line.

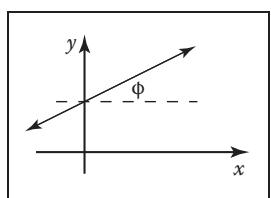


Figure 14.4. In angle-intercept space we parameterize the line by angle $\phi \in [-\pi/2, \pi/2]$ rather than slope.

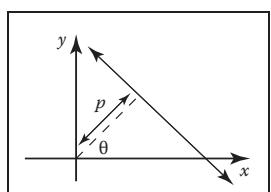


Figure 14.5. The normal coordinates of a line use the normal distance to the origin and an angle to specify a line.

对于在 (u, v) 空间中参数化的线的空间

$$ux + vy + 1 = 0,$$

适当的措施是

$$d\mu = \frac{du \ dv}{(u^2 + v^2)^{\frac{3}{2}}}.$$

对于以 (a, b) 、 x 截距和 y 截距参数化的线，度量为

$$d\mu = \frac{ab \ da \ db}{(a^2 + b^2)^{\frac{3}{2}}}.$$

请注意，这些空格中的任何一个都是同样有效的指定行的方法，哪个最好取决于具体情况。然而，人们可能想知道是否存在一个坐标系统，其中一组线的度量只是对偶空间中的一个区域。事实上，有这样一个坐标系统，它非常简单；它是法线坐标，根据从原点到线的法线距离指定一条线，以及线的法线相对于 x 轴的角度（图 14.5）。这些线的隐含方程是

$$x \cos \theta + y \sin \theta - p = 0.$$

事实上，这个空间的衡量标准是

$$d\mu = dp \ d\theta.$$

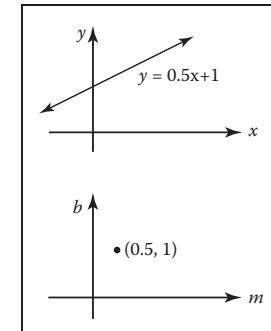
我们将在后面的章节中使用这些措施来选择公平的随机线。

14.1.3 示例：3D 线的度量

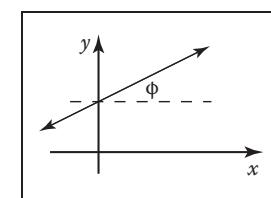
在 3D 中，有许多方法可以参数化线。也许，最简单的方法是使用它们与特定平面的交集以及它们的方向的一些规范。例如，我们可以绘制与 xy 平面的交点以及其方向的球面坐标。因此，每条线将被指定为 (x, y, θ, ϕ) 四倍。这表明 3D 中的线是 4D 实体，即它们可以被描述为 4D 空间中的点。

线的微分度量不应随 (x, y) 而变化，但具有相等横截面的线束应具有相等的度量。因此，公平的差异度量是

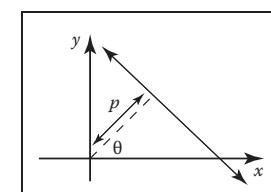
$$d\mu = dx \ dy \ \sin \theta \ d\theta \ d\phi.$$



(x, y) 空间中 $y = mx + b$ 线上的点集也可以由 (m, b) 空间中的单个点重新表示，因此顶点和底点表示相同的几何实体：2D 线。



在角截距空间中我们用角 ϕ [$-\pi/2, \pi/2$) 而不是斜率对线进行了参数化。



线的法线坐标使用到原点的法线距离和角度来指定线。

Another way to parameterize lines is to chart the intersection with two parallel planes. For example, if the line intersects the plane $z = 0$ at $(x = u, y = v)$ and the plane $z = 1$ at $(x = s, y = t)$, then the line can be described by the quadruple (u, v, s, t) . Note, that like the previous parameterization, this one is degenerate for lines parallel to the xy plane. The differential measure is more complicated for this parameterization although it can be approximated as

$$d\mu \approx du dv a ds dt,$$

for bundles of lines nearly parallel to the z -axis. This is the measure often implicitly used in image-based rendering.

For sets of lines that intersect a sphere, we can use the parameterization of the two points where the line intersects the sphere. If these are in spherical coordinates, then the point can be described by the quadruple $(\theta_1, \phi_1, \theta_2, \phi_2)$ and the measure is just the differential area associated with each point:

$$d\mu = \sin \theta_1 d\theta_1 d\phi_1 \sin \theta_2 d\theta_2 d\phi_2.$$

This implies that picking two uniform random endpoints on the sphere results in a line with uniform density. This observation was used to compute form-factors by Mateu Sbert in his dissertation (Sbert, 1997).

Note that sometimes we want to parameterize directed lines, and sometimes we want the order of the endpoints not to matter. This is a bookkeeping detail that is especially important for rendering applications where the amount of light flowing along a line is different in the two directions along the line.

14.2 Continuous Probability

Many graphics algorithms use probability to construct random samples to solve integration and averaging problems. This is the domain of applied continuous probability which has basic connections to measure theory.

14.2.1 One-Dimensional Continuous Probability Density Functions

Loosely speaking, a *continuous random variable* x is a scalar or vector quantity that “randomly” takes on some value from the real line $\mathbb{R} = (-\infty, +\infty)$. The behavior of x is entirely described by the distribution of values it takes. This distribution of values can be quantitatively described by

另一种参数化线的方法是用两个平行平面绘制交点。例如，如果线在($x=u$ $y=v$)处与平面 $z=0$ 相交并且在($x=s$ $y=t$)处与平面 $z=1$ 相交，那么线可以通过四元组(u v s t)来描述。请注意，与之前的参数化一样，对于平行于 xy 平面的线，此参数化是退化的。对于这种参数化，差分度量更复杂，尽管它可以近似为

$$d\mu \approx du dv a ds dt,$$

对于几乎平行于 z 轴的线束。这是在基于图像的渲染中经常隐含地使用的度量。

对于与球体相交的线集，我们可以使用线与球体相交的两个点的参数化。如果这些是球面坐标，那么该点可以用四元组 $(\theta_1, \phi_1, \theta_2, \phi_2)$ 来描述，并且度量只是与每个点相关的差分区域：

$$d\mu = \sin \theta_1 d\theta_1 d\phi_1 \sin \theta_2 d\theta_2 d\phi_2.$$

这意味着在球体上选择两个均匀的随机端点会导致具有均匀密度的线。这一观察结果被Mateu Sbert在他的论文 (Sbert, 1997) 中用于计算形状因子。

请注意，有时我们希望参数化有向线，有时我们希望端点的顺序无关紧要。这是一个笔记细节，对于沿一条线流动的光量在沿该线的两个方向上不同的渲染应用程序尤其重要。

14.2 连续概率

许多图形算法使用概率构造随机样本来解决积分和平均问题。这是应用连续概率的领域，它与测量理论有基本的联系。

14.2.1 一维连续概率密度函数

松散地说，连续随机变量 x 是标量或矢量量
实线 $R=(-\infty, +\infty)$ 。 x 的行为完全由它所采用的值的分布来描述。这种值的分布可以通过以下方式定量描述



the *probability density function* (pdf), p , associated with x (the relationship is denoted $x \sim p$). The probability that x assumes a particular value in some interval $[a, b]$ is given by the following integral:

$$\text{Probability}(x \in [a, b]) = \int_a^b p(x)dx. \quad (14.1)$$

Loosely speaking, the probability density function p describes the relative likelihood of a random variable taking a certain value; if $p(x_1) = 6.0$ and $p(x_2) = 3.0$, then a random variable with density p is twice as likely to have a value “near” x_1 than it is to have a value near x_2 . The density p has two characteristics:

$$p(x) \geq 0 \quad (\text{probability is nonnegative}), \quad (14.2)$$

$$\int_{-\infty}^{+\infty} p(x)dx = 1 \quad (\text{Probability}(x \in \mathbb{R}) = 1). \quad (14.3)$$

As an example, the *canonical* random variable ξ takes on values between zero (inclusive) and one (non-inclusive) with uniform probability (here *uniform* simply means each value for ξ is equally likely). This implies that the probability density function q for ξ is

$$q(\xi) = \begin{cases} 1 & \text{if } 0 \leq \xi < 1, \\ 0 & \text{otherwise,} \end{cases}$$

The space over which ξ is defined is simply the interval $[0, 1]$. The probability that ξ takes on a value in a certain interval $[a, b] \in [0, 1]$ is

$$\text{Probability}(a \leq \xi \leq b) = \int_a^b 1 dx = b - a.$$

14.2.2 One-Dimensional Expected Value

The average value that a real function f of a one-dimensional random variable with underlying pdf p will take on is called its *expected value*, $E(f(x))$ (sometimes written $Ef(x)$):

$$E(f(x)) = \int f(x)p(x)dx.$$

The expected value of a one-dimensional random variable can be calculated by setting $f(x) = x$. The expected value has a surprising and useful property: the



概率密度函数 (pdf) , p , 与 x 相关联 (关系记为 $x \in p$) 。 X 在某个区间 $[a, b]$ 中假设特定值的概率由以下积分给出:

$$\text{Probability}(x \in [a, b]) = \int_a^b p(x)dx. \quad (14.1)$$

松散地说, 概率密度函数 p 描述了随机变量取某个值的相对似然性; 如果 $p(x_1) = 6.0$ 和 $p(x_2) = 3.0$, 那么密度为 p 的随机变量在 x_1 附近具有值的可能性是在 x_2 附近具有值的可能性的两倍。密度 p 具有两个特性:

$$p(x) \geq 0 \quad (\text{probability is nonnegative}), \quad (14.2)$$

$$\int_{-\infty}^{+\infty} p(x)dx = 1 \quad (\text{Probability}(x \in \mathbb{R}) = 1). \quad (14.3)$$

作为一个例子, 规范随机变量 ξ 以均匀的概率 (这里均匀只是意味着 ξ 的每个值都具有相同的可能性) 接受零 (包含) 和一 (非包含) 之间的值。这意味着 ξ 的概率密度函数 q 是

$$q(\xi) = \begin{cases} 1 & \text{if } 0 \leq \xi < 1, \\ 0 & \text{otherwise,} \end{cases}$$

定义 ξ 的空间只是区间 $[0, 1]$ 。 \exists 在某个区间 $[a, b] \in [0, 1]$ 中取值的概率为

$$\text{Probability}(a \leq \xi \leq b) = \int_a^b 1 dx = b - a.$$

14.2.2 一维期望值

具有底层 pdf p 的一维随机变量的实函数 f 将承担的平均值称为其期望值, $E(f(x))$ (有时写为 $Ef(x)$) :

$$E(f(x)) = \int f(x)p(x)dx.$$

一维随机变量的期望值可以通过设置 $f(x) = x$ 来计算。期望值有一个令人惊讶和有用的属性:

expected value of the sum of two random variables is the sum of the expected values of those variables:

$$E(x + y) = E(x) + E(y),$$

for random variables x and y . Because functions of random variables are themselves random variables, this linearity of expectation applies to them as well:

$$E(f(x) + g(y)) = E(f(x)) + E(g(y)).$$

An obvious question to ask is whether this property holds if the random variables being summed are correlated (variables that are not correlated are called *independent*). This linearity property in fact does hold *whether or not* the variables are independent! This summation property is vital for most Monte Carlo applications.

14.2.3 Multidimensional Random Variables

The discussion of random variables and their expected values extends naturally to multidimensional spaces. Most graphics problems will be in such higher-dimensional spaces. For example, many lighting problems are phrased on the surface of the hemisphere. Fortunately, if we define a measure μ on the space the random variables occupy, everything is very similar to the one-dimensional case. Suppose the space S has associated measure μ ; for example S is the surface of a sphere and μ measures area. We can define a pdf $p : S \mapsto \mathbb{R}$, and if x is a random variable with $x \sim p$, then the probability that x will take on a value in some region $S_i \subset S$ is given by the integral

$$\text{Probability}(x \in S_i) = \int_{S_i} p(x) d\mu.$$

Here *Probability (event)* is the probability that *event* is true, so the integral is the probability that x takes on a value in the region S_i .

In graphics, S is often an area ($d\mu = dA = dx dy$) or a set of directions (points on a unit sphere: $d\mu = d\omega = \sin \theta d\theta d\phi$). As an example, a two-dimensional random variable α is a uniformly distributed random variable on a disk of radius R . Here *uniformly* means uniform with respect to area, e.g., the way a bad dart player's hits would be distributed on a dart board. Since it is uniform, we know that $p(\alpha)$ is some constant. From the fact that the area of the disk is πr^2 and that the total probability is one, we can deduce that

$$p(\alpha) = \frac{1}{\pi R^2}.$$

两个随机变量之和的期望值是这些变量的期望值之和:

$$E(x + y) = E(x) + E(y),$$

为随机变量 x 和 y 。因为随机变量的函数本身就是随机变量，所以期望的线性也适用于它们:

$$E(f(x) + g(y)) = E(f(x)) + E(g(y)).$$

一个显而易见的问题是，如果被求和的随机变量是相关的（不相关的变量被称为独立的），这个属性是否成立。这个线性属性实际上确实保持变量是否独立！此求和属性对于大多数蒙特卡罗应用程序至关重要。

14.2.3 多维随机变量

对随机变量及其期望值的讨论自然延伸到多维空间。

大多数图形问题都会在这样的更高维空间中。例如，许多照明问题都表现在半球的表面上。幸运的是，如果我们在随机变量占据的空间上定义度量 μ ，则一切都与一维情况非常相似。假设空间 S 具有相关的度量 μ ；例如 S 是球体的表面和 μ 度量区域。我们可以定义一个pdf $p : S \rightarrow \mathbb{R}$ ，如果 x 是一个带有 $x \in p$ 的随机变量，那么 x 在某些区域 $S_i \in S$ 中取值的概率由积分给出

$$\text{Probability}(x \in S_i) = \int_{S_i} p(x) d\mu.$$

这里概率（事件）是事件为真的概率，因此积分是 x 在区域 S_i 中取值的概率。

在图形中， S 通常是一个区域 ($d\mu=dA=dx dy$) 或一组方向 (单位球体上的点: $d\mu=d\omega=\sin\theta d\theta d\phi$)。作为一个例子，二维随机变量 α 是半径 R 的盘上均匀分布的随机变量。这里统一的意思是关于区域的统一，例如，一个坏的飞镖运动员的命中将分布在飞镖板上的方式。由于它是均匀的，我们知道 $p(\alpha)$ 是一些常数。从磁盘的面积是 πr^2 并且总概率是1的事实，我们可以推断出

$$p(\alpha) = \frac{1}{\pi R^2}.$$



This means that the probability that α is in a certain subset S_1 of the disk is just

$$\text{Probability}(\alpha \in S_1) = \int_{S_1} \frac{1}{\pi R^2} dA.$$

This is all very abstract. To actually use this information, we need the integral in a form we can evaluate. Suppose S_i is the portion of the disk closer to the center than the perimeter. If we convert to polar coordinates, then α is represented as a (r, ϕ) pair, and S_1 is the region where $r < R/2$. Note, that just because α is uniform, it does not imply that ϕ or r are necessarily uniform (in fact, ϕ is uniform, and r is not uniform). The differential area dA is just $r dr d\phi$. Thus,

$$\text{Probability}\left(r < \frac{R}{2}\right) = \int_0^{2\pi} \int_0^{\frac{R}{2}} \frac{1}{\pi R^2} r dr d\phi = 0.25.$$

The formula for expected value of a real function applies to the multidimensional case:

$$E(f(x)) = \int_S f(x)p(x)d\mu,$$

where $x \in S$ and $f : S \mapsto \mathbb{R}$, and $p : S \mapsto \mathbb{R}$. For example, on the unit square $S = [0, 1] \times [0, 1]$ and $p(x, y) = 4xy$, the expected value of the x coordinate for $(x, y) \sim p$ is

$$\begin{aligned} E(x) &= \int_S f(x, y)p(x, y)dA \\ &= \int_0^1 \int_0^1 4x^2y dx dy \\ &= \frac{2}{3}. \end{aligned}$$

Note that here $f(x, y) = x$.

14.2.4 Variance

The *variance*, $V(x)$, of a one-dimensional random variable is, by definition, the expected value of the square of the difference between x and $E(x)$:

$$V(x) \equiv E([x - E(x)]^2).$$

Some algebraic manipulation gives the non-obvious expression:

$$V(x) = E(x^2) - [E(x)]^2.$$



这意味着 α 在磁盘的某个子集 S_1 中的概率只是

$$\text{Probability}(\alpha \in S_1) = \int_{S_1} \frac{1}{\pi R^2} dA.$$

这一切都非常抽象。要实际使用这些信息，我们需要以我们可以评估的形式进行积分。假设 S_i 是磁盘比周边更靠近中心的部分。如果我们转换为极坐标，那么 α 表示为 (r, ϕ) 对， S_1 是 $r < R/2$ 的区域。注意，仅仅因为 α 是均匀的，并不意味着 ϕ 或 r 必然是均匀的（实际上， ϕ 是均匀的，而 r 不是均匀的）。差分区域 dA 只是 $r dr d\phi$ 。因此

$$\text{Probability}\left(r < \frac{R}{2}\right) = \int_0^{2\pi} \int_0^{\frac{R}{2}} \frac{1}{\pi R^2} r dr d\phi = 0.25.$$

实函数期望值公式适用于多维情况：

$$E(f(x)) = \int_S f(x)p(x)d\mu,$$

其中 $x \in S$ 和 $f : S \mapsto \mathbb{R}$, 和 $p : S \mapsto \mathbb{R}$ 。例如，在单位平方 S 上 $= [0, 1] \times [0, 1]$ 且 $p(x, y) = 4xy$, 则 x 坐标对 $(x, y) \in p$ 的期望值为

$$\begin{aligned} E(x) &= \int_S f(x, y)p(x, y)dA \\ &= \int_0^1 \int_0^1 4x^2y dx dy \\ &= \frac{2}{3}. \end{aligned}$$

注意这里 $f(x, y) = x$ 。

14.2.4 Variance

根据定义，一维随机变量的方差 $V(x)$ 是 x 和 $E(x)$ 之间差值平方的期望值：

$$V(x) \equiv E([x - E(x)]^2).$$

一些代数操作给出了非显而易见的表达式：

$$V(x) = E(x^2) - [E(x)]^2.$$

The expression $E([x - E(x)]^2)$ is more useful for thinking intuitively about variance, while the algebraically equivalent expression $E(x^2) - [E(x)]^2$ is usually convenient for calculations. The variance of a sum of random variables is the sum of the variances *if the variables are independent*. This summation property of variance is one of the reasons it is frequently used in analysis of probabilistic models. The square root of the variance is called the *standard deviation*, σ , which gives some indication of expected absolute deviation from the expected value.

14.2.5 Estimated Means

Many problems involve sums of independent random variables x_i , where the variables share a common density p . Such variables are said to be *independent identically distributed* (iid) random variables. When the sum is divided by the number of variables, we get an estimate of $E(x)$:

$$E(x) \approx \frac{1}{N} \sum_{i=1}^N x_i.$$

As N increases, the variance of this estimate decreases. We want N to be large enough so that we have confidence that the estimate is “close enough.” However, there are no sure things in Monte Carlo; we just gain statistical confidence that our estimate is good. To be sure, we would have to have $N = \infty$. This confidence is expressed by the *Law of Large Numbers*:

$$\text{Probability} \left[E(x) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N x_i \right] = 1.$$

14.3 Monte Carlo Integration

In this section, the basic Monte Carlo solution methods for definite integrals are outlined. These techniques are then straightforwardly applied to certain integral problems. All of the basic material of this section is also covered in several of the classic Monte Carlo texts. (See the Notes section at the end of this chapter.)

As discussed earlier, given a function $f : S \mapsto \mathbb{R}$ and a random variable $x \sim p$, we can approximate the expected value of $f(x)$ by a sum:

$$E(f(x)) = \int_{x \in S} f(x)p(x)d\mu \approx \frac{1}{N} \sum_{i=1}^N f(x_i). \quad (14.4)$$

表达式 $E([x - E(x)]^2)$ 对于直观地思考方差更有用，而代数等价表达式 $E(x^2) - [E(x)]^2$ 通常便于计算。随机变量之和的方差是变量独立时的方差之和。方差的这种求和属性是它在概率模型分析中经常使用的原因之一。方差的平方根称为标准偏差 σ ，它给出了预期绝对偏差与期望值的一些指示。

14.2.5 估计手段

许多问题涉及独立随机变量 x_i 的总和，其中变量共享公共密度 p 。这样的变量被认为是独立的相同分布 (iid) 随机变量。当总和除以变量数时，我们得到 $E(x)$ 的估计值：

$$E(x) \approx \frac{1}{N} \sum_{i=1}^N x_i.$$

随着 N 的增加，该估计的方差减小。我们希望 N 足够大，以便我们有信心估计“足够接近。”然而，蒙特卡洛没有确定的事情；我们只是获得统计上的信心，我们的估计是好的。可以肯定的是，我们必须有 $N = \infty$ 。这种信心由大数定律表示：

$$\text{Probability} \left[E(x) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N x_i \right] = 1.$$

14.3 蒙特卡罗积分

本节概述了定积分的基本蒙特卡罗解法。然后将这些技术直接应用于某些积分问题。本节的所有基本材料也包括在一些经典的蒙特卡洛文本。（请参阅本章末尾的注释部分。）

如前所述，给定一个函数 $f: S \mapsto \mathbb{R}$ 和一个随机变量 $x \sim p$ ，我们可以用一个和来近似 $f(x)$ 的期望值：

$$E(f(x)) = \int_{x \in S} f(x)p(x)d\mu \approx \frac{1}{N} \sum_{i=1}^N f(x_i). \quad (14.4)$$

Because the expected value can be expressed as an integral, the integral is also approximated by the sum. The form of Equation (14.4) is a bit awkward; we would usually like to approximate an integral of a single function g rather than a product fp . We can accomplish this by substituting $g = fp$ as the integrand:

$$\int_{x \in S} g(x)d\mu \approx \frac{1}{N} \sum_{i=1}^N \frac{g(x_i)}{p(x_i)}. \quad (14.5)$$

For this formula to be valid, p must be positive when g is nonzero.

So to get a good estimate, we want as many samples as possible, and we want the g/p to have a low variance (g and p should have a similar shape). Choosing p intelligently is called *importance sampling*, because if p is large where g is large, there will be more samples in important regions. Equation (14.4) also shows the fundamental problem with Monte Carlo integration: *diminishing return*. Because the variance of the estimate is proportional to $1/N$, the standard deviation is proportional to $1/\sqrt{N}$. Since the error in the estimate behaves similarly to the standard deviation, we will need to quadruple N to halve the error.

Another way to reduce variance is to partition S , the domain of the integral, into several smaller domains S_i , and evaluate the integral as a sum of integrals over the S_i . This is called *stratified sampling*, the technique that jittering employs in pixel sampling (Chapter 4). Normally only one sample is taken in each S_i (with density p_i), and in this case the variance of the estimate is:

$$\text{var} \left(\sum_{i=1}^N \frac{g(x_i)}{p_i(x_i)} \right) = \sum_{i=1}^N \text{var} \left(\frac{g(x_i)}{p_i(x_i)} \right). \quad (14.6)$$

It can be shown that the variance of stratified sampling is never higher than unstratified if all strata have equal measure:

$$\int_{S_i} p(x)d\mu = \frac{1}{N} \int_S p(x)d\mu.$$

The most common example of stratified sampling in graphics is jittering for pixel sampling as discussed in Section 13.4.

As an example of the Monte Carlo solution of an integral I , set $g(x)$ equal to x over the interval $(0, 4)$:

$$I = \int_0^4 x dx = 8. \quad (14.7)$$

The impact of the shape of the function p on the variance of the N sample estimates is shown in Table 14.1. Note that the variance is reduced when the shape of p is similar to the shape of g . The variance drops to zero if $p = g/I$, but

因为期望值可以表示为积分，所以积分也由和近似。方程 (14.4) 的形式有点尴尬；我们通常希望近似单个函数 g 的积分，而不是乘积 fp 。我们可以通过将 $g=fp$ 替换为被积函数来实现这一点：

$$\int_{x \in S} g(x)d\mu \approx \frac{1}{N} \sum_{i=1}^N \frac{g(x_i)}{p(x_i)}. \quad (14.5)$$

为了使此公式有效，当 g 非零时， p 必须为正。

因此，为了获得良好的估计，我们希望尽可能多的样本，并且我们希望 gp 具有低方差 (g 和 p 应该具有相似的形状)。智能地选择 p 被称为重要性抽样，因为如果 p 在 g 大的地方大，那么重要区域的样本就会更多。方程 (14.4) 还显示了蒙特卡洛积分的基本问题：收益递减。由于估计值的方差与 N 成正比，标准偏差与 $1/N$ 成正比

N 。由于估计中的误差与标准偏差的行为类似，我们将需要将 N 加倍以将误差减半。

减少方差的另一种方法是将积分的域 S 划分为几个较小的域 S_i ，并将积分评估为 S_i 上的积分之和。这称为分层采样，即抖动在像素采样中使用的技术（第 4 章）。通常在每个 S_i 中只取一个样本（具有密度 p_i ），并且在这种情况下估计的方差为：

$$\text{var} \left(\sum_{i=1}^N \frac{g(x_i)}{p_i(x_i)} \right) = \sum_{i=1}^N \text{var} \left(\frac{g(x_i)}{p_i(x_i)} \right). \quad (14.6)$$

可以看出，如果所有地层具有相等的度量，分层采样的方差永远不会高于未分层：

$$\int_{S_i} p(x)d\mu = \frac{1}{N} \int_S p(x)d\mu.$$

图形分层采样最常见的例子是像素采样的抖动，如第 13.4 节所述。

作为积分 I 的蒙特卡罗解的一个例子，在间隔 $(0, 4)$ 上设置 $g(x)$ 等于 x ：

$$I = \int_0^4 x dx = 8. \quad (14.7)$$

函数 p 的形状对 N 个样本估计值方差的影响见表 14.1。注意，当 p 的形状类似于 g 的形状时，方差减小。方差下降到零，如果 $p=g/I$ ，但

Method	Sampling function	Variance	Samples needed for standard error of 0.008
importance	$(6 - x)/(16)$	$56.8N^{-1}$	887,500
importance	$1/4$	$21.3N^{-1}$	332,812
importance	$(x + 2)/16$	$6.3N^{-1}$	98,437
importance	$x/8$	0	1
stratified	$1/4$	$21.3N^{-3}$	70

Table 14.1. Variance for Monte Carlo estimate of $\int_0^4 x dx$.

I is not usually known or we would not have to resort to Monte Carlo. One important principle illustrated in Table 14.1 is that stratified sampling is often *far* superior to importance sampling (Mitchell, 1996). Although the variance for this stratification on I is inversely proportional to the cube of the number of samples, there is no general result for the behavior of variance under stratification. There are some functions for which stratification does no good. One example is a white noise function, where the variance is constant for all regions. On the other hand, most functions will benefit from stratified sampling, because the variance in each subcell will usually be smaller than the variance of the entire domain.

14.3.1 Quasi–Monte Carlo Integration

A popular method for quadrature is to replace the random points in Monte Carlo integration with *quasi-random* points. Such points are deterministic, but are in some sense uniform. For example, on the unit square $[0, 1]^2$, a set of N quasi-random points should have the following property on a region of area A within the square:

$$\text{number of points in the region} \approx AN.$$

For example, a set of regular samples in a lattice has this property.

Quasi-random points can improve performance in many integration applications. Sometimes care must be taken to make sure that they do not introduce aliasing. It is especially nice that, in any application where calls are made to random or stratified points in $[0, 1]^d$, one can substitute d -dimensional quasi-random points with no other changes.

The key intuition motivating quasi–Monte Carlo integration is that when estimating the average value of an integrand, any set of sample points will do, provided they are “fair.”

Method	采样函数	Variance	所需样本 标准误差0.008
importance	$(6 - x)/(16)$	$56.8N^{-1}$	887,500
importance	$1/4$	$21.3N^{-1}$	332,812
importance	$(x + 2)/16$	$6.3N^{-1}$	98,437
importance	$x/8$	0	1
stratified	$1/4$	$21.3N^{-3}$	70

表14.1。蒙特卡罗估计的方差

$$\int_0^4 x dx.$$

我通常不知道，否则我们就不必求助于蒙特卡洛。表14.1中说明的一个重要原则是，分层抽样往往远远优于重要抽样（Mitchell, 1996）。虽然对于 I 上的这种分层的方差与样本数量的立方体成反比，但是对于分层下的方差行为没有一般的结果。有一些功能对分层没有好处。一个例子是白噪声函数，其中所有区域的方差都是恒定的。另一方面，大多数函数将受益于分层采样，因为每个子小区中的方差通常会小于整个域的方差。

14.3.1 准蒙特卡洛积分

正交的一种流行方法是用准随机点替换蒙特卡洛积分中的随机点。这些点是确定性的，但在某种意义上是统一的。例如，在单位正方形 $[0, 1]^2$ 上，一组 N 个准对称点应在正方形内区域 A 的区域上具有以下属性：

$$\text{区域中的点数} \triangleq AN.$$

例如，晶格中的一组规则样本具有此属性。

准随机点可以提高许多集成应用的性能。有时必须小心确保它们不会引入别名。特别好的是，在任何调用 $[0, 1]^d$ 中的random或分层点的应用程序中，都可以替换 d 维准随机点，而无需其他更改。

激励准蒙特卡洛积分的关键直觉是，当estimating一个被积函数的平均值时，任何一组样本点都会做，因为它们是“公平的。”



14.4 Choosing Random Points

We often want to generate sets of random or pseudorandom points on the unit square for applications such as distribution ray tracing. There are several methods for doing this, e.g., jittering (see Section 13.4). These methods give us a set of N reasonably equidistributed points on the unit square $[0, 1]^2$: (u_1, v_1) through (u_N, v_N) .

Sometimes, our sampling space may not be square (e.g., a circular lens), or may not be uniform (e.g., a filter function centered on a pixel). It would be nice if we could write a mathematical transformation that would take our equidistributed points (u_i, v_i) as input and output a set of points in our desired sampling space with our desired density. For example, to sample a camera lens, the transformation would take (u_i, v_i) and output (r_i, ϕ_i) such that the new points are approximately equidistributed on the disk of the lens. While we might be tempted to use the transform

$$\begin{aligned}\phi_i &= 2\pi u_i, \\ r_i &= v_i R,\end{aligned}$$

it has a serious problem. While the points do cover the lens, they do so nonuniformly (Figure 14.6). What we need in this case is a transformation that takes equal-area regions to equal-area regions—one that takes uniform sampling distributions on the square to uniform distributions on the new domain.

There are several ways to generate such nonuniform points or uniform points on non-rectangular domains, and the following sections review the three most often used: function inversion, rejection, and Metropolis.

14.4.1 Function Inversion

If the density $f(x)$ is one-dimensional and defined over the interval $x \in [x_{\min}, x_{\max}]$, then we can generate random numbers α_i that have density f from a set of uniform random numbers ξ_i , where $\xi_i \in [0, 1]$. To do this, we need the cumulative probability distribution function $P(x)$:

$$\text{Probability}(\alpha < x) = P(x) = \int_{x_{\min}}^x f(x') d\mu.$$

To get α_i , we simply transform ξ_i :

$$\alpha_i = P^{-1}(\xi_i),$$

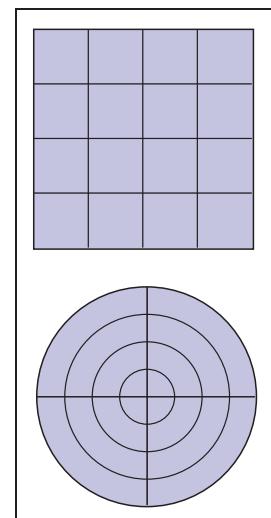


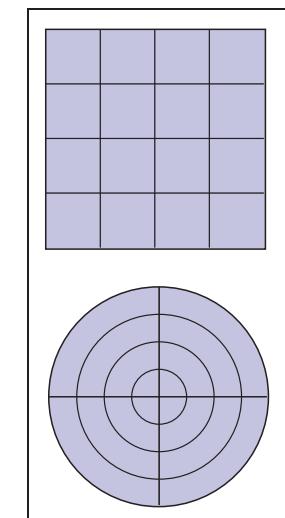
Figure 14.6. The transform that takes the horizontal and vertical dimensions uniformly to (r, ϕ) does not preserve relative area; not all of the resulting areas are the same.



14.4 选择随机点

我们经常希望在单位正方形上生成随机或伪随机点集，以用于分布光线跟踪等应用。有几种方法可以做到这一点，例如抖动（见第13.4节）。这些方法给出了一组在单位平方 $[0, 1]^2$ 上合理分布的N个点： (u_1, v_1) 通过 (u_N, v_N) 。

有时，我们的采样空间可能不是正方形（例如，圆形透镜），或者可能不均匀（例如，以像素为中心的滤波器函数）。如果我们可以编写一个数学变换，将我们的等分布点 (u_i, v_i) 作为输入，并以我们所需的密度输出我们所需的采样空间中的一组点，那将是很好的。例如，要对相机镜头进行采样，变换将采用 (u_i, v_i) 和输出 (r_i, ϕ_i) ，以便新点在镜头磁盘上近似等分布。虽然我们可能会试图使用转换



将水平和垂直尺寸统一为 (r, ϕ) 的变换不保留relative区域；并非所有结果区域都是相同的。

$$\begin{aligned}\phi_i &= 2\pi u_i, \\ r_i &= v_i R,\end{aligned}$$

它有一个严重的问题。虽然点确实覆盖了镜头，但它们是非固定的（图14.6）。在这种情况下，我们需要的是将等面积区域转换为等面积区域—将正方形上的均匀采样distribuits转换为新域上的均匀分布。

有几种方法可以在非矩形域上生成这种非均匀点或均匀点，以下部分回顾了最常用的三种方法：函数反转，拒绝和大都会。

14.4.1 函数反转

如果密度 $f(x)$ 是一维的，并且在间隔 $x \in [x_{\min}, x_{\max}]$ 上定义，那么我们可以从一组均匀随机数 ξ_i 中生成具有密度 f 的随机数 α_i ，其中 $\xi_i \in [0, 1]$ 。为此，我们需要累积概率分布函数 $P(x)$ ：

$$\text{Probability}(\alpha < x) = P(x) = \int_{x_{\min}}^x f(x') d\mu.$$

为了得到 α_i ，我们简单地变换 ξ_i ：

$$\alpha_i = P^{-1}(\xi_i),$$

where P^{-1} is the inverse of P . If P is not analytically invertible, then numerical methods will suffice, because an inverse exists for all valid probability distribution functions.

Note that analytically inverting a function is more confusing than it should be due to notation. For example, if we have the function

$$y = x^2,$$

for $x > 0$, then the inverse function is expressed in terms of y as a function of x :

$$x = \sqrt{y}.$$

When the function is analytically invertible, it is almost always that simple. However, things are a little more opaque with the standard notation:

$$\begin{aligned} f(x) &= x^2, \\ f^{-1}(x) &= \sqrt{x}. \end{aligned}$$

Here x is just a dummy variable. You may find it easier to use the less standard notation:

$$\begin{aligned} y &= x^2, \\ x &= \sqrt{y}, \end{aligned}$$

while keeping in mind that these are inverse functions of each other.

For example, to choose random points x_i that have density

$$p(x) = \frac{3x^2}{2}$$

on $[-1, 1]$, we see that

$$P(x) = \frac{x^3 + 1}{2},$$

and

$$P^{-1}(x) = \sqrt[3]{2x - 1},$$

so we can "warp" a set of canonical random numbers (ξ_1, \dots, ξ_N) to the properly distributed numbers

$$(x_1, \dots, x_N) = (\sqrt[3]{2\xi_1 - 1}, \dots, \sqrt[3]{2\xi_N - 1}).$$

Of course, this same warping function can be used to transform "uniform" jittered samples into nicely distributed samples with the desired density.

其中 P^{-1} 是 P 的倒数。如果 P 在解析上不可逆，那么数值方法就足够了，因为所有有效的概率分布函数都存在逆。

请注意，由于符号的原因，解析反演函数比它应该更令人困惑。例如，如果我们有函数

$$y = x^2,$$

对于 $x > 0$, 则反函数以 y 表示为 x 的函数:

$$x = \sqrt{y}.$$

当函数在解析上是可反转的时，它几乎总是那么简单。但是，使用标准符号时，事情有点不透明:

$$\begin{aligned} f(x) &= x^2, \\ f^{-1}(x) &= \sqrt{x}. \end{aligned}$$

这里 x 只是一个虚拟变量。您可能会发现使用不太标准的符号更容易:

$$\begin{aligned} y &= x^2, \\ x &= \sqrt{y}, \end{aligned}$$

同时请记住，这些是彼此的反函数。例如，要选择具有密度的随机点 x_i

$$p(x) = \frac{3x^2}{2}$$

在 $[-1, 1]$ 上，我们看到

$$P(x) = \frac{x^3 + 1}{2},$$

and

$$P^{-1}(x) = \sqrt[3]{2x - 1},$$

因此，我们可以将一组规范随机数 (ξ_1, \dots, ξ_N) "扭曲" 到正确分布的数字

$$(x_1, \dots, x_N) = (\sqrt[3]{2\xi_1 - 1}, \dots, \sqrt[3]{2\xi_N - 1}).$$

当然，这种相同的翘曲函数可用于将"均匀"抖动的样本转换成具有所需密度的良好分布的样本。



If we have a random variable $\alpha = (\alpha_x, \alpha_y)$ with two-dimensional density (x, y) defined on $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$, then we need the two-dimensional distribution function:

$$\text{Probability}(\alpha_x < x \text{ and } \alpha_y < y) = F(x, y) = \int_{y_{\min}}^y \int_{x_{\min}}^x f(x', y') d\mu(x', y').$$

We first choose an x_i using the marginal distribution $F(x, y_{\max})$ and then choose y_i according to $F(x_i, y)/F(x_i, y_{\max})$. If $f(x, y)$ is separable (expressible as $g(x)h(y)$), then the one-dimensional techniques can be used on each dimension.

Returning to our earlier example, suppose we are sampling uniformly from the disk of radius R , so $p(r, \phi) = 1/(\pi R^2)$. The two-dimensional distribution function is

$$\text{Probability}(r < r_0 \text{ and } \phi < \phi_0) = F(r_0, \phi_0) = \int_0^{\phi_0} \int_0^{r_0} \frac{r dr d\phi}{\pi R^2} = \frac{\phi r^2}{2\pi R^2}.$$

This means that a canonical pair (ξ_1, ξ_2) can be transformed to a uniform random point on the disk:

$$\begin{aligned}\phi &= 2\pi\xi_1, \\ r &= R\sqrt{\xi_2}.\end{aligned}$$

This mapping is shown in Figure 14.7.

To choose reflected ray directions for some realistic rendering applications, we choose points on the unit hemisphere according to the density:

$$p(\theta, \phi) = \frac{n+1}{2\pi} \cos^n \theta,$$

where n is a Phong-like exponent, θ is the angle from the surface normal and $\theta \in [0, \pi/2]$ (is on the upper hemisphere) and ϕ is the azimuthal angle ($\phi \in [0, 2\pi]$).

The cumulative distribution function is

$$P(\theta, \phi) = \int_0^\phi \int_0^\theta p(\theta', \phi') \sin \theta' d\theta' d\phi'. \quad (14.8)$$

The $\sin \theta'$ term arises because, on the sphere, $d\omega = \cos \theta d\theta d\phi$. When the marginal densities are found, p (as expected) is separable, and we find that a (ξ_1, ξ_2) pair of canonical random numbers can be transformed to a direction by

$$\begin{aligned}\theta &= \arccos \left((1 - \xi_1)^{\frac{1}{n+1}} \right), \\ \phi &= 2\pi\xi_2.\end{aligned}$$



如果我们有一个随机变量 $\alpha = (\alpha_x, \alpha_y)$ ，其二维密度 (x, y) 定义在 $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ 上，那么我们需要二维分布函数：

$$\text{概率}(\alpha_x < x \text{ 和 } \alpha_y < y) = F(x, y) = \int_{y_{\min}}^y \int_{x_{\min}}^x f(x', y') d\mu(x', y').$$

我们首先使用边际分布 $F(x, y_{\max})$ 选择一个 x_i ，然后根据 $F(x_i, y)$ 从 y_{\min} 到 y_{\max} 选择 y_i 。如果 $f(x, y)$ 是可分离的（可表达为 $g(x)h(y)$ ），那么一维技术可以用于每个维度。回到我们前面的例子，假设我们从半径 R 的磁盘均匀采样，所以 $p(r, \phi) = 1/(\pi R^2)$ 。二维分布函数为

$$\text{概率}(r < r_0 \text{ 且 } \phi < \phi_0) = F(r_0, \phi_0) = \int_0^{\phi_0} \int_0^{r_0} \frac{r dr d\phi}{\pi R^2} = \frac{\phi r^2}{2\pi R^2}.$$

这意味着规范对 (ξ_1, ξ_2) 可以转换为磁盘上的均匀随机点：

$$\begin{aligned}\phi &= 2\pi\xi_1, \\ r &= R\sqrt{\xi_2}.\end{aligned}$$

该映射如图 14.7 所示。

要为一些逼真的渲染应用选择反射射线方向，我们根据密度在单位半球上选择点：

$$p(\theta, \phi) = \frac{n+1}{2\pi} \cos^n \theta,$$

其中 n 是一个类似 Phong 的指数， θ 是与表面法线的角度， $\theta \in [0, \pi/2]$ （在上半球）， ϕ 是方位角 $(\phi \in [0, 2\pi])$ 。累积分布函数为

$$P(\theta, \phi) = \int_0^\phi \int_0^\theta p(\theta', \phi') \sin \theta' d\theta' d\phi'. \quad (14.8)$$

当发现边际密度时， p （如预期的那样）是可分的，我们发现 $a(\xi_1, \xi_2)$ 对规范随机数可以通过以下方式转换为方向

$$\begin{aligned}\theta &= \arccos \left((1 - \xi_1)^{\frac{1}{n+1}} \right), \\ \phi &= 2\pi\xi_2.\end{aligned}$$

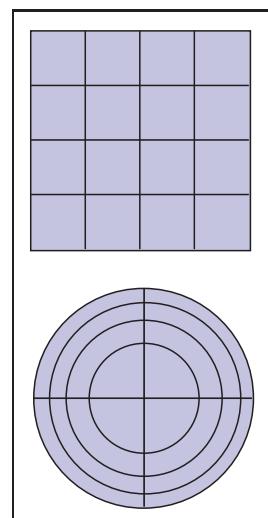
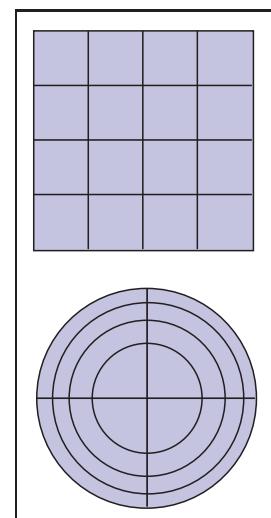


Figure 14.7. A mapping that takes equal area regions in the unit square to equal area regions in the disk.



将单位正方形中的等面积区域带到磁盘中的等面积区域的映射。

Again, a nice thing about this is that a set of jittered points on the unit square can be easily transformed to a set of jittered points on the hemisphere with the desired distribution. Note that if n is set to 1, we have a diffuse distribution, as is often needed.

Often we must map the point on the sphere into an appropriate direction with respect to a uvw basis. To do this, we can first convert the angles to a unit vector \vec{a} :

$$\mathbf{a} = (\cos \phi \sin \theta, \sin \phi \sin \theta, \cos \theta)$$

As an efficiency improvement, we can avoid taking trigonometric functions of inverse trigonometric functions (e.g., $\cos(\arccos \theta)$). For example, when $n = 1$ (a diffuse distribution), the vector \mathbf{a} simplifies to

$$\mathbf{a} = (\cos(2\pi\xi_1)\sqrt{\xi_2}, \sin(2\pi\xi_1)\sqrt{\xi_2}, \sqrt{1-\xi_2})$$

14.4.2 Rejection

A *rejection* method chooses points according to some simple distribution and rejects some of them that are in a more complex distribution. There are several scenarios where rejection is used, and we show some of these by example.

Suppose we want uniform random points within the unit circle. We can first choose uniform random points $(x, y) \in [-1, 1]^2$ and reject those outside the circle. If the function $r()$ returns a canonical random number, then the procedure is:

```
done = false
while (not done) do
    x = -1 + 2r()
    y = -1 + 2r()
    if ( $x^2 + y^2 < 1$ ) then
        done = true
```

If we want a random number $x \sim p$ and we know that $p : [a, b] \mapsto \mathbb{R}$, and that for all x , $p(x) < m$, then we can generate random points in the rectangle $[a, b] \times [0, m]$ and take those where $y < p(x)$:

```
done = false
while (not done) do
    x = a + r()(b - a)
    y = r()m
    if ( $y < p(x)$ ) then
        done = true
```

同样，关于这一点的一个好处是，单位正方形上的一组抖动点可以很容易地转换为具有所需分布的半球上的一组抖动点。请注意，如果n设置为1，我们有一个扩散分布，这是经常需要的。

通常，我们必须将球体上的点映射到相对于uvw基础的适当方向。为此，我们可以首先将角度转换为单位矢量 \mathbf{a} :

$$\mathbf{a} = (\cos \phi \sin \theta, \sin \phi \sin \theta, \cos \theta)$$

作为效率改进，我们可以避免取反三角函数的三角函数（例如， $\cos(\arccos \theta)$ ）。例如，当n=1（漫射分布）时，矢量a简化为

$$\mathbf{a} = (\cos(2\pi\xi_1)\sqrt{\xi_2}, \sin(2\pi\xi_1)\sqrt{\xi_2}, \sqrt{1-\xi_2})$$

14.4.2 Rejection

拒绝方法根据一些简单的分布来选择点，并将其中一些分布在更复杂的分布中。有几种使用拒绝的情况，我们通过示例展示了其中的一些情况。

假设我们想要单位圆内的均匀随机点。我们可以首先选择均匀随机点 $(x, y) \in [-1, 1]^2$ 并拒绝circle之外的那些。如果函数r()返回规范随机数，则过程为：

```
done=falsewhile(notd
one)dox=-1+2r()y=-1
+2r()if(x2+y2<1)thend
one=true
```

如果我们想要一个随机数 $x \neq p$ 并且我们知道 $p : [a, b] \rightarrow \mathbb{R}$ ，并且对于所有 x , $p(x) < m$ ，那么我们可以在矩形 $[a, b] \times [0, m]$ 中生成随机点，并取 $y < p(x)$ 的那些：

```
done=falsewhile(no
tdone)dox=a+r()(b
a)y=r()mif(y<p(x))th
endone=true
```

This same idea can be applied to take random points on the surface of a sphere. To pick a random unit vector with uniform directional distribution, we first pick a random point in the unit sphere and then treat that point as a direction vector by taking the unit vector in the same direction:

```
done = false
while (not done) do
    x = -1 + 2r()
    y = -1 + 2r()
    z = -1 + 2r()
    if ((l =  $\sqrt{x^2 + y^2 + z^2}$ ) < 1) then
        done = true
    x = x/l
    y = y/l
    z = z/l
```

Although the rejection method is usually simple to code, it is rarely compatible with stratification. For this reason, it tends to converge more slowly and should thus be used mainly for debugging, or in particularly difficult circumstances.

14.4.3 Metropolis

The *Metropolis* method uses random *mutations* to produce a set of samples with a desired density. This concept is used extensively in the *Metropolis Light Transport* algorithm referenced in the chapter notes. Suppose we have a random point x_0 in a domain S . Further, suppose for any point x , we have a way to generate random $y \sim p_x$. We use the marginal notation $p_x(y) \equiv p(x \rightarrow y)$ to denote this density function. Now, suppose we let x_1 be a random point in S selected with underlying density $p(x_0 \rightarrow x_1)$. We generate x_2 with density $p(x_1 \rightarrow x_0)$ and so on. In the limit, where we generate an infinite number of samples, it can be proved that the samples will have some underlying density determined by p regardless of the initial point x_0 .

Now, suppose we want to choose p such that the underlying density of samples to which we converge is proportional to a function $f(x)$ where f is a nonnegative function with domain S . Further, suppose we can evaluate f , but we have little or no additional knowledge about its properties (such functions are common in graphics). Also, suppose we have the ability to make “transitions” from x_i to x_{i+1} with underlying density function $t(x_i \rightarrow x_{i+1})$. To add flexibility, further suppose we add the potentially nonzero probability that x_i transitions to itself, i.e., $x_{i+1} = x_i$. We phrase this as generating a potential candidate $y \sim t(x_i \rightarrow y)$

同样的想法也可以应用于在球体表面上随机取点。要选择具有均匀方向分布的随机单位向量，我们首先在单位球体中挑选一个随机点，然后通过在相同方向上取单位向量来将该点视为方向向量：

```
done=falsewhile(
notdone)dox=-1+
2r()y=-1+2r()z=-1
+2r()if((l=
x2+y2+z2)<1)则done=true
x=xly=y
zl=z
```

尽管拒绝方法通常编码简单，但很少与分层兼容。由于这个原因，它倾向于收敛得更慢，因此应该主要用于调试，或者在特别困难的情况下使用。

14.4.3 Metropolis

Metropolis方法使用随机突变产生一组具有所需密度的样品。这一概念在章节注释中引用的MetropolisLightTransport算法中被广泛使用。假设我们在域S中有一个随机点 x_0 。此外，假设对于任何点 x ，我们有一种方法来生成随机 $y \in p_x$ 。我们用边际符号 $p_x(y) \div p(x \rightarrow y)$ 来表示这个密度函数。现在，假设我们让 x_1 是 S 中的随机点，选择底层密度 $p(x_0 \rightarrow x_1)$ 。我们以密度 $p(x_1 \rightarrow x_0)$ 等生成 x_2 。在极限中，我们生成无限数量的样本，可以证明样本将具有由 p 确定的一些底层密度，而不管初始点 x_0 。

现在，假设我们要选择 p ，使得我们收敛的样本的底层密度与函数 $f(x)$ 成正比，其中 f 是域 S 的非负函数。此外，假设我们可以评估 f ，但我们很少或没有关于其属性的额外知识（此类函数在图形中很常见）。此外，假设我们有能力用底层密度函数 $t(x_i \rightarrow x_{i+1})$ 从 x_i 到 x_{i+1} 进行“过渡”。为了增加灵活性，进一步假设我们添加 x_i 转换到自身的潜在非零概率，即 $x_{i+1} = x_i$ 。我们将其称为生成潜在候选 $y \in t(x_i \rightarrow y)$

and “accepting” this candidate (i.e., $x_{i+1} = y$) with probability $a(x_i \rightarrow y)$ and rejecting it (i.e., $x_{i+1} = x_i$) with probability $1 - a(x_i \rightarrow y)$. Note that the sequence x_0, x_1, x_2, \dots will be a random set, but there will be some correlation among samples. They will still be suitable for Monte Carlo integration or density estimation, but analyzing the variance of those estimates is much more challenging.

Now, suppose we are given a transition function $t(x \rightarrow y)$ and a function $f(x)$ of which we want to mimic the distribution, can we use $a(y \rightarrow x)$ such that the points are distributed in the shape of f ? Or more precisely,

$$\{x_0, x_1, x_2, \dots\} \sim \frac{f}{\int_s f}.$$

It turns out this can be forced by making sure the x_i are *stationary* in some strong sense. If you visualize a huge collection of sample points x , you want the “flow” between two points to be the same in each direction. If we assume the density of points near x and y are proportional to $f(x)$ and $f(y)$, respectively, then the flow in the two directions should be the same:

$$\begin{aligned}\text{flow}(x \rightarrow y) &= kf(x)t(x \rightarrow y)a(x \rightarrow y), \\ \text{flow}(y \rightarrow x) &= kf(y)t(y \rightarrow x)a(y \rightarrow x),\end{aligned}$$

where k is some positive constant. Setting these two flows constant gives a constraint on a :

$$\frac{a(y \rightarrow x)}{a(x \rightarrow y)} = \frac{f(x)t(x \rightarrow y)}{f(y)t(y \rightarrow x)}.$$

Thus, if either $a(y \rightarrow x)$ or $a(x \rightarrow y)$ is known, so is the other. Making them larger improves the chance of acceptance, so the usual technique is to set the larger of the two to 1.

A difficulty in using the Metropolis sample generation technique is that it is hard to estimate how many points are needed before the set of points is “good.” Things are accelerated if the first n points are discarded, although choosing n wisely is nontrivial.

14.4.4 Example: Choosing Random Lines in the Square

As an example of the full process of designing a sampling strategy, consider the problem of finding random lines that intersect the unit square $[0, 1]^2$. We want this process to be fair; that is, we would like the lines to be uniformly distributed within the square. Intuitively, we can see that there is some subtlety to this problem; there are “more” lines at an oblique angle than in horizontal or vertical directions. This is because the cross section of the square is not uniform.

并以概率 $a(x_i \rightarrow y)$ “接受”这个候选人（即， $x_{i+1}=y$ ），并以概率 $1 - a(x_i \rightarrow y)$ 重新将其（即， $x_{i+1}=x_i$ ）。注意，序列 $x_0 x_1 x_2 \dots$ 将是一个随机集合，但 samples 之间会有一些相关性。它们仍然适用于蒙特卡洛积分或密度估计，但分析这些估计的方差更具挑战性。

现在，假设我们得到一个过渡函数 $t(x \rightarrow y)$ 和一个我们想要模仿分布的函数 $f(x)$ ，我们可以使用 $a(y \rightarrow x)$ 使得点以 f 的形状分布吗？或者更确切地说

$$\{x_0, x_1, x_2, \dots\} \sim \frac{f}{\int_s f}.$$

事实证明，这可以通过确保 x_i 在某种强烈意义上是静止的来强制的。如果您可视化样本点 x 的巨大集合，则希望两个点之间的“流动”在每个方向上都是相同的。如果我们假设 x 和 y 附近的点的密度分别与 $f(x)$ 和 $f(y)$ 成正比，那么两个方向的流量应该是相同的：

$$\begin{aligned}\text{flow}(x \rightarrow y) &= kf(x)t(x \rightarrow y)a(x \rightarrow y), \\ \text{flow}(y \rightarrow x) &= kf(y)t(y \rightarrow x)a(y \rightarrow x),\end{aligned}$$

其中 k 是一些正常数。设置这两个流常数给出了一个 constraint 上：

$$\frac{a(y \rightarrow x)}{a(x \rightarrow y)} = \frac{f(x)t(x \rightarrow y)}{f(y)t(y \rightarrow x)}.$$

因此，如果已知 $a(y \rightarrow x)$ 或 $a(x \rightarrow y)$ ，则另一个也是如此。使它们变大会提高接受的机会，因此通常的技术是将两者中的较大者设置为 1。

使用 Metropolis 样本生成技术的一个困难是，很难估计在点集“好”之前需要多少点。“如果丢弃前 n 个点，事情就会加速，尽管明智地选择 n 并不重要。

14.4.4 示例：在正方形中选择随机线

作为设计采样策略的全过程的一个例子，考虑找到与单位 $[0, 1]^2$ 相交的随机线的问题。我们希望这个过程是公平的，也就是说，我们希望线条在广场内均匀分布。直观地说，我们可以看到这个问题有一些微妙之处：在斜角上比在水平或垂直方向上有“更多”的线条。这是因为正方形的横截面不均匀。

Our first goal is to find a function-inversion method, if one exists, and then to fall back on rejection or Metropolis if that fails. This is because we would like to have stratified samples in line space. We try using normal coordinates first, because the problem of choosing random lines in the square is just the problem of finding uniform random points in whatever part of (r, θ) space corresponds to lines in the square.

Consider the region where $-\pi/2 < \theta < 0$. What values of r correspond to lines that hit the square? For those angles, $r < \cos \theta$ are all the lines that hit the square as shown in Figure 14.8. Similar reasoning in the other four quadrants finds the region in (r, θ) space that must be sampled, as shown in Figure 14.9. The equation of the boundary of that region $r_{\max}(\theta)$ is

$$r_{\max}(\theta) = \begin{cases} 0 & \text{if } \theta \in [-\pi, -\frac{\pi}{2}], \\ \cos \theta & \text{if } \theta \in [-\frac{\pi}{2}, 0], \\ \sqrt{2} \cos(\theta - \frac{\pi}{4}) & \text{if } \theta \in [0, \frac{\pi}{2}], \\ \sin \theta & \text{if } \theta \in [\frac{\pi}{2}, \pi]. \end{cases}$$

Because the region under $r_{\max}(\theta)$ is a simple function bounded below by $r = 0$, we can sample it by first choosing θ according to the density function:

$$p(\theta) = \frac{r_{\max}(\theta)}{\int_{-\pi}^{\pi} r_{\max}(\theta) d\theta}.$$

The denominator here is 4. Now, we can compute the cumulative probability distribution function:

$$P(\theta) = \begin{cases} 0 & \text{if } \theta \in [-\pi, -\frac{\pi}{2}], \\ (1 + \sin \theta)/4 & \text{if } \theta \in [-\frac{\pi}{2}, 0], \\ (1 + \frac{\sqrt{2}}{2} \sin(\theta - \frac{\pi}{4}))/2 & \text{if } \theta \in [0, \frac{\pi}{2}], \\ (3 - \cos \theta)/4 & \text{if } \theta \in [\frac{\pi}{2}, \pi]. \end{cases}$$

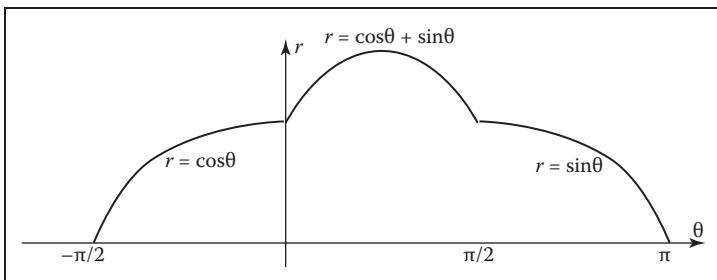


Figure 14.9. The maximum radius for lines hitting the unit square $[0,1]^2$ as a function of θ .

我们的第一个目标是找到一个函数反转方法，如果存在，然后如果失败，则回退拒绝或Metropolis。这是因为我们希望在线空间中分层样本。我们首先尝试使用法线坐标，因为在正方形中选择随机线的问题只是在 (r, θ) 空间的任何部分对应于正方形中的线中找到均匀随机点的问题。

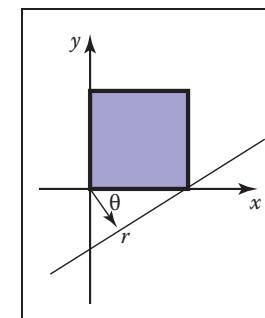


Figure 14.8. The largest distance r corresponds to a line hitting the square for $\theta \in [-\pi/2, 0]$. Because the square has sidelength one, $r = \cos \theta$.

考虑 $\pi/2 < \theta < 0$ 的区域。 R 的值对应于击中正方形的线？对于这些角度， $r < \cos \theta$ 是击中正方形的所有线，如图14.8所示。其他四个象限中的类似推理发现必须采样的 (r, θ) 空间中的区域，如图14.9所示。该区域的边界的方程 $r_{\max}(\theta)$ 是

$$r_{\max}(\theta) = \begin{cases} 0 & \text{if } \theta \in [-\pi, -\frac{\pi}{2}], \\ \cos \theta & \text{if } \theta \in [-\frac{\pi}{2}, 0], \\ \sqrt{2} \cos(\theta - \frac{\pi}{4}) & \text{if } \theta \in [0, \frac{\pi}{2}], \\ \sin \theta & \text{if } \theta \in [\frac{\pi}{2}, \pi]. \end{cases}$$

因为 $r_{\max}(\theta)$ 下的区域是一个以 $r=0$ 为界的简单函数，所以我们可以通过对密度函数选择 θ 来对其进行采样：

$$p(\theta) = \frac{r_{\max}(\theta)}{\int_{-\pi}^{\pi} r_{\max}(\theta) d\theta}.$$

这里的分母是4。现在，我们可以计算累积概率分布函数：

$$P(\theta) = \begin{cases} 0 & \text{if } \theta \in [-\pi, -\frac{\pi}{2}], \\ (1 + \sin \theta)/4 & \text{if } \theta \in [-\frac{\pi}{2}, 0], \\ (1 + \frac{\sqrt{2}}{2} \sin(\theta - \frac{\pi}{4}))/2 & \text{if } \theta \in [0, \frac{\pi}{2}], \\ (3 - \cos \theta)/4 & \text{if } \theta \in [\frac{\pi}{2}, \pi]. \end{cases}$$

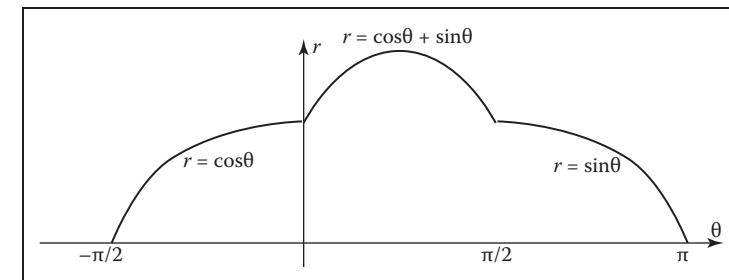
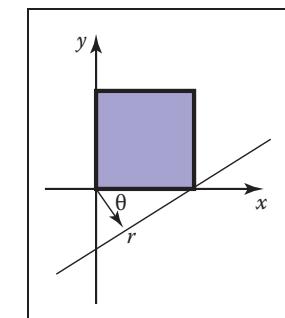


图14.9. 击中单位平方 $[0,1]^2$ 的线的最大半径作为 θ 的函数。



最大的距离 r 对应于一条线击中了 $\theta \in [\pi/2, 0]$ 的正方形。因为正方形有边长一， $r = \cos \theta$ 。

We can invert this by manipulating $\xi_1 = P(\theta)$ into the form $\theta = g(\xi_1)$. This yields

$$\theta = \begin{cases} \arcsin(4\xi_1 - 1) & \text{if } \xi_1 < \frac{1}{4}, \\ \arcsin\left(\frac{\sqrt{2}}{2}(2\xi_1 - 1)\right) + \frac{\pi}{4} & \text{if } \xi_1 \in [\frac{1}{4}, \frac{3}{4}], \\ \arccos(3 - 4\xi_1) & \text{if } \xi_1 > \frac{3}{4}. \end{cases}$$

Once we have θ , then r is simply:

$$r = \xi_2 r_{\max}(\theta).$$

As discussed earlier, there are many parameterizations of the line, and each has an associated “fair” measure. We can generate random lines in any of these spaces as well. For example, in slope-intercept space, the region that hits the square is shown in Figure 14.10. By similar reasoning to the normal space, the density function for the slope is

$$p(m) = \frac{1 + |m|}{4}$$

with respect to the differential measure

$$d\mu = \frac{dm}{(1 + m^2)^{\frac{3}{2}}}.$$

This gives rise to the cumulative distribution function:

$$P(m) = \begin{cases} \frac{1}{4} + \frac{m+1}{4\sqrt{1+m^2}} & \text{if } m < 0, \\ \frac{3}{4} + \frac{m-1}{4\sqrt{1+m^2}} & \text{if } m \geq 0. \end{cases}$$

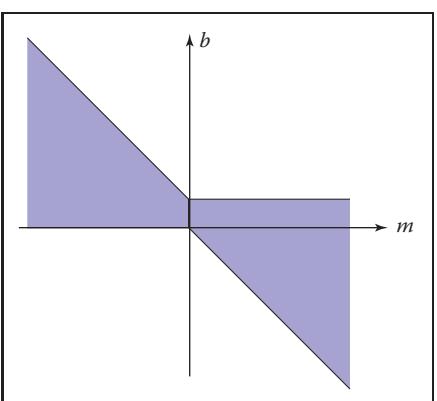


Figure 14.10. The region of (m,b) space that contains lines that intersect the unit square $[0,1]^2$.

我们可以通过将 $\xi_1 = p(\theta)$ 操纵成 $\theta = g(\xi_1)$ 的形式来反转这个。这会产生

$$\theta = \begin{cases} \arcsin(4\xi_1 - 1) & \text{if } \xi_1 < \frac{1}{4}, \\ \arcsin\left(\frac{\sqrt{2}}{2}(2\xi_1 - 1)\right) + \frac{\pi}{4} & \text{if } \xi_1 \in [\frac{1}{4}, \frac{3}{4}], \\ \arccos(3 - 4\xi_1) & \text{if } \xi_1 > \frac{3}{4}. \end{cases}$$

一旦我们有了 θ , 那么 r 就是简单的:

$$r = \xi_2 r_{\max}(\theta).$$

如前所述, 线有许多参数化, 每个参数化都有一个相关的“公平”度量。我们也可以在任何这些空间中生成随机线。例如, 在斜率-截距空间中, 击中正方形的区域如图14.10所示。通过与正常空间相似的推理, 斜率的密度函数为

$$p(m) = \frac{1 + |m|}{4}$$

相对于差分度量

$$d\mu = \frac{dm}{(1 + m^2)^{\frac{3}{2}}}.$$

这就产生了累积分布函数:

$$P(m) = \begin{cases} \frac{1}{4} + \frac{m+1}{4\sqrt{1+m^2}} & \text{if } m < 0, \\ \frac{3}{4} + \frac{m-1}{4\sqrt{1+m^2}} & \text{if } m \geq 0. \end{cases}$$

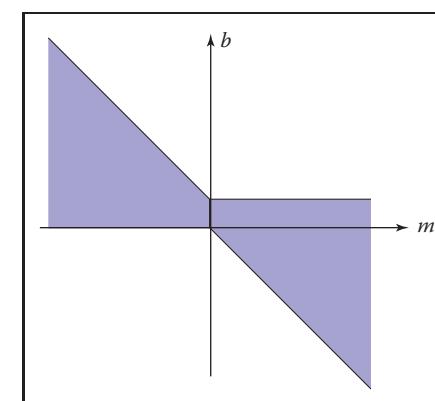


图14.10。 包含与单位正方形 $[0,1]^2$ 相交的线的 (m, b) 空间的区域。

These can be inverted by solving two quadratic equations. Given an m generated using ξ_1 , we then have

$$b = \begin{cases} (1-m)\xi_2 & \text{if } \xi < \frac{1}{2}, \\ -m + (1+m)\xi_2 & \text{otherwise.} \end{cases}$$

This is not a better way than using normal coordinates; it is just an alternative way.

Frequently Asked Questions

- This chapter discussed probability but not statistics. What is the distinction?

Probability is the study of how likely an event is. Statistics infers characteristics of large, but finite, populations of random variables. In that sense, statistics could be viewed as a specific type of applied probability.

- Is Metropolis sampling the same as the Metropolis Light Transport Algorithm?

No. The *Metropolis Light Transport* (Veach & Guibas, 1997) algorithm uses Metropolis sampling as part of its procedure, but it is specifically for rendering, and it has other steps as well.

Notes

The classic reference for geometric probability is *Geometric Probability* (Solomon, 1978). Another method for picking random edges in a square is given in *Random-Edge Discrepancy of Supersampling Patterns* (Dobkin & Mitchell, 1993). More information on quasi-Monte Carlo methods for graphics can be found in *Efficient Multidimensional Sampling* (Kollig & Keller, 2002). Three classic and very readable books on Monte Carlo methods are *Monte Carlo Methods* (Hammersley & Handscomb, 1964), *Monte Carlo Methods, Basics* (Kalos & Whitlock, 1986), and *The Monte Carlo Method* (Sobel, Stone, & Messer, 1975).

Exercises

1. What is the average value of the function xyz in the unit cube $(x, y, z) \in [0, 1]^3$?

这些可以通过求解两个二次方程来反转。给定使用 ξ_1 生成的 m , 我们然后有

$$b = \begin{cases} (1-m)\xi_2 & \text{if } \xi < \frac{1}{2}, \\ -m + (1+m)\xi_2 & \text{otherwise.} \end{cases}$$

这不是比使用正常坐标更好的方法; 它只是一种替代方法。

常见问题

- 本章讨论的是概率，而不是统计。什么是

概率是研究事件的可能性。统计推断随机变量的大型但有限的群体的特征。从这个意义上说，统计数字可以被视为一种特定类型的应用概率。

- *Metropolissampling是否与MetropolisLightTransport算法相同？

非也。MetropolisLightTransport (Veach & Guibas, 1997) 算法使用Metropolissampling作为其过程的一部分，但它专门用于渲染，并且它还有其他步骤。

Notes

几何概率的经典参考是几何概率 (Solomon, 1978)。在超采样模式的随机边缘差异中给出了在正方形中拾取随机边缘的另一种方法 (Dobkin & Mitchell, 1993)。关于图形的准蒙特卡洛方法的更多信息可以在 Efficient Multidimensional Sampling (Kollig & Keller, 2002) 中找到。关于蒙特卡洛方法的三本经典且非常可读的书籍是蒙特卡洛方法 (Hammersley & Handscomb, 1964)，蒙特卡洛方法，基础 (Kalos & Whitlock, 1986) 和蒙特卡洛方法 (Sobel, Stone & Messer, 1975)。

Exercises

1. 单位立方体 (x, y, z) 中的函数 xyz 的平均值 $\in [0, 1]^3$ 是多少？



2. What is the average value of r on the unit-radius disk: $(r, \phi) \in [0, 1] \times [0, 2\pi]$?
3. Show that the uniform mapping of canonical random points (ξ_1, ξ_2) to the barycentric coordinates of any triangle is: $\beta = 1 - \sqrt{1 - \xi_1}$, and $\gamma = (1 - u)\xi_2$.
4. What is the average length of a line inside the unit square? Verify your answer by generating ten million random lines in the unit square and averaging their lengths.
5. What is the average length of a line inside the unit cube? Verify your answer by generating ten million random lines in the unit cube and averaging their lengths.
6. Show from the definition of variance that $V(x) = E(x^2) - [E(x)]^2$.



2. 单位半径盘上 r 的平均值是多少: $(r, \phi) \in [0, 1] \times [0, 2\pi]$?
3. 表明规范随机点 $(\approx 1, \approx 2)$ 到任何三角形的重心坐标的均匀映射为: $\beta = 1 - \sqrt{1 - \xi_1}$, $\gamma = (1 - u)\xi_2$ 。
4. 单位正方形内一条线的平均长度是多少? 通过在单位平方中生成一千万条随机线并平均它们的长度来验证您的答案。
5. 单位立方体内一条线的平均长度是多少? 通过在单位立方体中生成一千万条随机线并平均它们的长度来验证您的答案。
6. 从方差的定义表明, $V(x) = E(x^2) - [E(x)]^2$ 。



Curves

15.1 Curves

Intuitively, think of a *curve* as something you can draw with a pen. The curve is the set of points that the pen traces over an interval of time. While we usually think of a pen writing on paper (e.g., a curve that is in a 2D space), the pen could move in 3D to generate a *space curve*, or you could imagine the pen moving in some other kind of space.

Mathematically, definitions of curve can be seen in at least two ways:

1. the continuous image of some interval in an n -dimensional space;
2. a continuous map from a one-dimensional space to an n -dimensional space.

Both of these definitions start with the idea of an interval range (the time over which the pen traces the curve). However, there is a significant difference: in the first definition, the curve is the set of points the pen traces (the image), while in the second definition, the curve is the mapping between time and that set of points. For this chapter, we use the first definition.

A curve is an infinitely large set of points. The points in a curve have the property that any point has two neighbors, except for a small number of points that have one neighbor (these are the endpoints). Some curves have no endpoints, either because they are infinite (like a line) or they are *closed* (loop around and connect to themselves).



Curves

15.1 Curves

直观地说，把曲线想象成你可以用钢笔画的东西。曲线是笔在一段时间内跟踪的一组点。虽然我们通常认为笔在纸上书写（例如，在2D空间中的曲线），但笔可以在3D中移动以生成空间曲线，或者您可以想象笔在其他类型的空间中移动。

在数学上，曲线的定义至少可以通过两种方式看到：

1. n 维空间中某个区间的连续图像；
2. 从一维空间到 n 维空间的连续映射。

这两个定义都是从间隔范围（笔跟踪曲线的时间）的概念开始的。然而，有一个显着的区别：在第一个定义中，曲线是笔迹线（图像）的一组点，而在第二个定义中，曲线是时间和该组点之间的映射。对于本章，我们使用第一个定义。

曲线是无限大的一组点。曲线中的点具有任何点都有两个邻居的属性，除了少数有一个邻居的点（这些是端点）。有些曲线没有端点，要么是因为它们是无限的（就像一条线），要么是闭合的（循环并连接到它们自己）。

Because the “pen” of the curve is thin (infinitesimally), it is difficult to create filled regions. While space-filling curves are possible (by having them fold over themselves infinitely many times), we do not consider such mathematical oddities here. Generally, we think of curves as the outlines of things, not the “insides.”

The problem that we need to address is how to specify a curve—to give a name or representation to a curve so that we can represent it on a computer. For some curves, the problem of naming them is easy since they have known shapes: line segments, circles, elliptical arcs, etc. A general curve that does not have a “named” shape is sometimes called a *free-form* curve. Because a free-form curve can take on just about any shape, they are much harder to specify.

There are three main ways to specify curves mathematically:

1. **Implicit** curve representations define the set of points on a curve by giving a procedure that can test to see if a point is on the curve. Usually, an implicit curve representation is defined by an *implicit function* of the form

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

so that the curve is the set of points for which this equation is true. Note that the implicit function f is a scalar function (it returns a single real number).

2. **Parametric** curve representations provide a mapping from a *free parameter* to the set of points on the curve. That is, this free parameter provides an index to the points on the curve. The parametric form of a curve is a function that assigns positions to values of the free parameter. Intuitively, if you think of a curve as something you can draw with a pen on a piece of paper, the free parameter is time, ranging over the interval from the time that we began drawing the curve to the time that we finish. The *parametric function* of this curve tells us where the pen is at any instant in time:

$$(x, y) = \mathbf{f}(t).$$

Note that the parametric function is a vector-valued function. This example is a 2D curve, so the output of the function is a 2-vector; in 3D it would be a 3-vector.

3. **Generative or procedural** curve representations provide procedures that can generate the points on the curve that do not fall into the first two categories. Examples of generative curve descriptions include subdivision schemes and fractals.

Remember that a curve is a set of points. These representations give us ways to specify those sets. Any curve has many possible representations. For this

由于曲线的“笔”很薄（无限），因此很难创建填充区域。虽然空间填充曲线是可能的（通过让它们无限多次折叠），但我们在这里不考虑这种数学上的奇怪之处。一般来说，我们认为曲线是事物的轮廓，而不是“内部”。

我们需要解决的问题是如何指定曲线—为曲线命名或表示，以便我们可以在计算机上表示它。对于某些曲线，命名它们的问题很容易，因为它们具有已知的形状：线段，圆形，椭圆弧等。不具有“命名”形状的一般曲线有时称为自由形状曲线。因为自由曲线可以呈现几乎任何形状，所以它们更难指定。

以数学方式指定曲线有三种主要方法：

1. 隐式曲线表示通过给出一个可以测试曲线上的点是否进入的过程来定义曲线上的点集。通常，隐式曲线表示由形式的隐式函数定义

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

因此，曲线是该方程为真的点集。请注意，隐式函数 f 是一个标量函数（它返回单个实数）。

2. 参数曲线表示提供了从自由参数到曲线上点集的映射。也就是说，这个自由参数为曲线上点提供了一个索引。曲线的参数形式是为自由参数的值分配位置的函数。直观地说，如果你认为一条曲线是你可以用笔在一张纸上画的东西，自由参数是时间，从我们开始画曲线到我们完成的时间的间隔。这条曲线的参数函数告诉我们笔在任何时刻的位置：

$$(x, y) = \mathbf{f}(t).$$

请注意，参数函数是矢量值函数。这个例子是一个2D曲线，所以函数的输出是一个2向量；在3D中，它将是一个3向量。

3. 生成或程序性曲线表示提供了可以生成曲线上不属于前两类的点的过程。生成曲线描述的示例包括细分方案和分形。

请记住，曲线是一组点。这些表示为我们提供了指定这些集合的方法。任何曲线都有许多可能的表示。为了这个



reason, mathematicians typically are careful to distinguish between a curve and its representations. In computer graphics we are often sloppy, since we usually only refer to the representation, not the actual curve itself. So when someone says “an implicit curve,” they are either referring to the curve that is represented by some implicit function or to the implicit function that is one of the representations of some curve. Such distinctions are not usually important, unless we need to consider different representations of the same curve. We will consider different curve representations in this chapter, so we will be more careful. When we use a term like “polynomial curve,” we will mean the curve that can be represented by the polynomial.

By the definition given at the beginning of the chapter, for something to be a curve it must have a parametric representation. However, many curves have other representations. For example, a circle in 2D with its center at the origin and radius equal to 1 can be written in implicit form as

$$f(x, y) = x^2 + y^2 - 1 = 0,$$

or in parametric form as

$$(x, y) = \mathbf{f}(t) = (\cos t, \sin t), \quad t \in [0, 2\pi].$$

The parametric form need not be the most convenient representation for a given curve. In fact, it is possible to have curves with simple implicit or generative representations for which it is difficult to find a parametric representation.

Different representations of curves have advantages and disadvantages. For example, parametric curves are much easier to draw, because we can sample the free parameter. Generally, parametric forms are the most commonly used in computer graphics since they are easier to work with. Our focus will be on parametric representations of curves.

15.1.1 Parameterizations and Reparameterizations

A *parametric curve* refers to the curve that is given by a specific parametric function over some particular interval. To be more precise, a parametric curve has a given function that is a mapping from an interval of the parameters. It is often convenient to have the parameter run over the unit interval from 0 to 1. When the free parameter varies over the unit interval, we often denote the parameter as u .

If we view the parametric curve to be a line drawn with a pen, we can consider $u = 0$ as the time when the pen is first set down on the paper and the unit of time to be the amount of time it takes to draw the curve ($u = 1$ is the end of the curve).



原因，数学家通常会小心区分曲线及其表示。在计算机图形学中，我们经常马虎，因为我们通常只指表示，而不是实际的曲线本身。因此，当有人说“隐式曲线”时，他们要么是指由某种隐式函数表示的曲线，要么是指作为某种曲线的表示之一的隐式函数。这种区别通常并不重要，除非我们需要考虑相同曲线的不同表示。我们将在本章中考虑不同的曲线表示，因此我们会更加小心。当我们使用像“多项式曲线”这样的术语时，我们将指可以由多项式表示的曲线。

根据本章开头给出的定义，要使某物成为曲线，它必须具有参数表示。然而，许多曲线具有其他表示。例如，2D中的圆心在原点且半径等于1的圆可以隐式形式写为

$$f(x, y) = x^2 + y^2 - 1 = 0,$$

或以参数形式为

$$(x, y) = \mathbf{f}(t) = (\cos t, \sin t), \quad t \in [0, 2\pi].$$

参数形式不一定是给定曲线最方便的表示形式。实际上，可以使用具有简单隐式或生成表示的曲线，对于这些曲线很难找到参数表示。

曲线的不同表示有优点和缺点。例如，参数化曲线更容易绘制，因为我们可以对自由参数进行采样。通常，参数化表单是computer图形中最常用的，因为它们更易于使用。我们的重点将是曲线的参数表示。

15.1.1 Parameterizations and Reparameterizations

参数曲线是指由特定参数函数在某个特定区间内给出的曲线。更准确地说，参数曲线具有给定的函数，该函数是参数间隔的映射。让参数在从0到1的单位间隔内运行通常很方便。当自由参数在单位间隔内变化时，我们通常将参数表示为 u 。如果我们把参数曲线看作是用笔画的一条线，我们可以把 $u=0$ 看作是笔第一次放在纸上的时间，而时间单位是绘制曲线所需的时间（ $u=1$ 是曲线的结束）。

The curve can be specified by a function that maps time (in these unit coordinates) to positions. Basically, the specification of the curve is a function that can answer the question, "Where is the pen at time u ?"

If we are given a function $\mathbf{f}(t)$ that specifies a curve over interval $[a, b]$, we can easily define a new function $\mathbf{f}_2(u)$ that specifies the same curve over the unit interval. We can first define

$$g(u) = a + (b - a)u,$$

and then

$$\mathbf{f}_2(u) = \mathbf{f}(g(u)).$$

The two functions, \mathbf{f} and \mathbf{f}_2 both represent the same curve; however, they provide different *parameterizations* of the curve. The process of creating a new parameterization for an existing curve is called *reparameterization*, and the mapping from old parameters to the new ones (g , in this example) is called the *reparameterization function*.

If we have defined a curve by some parameterization, infinitely many others exist (because we can always reparameterize). Being able to have multiple parameterizations of a curve is useful, because it allows us to create parameterizations that are convenient. However, it can also be problematic, because it makes it difficult to compare two functions to see if they represent the same curve.

The essence of this problem is more general: the existence of the free parameter (or the element of time) adds an invisible, potentially unknown element to our representation of the curves. When we look at the curve after it is drawn, we don't necessarily know the timing. The pen might have moved at a constant speed over the entire time interval, or it might have started slowly and sped up. For example, while $u = 0.5$ is halfway through the parameter space, it may not be halfway along the curve if the motion of the pen starts slowly and speeds up at the end. Consider the following representations of a very simple curve:

$$\begin{aligned} (x, y) &= \mathbf{f}(u) = (u, u), \\ (x, y) &= \mathbf{f}(u) = (u^2, u^2), \\ (x, y) &= \mathbf{f}(u) = (u^5, u^5). \end{aligned}$$

All three functions represent the same curve on the unit interval; however when u is not 0 or 1, $\mathbf{f}(u)$ refers to a different point depending on the representation of the curve.

If we are given a parameterization of a curve, we can use it directly as our specification of the curve, or we can develop a more convenient parameterization. Usually, the *natural parameterization* is created in a way that is convenient (or

曲线可以通过将时间（在这些单位坐标中）映射到位置的函数来指定。基本上，曲线的规范是一个函数，可以回答这个问题，“时间u的笔在哪里？”

如果给出一个函数 $f(t)$ 来指定区间 $[a, b]$ 上的曲线，我们可以很容易地定义一个新函数 $f_2(u)$ 来指定单位区间上的相同曲线。我们可以先定义

$$g(u) = a + (b - a)u,$$

然后

$$\mathbf{f}_2(u) = \mathbf{f}(g(u)).$$

两个函数 f 和 f_2 都表示同一条曲线；但是，它们提供了曲线的不同参数化。为现有曲线创建新参数化的过程称为重新参数化，从旧参数到新参数（在本例中为 g ）的映射称为重新参数化函数。

如果我们通过一些参数化定义了一条曲线，那么存在无限多的其他曲线（因为我们总是可以重新参数化）。能够有一条曲线的多个参数化是有用的，因为它允许我们创建方便的参数化。然而，它也可能是有问题的，因为它使得很难比较两个函数以查看它们是否表示相同的曲线。

这个问题的本质更普遍：自由参数（或时间元素）的存在为我们对曲线的表示增加了一个不可见的，可能未知的元素。当我们看曲线绘制后，我们不一定知道时机。笔可能在整个时间间隔内以恒定的速度移动，或者它可能开始缓慢并加速。例如，当 $u=0$ 时， 5 是参数空间的一半，如果笔的运动开始缓慢并在结束时加速，则可能不是曲线的一半。考虑一个非常简单的曲线的以下表示：

$$\begin{aligned} (x, y) &= \mathbf{f}(u) = (u, u), \\ (x, y) &= \mathbf{f}(u) = (u^2, u^2), \\ (x, y) &= \mathbf{f}(u) = (u^5, u^5). \end{aligned}$$

所有三个函数在单位间隔上表示相同的曲线；然而，当 u 不是0或1时， $\mathbf{f}(u)$ 指的是不同的点，具体取决于曲线的表示。

如果给我们一个曲线的参数化，我们可以直接使用它作为我们对曲线的规范，或者我们可以开发一个更方便的参数化。通常，自然参数化是以方便的方式创建的（或



natural) for specifying the curve, so we don't have to know about how the speed changes along the curve.

If we know that the pen moves at a constant velocity, then the values of the free parameters have more meaning. Halfway through parameter space is halfway along the curve. Rather than measuring time, the parameter can be thought to measure length along the curve. Such parameterizations are called *arc-length* parameterizations because they define curves by functions that map from the distance along the curve (known as the arc length) to positions. We often use the variable s to denote an arc-length parameter.

Technically, a parameterization is an arc-length parameterization if the magnitude of its *tangent* (that is, the derivative of the parameterization with respect to the parameter) has constant magnitude. Expressed as an equation,

$$\left| \frac{d\mathbf{f}(s)}{ds} \right|^2 = c.$$

Computing the length along a curve can be tricky. In general, it is defined by the integral of the magnitude of the derivative (intuitively, the magnitude of the derivative is the velocity of the pen as it moves along the curve). So, given a value for the parameter v , you can compute s (the arc-length distance along the curve from the point $\mathbf{f}(0)$ to the point $\mathbf{f}(v)$) as

$$s = \int_0^v \left| \frac{d\mathbf{f}(t)}{dt} \right|^2 dt, \quad (15.1)$$

where $\mathbf{f}(t)$ is a function that defines the curve with a natural parameterization.

Using the arc-length parameterization requires being able to solve Equation (15.1) for t , given s . For many of the kinds of curves we examine, it cannot be done in a closed-form (simple) manner and must be done numerically.

Generally, we use the variable u to denote free parameters that range over the unit interval, s to denote arc-length free parameters, and t to represent parameters that aren't one of the other two.

15.1.2 Piecewise Parametric Representations

For some curves, defining a parametric function that represents their shape is easy. For example, lines, circles, and ellipses all have simple functions that define the points they contain in terms of a parameter. For many curves, finding a function that specifies their shape can be hard. The main strategy that we use to create complex curves is divide-and-conquer: we break the curve into a number of simpler smaller pieces, each of which has a simple description.



自然) 用于指定曲线，因此我们不必知道速度如何沿着曲线变化。

如果我们知道笔以恒定的速度移动，那么自由参数的值就有更多的意义。参数空间的中途是沿着曲线的中途。而不是测量时间，参数可以被认为是测量沿着曲线的长度。这种参数化称为弧长参数化，因为它们通过函数定义曲线，这些函数从沿着曲线的偏差（称为弧长）映射到位置。我们经常使用变量 s 来表示弧长参数。

从技术上讲，参数化是弧长参数化，如果其切线的磁导（即参数化相对于参数的导数）具有恒定的幅度。表示为方程

$$\left| \frac{d\mathbf{f}(s)}{ds} \right|^2 = c.$$

计算曲线的长度可能很棘手。一般来说，它由导数的量值的积分定义（直观地说，导数的量值是笔沿曲线移动时的速度）。因此，给定参数 v 的值，您可以计算 s （沿曲线从点 $\mathbf{f}(0)$ 到点 $\mathbf{f}(v)$ 的弧长距离）为

$$s = \int_0^v \left| \frac{d\mathbf{f}(t)}{dt} \right|^2 dt, \quad (15.1)$$

其中 $\mathbf{f}(t)$ 是用自然参数化定义曲线的函数。

使用弧长参数化需要能够求解给定 s 的 t 的方程 (15.1)。对于我们检查的许多类型的曲线，它不能以封闭形式（简单）的方式完成，必须以数字方式完成。

通常，我们使用变量 u 表示在单位间隔范围内的自由参数， s 表示弧长自由参数， t 表示不是其他两个参数之一的参数。

15.1.2 分段参数表示

对于某些曲线，定义表示其形状的参数函数很容易。例如，线、圆和椭圆都具有简单的函数，这些函数根据参数定义它们包含的点。对于许多曲线，找到指定其形状的函数可能很难。我们用来创建复杂曲线的主要策略是分而治之：我们将曲线分解成一些更简单的小块，每个小块都有一个简单的描述。

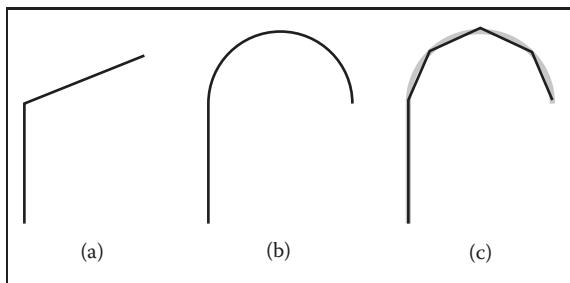


Figure 15.1. (a) A curve that can be easily represented as two lines; (b) a curve that can be easily represented as a line and a circular arc; (c) a curve approximating curve (b) with five line segments.

For example, consider the curves in Figure 15.1. The first two curves are easily specified in terms of two pieces. In the case of the curve in Figure 15.1(b), we need two different kinds of pieces: a line segment and a circle.

To create a parametric representation of a compound curve (like the curve in Figure 15.1(b)), we need to have our parametric function switch between the functions that represent the pieces. If we define our parametric functions over the range $0 \leq u \leq 1$, then the curve in Figures 15.1(a) or (b) might be defined as

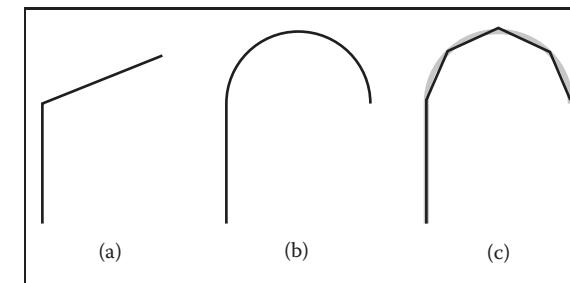
$$\mathbf{f}(u) = \begin{cases} \mathbf{f}_1(2u) & \text{if } u \leq 0.5, \\ \mathbf{f}_2(2u - 1) & \text{if } u > 0.5, \end{cases} \quad (15.2)$$

where \mathbf{f}_1 is a parameterization of the first piece, \mathbf{f}_2 is a parameterization of the second piece, and both of these functions are defined over the unit interval.

We need to be careful in defining the functions \mathbf{f}_1 and \mathbf{f}_2 to make sure that the pieces of the curve fit together. If $\mathbf{f}_1(1) \neq \mathbf{f}_2(0)$, then our curve pieces will not connect and will not form a single continuous curve.

To represent the curve in Figure 15.1(b), we needed to use two different types of pieces: a line segment and a circular arc. For simplicity's sake, we may prefer to use a single type of piece. If we try to represent the curve in Figure 15.1(b) with only one type of piece (line segments), we cannot exactly re-create the curve (unless we use an infinite number of pieces). While the new curve made of line segments (as in Figure 15.1(c)) may not be exactly the same shape as in Figure 15.1(b), it might be close enough for our use. In such a case, we might prefer the simplicity of using the simpler line segment pieces to having a curve that more accurately represents the shape.

Also, notice that as we use an increasing number of pieces, we can get a better approximation. In the limit (using an infinite number of pieces), we can exactly represent the original shape.



(a)可以容易地表示为两条线的曲线；(b)可以容易地表示为一条线和一条圆弧的曲线；(c)用五条线段近似曲线(b)的曲线。

例如，考虑图15.1中的曲线。前两条曲线很容易用两条来指定。在图15.1 (b) 中的曲线的情况下，我们需要两种不同的部分：线段和圆。

要创建复合曲线的参数表示（如图15.1 (b) 中的曲线），我们需要在表示各部分的函数之间切换参数函数。如果我们在 $0 \leq u \leq 1$ 的范围内定义参数函数，那么图15.1 (a) 或 (b) 中的曲线可能被定义为

$$\mathbf{f}(u) = \begin{cases} \mathbf{f}_1(2u) & \text{if } u \leq 0.5, \\ \mathbf{f}_2(2u - 1) & \text{if } u > 0.5, \end{cases} \quad (15.2)$$

其中 \mathbf{f}_1 是第一块的参数化， \mathbf{f}_2 是第二块的参数化，并且这两个函数都是在单位间隔上定义的。

我们在定义函数 \mathbf{f}_1 和 \mathbf{f}_2 时需要小心，以确保曲线拟合在一起。如果 $\mathbf{f}_1(1) = \mathbf{f}_2(0)$ ，那么我们的曲线片将不会连接，也不会形成单个连续曲线。

为了表示图15.1 (b) 中的曲线，我们需要使用两种不同类型的片段：线段和圆弧。为了简单起见，我们可能更喜欢使用单一类型的一块。如果我们试图用一种类型的片段（线段）来表示图15.1 (b) 中的曲线，我们不能完全重新创建曲线（除非我们使用无限数量的片段）。虽然由线段组成的新曲线（如图15.1 (c)）可能与图15.1 (b) 中的形状不完全相同，但它可能足够接近我们使用。在这种情况下，我们可能更喜欢使用更简单的线段片的简单性，而不是具有更准确地表示形状的曲线。

另外，请注意，当我们使用越来越多的碎片时，我们可以得到更好的近似值。在极限（使用无限数量的碎片）中，我们可以精确地表示原始形状。



One advantage to using a piecewise representation is that it allows us to make a tradeoff between

1. how well our represented curve approximates the real shape we are trying to represent;
2. how complicated the pieces that we use are;
3. how many pieces we use.

So, if we are trying to represent a complicated shape, we might decide that a crude approximation is acceptable and use a small number of simple pieces. To improve the approximation, we can choose between using more pieces and using more complicated pieces.

In computer graphics practice, we tend to prefer using relatively simple curve pieces (either line segments, arcs, or polynomial segments).

15.1.3 Splines

Before computers, when draftsmen wanted to draw a smooth curve, one tool they employed was a stiff piece of metal that they would bend into the desired shape for tracing. Because the metal would bend, not fold, it would have a smooth shape. The stiffness meant that the metal would bend as little as possible to make the desired shape. This stiff piece of metal was called a *spline*.

Mathematicians found that they could represent the curves created by a draftsman's spline with piecewise polynomial functions. Initially, they used the term spline to mean a smooth, piecewise polynomial function. More recently, the term spline has been used to describe any piecewise polynomial function. We prefer this latter definition.

For us, a *spline* is a piecewise polynomial function. Such functions are very useful for representing curves.

15.2 Curve Properties

To describe a curve, we need to give some facts about its properties. For "named" curves, the properties are usually specific according to the type of curve. For example, to describe a circle, we might provide its radius and the position of its center. For an ellipse, we might also provide the orientation of its major axis and the ratio of the lengths of the axes. For free-form curves however, we need to have a more general set of properties to describe individual curves.



使用分段表示的一个优点是它允许我们在

1. 我们所表示的曲线与我们试图表示的真实形状有多接近;
2. 我们使用的部件有多复杂;
3. 我们用了多少件。

所以, 如果我们试图表示一个复杂的形状, 我们可能会决定一个粗略的近似是可以接受的, 并使用少量的简单件。为了改进近似, 我们可以选择使用更多的块和使用更复杂的块。

在计算机图形学实践中, 我们倾向于使用相对简单的曲线片段 (线段, 圆弧或多项式段)。

15.1.3 Splines

在计算机之前, 当绘图员想要绘制一条平滑的曲线时, 他们使用的一种工具是一块坚硬的金属, 他们会弯曲成所需的形状进行跟踪。因为金属会弯曲, 而不是折叠, 它会有一个光滑的形状。刚度意味着金属会尽可能少地弯曲以形成所需的形状。这块坚硬的金属被称为花键。

数学家发现他们可以用分段多项式函数表示由草稿人的样条创建的曲线。最初, 他们使用术语样条来表示平滑的分段多项式函数。最近, 术语样条曲线已被用于描述任何分段多项式函数。我们更喜欢后一种定义。

对我们来说, 样条是分段多项式函数。这样的函数对于表示曲线非常有用。

15.2 曲线属性

为了描述一条曲线, 我们需要给出一些关于它的属性的事实。对于"命名"曲线, 属性通常根据曲线的类型而特定。例如, 为了描述一个圆, 我们可以提供它的半径和圆心的位置。对于椭圆, 我们还可以提供其长轴的方向和轴长度的比率。然而, 对于自由曲线, 我们需要有一组更通用的属性来描述单个曲线。

Some properties of curves are attributed to only a single location on the curve, while other properties require knowledge of the whole curve. For an intuition of the difference, imagine that the curve is a train track. If you are standing on the track on a foggy day, you can tell that the track is straight or curved and whether or not you are at an endpoint. These are *local* properties. You cannot tell whether or not the track is a closed curve, or crosses itself, or how long it is. We call this type of property, a *global* property.

The study of local properties of geometric objects (curves and surfaces) is known as *differential geometry*. Technically, to be a differential property, there are some mathematical restrictions about the properties (roughly speaking, in the train-track analogy, you would not be able to have a GPS or a compass). Rather than worry about this distinction, we will use the term *local* property rather than differential property.

Local properties are important tools for describing curves because they do not require knowledge about the whole curve. Local properties include

- continuity,
- position at a specific place on the curve,
- direction at a specific place on the curve,
- curvature (and other derivatives).

Often, we want to specify that a curve includes a particular point. A curve is said to *interpolate* a point if that point is part of the curve. A function f interpolates a value v if there is some value of the parameter u for which $f(t) = v$. We call the place of interpolation, that is the value of t , the *site*.

15.2.1 Continuity

It will be very important to understand the local properties of a curve where two parametric pieces come together. If a curve is defined using an equation like Equation (15.2), then we need to be careful about how the pieces are defined. If $f_1(1) \neq f_2(0)$, then the curve will be “broken”—we would not be able to draw the curve in a continuous stroke of a pen. We call the condition that the curve pieces fit together *continuity conditions* because if they hold, the curve can be drawn as a continuous piece. Because our definition of “curve” at the beginning of the chapter requires a curve to be continuous, technically a “broken curve” is not a curve.

曲线的某些属性仅归因于曲线上的单个位置，而其他属性则需要了解整个曲线。对于差异的直觉，想象曲线是火车轨道。如果您在雾天站在赛道上，您可以判断赛道是直的还是弯曲的，以及您是否处于端点。这些是本地属性。你不能告诉轨道是否是一个封闭的曲线，或交叉本身，或它有多长。我们将这种类型的属性称为全局属性。

几何对象（曲线和曲面）的局部属性的研究被称为微分几何。从技术上讲，作为一个微分属性，有一些关于属性的数学限制（粗略地说，在火车轨道类比中，您将无法拥有GPS或指南针）。而不是担心这种区别，我们将使用术语局部属性而不是差异属性。

局部属性是描述曲线的重要工具，因为它们不需要关于整个曲线的知识。本地属性包括

- continuity,
- *曲线上特定位置的位置
- *曲线上特定位置的方向
- *曲率（和其他衍生物）。

通常，我们希望指定曲线包含特定点。如果点是曲线的一部分，则称曲线为插值点。如果参数 u 的某个值为 $f(t) = v$ ，则函数 f 插值一个值 v 。我们称之为插值的地方，即 t 的值，即站点。

15.2.1 Continuity

理解曲线的局部属性是非常重要的，其中两个参数部分聚集在一起。如果使用方程 (15.2) 这样的方程定义曲线，那么我们需要小心如何定义这些部分。如果 $f_1(1) = f_2(0)$ ，那么曲线将被“打破”—我们将无法在笔的连续笔画中绘制曲线。我们将曲线片拟合在一起的条件称为连续性条件，因为如果它们成立，曲线可以绘制为连续片。因为我们在章节开头对“曲线”的定义要求曲线是连续的，所以从技术上讲，“断曲线”不是曲线。

In addition to the positions, we can also check that the derivatives of the pieces match correctly. If $f'_1(1) \neq f'_2(0)$, then the combined curve will have an abrupt change in its first derivative at the switching point; the first derivative will not be continuous. In general, we say that a curve is C^n continuous if all of its derivatives up to n match across pieces. We denote the position itself as the zeroth derivative, so that the C^0 continuity condition means that the positions of the curve are continuous, and C^1 continuity means that positions and first derivatives are continuous. The definition of curve requires the curve to be C^0 .

An illustration of some continuity conditions is shown in Figure 15.2. A discontinuity in the first derivative (the curve is C^0 but not C^1) is usually noticeable because it displays a sharp corner. A discontinuity in the second derivative is sometimes visually noticeable. Discontinuities in higher derivatives might matter, depending on the application. For example, if the curve represents a motion, an abrupt change in the second derivative is noticeable, so third derivative continuity is often useful. If the curve is going to have a fluid flowing over it (for example, if it is the shape for an airplane wing or boat hull), a discontinuity in the fourth or fifth derivative might cause turbulence.

The type of continuity we have just introduced (C^n) is commonly referred to as *parametric continuity* as it depends on the parameterization of the two curve pieces. If the “speed” of each piece is different, then they will not be continuous. For cases where we care about the shape of the curve, and not its parameterization, we define *geometric continuity* that requires that the derivatives of the curve pieces match when the curves are parameterized equivalently (for example, using an arc-length parameterization). Intuitively, this means that the corresponding derivatives must have the same direction, even if they have different magnitudes.

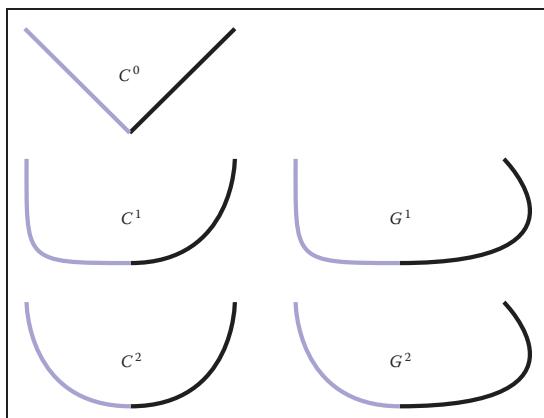


Figure 15.2. An illustration of various types of continuity between two curve segments.

除了位置，我们还可以检查棋子的导数是否正确匹配。如果 $f'(2)(0)$ ，那么组合曲线将在切换点处其一阶导数发生突然变化；一阶导数将不连续。一般来说，我们说一条曲线是 C^n 连续的，如果它的所有导数高达 n 跨块匹配。我们将位置本身表示为零导数，因此 C^0 连续性条件意味着曲线的位置是连续的，而 C^1 连续性意味着位置和一阶导数是连续的。曲线的定义要求曲线为 C^0 。

一些连续性条件的图示如图 15.2 所示。一阶导数中的 C^0 不连续（曲线为 C^0 但不是 C^1 ）通常很明显，因为它显示了一个尖角。二阶导数的不连续性有时在视觉上是明显的。根据应用的不同，高阶导数的不连续性可能会很大。例如，如果曲线表示一个运动，二阶导数的突然变化是明显的，所以三阶导数的一致性通常是有用的。如果曲线将有流体流过它（例如，如果它是飞机机翼或船体的形状），则第四阶或第五阶导数的不连续性可能导致湍流。

我们刚刚介绍的连续性类型 (C^n) 通常被称为参数连续性，因为它取决于两条曲线的参数化。如果每件作品的“速度”不同，那么它们就不会连续。对于我们关心曲线形状而不是其参数化的情况，我们定义几何连续性，要求曲线的导数在曲线等效参数化时匹配（例如，我们使用弧长参数化）。直观地说，这意味着相应的导数必须具有相同的方向，即使它们具有不同的幅度。

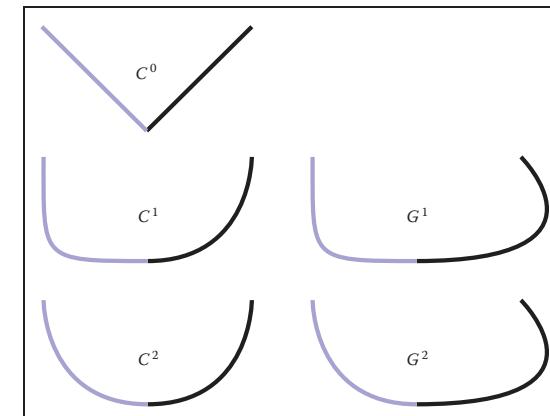


图 15.2. 两个曲线段之间的各种类型的连续性的图示。

So, if the C^1 continuity condition is

$$\mathbf{f}'_1(1) = \mathbf{f}'_2(0),$$

the G^1 continuity condition would be

$$\mathbf{f}'_1(1) = k \mathbf{f}'_2(0),$$

for some value of scalar k . Generally, geometric continuity is less restrictive than parametric continuity. A C^n curve is also G^n except when the parametric derivatives vanish.

15.3 Polynomial Pieces

The most widely used representations of curves in computer graphics is done by piecing together basic elements that are defined by polynomials and called polynomial pieces. For example, a line element is given by a linear polynomial. In Section 15.3.1, we give a formal definition and explain how to put pieces of polynomial together.

15.3.1 Polynomial Notation

Polynomials are functions of the form

$$f(t) = a_0 + a_1 t + a_2 t^2 + \dots + a_n t^n. \quad (15.3)$$

The a_i are called the *coefficients*, and n is called the degree of the polynomial if $a_n \neq 0$. We also write Equation (15.3) in the form

$$\mathbf{f}(t) = \sum_{i=0}^n \mathbf{a}_i t^i. \quad (15.4)$$

We call this the *canonical form* of the polynomial.

We can generalize the canonical form to

$$\mathbf{f}(t) = \sum_{i=0}^n \mathbf{c}_i b_i(t), \quad (15.5)$$

where $b_i(t)$ is a polynomial. We can choose these polynomials in a convenient form for different applications, and we call them *basis functions* or *blending functions* (see Section 15.3.5). In Equation (15.4), the t^i are the $b_i(t)$ of Equation (15.5). If the set of basis functions is chosen correctly, any polynomial of degree $n + 1$ can be represented by an appropriate choice of \mathbf{c} .

所以, 如果C1连续性条件是

$$\mathbf{f}'_1(1) = \mathbf{f}'_2(0),$$

G^1 连续性条件是

$$\mathbf{f}'_1(1) = k \mathbf{f}'_2(0),$$

对于标量k的一些值。通常, 几何连续性比参数连续性的限制性更小。 C^n 曲线也是 G^n , 除非参数导数消失。

15.3 多项式片

计算机图形学中最广泛使用的曲线表示是通过将由多项式定义的基本元素拼凑在一起完成的, 这些元素称为多项式块。例如, 线元素由线性多项式给出。在第15.3.1节中, 我们给出了一个正式的定义, 并解释了如何将多项式片段放在一起。

15.3.1 多项式表示法

多项式是形式的函数

$$f(t) = a_0 + a_1 t + a_2 t^2 + \dots + a_n t^n. \quad (15.3)$$

将 a_i 称为系数, 如果 $a_n=0$, 则将 n 称为多项式的程度。我们也写公式 (15.3) 的形式

$$\mathbf{f}(t) = \sum_{i=0}^n \mathbf{a}_i t^i. \quad (15.4)$$

我们称之为多项式的规范形式。

我们可以将规范形式概括为

$$\mathbf{f}(t) = \sum_{i=0}^n \mathbf{c}_i b_i(t), \quad (15.5)$$

其中 $b_i(t)$ 是多项式。我们可以为不同的应用以方便的形式选择这些多项式, 我们称它们为基函数或混合函数 (参见第15.3.5节)。在等式 (15.4) 中, t^i 是等式 (15.5) 的 $b_i(t)$ 。如果正确选择基函数集, 则任何 $n+1$ 度的多项式都可以用 \mathbf{c} 的适当选择来表示。

The canonical form does not always have convenient coefficients. For practical purposes, throughout this chapter, we will find sets of basis functions such that the coefficients are convenient ways to control the curves represented by the polynomial functions.

To specify a curve embedded in two dimensions, one can either specify two polynomials in t : one for how x varies with t and one for how y varies with t ; or specify a single polynomial where each of the \mathbf{a}_i is a 2D point. An analogous situation exists for any curve in an n -dimensional space.

15.3.2 A Line Segment

To introduce the concepts of piecewise polynomial curve representations, we will discuss line segments. In practice, line segments are so simple that the mathematical derivations will seem excessive. However, by understanding this simple case, things will be easier when we move on to more complicated polynomials.

Consider a line segment that connects point \mathbf{p}_0 to \mathbf{p}_1 . We could write the parametric function over the unit domain for this line segment as

$$\mathbf{f}(u) = (1 - u)\mathbf{p}_0 + u\mathbf{p}_1. \quad (15.6)$$

By writing this in vector form, we have hidden the dimensionality of the points and the fact that we are dealing with each dimension separately. For example, were we working in 2D, we could have created separate equations:

$$\begin{aligned} f_x(u) &= (1 - u)x_0 + ux_1, \\ f_y(u) &= (1 - u)y_0 + uy_1. \end{aligned}$$

The line that we specify is determined by the two endpoints, but from now on we will stick to vector notation since it is cleaner. We will call the vector of control parameters, \mathbf{p} , the *control points*, and each element of \mathbf{p} , a *control point*.

While describing a line segment by the positions of its endpoints is obvious and usually convenient, there are other ways to describe a line segment. For example,

1. the position of the center of the line segment, the orientation, and the length;
2. the position of one endpoint and the position of the second point relative to the first;
3. the position of the middle of the line segment and one endpoint.

规范形式并不总是具有方便的系数。为了实用的目的，在本章中，我们将找到基函数集，以便系数是控制多项式函数表示的曲线的方便方法。

要指定嵌入两个维度的曲线，可以在 t 中指定两个多项式：一个用于 x 如何随 t 变化，一个用于 y 如何随 t 变化；或者指定单个多项式，其中每个 a_i 都是对于 n 维空间中的任何曲线都存在类似的情况。

15.3.2 一个线段

为了介绍分段多项式曲线表示的概念，我们将讨论线段。在实践中，线段非常简单，以至于mathematical推导看起来会过度。但是，通过理解这个简单的情况，当我们转向更复杂的多项式时，事情会更容易。

考虑将点 p_0 连接到 p_1 的线段。我们可以将此线段的参数函数写在单位域上，如下所示

$$\mathbf{f}(u) = (1 - u)\mathbf{p}_0 + u\mathbf{p}_1. \quad (15.6)$$

通过以矢量形式写这一点，我们隐藏了点的维数性以及我们分别处理每个维度的事实。例如，如果我们在2D中工作，我们可以创建单独的方程：

$$\begin{aligned} f_x(u) &= (1 - u)x_0 + ux_1, \\ f_y(u) &= (1 - u)y_0 + uy_1. \end{aligned}$$

我们指定的行由两个端点决定，但从现在开始，我们将坚持矢量表示法，因为它更干净。我们将控制参数 p 的向量称为控制点， p 的每个元素称为控制点。虽然通过端点的位置描述线段是显而易见的并且通常是方便的，但还有其他方法来描述线段。例如

- 1.线段中心的位置、方位和长度;
- 2.一端点的位置和所述第二点相对于所述第一;
- 3.线段中间和一个端点的位置。

It is obvious that given one kind of a description of a line segment, we can switch to another one.

A different way to describe a line segment is using the canonical form of the polynomial (as discussed in Section 15.3.1),

$$\mathbf{f}(u) = \mathbf{a}_0 + u\mathbf{a}_1. \quad (15.7)$$

Any line segment can be represented either by specifying \mathbf{a}_0 and \mathbf{a}_1 or the endpoints (\mathbf{p}_0 and \mathbf{p}_1). It is usually more convenient to specify the endpoints, because we can compute the other parameters from the endpoints.

To write the canonical form as a vector expression, we define a vector \mathbf{u} that is a vector of the powers of u :

$$\mathbf{u} = [1 \ u \ u^2 \ u^3 \ \dots \ u^n],$$

so that Equation (15.4) can be written as

$$\mathbf{f}(u) = \mathbf{u} \cdot \mathbf{a}. \quad (15.8)$$

This vector notation will make transforming between different forms of the curve easier.

Equation (15.8) describes a curve segment by the set of polynomial coefficients for the simple form of the polynomial. We call such a representation the *canonical form*. We will denote the parameters of the canonical form by \mathbf{a} .

While it is mathematically simple, the canonical form is not always the most convenient way to specify curves. For example, we might prefer to specify a line segment by the positions of its endpoints. If we want to define \mathbf{p}_0 to be the beginning of the segment (where the segment is when $u = 0$) and \mathbf{p}_1 to be the end of the line segment (where the line segment is at $u = 1$), we can write

$$\begin{aligned} \mathbf{p}_0 &= \mathbf{f}(0) = [1 \ 0] \cdot [\mathbf{a}_0 \ \mathbf{a}_1], \\ \mathbf{p}_1 &= \mathbf{f}(1) = [1 \ 1] \cdot [\mathbf{a}_0 \ \mathbf{a}_1]. \end{aligned} \quad (15.9)$$

We can solve these equations for \mathbf{a}_0 and \mathbf{a}_1 :

$$\begin{aligned} \mathbf{a}_0 &= \mathbf{p}_0, \\ \mathbf{a}_1 &= \mathbf{p}_1 - \mathbf{p}_0. \end{aligned}$$

Matrix Form for Polynomials

While this first example was easy enough to solve, for more complicated examples it will be easier to write Equation (15.9) in the form

$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \end{bmatrix}.$$

很明显，给定一种线段的描述，我们可以切换到另一种。

描述线段的另一种方法是使用多项式的规范形式（如第15.3.1节所述）

$$\mathbf{f}(u) = \mathbf{a}_0 + u\mathbf{a}_1. \quad (15.7)$$

任何线段都可以通过指定 \mathbf{a}_0 和 \mathbf{a}_1 或端点 (\mathbf{p}_0 和 \mathbf{p}_1) 来表示。指定端点通常更方便，因为我们可以从端点计算其他参数。

为了将规范形式写成向量表达式，我们定义了一个向量 \mathbf{u} ，它是 u 的幂的向量：

$$\mathbf{u} = [1 \ u \ u^2 \ u^3 \ \dots \ u^n],$$

这样方程 (15.4) 可以写成

$$\mathbf{f}(u) = \mathbf{u} \cdot \mathbf{a}. \quad (15.8)$$

这种矢量表示法将使曲线的不同形式之间的转换更容易。

方程(15.8)描述了一个曲线段，由多项式系数的集合为多项式的简单形式。我们将这种表示称为规范形式。我们将用 \mathbf{a} 表示规范形式的参数。

虽然它在数学上很简单，但规范形式并不总是指定曲线的最方便方法。例如，我们可能更喜欢按端点的位置指定线段。如果我们想定义 \mathbf{p}_0 是线段的开始（其中线段在 $u=0$ 时）和 \mathbf{p}_1 是线段的结束（其中线段在 $u=1$ ），我们可以写

$$\begin{aligned} \mathbf{p}_0 &= \mathbf{f}(0) = [1 \ 0] \cdot [\mathbf{a}_0 \ \mathbf{a}_1], \\ \mathbf{p}_1 &= \mathbf{f}(1) = [1 \ 1] \cdot [\mathbf{a}_0 \ \mathbf{a}_1]. \end{aligned} \quad (15.9)$$

我们可以解决这些方程为一个0和一个1：

$$\begin{aligned} \mathbf{a}_0 &= \mathbf{p}_0, \\ \mathbf{a}_1 &= \mathbf{p}_1 - \mathbf{p}_0. \end{aligned}$$

多项式的矩阵形式

虽然第一个例子很容易解决，但对于更复杂的例子，将方程 (15.9) 写成 $wetp_0p_1$ 形式会更容易



Alternatively, we can write

$$\mathbf{p} = \mathbf{C} \mathbf{a}, \quad (15.10)$$

where we call \mathbf{C} , the *constraint matrix*.¹ If having vectors of points bothers you, you can consider each dimension independently (so that \mathbf{p} is $[x_0 \ x_1]$ or $[y_0 \ y_1]$ and \mathbf{a} is handled correspondingly).

We can solve Equation (15.10) for \mathbf{a} by finding the inverse of \mathbf{C} . This inverse matrix which we will denote by \mathbf{B} is called the *basis matrix*. The basis matrix is very handy since it tells us how to convert between the convenient parameters \mathbf{p} and the canonical form \mathbf{a} , and, therefore, gives us an easy way to evaluate the curve

$$\mathbf{f}(u) = \mathbf{u} \mathbf{B} \mathbf{p}.$$

We can find a basis matrix for whatever form of the curve that we want, providing that there are no nonlinearities in the definition of the parameters. Examples of nonlinearly defined parameters include the length and angle of the line segment.

Now, suppose we want to parameterize the line segment so that \mathbf{p}_0 is the half-way point ($u = 0.5$), and \mathbf{p}_1 is the ending point ($u = 1$). To derive the basis matrix for this parameterization, we set

$$\begin{aligned}\mathbf{p}_0 &= \mathbf{f}(0.5) = 1 \mathbf{a}_0 + 0.5 \mathbf{a}_1, \\ \mathbf{p}_1 &= \mathbf{f}(1) = 1 \mathbf{a}_0 + 1 \mathbf{a}_1.\end{aligned}$$

So

$$\mathbf{C} = \begin{bmatrix} 1 & .5 \\ 1 & 1 \end{bmatrix},$$

and therefore

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 2 & -1 \\ -2 & 2 \end{bmatrix}.$$

15.3.3 Beyond Line Segments

Line segments are so simple that finding a basis matrix is trivial. However, it was good practice for curves of higher degree. First, let's consider quadratics (curves of degree two). The advantage of the canonical form (Equation (15.4)) is that it works for these more complicated curves, just by letting n be a larger number.

¹We assume the form of a vector (row or column) is obvious from the context, and we will skip all of the transpose symbols for vectors.



或者，我们可以写

$$\mathbf{p} = \mathbf{C} \mathbf{a}, \quad (15.10)$$

我们称之为c的约束矩阵。如果有点的向量困扰你，你可以独立地考虑每个维度（这样p是[x0x1]或[y0y1]并且a相应地处理）。

我们可以通过找到C的倒数来求解A的方程（15.10）。我们将用B表示的这个逆矩阵称为基矩阵。基矩阵非常方便，因为它告诉我们如何在方便的参数p和规范形式a之间进行转换，因此为我们提供了一种评估曲线的简单方法

$$\mathbf{f}(u) = \mathbf{u} \mathbf{B} \mathbf{p}.$$

我们可以为我们想要的曲线的任何形式找到一个基矩阵，前提是参数的定义中没有非线性。非线性定义的参数的示例包括线段的长度和角度。现在，假设我们要参数化线段，以便p0是中途点（u=0.5），而p1是终点（u=1）。为了推导出这种参数化的基础矩阵，我们设置

$$\begin{aligned}\mathbf{p}_0 &= \mathbf{f}(0.5) = 1 \mathbf{a}_0 + 0.5 \mathbf{a}_1, \\ \mathbf{p}_1 &= \mathbf{f}(1) = 1 \mathbf{a}_0 + 1 \mathbf{a}_1.\end{aligned}$$

So

$$\mathbf{C} = \begin{bmatrix} 1 & .5 \\ 1 & 1 \end{bmatrix},$$

因此

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 2 & -1 \\ -2 & 2 \end{bmatrix}.$$

15.3.3 超越线段

线段非常简单，以至于找到基矩阵是微不足道的。然而，对于更高程度的曲线，这是一个很好的做法。首先，让我们考虑quadratics（二度曲线）。规范形式（方程（15.4））的优点是它适用于这些更复杂的曲线，只需让n是一个更大的数字。

¹我们假设向量（行或列）的形式从上下文中显而易见，我们将跳过向量的所有转置符号。

A quadratic (a degree-two polynomial) has three coefficients, \mathbf{a}_0 , \mathbf{a}_1 , and \mathbf{a}_2 . These coefficients are not convenient for describing the shape of the curve. However, we can use the same basis matrix method to devise more convenient parameters. If we know the value of u , Equation (15.4) becomes a linear equation in the parameters, and the linear algebra from the last section still works.

Suppose that we wanted to describe our curves by the position of the beginning ($u = 0$), middle² ($u = 0.5$), and end ($u = 1$). Entering the appropriate values into Equation (15.4):

$$\begin{aligned} \mathbf{p}_0 &= \mathbf{f}(0) = \mathbf{a}_0 + 0^1 \quad \mathbf{a}_1 + 0^2 \quad \mathbf{a}_2, \\ \mathbf{p}_1 &= \mathbf{f}(0.5) = \mathbf{a}_0 + 0.5^1 \quad \mathbf{a}_1 + 0.5^2 \quad \mathbf{a}_2, \\ \mathbf{p}_2 &= \mathbf{f}(1) = \mathbf{a}_0 + 1^1 \quad \mathbf{a}_1 + 1^2 \quad \mathbf{a}_2. \end{aligned}$$

So the constraint matrix is

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & .5 & .25 \\ 1 & 1 & 1 \end{bmatrix},$$

and the basis matrix is

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}.$$

There is an additional type of constraint (or parameter) that is sometimes convenient to specify: the derivative of the curve (with respect to its free parameter) at a particular value. Intuitively, the derivatives tell us how the curve is changing, so that the first derivative tells us what direction the curve is going, the second derivative tells us how quickly the curve is changing direction, etc. We will see examples of why it is useful to specify derivatives later.

For the quadratic,

$$\mathbf{f}(u) = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2,$$

the derivatives are simple:

$$\mathbf{f}'(u) = \frac{d\mathbf{f}}{du} = \mathbf{a}_1 + 2\mathbf{a}_2 u,$$

and

$$\mathbf{f}''(u) = \frac{d^2\mathbf{f}}{du^2} = \frac{d\mathbf{f}'}{du} = 2\mathbf{a}_2.$$

²Notice that this is the middle of the parameter space, which might not be the middle of the curve itself.

二次（二度多项式）有三个系数，一个0，一个1和一个2。

这些系数不便于描述曲线的形状。无论如何，我们可以使用相同的基矩阵方法来设计更方便的参数。如果我们知道 u 的值，方程 (15.4) 成为参数中的线性方程，最后一节的线性代数仍然有效。

假设我们想用开始ning ($u=0$)，中间2 ($u=0.5$) 的位置来描述我们的曲线。5)，并结束 ($u=1$)。在公式(15.4)中输入适当的值：

$$\begin{aligned} \mathbf{p}_0 &= \mathbf{f}(0) = \mathbf{a}_0 + 0^1 \quad \mathbf{a}_1 + 0^2 \quad \mathbf{a}_2, \\ \mathbf{p}_1 &= \mathbf{f}(0.5) = \mathbf{a}_0 + 0.5^1 \quad \mathbf{a}_1 + 0.5^2 \quad \mathbf{a}_2, \\ \mathbf{p}_2 &= \mathbf{f}(1) = \mathbf{a}_0 + 1^1 \quad \mathbf{a}_1 + 1^2 \quad \mathbf{a}_2. \end{aligned}$$

所以约束矩阵是

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & .5 & .25 \\ 1 & 1 & 1 \end{bmatrix},$$

而基矩阵为

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}.$$

还有一种额外类型的约束（或参数），有时可以用来指定：曲线在特定值上的导数（相对于其自由参数）。直观地说，导数告诉我们曲线是如何变化的，所以一阶导数告诉我们曲线的方向，二阶导数告诉我们曲线改变方向的速度，等等。我们将在后面看到为什么指定衍生物是有用的例子。对于二次

$$\mathbf{f}(u) = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2,$$

导数很简单：

$$\mathbf{f}'(u) = \frac{d\mathbf{f}}{du} = \mathbf{a}_1 + 2\mathbf{a}_2 u,$$

and

$$\mathbf{f}''(u) = \frac{d^2\mathbf{f}}{du^2} = \frac{d\mathbf{f}'}{du} = 2\mathbf{a}_2.$$

²请注意，这是参数空间的中间，它可能不是曲线本身的中间。



Or, more generally,

$$\begin{aligned}\mathbf{f}'(u) &= \sum_{i=1}^n i u^{i-1} \mathbf{a}_i, \\ \mathbf{f}''(u) &= \sum_{i=2}^n i(i-1) u^{i-2} \mathbf{a}_i.\end{aligned}$$

For example, consider a case where we want to specify a quadratic curve segment by the position, first, and second derivative at its middle ($u = 0.5$).

$$\begin{aligned}\mathbf{p}_0 &= \mathbf{f}(0.5) = \mathbf{a}_0 + 0.5^1 \mathbf{a}_1 + 0.5^2 \mathbf{a}_2, \\ \mathbf{p}_1 &= \mathbf{f}'(0.5) = \mathbf{a}_1 + 2 \cdot 0.5 \mathbf{a}_2, \\ \mathbf{p}_2 &= \mathbf{f}''(0.5) = 2 \mathbf{a}_2.\end{aligned}$$

The constraint matrix is

$$\mathbf{C} = \begin{bmatrix} 1 & .5 & .25 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix},$$

and the basis matrix is

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & -.5 & .125 \\ 0 & 1 & -.5 \\ 0 & 0 & .5 \end{bmatrix}.$$

15.3.4 Basis Matrices for Cubics

Cubic polynomials are popular in graphics (See Section 15.5). The derivations for the various forms of cubics are just like the derivations we've seen already in this section. We will work through one more example for practice.

A very useful form of a cubic polynomial is the *Hermite* form, where we specify the position and first derivative at the beginning and end, that is,

$$\begin{aligned}\mathbf{p}_0 &= \mathbf{f}(0) = \mathbf{a}_0 + 0^1 \mathbf{a}_1 + 0^2 \mathbf{a}_2 + 0^3 \mathbf{a}_3, \\ \mathbf{p}_1 &= \mathbf{f}'(0) = \mathbf{a}_1 + 2 \cdot 0^1 \mathbf{a}_2 + 3 \cdot 0^2 \mathbf{a}_3, \\ \mathbf{p}_2 &= \mathbf{f}(1) = \mathbf{a}_0 + 1^1 \mathbf{a}_1 + 1^2 \mathbf{a}_2 + 1^3 \mathbf{a}_3, \\ \mathbf{p}_3 &= \mathbf{f}'(1) = \mathbf{a}_1 + 2 \cdot 1^1 \mathbf{a}_2 + 3 \cdot 1^2 \mathbf{a}_3.\end{aligned}$$

或者, 更一般地说

$$\begin{aligned}\mathbf{f}'(u) &= \sum_{i=1}^n i u^{i-1} \mathbf{a}_i, \\ \mathbf{f}''(u) &= \sum_{i=2}^n i(i-1) u^{i-2} \mathbf{a}_i.\end{aligned}$$

例如, 考虑一种情况, 我们希望通过中间的位置, 一阶导数和二阶导数来指定二次曲线段 ($u=0.5$).

$$\begin{aligned}\mathbf{p}_0 &= \mathbf{f}(0.5) = \mathbf{a}_0 + 0.5^1 \mathbf{a}_1 + 0.5^2 \mathbf{a}_2, \\ \mathbf{p}_1 &= \mathbf{f}'(0.5) = \mathbf{a}_1 + 2 \cdot 0.5 \mathbf{a}_2, \\ \mathbf{p}_2 &= \mathbf{f}''(0.5) = 2 \mathbf{a}_2.\end{aligned}$$

约束矩阵为

$$\mathbf{C} = \begin{bmatrix} 1 & .5 & .25 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix},$$

而基矩阵为

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & -.5 & .125 \\ 0 & 1 & -.5 \\ 0 & 0 & .5 \end{bmatrix}.$$

15.3.4 立方体的基矩阵

三次多项式在图形中很流行 (见第15.5节)。各种形式的立方体的推导就像我们在本节中已经看到的推导一样。我们将通过一个实践的例子。

三次多项式的一个非常有用的形式是Hermite形式, 我们在开始和结束时指定位置和一阶导数, 即

$$\begin{aligned}\mathbf{p}_0 &= \mathbf{f}(0) = \mathbf{a}_0 + 0^1 \mathbf{a}_1 + 0^2 \mathbf{a}_2 + 0^3 \mathbf{a}_3, \\ \mathbf{p}_1 &= \mathbf{f}'(0) = \mathbf{a}_1 + 2 \cdot 0^1 \mathbf{a}_2 + 3 \cdot 0^2 \mathbf{a}_3, \\ \mathbf{p}_2 &= \mathbf{f}(1) = \mathbf{a}_0 + 1^1 \mathbf{a}_1 + 1^2 \mathbf{a}_2 + 1^3 \mathbf{a}_3, \\ \mathbf{p}_3 &= \mathbf{f}'(1) = \mathbf{a}_1 + 2 \cdot 1^1 \mathbf{a}_2 + 3 \cdot 1^2 \mathbf{a}_3.\end{aligned}$$

Thus, the constraint matrix is

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix},$$

and the basis matrix is

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix}.$$

We will discuss Hermite cubic splines in Section 15.5.2.

15.3.5 Blending Functions

If we know the basis matrix, \mathbf{B} , we can multiply it by the parameter vector, \mathbf{u} , to get a vector of functions

$$\mathbf{b}(u) = \mathbf{u} \mathbf{B}.$$

Notice that we denote this vector by $\mathbf{b}(u)$ to emphasize the fact that its value depends on the free parameter u . We call the elements of $\mathbf{b}(u)$ the *blending functions*, because they specify how to blend the values of the control point vector together:

$$\mathbf{f}(u) = \sum_{i=0}^n \mathbf{b}_i(u) \mathbf{p}_i. \quad (15.11)$$

It is important to note that for a chosen value of u , Equation (15.11) is a *linear* equation specifying a *linear blend* (or weighted average) of the control points. This is true no matter what degree polynomials are “hidden” inside of the \mathbf{b}_i functions.

Blending functions provide a nice abstraction for describing curves. Any type of curve can be represented as a linear combination of its control points, where those weights are computed as some arbitrary functions of the free parameter.

15.3.6 Interpolating Polynomials

In general, a polynomial of degree n can interpolate a set of $n + 1$ values. If we are given a vector $\mathbf{p} = (p_0, \dots, p_n)$ of points to interpolate and a vector $\mathbf{t} = (t_0, \dots, t_n)$ of increasing parameter values, $t_i \neq t_j$, we can use the methods

因此, 约束矩阵为

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix},$$

而基矩阵为

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & 3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix}.$$

我们将在第15.5.2节中讨论Hermite立方样条。

15.3.5 混合函数

如果我们知道基矩阵 \mathbf{B} , 我们可以将它乘以参数向量 \mathbf{u} , 得到一个函数向量

$$\mathbf{b}(u) = \mathbf{u} \mathbf{B}.$$

请注意, 我们用 $\mathbf{b}(u)$ 来表示这个向量, 以强调它的值取决于自由参数 u 。我们将 $\mathbf{b}(u)$ 的元素称为混合函数, 因为它们指定了如何将控制点向量的值混合在一起:

$$\mathbf{f}(u) = \sum_{i=0}^n \mathbf{b}_i(u) \mathbf{p}_i. \quad (15.11)$$

需要注意的是, 对于所选的 u 值, 方程(15.11)是一个线性方程, 指定了控制点的线性混合 (或加权平均)。无论 \mathbf{b}_i 函数内部“隐藏”的多项式是什么程度都是如此。

混合函数为描述曲线提供了一个很好的抽象。任何类型的曲线都可以表示为其控制点的线性组合, 其中这些权重被计算为自由参数的一些任意函数。

15.3.6 插值多项式

一般来说, n 度的多项式可以内插一组 $n+1$ 个值。如果给出一个向量 $\mathbf{p}=(p_0, \dots, p_n)$ 的点进行插值和向量 $\mathbf{t}=(t_0, \dots, t_n)$ 增加参数值, $t_i=t_j$, 我们可以使用这些方法

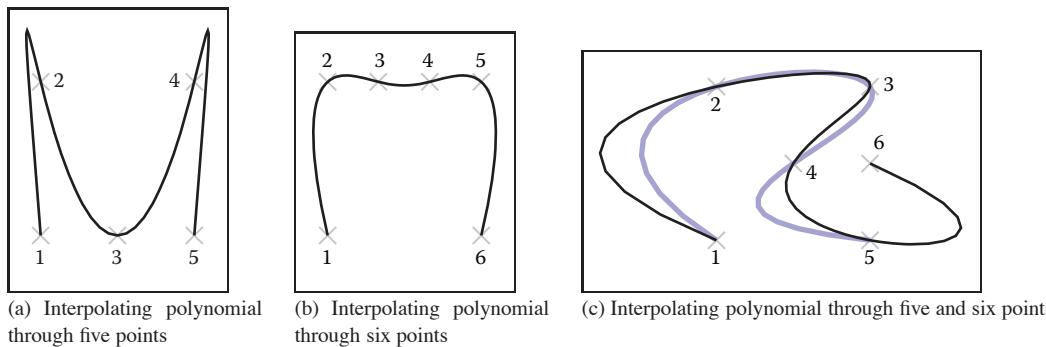


Figure 15.3. Interpolating polynomials through multiple points. Notice the extra wiggles and over-shooting between points. In (c), when the sixth point is added, it completely changes the shape of the curve due to the non-local nature of interpolating polynomials.

described in the previous sections to determine an $n+1 \times n+1$ basis matrix that gives us a function $f(t)$ such that $f(t_i) = p_i$. For any given vector t , we need to set up and solve an $n = 1 \times n+1$ linear system. This provides us with a set of $n+1$ basis functions that perform interpolation:

$$f(t) = \sum_{i=0}^n p_i b_i(t).$$

These interpolating basis functions can be derived in other ways. One particularly elegant way to define them is the *Lagrange form*:

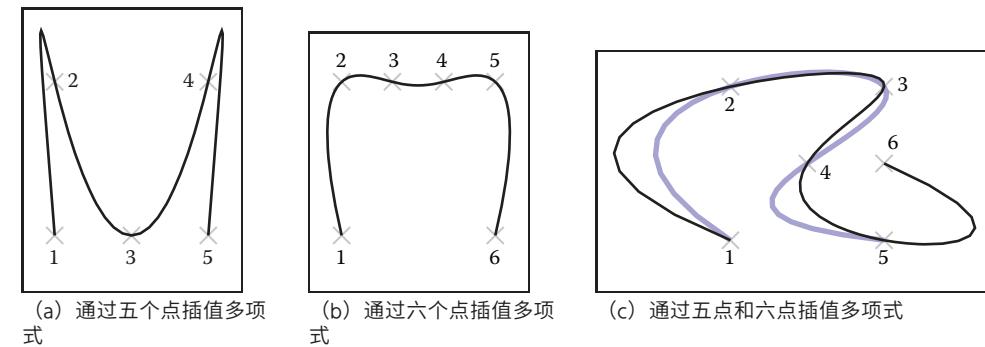
$$b_i = \prod_{j=0, j \neq i}^n \frac{x - t_j}{t_i - t_j}. \quad (15.12)$$

There are more computationally efficient ways to express the interpolating basis functions than the Lagrange form (see De Boor (1978) for details).

Interpolating polynomials provide a mechanism for defining curves that interpolate a set of points. Figure 15.3 shows some examples. While it is possible to create a single polynomial to interpolate any number of points, we rarely use high-order polynomials to represent curves in computer graphics. Instead, interpolating splines (piecewise polynomial functions) are preferred. Some reasons for this are considered in Section 15.5.3.

15.4 Putting Pieces Together

Now that we've seen how to make individual pieces of polynomial curves, we can consider how to put these pieces together.



通过多个点插值多项式。注意点之间的额外摆动和过度射击。在 (c) 中, 当添加第六个点时, 由于插值多项式的非局部性质, 它完全改变了曲线的形状。

在前面的章节中描述了确定一个 $n+1 \times n+1$ 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 t , 我们需要建立并求解一个 $n=1 \times n+1$ 线性系统。这为我们提供了一组执行插值的 $n+1$ 基函数:

$$f(t) = \sum_{i=0}^n p_i b_i(t).$$

这些内插基函数可以以其他方式导出。定义它们的一种特别优雅的方式是拉格朗日形式:

$$b_i = \prod_{j=0, j \neq i}^n \frac{x - t_j}{t_i - t_j}. \quad (15.12)$$

与拉格朗日形式相比, 有更多计算效率更高的方法来表达插值基函数 (详情请参阅DeBoor (1978))。

插值多项式提供了一种用于定义在terpolate一组点的曲线的机制。图15.3显示了一些示例。虽然可以创建单个多项式来插值任意数量的点, 但我们很少使用高阶多项式来表示计算机图形中的曲线。相反, 优先间极化样条(分段多项式函数)。第15.5.3节考虑了这方面的一些原因。

15.4 把碎片拼在一起

现在我们已经看到了如何制作多项式曲线的各个部分, 我们可以考虑如何将这些部分放在一起。

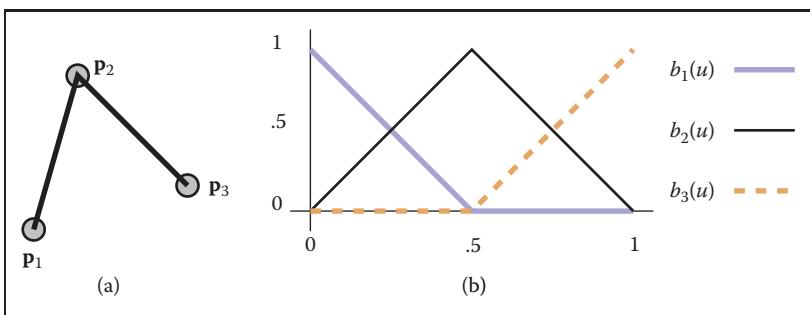


Figure 15.4. (a) Two line segments connect three points; (b) the blending functions for each of the points are graphed at right.

15.4.1 Knots

The basic idea of a piecewise parametric function is that each piece is only used over some parameter range. For example, if we want to define a function that has two piecewise linear segments that connect three points (as shown in Figure 15.4(a)), we might define

$$\mathbf{f}(u) = \begin{cases} \mathbf{f}_1(2u) & \text{if } 0 \leq u < \frac{1}{2}, \\ \mathbf{f}_2(2u - 1) & \text{if } \frac{1}{2} \leq u < 1, \end{cases} \quad (15.13)$$

where \mathbf{f}_1 and \mathbf{f}_2 are functions for each of the two line segments. Notice that we have rescaled the parameter for each of the pieces to facilitate writing their equations as

$$\mathbf{f}_1(u) = (1 - u)\mathbf{p}_1 + u\mathbf{p}_2.$$

For each polynomial in our piecewise function, there is a site (or parameter value) where it starts and ends. Sites where a piece function begins or ends are called *knots*. For the example in Equation (15.13), the values of the knots are 0, 0.5, and 1.

We may also write piecewise polynomial functions as the sum of basis functions, each scaled by a coefficient. For example, we can rewrite the two line segments of Equation (15.13) as

$$\mathbf{f}(u) = \mathbf{p}_1 b_1(u) + \mathbf{p}_2 b_2(u) + \mathbf{p}_3 b_3(u), \quad (15.14)$$

where the function $b_1(u)$ is defined as

$$b_1(u) = \begin{cases} 1 - 2u & \text{if } 0 \leq u < \frac{1}{2}, \\ 0 & \text{otherwise,} \end{cases}$$

and b_2 and b_3 are defined similarly. These functions are plotted in Figure 15.4(b).

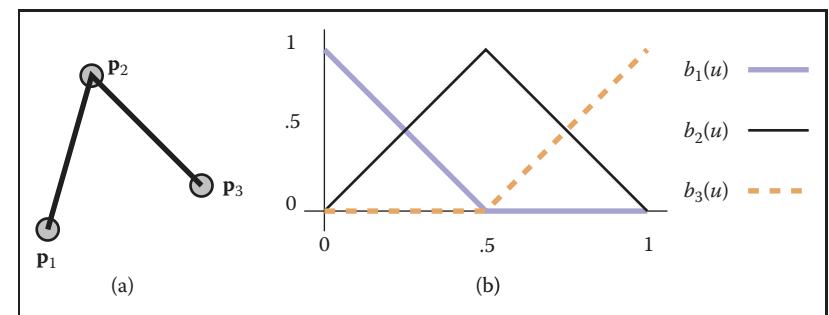


图15.4。(a)两条线段连接三个点；(b)每个点的混合函数在右边绘制。

15.4.1 Knots

分段参数函数的基本思想是每个分段只在某些参数范围内使用。例如，如果我们要定义一个函数，它有两个连接三个点的分段线性段（如图15.4 (a) 所示），我们可以定义

$$\mathbf{f}(u) = \begin{cases} \mathbf{f}_1(2u) & \text{if } 0 \leq u < \frac{1}{2}, \\ \mathbf{f}_2(2u - 1) & \text{if } \frac{1}{2} \leq u < 1, \end{cases} \quad (15.13)$$

其中 f_1 和 f_2 是两个线段中的每一个的函数。请注意，我们已经重新调整了每个部分的参数，以便于将它们的方程写成

$$\mathbf{f}_1(u) = (1 - u)\mathbf{p}_1 + u\mathbf{p}_2.$$

对于我们的分段函数中的每个多项式，都有一个开始和结束的站点（或参数值）。件功能开始或结束的站点称为结。对于公式 (15.13) 中的示例，结的值为0 0.5，和1。

我们也可以将分段多项式函数写为基函数的和，每个函数都按系数缩放。例如，我们可以将等式 (15.13) 的两条线段重写为

$$\mathbf{f}(u) = \mathbf{p}_1 b_1(u) + \mathbf{p}_2 b_2(u) + \mathbf{p}_3 b_3(u), \quad (15.14)$$

其中函数 $b_1(u)$ 定义为

$$b_1(u) = \begin{cases} 1 - 2u & \text{if } 0 \leq u < \frac{1}{2}, \\ 0 & \text{otherwise,} \end{cases}$$

和 b_2 和 b_3 的定义类似。这些函数在图15.4 (b) 中绘制。



The knots of a polynomial function are the combination of the knots of all of the pieces that are used to create it. The *knot vector* is a vector that stores all of the knot values in ascending order.

Notice that in this section we have used two different mechanisms for combining polynomial pieces: using independent polynomial pieces for different ranges of the parameter and blending together piecewise polynomial functions.

15.4.2 Using Independent Pieces

In Section 15.3, we defined pieces of polynomials over the unit parameter range. If we want to assemble these pieces, we need to convert from the parameter of the overall function to the value of the parameter for the piece. The simplest way to do this is to define the overall curve over the parameter range $[0, n]$ where n is the number of segments. Depending on the value of the parameter, we can shift it to the required range.

15.4.3 Putting Segments Together

If we want to make a single curve from two line segments, we need to make sure that the end of the first line segment is at the same location as the beginning of the next. There are three ways to connect the two segments (in order of simplicity):

1. Represent the line segment as its two endpoints, and then use the same point for both. We call this a *shared-point* scheme.
2. Copy the value of the end of the first segment to the beginning of the second segment every time that the parameters of the first segment change. We call this a *dependency* scheme.
3. Write an explicit equation for the connection, and enforce it through numerical methods as the other parameters are changed.

While the simpler schemes are preferable since they require less work, they also place more restrictions on the way the line segments are parameterized. For example, if we want to use the center of the line segment as a parameter (so that the user can specify it directly), we will use the beginning of each line segment and the center of the line segment as their parameters. This will force us to use the dependency scheme.

Notice that if we use a shared-point or dependency scheme, the total number of control points is less than $n * m$, where n is the number of segments and m



多项式函数的结是用于创建它的所有部分的结的组合。结向量是按升序存储所有结值的向量。

请注意，在本节中，我们使用了两种不同的组合多项式块的机制：对参数的不同范围使用独立的多项式块和将分段多项式函数混合在一起。

15.4.2 使用独立件

在第15.3节中，我们定义了单位参数范围内的多项式。如果我们想组装这些片段，我们需要从整体函数的参数转换为该片段的参数值。最简单的方法是定义参数范围 $[0, n]$ 上的整体曲线，其中 n 是段数。根据参数的值，我们可以将其移动到所需的范围。

15.4.3 将片段放在一起

如果我们想从两条线段制作一条曲线，我们需要确保第一条线段的末端与下一条线段的开头位于同一位置。有三种方法可以连接两个段（为了简单起见）：

1. 将线段表示为其两个端点，然后对两者使用相同的点。我们称之为共享点方案。
2. 每次第一段的参数发生变化时，将第一段末尾的值复制到第二段的开头。我们称之为依赖方案。
3. 为连接编写一个显式方程，并在其他参数更改时通过数值方法强制执行。

虽然更简单的方案更可取，因为它们需要更少的工作，但它们也对线段参数化的方式施加了更多限制。例如，如果我们想使用线段的中心作为参数（以便用户可以直接指定它），我们将使用每个线段的开头和线段的中心作为它们的参数。这将迫使我们使用依赖方案。

请注意，如果我们使用共享点或依赖方案，则控制点的总数小于 $n * m$ ，其中 n 是段数和 m

is the number of control points for each segment; many of the control points of the independent pieces will be computed as functions of other pieces. Notice that if we use either the shared-point scheme for lines (each segment uses its two endpoints as parameters and shares interior points with its neighbors), or if we use the dependency scheme (such as the example one with the first endpoint and midpoint), we end up with $n + 1$ controls for an n -segment curve.

Dependency schemes have a more serious problem. A change in one place in the curve can propagate through the entire curve. This is called a lack of *locality*. Locality means that if you move a point on a curve it will only affect a local region. The local region might be big, but it will be finite. If a curve's controls do not have locality, changing a control point may affect points infinitely far away.

To see locality, and the lack thereof, in action, consider two chains of line segments, as shown in Figure 15.5. One chain has its pieces parameterized by their endpoints and uses point-sharing to maintain continuity. The other has its pieces parameterized by an endpoint and midpoint and uses dependency propagation to keep the segments together. The two segment chains can represent the same curves: they are both a set of n connected line segments. However, because of locality issues, the endpoint-shared form is likely to be more convenient for the user. Consider changing the position of the first control point in each chain. For the endpoint-shared version, only the first segment will change, while all of the segments will be affected in the midpoint version, as in Figure 15.5. In fact, for any point moved in the endpoint-shared version, at most two line segments will change. In the midpoint version, all segments after the control point that is moved will change, even if the chain is infinitely long.

In this example, the dependency propagation scheme was the one that did not have local control. This is not always true. There are direct sharing schemes that are not local and propagation schemes that are local.

We emphasize that locality is a convenience of control issue. While it is inconvenient to have the entire curve change every time, the same changes can be made to the curve. It simply requires moving several points in unison.

15.5 Cubics

In graphics, when we represent curves using piecewise polynomials, we usually use either line segments or cubic polynomials for the pieces. There are a number of reasons why cubics are popular in computer graphics:

- Piecewise cubic polynomials allow for C^2 continuity, which is generally sufficient for most visual tasks. The C^1 smoothness that quadratics offer is

是每个分段的控制点的数量;独立部分的许多控制点将作为其他部分的函数计算。请注意，如果我们使用线的共享点方案（每个段使用它的两个端点作为参数并与其邻居共享内部点），或者如果我们使用依赖方案（例如第一个端点和中点的示例），我们最终会得到 $n+1$ 个 n 段曲线的控件。

依赖方案有一个更严重的问题。曲线中某个位置的变化可以传播到整个曲线。这被称为缺乏地方性。局部性意味着如果移动曲线上的点，它只会影响局部区域。当地可能很大，但它将是有限的。如果曲线的控件不具有局部性，则更改控制点可能会影响无限远的点。为了看到局部性，以及缺乏，在行动中，考虑两条线段链，如图15.5所示。一个链的各个部分由端点参数化，并使用点共享来保持连续性。另一个由端点和中点参数化其片段，并使用依赖项propagation将片段保持在一起。这两条线段链可以表示相同的曲线：它们都是一组 n 个相连的线段。但是，由于局部性问题，端点共享表单可能对用户更方便。考虑改变每个链中的第一个控制点的位置。对于端点共享版本，只有第一个段将更改，而所有段将在中点版本中受到影响，如图15.5所示。实际上，对于在端点共享版本中移动的任何点，最多会有两条线段发生变化。在中点版本中，移动的控制点之后的所有线段都会发生变化，即使链是无限长的。

在此示例中，依赖关系传播方案是没有本地控制的方案。这并不总是正确的。有不是本地的直接共享方案和本地的传播方案。

我们强调地方性是一个方便控制的问题。虽然每次更改整个曲线都很方便，但可以对曲线进行相同的更改。它只需要齐声移动几个点。

15.5 Cubics

在图形中，当我们使用分段多项式表示曲线时，我们通常使用线段或三次多项式来表示块。立方体在计算机图形学中受欢迎的原因有很多：

*分段三次多项式允许 C^2 连续性，这对于大多数视觉任务来说通常是足够的。Quadratics提供的 C^1 平滑度是

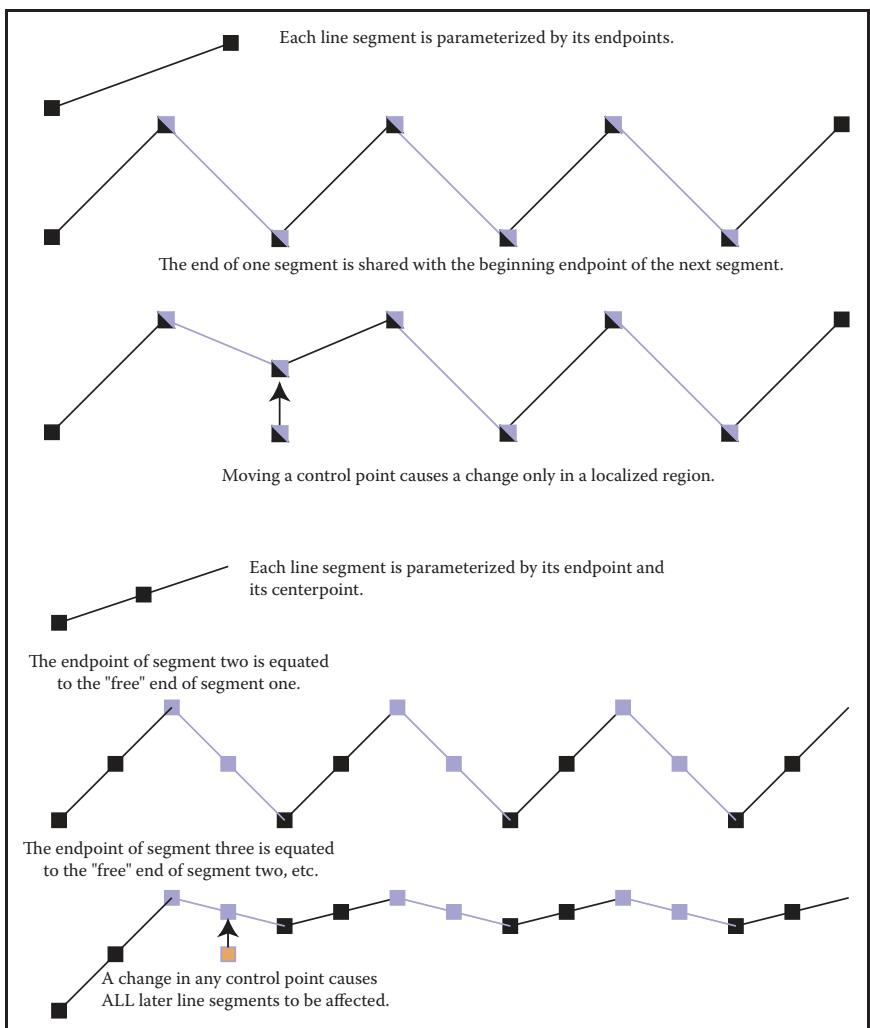


Figure 15.5. A chain of line segments with local control and one with non-local control.

often insufficient. The greater smoothness offered by higher-order polynomials is rarely important.

- Cubic curves provide the minimum-curvature interpolants to a set of points. That is, if you have a set of $n + 3$ points and define the “smoothest” curve that passes through them (that is the curve that has the minimum curvature over its length), this curve can be represented as a piecewise cubic with n segments.

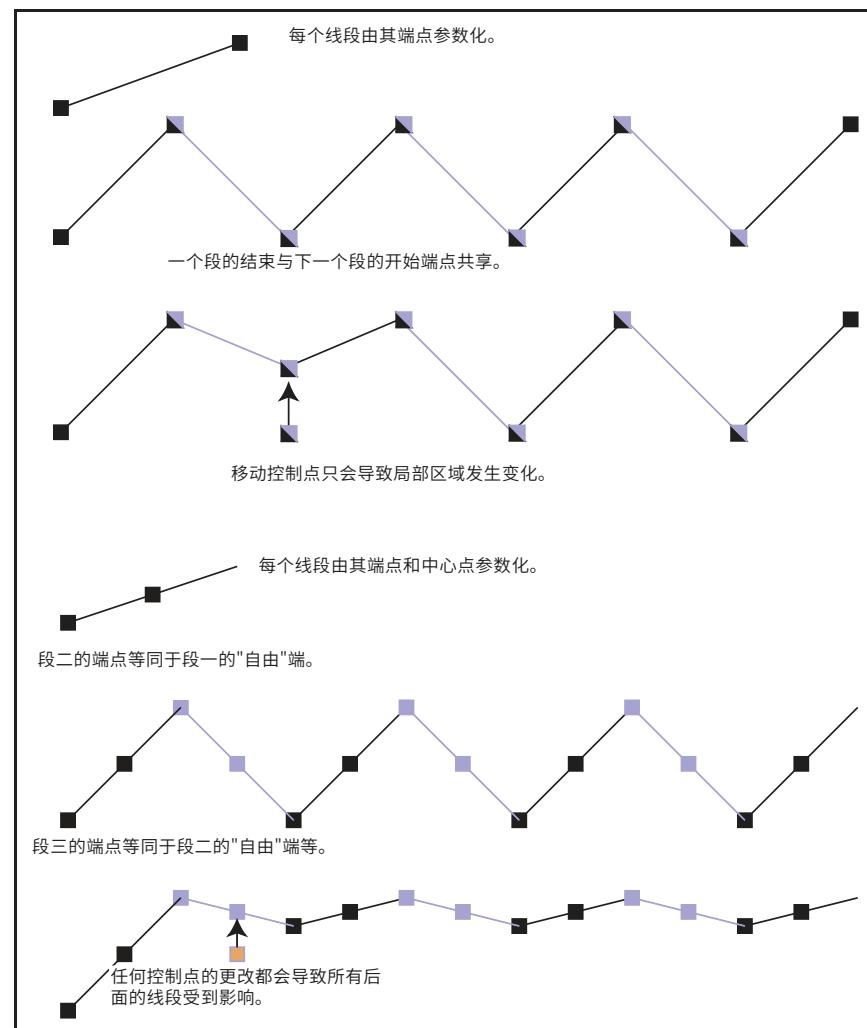


图15.5. 具有局部控制的线段链和具有非局部控制的线段链。

常不足。高阶多项式提供的更大平滑度很少重要。

*三次曲线为一组点提供最小曲线插值。也就是说，如果您有一组 $n+3$ 点并定义穿过它们的“最平滑”曲线（即在其长度上具有最小曲率的曲线），则此曲线可以表示为具有 n 段的分段三次。

- Cubic polynomials have a nice symmetry where position and derivative can be specified at the beginning and end.
- Cubic polynomials have a nice tradeoff between the numerical issues in computation and the smoothness.

Notice that we do not have to use cubics; they just tend to be a good tradeoff between the amount of smoothness and complexity. Different applications may have different tradeoffs. We focus on cubics since they are the most commonly used.

The canonical form of a cubic polynomial is

$$\mathbf{f}(u) = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2 + \mathbf{a}_3 u^3.$$

As we discussed in Section 15.3, these canonical form coefficients are not a convenient way to describe a cubic segment.

We seek forms of cubic polynomials for which the coefficients are a convenient way to control the resulting curve represented by the cubic. One of the main conveniences will be to provide ways to ensure the connectedness of the pieces and the continuity between the segments.

Each cubic polynomial piece requires four coefficients or control points. That means for a piecewise polynomial with n pieces, we may require up to $4n$ control points if no sharing between segments is done or dependencies used. More often, some part of each segment is either shared or depends on an adjacent segment, so the total number of control points is much lower. Also, note that a control point might be a position or a derivative of the curve.

Unfortunately, there is no single "best" representation for a piecewise cubic. It is not possible to have a piecewise polynomial curve representation that has all of the following desirable properties:

1. each piece of the curve is a cubic;
2. the curve interpolates the control points;
3. the curve has local control;
4. the curve has C^2 continuity.

We can have any three of these properties, but not all four; there are representations that have any combination of three. In this book, we will discuss cubic B-splines that do not interpolate their control points (but have local control and are C^2); Cardinal splines and Catmull-Rom splines that interpolate their control

- *三次多项式具有很好的对称性，其中位置和导数可以在开始和结束时指定。
- *三次多项式在计算中的数值问题和平滑之间有一个很好的权衡。

请注意，我们不必使用立方体；他们只是倾向于平滑度和复杂性之间的一个很好的权衡。不同的应用可能有不同的权衡。我们专注于立方体，因为它们是最常用的。

三次多项式的规范形式是

$$\mathbf{f}(u) = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2 + \mathbf{a}_3 u^3.$$

正如我们在第15.3节中所讨论的，这些规范形式系数不是描述立方段的一种方便的方式。

我们寻求三次多项式的形式，其系数是控制由三次表示的所得曲线的一种方便方式。其中一个主要的便利将是提供确保件的连通性和段之间的连续性的方法。

每个三次多项式片需要四个系数或控制点。这意味着对于具有 n 个分段的分段多项式，如果没有在分段之间进行共享或使用依赖关系，我们可能需要最多 $4n$ 个控制点。更常见的是，每个段的某些部分要么是共享的，要么依赖于相邻段，因此控制点的总数要低得多。另外，请注意，控制点可能是曲线的位置或导数。

不幸的是，分段立方没有单一的"最佳"表示。
不可能具有具有以下所有期望属性的分段多项式曲线表示：

- 1.每条曲线都是一个立方体;
- 2.曲线对控制点进行插值;
- 3.曲线具有局部控制;
- 4.曲线具有 C^2 连续性。

我们可以有这些属性中的任何三个，但不是全部四个；有三个任意组合的表示。在本书中，我们将讨论不插值其控制点（但具有局部控制并且是 C^2 ）的三次B样条线；插值其控制的Cardinal样条线和Catmull-Rom样条线

points and offer local control, but are not C^2 ; and natural cubics that interpolate and are C^2 , but do not have local control.

The continuity properties of cubics refer to the continuity between the segments (at the knot points). The cubic pieces themselves have infinite continuity in their derivatives (the way we have been talking about continuity so far). Note that if you have a lot of control points (or knots), the curve can be wiggly, which might not seem “smooth.”

15.5.1 Natural Cubics

With a piecewise cubic curve, it is possible to create a C^2 curve. To do this, we need to specify the position and first and second derivative at the beginning of each segment (so that we can make sure that it is the same as at the end of the previous segment). Notice that each curve segment receives three out of its four parameters from the previous curve in the chain. These C^2 continuous chains of cubics are sometimes referred to as *natural* cubic splines.

For one segment of the natural cubic, we need to parameterize the cubic by the positions of its endpoints and the first and second derivative at the beginning point. The control points are therefore

$$\begin{aligned} p_0 &= f(0) = \mathbf{a}_0 + 0^1 \mathbf{a}_1 + 0^2 \mathbf{a}_2 + 0^3 \mathbf{a}_3, \\ p_1 &= f'(0) = 1^1 \mathbf{a}_1 + 2 \cdot 0^1 \mathbf{a}_2 + 3 \cdot 0^2 \mathbf{a}_3, \\ p_2 &= f''(0) = 2 \cdot 1^1 \mathbf{a}_2 + 6 \cdot 0^1 \mathbf{a}_3, \\ p_3 &= f(1) = \mathbf{a}_0 + 1^1 \mathbf{a}_1 + 1^2 \mathbf{a}_2 + 1^3 \mathbf{a}_3. \end{aligned}$$

Therefore, the constraint matrix is

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

and the basis matrix is

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & .5 & 0 \\ -1 & -1 & -.5 & 1 \end{bmatrix}.$$

Given a set of n control points, a natural cubic spline has $n-1$ cubic segments. The first segment uses the control points to define its beginning position, ending position, and first and second derivative at the beginning. A dependency scheme

点并提供局部控制，但不是 C^2 ;以及插值并且是 C^2 的自然立方体，但没有局部控制。

立方体的连续性属性是指缝之间的连续性（在结点）。立方片本身在其衍生物中具有无限的连续性（到目前为止我们一直在谈论连续性的方式）。请注意，如果你有很多控制点（或结），曲线可以是摆动的，这可能看起来不是“平滑的”。

15.5.1 Natural Cubics

使用分段三次曲线，可以创建 C^2 曲线。要做到这一点，我们需要在每个段的开头指定位置以及一阶和二阶导数（这样我们就可以确保它与前一段的末尾相同）。请注意，每个曲线段从链中的前一条曲线接收其四个参数中的三个。立方体的这些 C^2 连续链有时被称为天然立方样条。

对于自然立方的一个段，我们需要通过其端点的位置以及起点处的一阶和二阶导数来参数化立方。因此，控制点是

$$\begin{aligned} p_0 &= f(0) = \mathbf{a}_0 + 0^1 \mathbf{a}_1 + 0^2 \mathbf{a}_2 + 0^3 \mathbf{a}_3, \\ p_1 &= f'(0) = 1^1 \mathbf{a}_1 + 2 \cdot 0^1 \mathbf{a}_2 + 3 \cdot 0^2 \mathbf{a}_3, \\ p_2 &= f''(0) = 2 \cdot 1^1 \mathbf{a}_2 + 6 \cdot 0^1 \mathbf{a}_3, \\ p_3 &= f(1) = \mathbf{a}_0 + 1^1 \mathbf{a}_1 + 1^2 \mathbf{a}_2 + 1^3 \mathbf{a}_3. \end{aligned}$$

因此，约束矩阵为

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

而基矩阵为

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & .5 & 0 \\ -1 & -1 & -.5 & 1 \end{bmatrix}.$$

给定一组 n 个控制点，自然三次样条具有 $n-1$ 个立方段。
第一段使用控制点来定义其开始位置、结束位置以及开始时的一阶导数和二阶导数。依赖方案

copies the position, and first and second derivative of the end of the first segment for use in the second segment.

A disadvantage of natural cubic splines is that they are not local. Any change in any segment may require the entire curve to change (at least the part after the change was made). To make matters worse, natural cubic splines tend to be ill-conditioned: a small change at the beginning of the curve can lead to large changes later. Another issue is that we only have control over the derivatives of the curve at its beginning. Segments after the beginning of the curve determine their derivatives from their beginning point.

15.5.2 Hermite Cubics

Hermite cubic polynomials were introduced in Section 15.3.4. A segment of a cubic Hermite spline allows the positions and first derivatives of both of its endpoints to be specified. A chain of segments can be linked into a C^1 spline by using the same values for the position and derivative of the end of one segment and for the beginning of the next.

Given a set of n control points, where every other control point is a derivative value, a cubic Hermite spline contains $(n-2)/2$ cubic segments. The spline interpolates the points, as shown in Figure 15.6, but can guarantee only C^1 continuity.

Hermite cubics are convenient because they provide local control over the shape, and provide C^1 continuity. However, since the user must specify both positions and derivatives, a special interface for the derivatives must be provided. One possibility is to provide the user with points that represent where the derivative vectors would end if they were “placed” at the position point.

15.5.3 Cardinal Cubics

A *cardinal cubic spline* is a type of C^1 interpolating spline made up of cubic polynomial segments. Given a set of n control points, a cardinal cubic spline uses

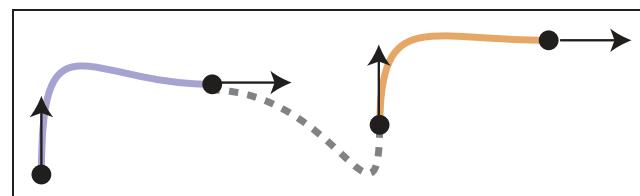


Figure 15.6. A Hermite cubic spline made up of three segments.

复制位置，以及第一段末端的一阶和二阶导数以用于第二段。

天然立方样条的缺点是它们不是局部的。任何段的任何改变都可能需要整个曲线改变（至少改变后的部分）。更糟糕的是，自然三次样条往往是病态的：曲线开始时的小变化可能会导致以后的大变化。另一个问题是，我们只能控制曲线开始时的导数。曲线开始后的线段从它们的起点确定它们的导数。

15.5.2 Hermite Cubics

15.3.4节介绍了Hermite立方多项式。立方Hermite样条的一段允许指定其两个端点的位置和一阶导数。通过对一个段的末端和下一个段的开始使用相同的值，可以将一个段链链接到C1样条中。

给定一组 n 个控制点，其中每个其他控制点是一个导数值，一个三次Hermite样条包含 $(n-2)/2$ 个立方段。样条线对点进行极化，如图15.6所示，但只能保证 C^1 的连续性。Hermite cubics很方便，因为它们提供对形状的局部控制，并提供 C^1 连续性。但是，由于用户必须同时指定采购订单和衍生品，因此必须提供衍生品的特殊接口。一种可能性是为用户提供点，这些点表示如果派生向量被“放置”在位置点上，它们将在哪里结束。

15.5.3 Cardinal Cubics

基数三次样条是一种由三次多项式段组成的 C^1 插值样条。给定一组 n 个控制点，基数三次样条使用

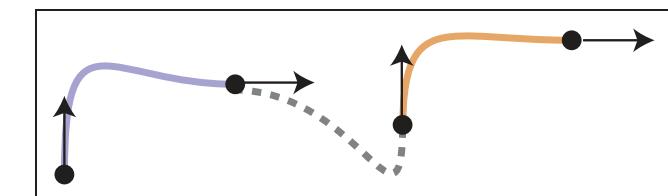


图15.6。由三段组成的Hermite立方样条。

$n - 2$ cubic polynomial segments to interpolate all of its points except for the first and last.

Cardinal splines have a parameter called *tension* that controls how “tight” the curve is between the points it interpolates. The tension is a number in the range $[0, 1]$ that controls how the curve bends toward the next control point. For the important special case of $t = 0$, the splines are called *Catmull-Rom* splines.

Each segment of the cardinal spline uses four control points. For segment i , the points used are $i, i + 1, i + 2$, and $i + 3$ as the segments share three points with their neighbors. Each segment begins at its second control point and ends at its third control point. The derivative at the beginning of the curve is determined by the vector between the first and third control points, while the derivative at the end of the curve is given by the vector between the second and fourth points, as shown in Figure 15.7.

The tension parameter adjusts how much the derivatives are scaled. Specifically, the derivatives are scaled by $(1 - t)/2$. The constraints on the cubic are therefore

$$\begin{aligned} f(0) &= p_2, \\ f(1) &= p_3, \\ f'(0) &= \frac{1}{2}(1-t)(p_3 - p_1), \\ f'(1) &= \frac{1}{2}(1-t)(p_4 - p_2). \end{aligned}$$

Solving these equations for the control points (defining $s = (1 - t)/2$) gives

$$\begin{aligned} p_0 &= f(1) - \frac{2}{1-t}f'(0) = a_0 + (1 - \frac{1}{s})a_1 + a_2 + a_3, \\ p_1 &= f(0) = a_0, \\ p_2 &= f(1) = a_0 + a_1 + a_2 + a_3, \\ p_3 &= f(0) + \frac{1}{s}f'(1) = a_0 + \frac{1}{s}a_1 + 2\frac{1}{s}a_2 + 3\frac{1}{s}a_3. \end{aligned}$$

This yields the cardinal matrix

$$B = C^{-1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -s & 0 & s & 0 \\ 2s & s-3 & 3-2s & -s \\ -s & 2-s & s-2 & s \end{bmatrix}.$$

Since the third point of segment i is the second point of segment $i+1$, adjacent segments of the cardinal spline connect. Similarly, the same points are used to specify the first derivative of each segment, providing C^1 continuity.

Cardinal splines are useful, because they provide an easy way to interpolate a set of points with C^1 continuity and local control. They are only C^1 , so they sometimes get “kinks” in them. The tension parameter gives some control over what happens between the interpolated points, as shown in Figure 15.8, where a

$n-2$ 个三次多项式分段来插值除第一个和最后一个以外的所有点。

基数样条有一个称为张力的参数，用于控制曲线在其插值的点之间的“紧密”程度。张力是范围 $[0, 1]$ 中的一个数字，用于控制曲线如何向下一个控制点弯曲。对于 $t=0$ 的重要特例，样条称为Catmull-Rom样条。

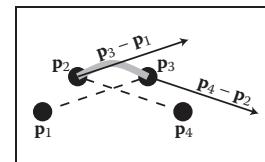


Figure 15.7. A segment of a cardinal cubic spline interpolates its second and third control points (p_2 and p_3), and uses its other points to determine the derivatives at the beginning and end.

基数样条的每个段使用四个控制点。对于段*i*，使用的点是*i, i+1, i+2*和*i+3*，因为段与其邻居共享三个点。每个段在其第二控制点处开始并在其第三控制点处结束。曲线开始处的导数确定

由第一和第三控制点之间的矢量给出，而曲线末端的导数由第二和第四点之间的矢量给出，如图15.7所示。

张力参数调整导数的比例。具体而言，导数按 $(1 - t)/2$ 缩放。因此，对立方的约束是

$$\begin{aligned} f(0) &= p_2, \\ f(1) &= p_3, \\ f'(0) &= \frac{1}{2}(1-t)(p_3 - p_1), \\ f'(1) &= \frac{1}{2}(1-t)(p_4 - p_2). \end{aligned}$$

为控制点求解这些方程（定义 $s = (1-t)/2$ ）给出

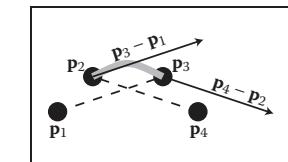
$$\begin{aligned} p_0 &= f(1) - \frac{2}{1-t}f'(0) = a_0 + (1 - \frac{1}{s})a_1 + a_2 + a_3, \\ p_1 &= f(0) = a_0, \\ p_2 &= f(1) = a_0 + a_1 + a_2 + a_3, \\ p_3 &= f(0) + \frac{1}{s}f'(1) = a_0 + \frac{1}{s}a_1 + 2\frac{1}{s}a_2 + 3\frac{1}{s}a_3. \end{aligned}$$

这产生了基数矩阵

$$B = C^{-1} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -s & 0 & s & 0 \\ 2s & s-3 & 3-2s & -s \\ -s & 2-s & s-2 & s \end{bmatrix}.$$

由于线段*i*的第三点是线段*i+1*的第二点，相邻线段的基数样条连接。同样，相同的点用于指定每个段的一阶导数，提供 C^1 连续性。

基数样条很有用，因为它们提供了一种简单的方法来插值具有 C^1 连续性和局部控制的一组点。他们只是 C^1 ，所以他们有时会在其中得到“扭结”。张力参数可以控制插值点之间发生的情况，如图15.8所示，其中



一个基数三次样条的一段在 interpolating它的第二个和第三个控制点(p_2 和 p_3)，并使用它的其他点来阻止挖掘在被轧和结束的导数。

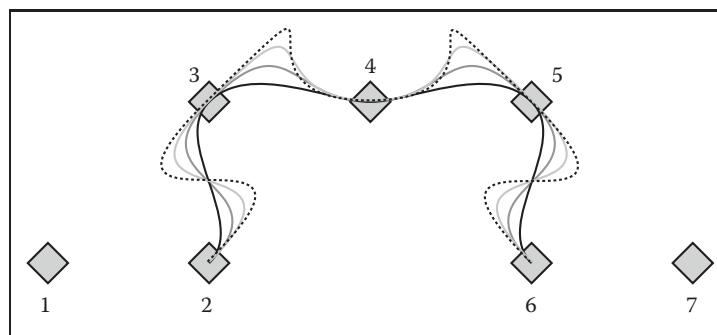


Figure 15.8. Cardinal splines through seven control points with varying values of tension parameter t .

set of cardinal splines through a set of points is shown. The curves use the same control points, but they use different values for the tension parameters. Note that the first and last control points are not interpolated.

Given a set of n points to interpolate, you might wonder why we might prefer to use a cardinal cubic spline (that is a set of $n - 2$ cubic pieces) rather than a single, order n polynomial as described in Section 15.3.6. Some of the disadvantages of the interpolating polynomial are:

- The interpolating polynomial tends to overshoot the points, as seen in Figure 15.9. This overshooting gets worse as the number of points grows larger. The cardinal splines tend to be well behaved in between the points.
- Control of the interpolating polynomial is not local. Changing a point at the beginning of the spline affects the entire spline. Cardinal splines are local: any place on the spline is affected by its four neighboring points at most.
- Evaluation of the interpolating polynomial is not local. Evaluating a point on the polynomial requires access to all of its points. Evaluating a point on the piecewise cubic requires a fixed small number of computations, no matter how large the total number of points is.

There are a variety of other numerical and technical issues in using interpolating splines as the number of points grows larger. See De Boor (2001) for more information.

A cardinal spline has the disadvantage that it does not interpolate the first or last point, which can be easily fixed by adding an extra point at either end of the sequence. The cardinal spline also is not as continuous—providing only C^1 continuity at the knots.

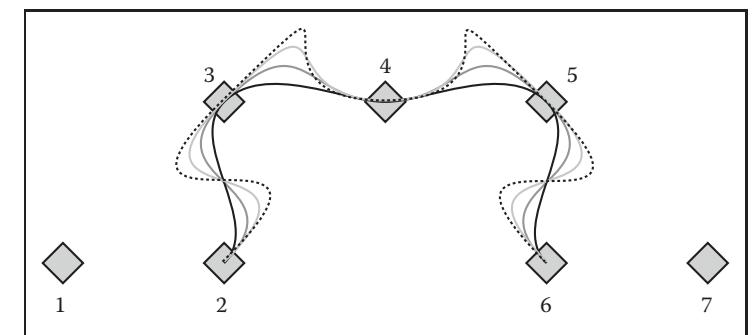


Figure 15.8. 通过七个控制点的基数样条具有不同的张力参数值

显示了通过一组点的基数样条集。曲线使用相同的控制点，但它们对张力参数使用不同的值。请注意，第一个和最后一个控制点不进行插值。

给定一组要插值的 n 个点，您可能想知道为什么我们可能更喜欢使用基数三次样条（即一组 $n - 2$ 立方块）而不是single，阶数 n 多项式，如第15.3.6节所述。插值多项式的一些缺点是：

*插值多项式倾向于超过点，如图15.9所示。随着点数的增加，这种超调会变得更糟。基本样条在点之间往往表现良好。

*插值多项式的控制不是局部的。更改样条曲线开头的点会影响整个样条曲线。基数样条是局部的：样条上的任何地方最多受其四个相邻点的影响。

*插值多项式的评估不是局部的。在多项式上评估一个点需要访问其所有点。计算分段三次上的点需要固定的少量计算，无论点的总数有多大。

随着点的数量越来越大，使用插值样条还存在许多其他数值和技术问题。更多信息见DeBoor(2001)。

基数样条的缺点是它不插值第一个或最后一个点，这可以很容易地通过在序列的任何一端添加一个额外的点来固定。基数样条也不是连续的只提供 C^1 结处的连续性。

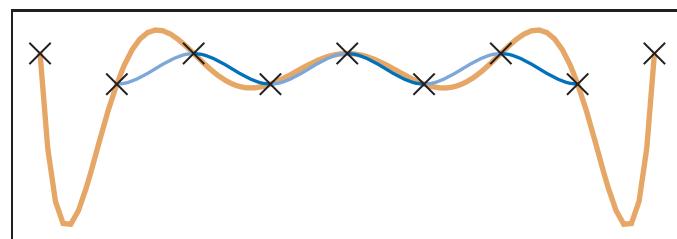


Figure 15.9. Splines interpolating nine control points (marked with small crosses). The thick orange line shows an interpolating polynomial. The thin line shows a Catmull-Rom spline. The latter is made of seven cubic segments, which are each shown in alternating blue tones.

15.6 Approximating Curves

It might seem like the easiest way to control a curve is to specify a set of points for it to interpolate. In practice, however, interpolation schemes often have undesirable properties because they have less continuity and offer no control of what happens between the points. Curve schemes that only approximate the points are often preferred. With an approximating scheme, the control points influence the shape of the curve, but do not specify it exactly. Although we give up the ability to directly specify points for the curve to pass through, we gain better behavior of the curve and local control. Should we need to interpolate a set of points, the positions of the control points can be computed such that the curve passes through these interpolation points.

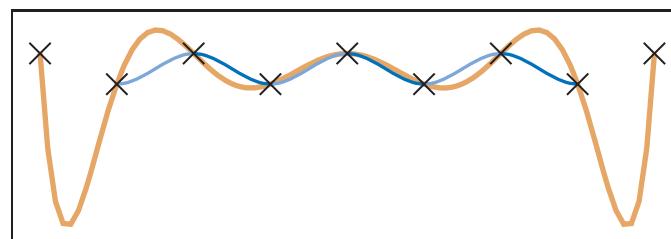
The two most important types of approximating curves in computer graphics are Bézier curves and B-spline curves.

15.6.1 Bézier Curves

Bézier curves are one of the most common representations for free-form curves in computer graphics. The curves are named for Pierre Bézier, one of the people who was instrumental in their development. Bézier curves have an interesting history where they were concurrently developed by several independent groups.

A Bézier curve is a polynomial curve that approximates its control points. The curves can be a polynomial of any degree. A curve of degree d is controlled by $d + 1$ control points. The curve interpolates its first and last control points, and the shape is directly influenced by the other points.

Often, complex shapes are made by connecting a number of Bézier curves of low degree, and in computer graphics, cubic ($d = 3$) Bézier curves are commonly used for this purpose. Many popular illustration programs, such as Adobe Illus-



样条插值九个控制点（用小十字标记）。粗橙色线显示插值多项式。细线显示Catmull-Rom样条。后者由七个立方段组成，每个段都以交替的蓝色色调显示。

15.6 近似曲线

控制曲线的最简单方法似乎是指定一组点以进行插值。然而，在实践中，插值方案通常具有不可取的属性，因为它们具有较少的连续性，并且无法控制点之间发生的情况。通常优先仅近似点的曲线方案。在近似方案中，控制点会影响曲线的形状，但不会精确指定它。虽然我们放弃了直接指定曲线通过点的能力，但我们获得了更好的曲线行为和局部控制。如果我们需要插值一组点，可以计算控制点的位置，使曲线通过这些插值点。

计算机图形学中两种最重要的近似曲线类型是贝塞尔曲线和B样条曲线。

15.6.1 Bézier Curves

贝塞尔曲线是计算机图形学中自由曲线最常见的表示形式之一。这些曲线以皮埃尔·贝塞尔 (Pierre Bezier) 的名字命名，贝塞尔 (Pierre Bezier) 是他们发展的重要人物之一。贝塞尔曲线有一个有趣的历史，它们是由几个独立的团体同时开发的。贝塞尔曲线是近似其控制点的多项式曲线。曲线可以是任何程度的多项式。度 d 的曲线由 $d+1$ 个控制点控制。曲线插值其第一个和最后一个控制点，形状直接受其他点的影响。

通常，通过连接一些低度的贝塞尔曲线来制作复杂的形状，并且在计算机图形学中，立方 ($d=3$) 贝塞尔曲线通常用于此目的。许多流行的插图程序，如Adobe Illus-

trator, and font representation schemes, such as that used in Postscript, use cubic Bézier curves. Bézier curves are extremely popular in computer graphics because they are easy to control, have a number of useful properties, and there are very efficient algorithms for working with them.

Bézier curves are constructed such that:

- The curve interpolates the first and last control points, with $u = 0$ and 1 , respectively.
- The first derivative of the curve at its beginning (end) is determined by the vector between the first and second (next to last and last) control points. The derivatives are given by the vectors between these points scaled by the degree of the curve.
- Higher derivatives at the beginning (end) of the curve depend on the points at the beginning (end) of the curve. The n^{th} derivative depends on the first (last) $n + 1$ points.

For example, consider the Bézier curve of degree 3 (cubic) as in Figure 15.10. The curve has four ($d + 1$) control points. It begins at the first control point (p_0) and ends at the last (p_1). The first derivative at the beginning is proportional to the vector between the first and second control points ($p_1 - p_0$). Specifically, $f'(0) = 3(p_1 - p_0)$. Similarly, the first derivative at the end of the curve is given by $f'(1) = 3(p_3 - p_2)$. The second derivative at the beginning of the curve can be determined from control points p_0 , p_1 and p_2 .

Using the facts about Bézier cubics in the preceding paragraph, we can use the methods of Section 15.5 to create a parametric function for them. The definitions

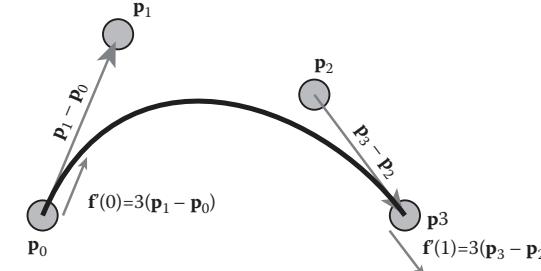


Figure 15.10. A cubic Bézier curve is controlled by four points. It interpolates the first and last, and the beginning and final derivatives are three times the vectors between the first two (or last two) points.

trator和字体表示方案（如Postscript中使用的方案）使用三次贝塞尔曲线。贝塞尔曲线在计算机图形学中非常受欢迎，因为它们易于控制，具有许多有用的属性，并且有非常有效的算法来处理它们。贝塞尔曲线的构造使得：

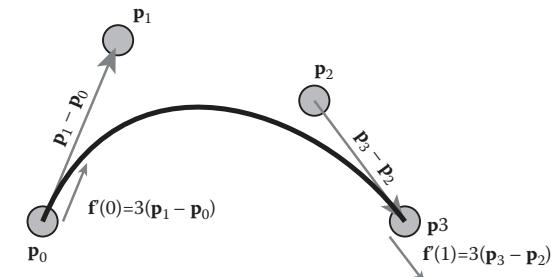
*曲线内插第一个和最后一个控制点，分别为 $u=0$ 和 1 。

*曲线在其开始（结束）处的一阶导数由第一个和第二个（紧挨着最后一个和最后一个）控制点之间的矢量确定。导数由这些点之间的矢量按曲线的程度缩放给出。

*曲线开始（结束）处的较高导数取决于曲线开始（结束）处的点。第 n 个导数取决于第一个（最后一个） $n+1$ 点。

例如，考虑3度（立方）的贝塞尔曲线，如图15.10所示。曲线有四个($d+1$)控制点。它从第一个控制点(p_0)开始，到最后一个(p_1)结束。开始时的一阶导数与第一和第二控制点(p_1 p_0)之间的矢量成比例。具体而言， $f'(0)=3(p_1-p_0)$ 。类似地，曲线末端的一阶导数由 $f'(1)=3(p_3-p_2)$ 给出。曲线开始处的二阶导数可以从控制点 p_0 、 p_1 和 p_2 确定。

利用上一段关于贝塞尔立方体的事实，我们可以使用第15.5节的方法为它们创建一个参数函数。定义



三次贝塞尔曲线由四个点控制。它插值第一个和最后一个，开始和最后一个导数是前两个（或最后两个）点之间矢量的三倍。



of the beginning and end interpolation and derivatives give

$$\begin{aligned} \mathbf{p}_0 &= \mathbf{f}(0) = \mathbf{a}_3 0^3 + \mathbf{a}_2 0^2 + \mathbf{a}_1 0 + \mathbf{a}_0, \\ \mathbf{p}_3 &= \mathbf{f}(1) = \mathbf{a}_3 1^3 + \mathbf{a}_2 1^2 + \mathbf{a}_1 1 + \mathbf{a}_0, \\ 3(\mathbf{p}_1 - \mathbf{p}_0) &= \mathbf{f}'(0) = 3\mathbf{a}_3 0^2 + 2\mathbf{a}_2 0 + \mathbf{a}_1, \\ 3(\mathbf{p}_3 - \mathbf{p}_2) &= \mathbf{f}'(1) = 3\mathbf{a}_3 1^2 + 2\mathbf{a}_2 1 + \mathbf{a}_1. \end{aligned}$$

This can be solved for the basis matrix

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix},$$

and then written as

$$\mathbf{f}(u) = (1 - 3u + 3u^2 - u^3)\mathbf{p}_0 + (3u - 6u^2 + 3u^3)\mathbf{p}_1 + (3u^2 - 3u^3)\mathbf{p}_2 + (u^3)\mathbf{p}_3,$$

or

$$\mathbf{f}(u) = \sum_{i=0}^d b_{i,3} \mathbf{p}_i,$$

where the $b_{i,3}$ are the Bézier blending functions of degree 3:

$$\begin{aligned} b_{0,3} &= (1-u)^3, \\ b_{1,3} &= 3u(1-u)^2, \\ b_{2,3} &= 3u^2(1-u), \\ b_{3,3} &= u^3. \end{aligned}$$

Fortunately, the blending functions for Bézier curves have a special form that works for all degrees. These functions are known as the *Bernstein basis polynomials* and have the general form

$$b_{k,n}(u) = C(n, k) u^k (1-u)^{(n-k)},$$

where n is the order of the Bézier curve, and k is the blending function number between 0 and n (inclusive). $C(n, k)$ are the binomial coefficients:

$$C(n, k) = \frac{n!}{k! (n-k)!}.$$

Given the positions of the control points \mathbf{p}_k , the function to evaluate the Bézier curve of order n (with $n+1$ control points) is

$$\mathbf{p}(u) = \sum_{k=0}^n p_k C(n, k) u^k (1-u)^{(n-k)}.$$

Some Bézier segments are shown in Figure 15.11.



开始和结束插值和导数给出

$$\begin{aligned} \mathbf{p}_0 &= \mathbf{f}(0) = \mathbf{a}_3 0^3 + \mathbf{a}_2 0^2 + \mathbf{a}_1 0 + \mathbf{a}_0, \\ \mathbf{p}_3 &= \mathbf{f}(1) = \mathbf{a}_3 1^3 + \mathbf{a}_2 1^2 + \mathbf{a}_1 1 + \mathbf{a}_0, \\ 3(\mathbf{p}_1 - \mathbf{p}_0) &= \mathbf{f}'(0) = 3\mathbf{a}_3 0^2 + 2\mathbf{a}_2 0 + \mathbf{a}_1, \\ 3(\mathbf{p}_3 - \mathbf{p}_2) &= \mathbf{f}'(1) = 3\mathbf{a}_3 1^2 + 2\mathbf{a}_2 1 + \mathbf{a}_1. \end{aligned}$$

这可以为基矩阵求解

$$\mathbf{B} = \mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix},$$

然后写成

$$\mathbf{f}(u) = (1 - 3u + 3u^2 - u^3)\mathbf{p}_0 + (3u - 6u^2 + 3u^3)\mathbf{p}_1 + (3u^2 - 3u^3)\mathbf{p}_2 + (u^3)\mathbf{p}_3,$$

or

$$\mathbf{f}(u) = \sum_{i=0}^d b_{i,3} \mathbf{p}_i,$$

其中 $b_{i,3}$ 是度3的贝塞尔混合函数:

$$\begin{aligned} b_{0,3} &= (1-u)^3, \\ b_{1,3} &= 3u(1-u)^2, \\ b_{2,3} &= 3u^2(1-u), \\ b_{3,3} &= u^3. \end{aligned}$$

幸运的是，贝塞尔曲线的混合函数具有适用于所有度数的特殊形式。这些函数被称为伯恩斯坦基多项式，具有一般形式

$$b_{k,n}(u) = C(n, k) u^k (1-u)^{(n-k)},$$

其中 n 是贝塞尔曲线的阶数， k 是 0 到 n (含) 之间的混合函数数。 $C(n, k)$ 是二项式系数:

$$C(n, k) = \frac{n!}{k! (n-k)!}.$$

给定控制点 p_k 的位置，评估阶数 n (具有 $n+1$ 个控制点) 的贝塞尔曲线的函数为

$$\mathbf{p}(u) = \sum_{k=0}^n p_k C(n, k) u^k (1-u)^{(n-k)}.$$

一些贝塞尔曲线段如图 15.11 所示。

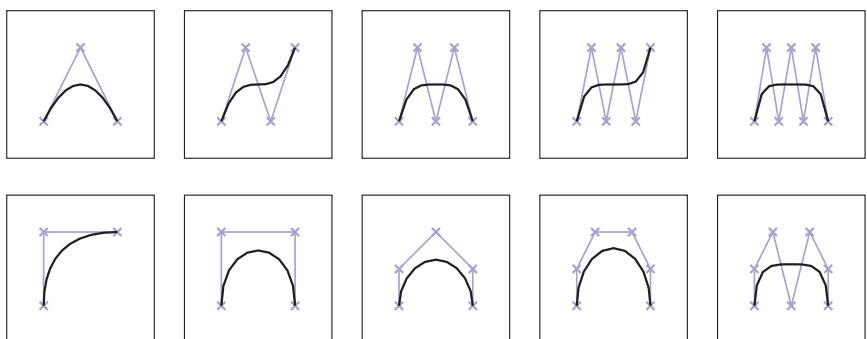
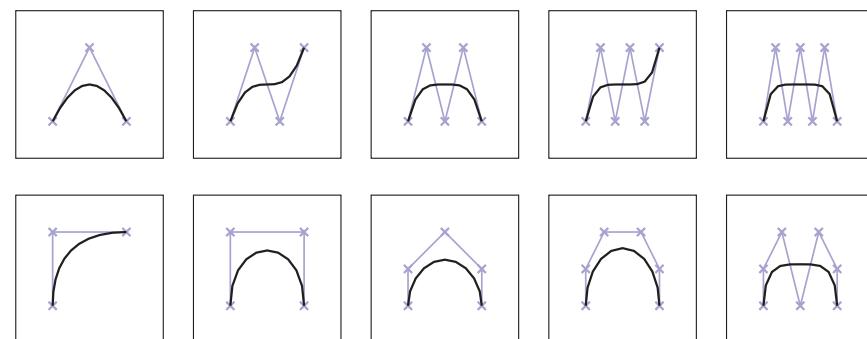


Figure 15.11. Various Bézier segments of degree 2–6. The control points are shown with crosses, and the control polygons (line segments connecting the control points) are also shown.

Bézier segments have several useful properties:

- The curve is bounded by the convex hull of the control points.
- Any line intersects the curve no more than it intersects the set of line segments connecting the control points. This is called the *variation diminishing* property. This property is illustrated in Figure 15.12.
- The curves are symmetric: reversing the order of the control points yields the same curve, with a reversed parameterization.
- The curves are *affine invariant*. This means that translating, scaling, rotating, or skewing the control points is the same as performing those operations on the curve itself.
- There are good simple algorithms for evaluating and subdividing Bézier curves into pieces that are themselves Bézier curves. Because subdivision



度2–6的各种贝塞尔分段。控制点用交叉示出，并且控制多边形（连接控制点的线段）也被示出。

贝塞尔线段有几个有用的属性：

- *曲线以控制点的凸包为界。
- *任何线与曲线相交的次数不超过与连接控制点的线段集合相交的次数。这称为变异递减属性。该属性如图15.12所示。
- *曲线是对称的：反转控制点的顺序会产生相同的曲线，并使用反向参数化。
- *曲线是仿射不变的。这意味着平移、缩放、旋转或倾斜控制点与对曲线本身执行这些操作相同。
- 有很好的简单算法来评估和细分贝塞尔曲线本身就是贝塞尔曲线的部分。因为细分

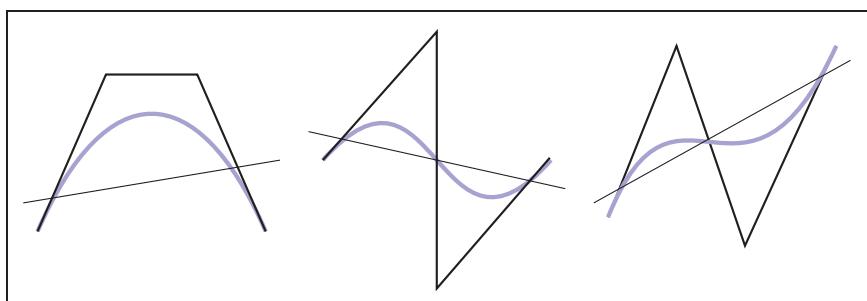


Figure 15.12. The *variation diminishing* property of Bézier curves means that the curve does not cross a line more than its control polygon does. Therefore, if the control polygon has no “wiggles,” the curve will not have them either. B-splines (Section 15.6.2) also have this property.

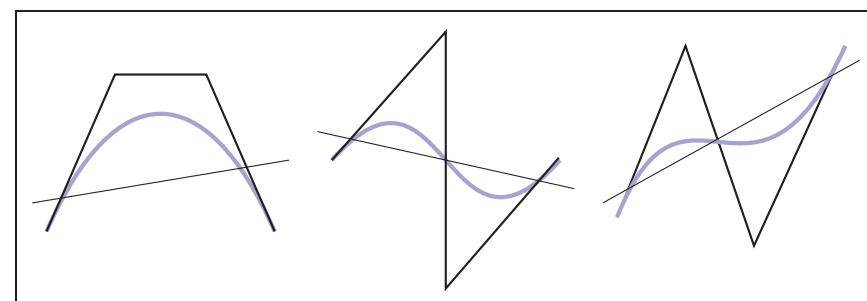


图15.12. 贝塞尔曲线的变异递减属性意味着曲线不会比其控制多边形更多地穿过一条线。因此，如果控制多边形没有“摆动”，则曲线也不会有它们。B-样条（第15.6.2节）也具有此属性。



can be done effectively using the algorithm described later, a divide and conquer approach can be used to create effective algorithms for important tasks such as rendering Bézier curves, approximating them with line segments, and determining the intersection between two curves.

When Bézier segments are connected together to make a spline, connectivity between the segments is created by sharing the endpoints. However, continuity of the derivatives must be created by positioning the other control points. This provides the user of a Bézier spline with control over the smoothness. For C^1 continuity, the second-to-last point of the first curve and the second point of the second curve must be collinear with the equated endpoints. For C^1 continuity, the distances between the points must be equal as well. This is illustrated in Figure 15.13. Higher degrees of continuity can be created by properly positioning more points.

Geometric Intuition for Bézier Curves

Bézier curves can be derived from geometric principles, as well as from the algebraic methods described above. We outline the geometric principles because they provide intuition on how Bézier curves work.

Imagine that we have a set of control points from which we want to create a smooth curve. Simply connecting the points with lines (to form the control polygon) will lead to something that is non-smooth. It will have sharp corners. We could imagine “smoothing” this polygon by cutting off the sharp corners, yielding a new polygon that is smoother, but still not “smooth” in the mathematical sense (since the curve is still a polygon, and therefore only C^1). We can repeat this process, each time yielding a smoother polygon, as shown in Figure 15.14. In the limit, that is if we repeated the process infinitely many times, we would obtain a C^1 smooth curve.

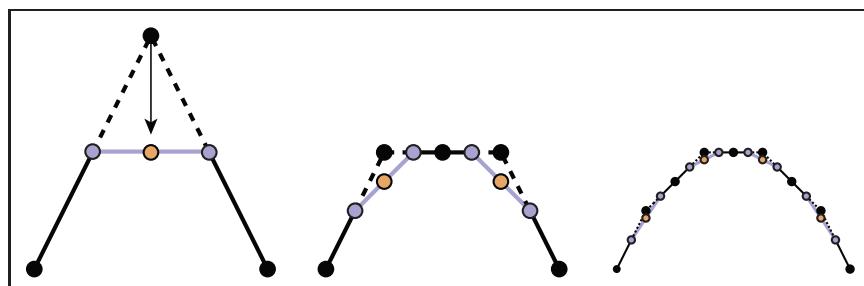


Figure 15.14. Subdivision procedure for quadratic Béziers. Each line segment is divided in half and these midpoints are connected (blue points and lines). The interior control point is moved to the midpoint of the new line segment (orange point).



可以使用后面描述的算法有效地完成，分而治之的方法可以用于为重要任务创建有效的算法，例如渲染贝塞尔曲线，用线段近似它们，并确定两条曲线之

当贝塞尔线段连接在一起形成样条曲线时，线段之间的连通性是通过共享端点来创建的。但是，必须通过定位其他控制点来创建导数的连续性。这为贝塞尔样条曲线的用户提供了对平滑度的控制。对于 G^1

连续性，第一条曲线的倒数第二点和第二条曲线必须与等效端点共线。对于 C^1 连续性，点之间的距离也必须相等。这在图15.13中说明。通过正确定位更多点，可以创建更高程度的连续性。

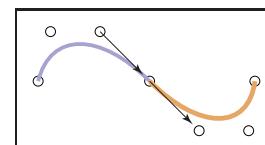
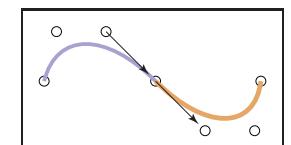


Figure 15.13. Two Bézier segments connect to form a C^1 spline, because the vector between the last two points of the first segment is equal to the vector between the first two points of the second segment.

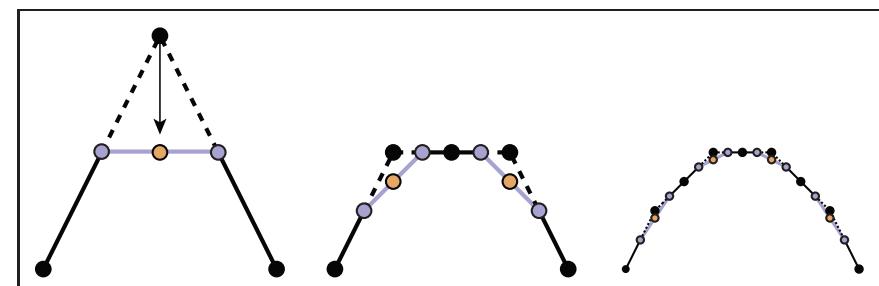


两个贝塞尔线段连接形成 C^1 样条曲线，因为第一线段的最后两点之间的矢量等于第二线段的前两点之间的矢量。

贝塞尔曲线的几何直觉

贝塞尔曲线可以从几何原理推导出来，也可以从上面描述的algebraic方法推导出来。我们概述了几何原理，因为它们提供了贝塞尔曲线如何工作的直觉。

想象一下，我们有一组控制点，我们想从中创建一个平滑的曲线。简单地将点与线连接起来（形成控制多边形）会导致一些不平滑的东西。它会有尖角。我们可以想象通过切断尖角来“平滑”这个多边形，产生一个更平滑的新多边形，但在数学意义上仍然不“平滑”（因为曲线仍然是多边形，因此只有 C^1 ）。我们可以重复这个过程，每次产生一个更平滑的多边形，如图15.14所示。在极限中，即如果我们无限多次重复该过程，我们将获得 C^1 平滑曲线。



二次贝塞尔的细分程序。每个线段被分成两半并连接这些中点（蓝色点和线）。内部控制点移动到新线段的中点（橙色点）。

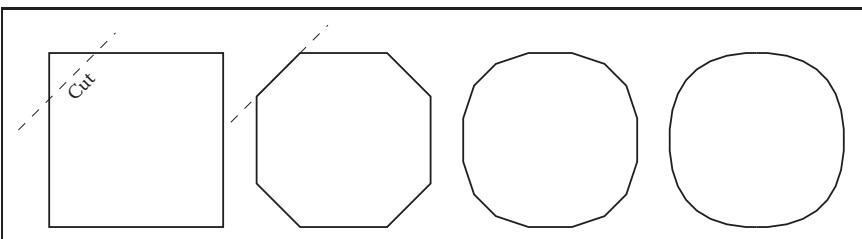


Figure 15.15. By repeatedly cutting the corners off a polygon, we approach a smooth curve.

What we have done with corner cutting is defining a *subdivision scheme*. That is, we have defined curves by a process for breaking a simpler curve into smaller pieces (e.g., subdividing it). The resulting curve is the *limit curve* that is achieved by applying the process infinitely many times. If the subdivision scheme is defined correctly, the result will be a smooth curve, and it will have a parametric form.

Let us consider applying corner cutting to a single corner. Given three points (\mathbf{p}_0 , \mathbf{p}_1 , \mathbf{p}_2), we repeatedly “cut off the corners” as shown in Figure 15.15. At each step, we divide each line segment in half, connect the midpoints, and then move the corner point to the midpoint of the new line segment. Note that in this process, new points are introduced, moved once, and then remain in this position for any remaining iterations. The endpoints never move.

If we compute the “new” position for \mathbf{p}_2 as the midpoint of the midpoints, we get the expression

$$\mathbf{p}'_2 = \frac{1}{2}\left(\frac{1}{2}\mathbf{p}_0 + \frac{1}{2}\mathbf{p}_1\right) + \frac{1}{2}\left(\frac{1}{2}\mathbf{p}_1 + \frac{1}{2}\mathbf{p}_2\right).$$

The construction actually works for other proportions of distance along each segment. If we let u be the distance between the beginning and the end of each segment where we place the middle point, we can rewrite this expression as

$$\mathbf{p}(u) = (1-u)((1-u)\mathbf{p}_0 + u\mathbf{p}_1) + u((1-u)\mathbf{p}_1 + u\mathbf{p}_2).$$

Regrouping terms gives the quadratic Bézier function:

$$\mathbf{B}_2(u) = (1-u)^2\mathbf{p}_0 + 2u(1-u)\mathbf{p}_1 + u^2\mathbf{p}_2.$$

The de Casteljau Algorithm

One nice feature of Bézier curves is that there is a very simple and general method for computing and subdividing them. The method, called the *de Casteljau algorithm*, uses a sequence of linear interpolations to compute the positions along the

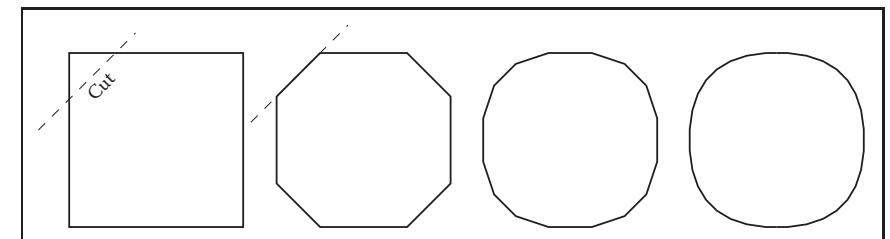


图15.15。通过反复切割多边形的角 我们接近一个平滑的曲线。

我们所做的角切割是定义一个细分方案。也就是说，我们通过将更简单的曲线分解为更小的部分（例如，细分它）的过程来定义曲线。所得曲线是通过无限多次应用该过程而实现的极限曲线。如果细分方案是正确的，结果将是一个平滑的曲线，它将有一个参数形式。

让我们考虑将角切割应用于单个角。给定三个点 (\mathbf{p}_0 , \mathbf{p}_1 , \mathbf{p}_2)，我们反复“切断角落”，如图15.15所示。在每个步骤中，我们将每个线段分成两半，连接中点，然后将角点移动到新线段的中点。请注意，在此过程中，新点被引入，移动一次，然后保留在该位置以进行任何剩余的迭代。端点永远不会移动。

如果我们计算 \mathbf{p}_2 的“新”位置作为中点的中点，我们得到表达式

$$\mathbf{p}'_2 = \frac{1}{2}\left(\frac{1}{2}\mathbf{p}_0 + \frac{1}{2}\mathbf{p}_1\right) + \frac{1}{2}\left(\frac{1}{2}\mathbf{p}_1 + \frac{1}{2}\mathbf{p}_2\right).$$

该结构实际上适用于沿每个段的其他比例的距离。如果我们让 u 是我们放置中间点的每个线段的开始和结束之间的距离，我们可以将此表达式重写为

$$\mathbf{p}(u) = (1-u)((1-u)\mathbf{p}_0 + u\mathbf{p}_1) + u((1-u)\mathbf{p}_1 + u\mathbf{p}_2).$$

重新组合项给出二次贝塞尔函数：

$$\mathbf{B}_2(u) = (1-u)^2\mathbf{p}_0 + 2u(1-u)\mathbf{p}_1 + u^2\mathbf{p}_2.$$

DeCasteljau算法

贝塞尔曲线的一个很好的特点是有一个非常简单和通用的方法来计算和细分它们。该方法称为deCasteljau算法，使用一系列线性插值来计算沿着

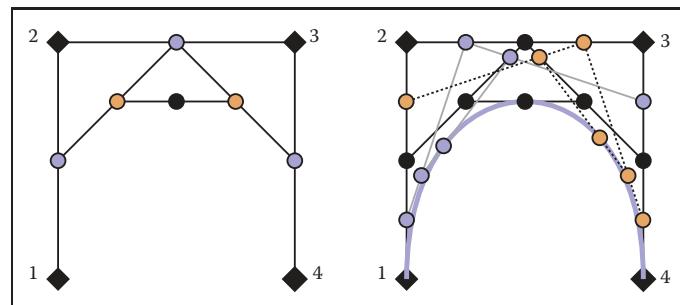


Figure 15.16. An illustration of the de Casteljau algorithm for a cubic Bézier. The left-hand image shows the construction for $u = 0.5$. The right-hand image shows the construction for 0.25, 0.5, and 0.75.

Bézier curve of arbitrary order. It is the generalization of the subdivision scheme described in the previous section.

The de Casteljau algorithm begins by connecting every adjacent set of points with lines, and finding the point on these lines that is the u interpolation, giving a set of $n - 1$ points. These points are then connected with straight lines, those lines are interpolated (again by u), giving a set of $n - 2$ points. This process is repeated until there is one point. An illustration of this process is shown in Figure 15.16.

The process of computing a point on a Bézier segment also provides a method for dividing the segment at the point. The intermediate points computed during the de Casteljau algorithm form the new control points of the new, smaller segments, as shown in Figure 15.17.

The existence of a good algorithm for dividing Bézier curves makes divide-and-conquer algorithms possible. For example, when drawing a Bézier curve

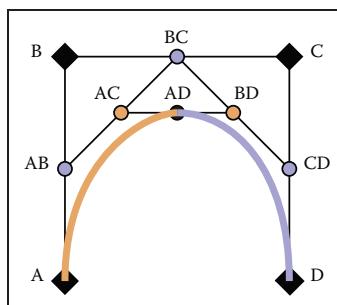


Figure 15.17. The de Casteljau algorithm is used to subdivide a cubic Bézier segment. The initial points (black diamonds A, B, C, and D) are linearly interpolated to yield blue circles (AB, BC, CD), which are linearly interpolated to yield orange circles (AC, BD), which are linearly interpolated to give the point on the cubic AD. This process also has subdivided the Bézier segment with control points A, B, C, D into two Bézier segments with control points A, AB, AC, AD and AD, BD, CD, D.

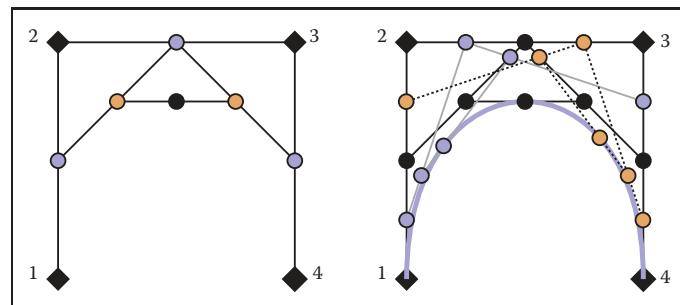
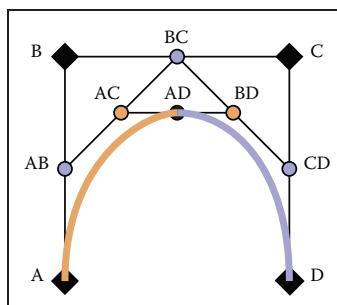


图15.16。立方贝塞尔的deCasteljau算法的说明。左图显示了 $u=0.5$ 的构造。右图显示了0.25、0.5和0.75的构造。

任意顺序的贝塞尔曲线。它是上一节中描述的细分方案的概括。

DeCasteljau算法首先用线连接每个相邻的点集，并在这些线上找到 u 插值的点，给出一组 $n-1$ 个点。然后用直线连接这些点，这些线被插值（再次由 u ），给出一组 $n-2$ 个点。重复这个过程，直到有一个点。图15.16显示了这个过程的说明。在贝塞尔线段上计算点的过程还提供了在点处划分线段的方法。在deCasteljau算法期间计算的中间点构成了新的较小seg的新控制点，如图15.17所示。

分割贝塞尔曲线的良好算法的存在使得分而治之算法成为可能。例如，绘制贝塞尔曲线时



DeCasteljau算法用于细分三次贝塞尔线段。初始点（黑钻石A, B, C和D）被线性插值以产生蓝色圆圈（AB, BC, CD），其被线性插值以产生橙色圆圈（AC, BD），其被线性插值以给出立方AD上的点。该过程还将具有控制点A, B, C, D的贝塞尔线段细分为具有控制点A, AB, AC, AD和AD, BD, CD, D的两个贝塞尔线段。

segment, it is easy to check if the curve is close to being a straight line because it is bounded by its convex hull. If the control points of the curve are all close to being colinear, the curve can be drawn as a straight line. Otherwise, the curve can be divided into smaller pieces, and the process can be repeated. Similar algorithms can be used for determining the intersection between two curves. Because of the existence of such algorithms, other curve representations are often converted to Bézier form for processing.

15.6.2 B-Splines

B-splines provide a method for approximating a set of n points with a curve made up of polynomials of degree d that gives $C^{(d-1)}$ continuity. Unlike the Bézier splines of the previous section, B-splines allow curves to be generated for any desired degree of continuity (almost up to the number of points). Because of this, B-splines are a preferred way to specify very smooth curves (high degrees of continuity) in computer graphics. If we want a C^2 or higher curve through an arbitrary number of points, B-splines are probably the right method.

We can represent a curve using a linear combination of B-spline basis functions. Since these basis functions are themselves splines, we call them basis splines or B-splines for short. Each B-spline or basis function is made up of a set of $d + 1$ polynomials each of degree d . The methods of B-splines provide general procedures for defining these functions.

The term B-spline specifically refers to one of the basis functions, not the function created by the linear combination of a set of B-splines. However, there is inconsistency in how the term is used in computer graphics. Commonly, a “B-spline curve” is used to mean a curve represented by the linear combination of B-splines.

The idea of representing a polynomial as the linear combination of other polynomials has been discussed in Section 15.3.1 and 15.3.5. Representing a spline as a linear combination of other splines was shown in Section 15.4.1. In fact, the example given is a simple case of a B-spline.

The general notation for representing a function as a linear combination of other functions is

$$\mathbf{f}(t) = \sum_{i=1}^n \mathbf{p}_i b_i(t), \quad (15.15)$$

where the \mathbf{p}_i are the coefficients and the b_i are the basis functions. If the coefficients are points (e.g., 2 or 3 vectors), we refer to them as control points. The key to making such a method work is to define the b_i appropriately. B-splines provide a very general way to do this.

段，很容易检查曲线是否接近是一条直线，因为它以其凸包为界。如果曲线的控制点都接近共线，则曲线可以绘制为直线。否则，可以将曲线分成更小的块，并且可以重复该过程。类似的算法可用于确定两条曲线之间的交点。由于这类算法的存在，其他曲线表示往往被转换为贝塞尔形式进行处理。

15.6.2 B-Splines

B样条提供了一种方法，用于用由 d 度的多项式组成的曲线来近似一组 n 点，该曲线赋予 $C^{(d-1)}$ 连续性。与上一节的贝塞尔样条不同，B样条允许为任何所需的连续性程度（几乎高达点的数量）生成曲线。因此，B样条是在计算机图形学中指定非常平滑曲线（高度连续性）的首选方法。如果我们希望通过任意数量的点获得 C^2 或更高的曲线，B样条可能是正确的方法。

我们可以用B样条基函数的线性组合来表示一条曲线。由于这些基函数本身就是样条，我们称之为基样条或简称B样条。每个B样条或基函数由一组 $d+1$ 多项式组成，每个多项式为度 d 。B样条的方法提供了定义这些函数的一般程序。

术语B样条特指基函数之一，而不是由一组B样条的线性组合创建的函数。然而，该术语在计算机图形学中的使用方式存在不一致性。通常，“B样条曲线”用于表示由B样条的线性组合表示的曲线。

将多项式表示为其他多项命名法的线性组合的想法已在第15.3.1和15.3.5节中讨论过。将样条表示为其他样条的线性组合见第15.4.1节。实际上，给出的例子是B样条的简单情况。

将函数表示为其他函数的线性组合的一般表示法是

$$\mathbf{f}(t) = \sum_{i=1}^n \mathbf{p}_i b_i(t), \quad (15.15)$$

其中 p_i 是系数， b_i 是基函数。如果系数是点（例如，2或3个矢量），我们将它们称为控制点。使这种方法起作用的关键是适当地定义 b_i 。B样条提供了一个非常一般的方法来做到这一点。

A set of B-splines can be defined for a number of coefficients n and a parameter value k .³ The value of k is one more than the degree of the polynomials used to make the B-splines ($k = d + 1$.)

B-splines are important because they provide a very general method for creating functions (that will be useful for representing curves) that have a number of useful properties. A curve with n points made with B-splines with parameter value k :

- is $C^{(k-2)}$ continuous;
- is made of polynomials of degree $k - 1$;
- has local control—any site on the curve only depends on k of the control points;
- is bounded by the convex hull of the points;
- exhibits the variation diminishing property illustrated in Figure 15.12.

A curve created using B-splines does not necessarily interpolate its control points.

We will introduce B-splines by first looking at a specific, simple case to introduce the concepts. We will then generalize the methods and show why they are interesting. Because the method for computing B-splines is very general, we delay introducing it until we have shown what these generalizations are.

Uniform Linear B-Splines

Consider a set of basis functions of the following form:

$$b_{i,2}(t) = \begin{cases} t - i & \text{if } i \leq t < i + 1, \\ 2 - t + i & \text{if } i + 1 \leq t \leq i + 2, \\ 0 & \text{otherwise.} \end{cases} \quad (15.16)$$

Each of these functions looks like a little triangular “hat” between i and $i + 2$ with its peak at $i + 1$. Each is a piecewise polynomial, with knots at i , $i + 1$, and $i + 2$. Two of them are graphed in Figure 15.18.

Each of these functions $b_{i,2}$ is a first-degree (linear) B-spline. Because we will consider B-splines of other parameter values later, we denote these with the 2 in the subscript.

³The B-spline parameter is actually the *order* of the polynomials used in the B-splines. While this terminology is not uniform in the literature, the use of the B-spline parameter k as a value one greater than the polynomial degree is widely used, although some texts (see the chapter notes) write all of the equations in terms of polynomial degree.

可以为若干系数n和一个参数值k定义一组B样条。3k的值比用于使B样条的多项式的程度多一个 ($k=d+1$ 。)

B样条很重要，因为它们为具有许多有用属性的creating函数（这对于表示曲线很有用）提供了非常通用的方法。用参数值为k的B样条制成的具有n个点的曲线：

- is $C^{(k-2)}$ continuous;
- 是由度 $k - 1$ 的多项式;
- 具有局部控制—曲线上任何站点仅取决于 k 个控制点;
- 以点的凸包为界;
- 显示图 15.12 所示的变异递减性质。

使用b样条创建的曲线不一定对其控制点进行插值。我们将首先通过一个具体的、简单的案例来介绍B样条来介绍这些概念。然后，我们将概括这些方法，并说明它们为什么有趣。因为计算B样条的方法非常通用，所以我们推迟引入它，直到我们展示了这些概括是什么。

均匀线性B样条

考虑以下形式的一组基函数：

$$b_{i,2}(t) = \begin{cases} t - i & \text{if } i \leq t < i + 1, \\ 2 - t + i & \text{if } i + 1 \leq t \leq i + 2, \\ 0 & \text{otherwise.} \end{cases} \quad (15.16)$$

这些函数中的每一个看起来都像 i 和 $i + 2$ 之间的一个小三角形“帽子”，其峰值在 $i + 1$ 。每个都是分段多项式，在 i , $i + 1$ 和 $i + 2$ 处有结。其中两个图表如图 15.18 所示。

这些函数 $b_{i,2}$ 中的每一个都是一级（线性）B样条。因为我们稍后会考虑其他参数值的B样条，所以我们用下标中的2来表示它们。

3B样条参数实际上是B样条中使用的多项式的阶数。虽然这一术语在文献中并不统一，但使用B样条参数k作为大于多项式度的值被广泛使用，尽管一些文本（见章节注释）用多项式度写出了所有的方程。

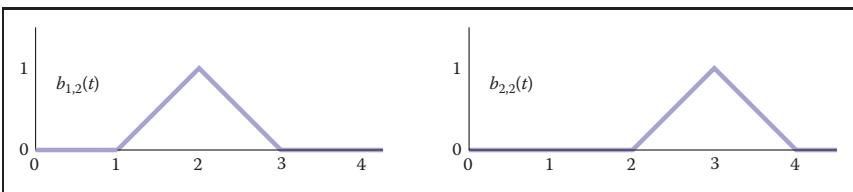


Figure 15.18. B-splines with $d = 1$ or $k = 2$.

Notice that we have chosen to put the lower edge of the B-spline (its first knot) at i . Therefore, the first knot of the first B-spline ($i = 1$) is at 1. Iteration over the B-splines or elements of the coefficient vector is from 1 to n (see Equation 15.15). When B-splines are implemented, as well as in many other discussions of them, they often are numbered from 0 to $n - 1$.

We can create a function from a set of n control points using Equation 15.15, with these functions used for the b_i to create an “overall function” that was influenced by the coefficients. If we were to use these ($k = 2$) B-splines to define the overall function, we would define a piecewise polynomial function that linearly interpolates the coefficients p_i between $t = k$ and $t = n + 1$. Note that while ($k = 2$) B-splines interpolate all of their coefficients, B-splines of higher degree do this under some specific conditions that we will discuss in Section 15.6.3.

Some properties of B-splines can be seen in this simple case. We will write these in the general form using k , the parameter, and n for the number of coefficients or control points:

- Each B-spline has $k + 1$ knots.
- Each B-spline is zero before its first knot and after its last knot.
- The overall spline has local control because each coefficient is only multiplied by one B-spline, and this B-spline is nonzero only between $k + 1$ knots.
- The overall spline has $n + k$ knots.
- Each B-spline is $C^{(k-2)}$ continuous, therefore the overall spline is $C^{(k-2)}$ continuous.
- The set of B-splines sums to 1 for all parameter values between knots k and $n + 1$. This range is where there are k B-splines that are nonzero. Summing to 1 is important because it means that the B-splines are shift invariant: translating the control points will translate the entire curve.

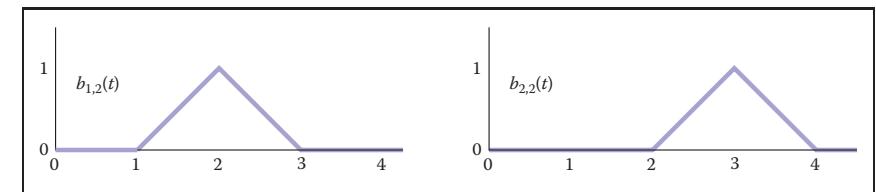


图15.18。B-具有d=1或k=2的样条。

请注意，我们选择将B样条的下边缘（它的第一个结）放在 i 处。因此，第一个b样条 ($i=1$) 的第一个结在1。在系数向量的b样条或元素上的迭代是从1到n (见公式15.15)。当实现B样条时，以及在对它们的许多其他讨论中，它们通常从0到n-1编号。

我们可以使用公式15.15从一组n个控制点创建一个函数，这些函数用于 b_i 创建一个由系数膨胀的“整体函数”。如果我们使用这些 ($k=2$) b样条来定义整体函数，我们将定义一个分段多项式函数，该函数在 $t=k$ 和 $t=n+1$ 之间线性插值系数 p_i 。请注意，虽然($k=2$)B-样条插值其所有系数，但更高程度的B-样条在我们将在第15.6.3节中讨论的某些特定条件下执行此操作。

在这个简单的例子中可以看到B样条的一些属性。我们将使用 k 、参数和 n 作为系数或控制点的数量，以一般的形式写这些：

- 每个B样条具有 $k+1$ 节。
- 每个b样条在其第一个结之前和最后一个结之后为零。
- *整体样条具有局部控制，因为每个系数仅乘以一个B样条，并且此B样条仅在 $k+1$ 节之间非零。
- *整体样条有 $n+k$ 个结。
- 每个B样条是 $C^{(k-2)}$ 连续的，因此整体样条是 $C^{(k-2)}$ 连续的。
- *对于结 k 和 $n+1$ 之间的所有参数值，B样条的集合总和为1。这个范围是存在非零的 k 个B样条的地方。求和到1很重要，因为这意味着B样条线是移位不变的：平移控制点将平移整个曲线。

- Between each of its knots, the B-spline is a single polynomial of degree $d = k - 1$. Therefore, the overall curve (that sums these together) can also be expressed as a single, degree d polynomial between any adjacent knots.

In this example, we have chosen the knots to be uniformly spaced. We will consider B-splines with nonuniform spacing later. When the knot spacing is uniform, each of the B-splines is identical except for being shifted. B-splines with uniform knot spacing are sometimes called *uniform B-splines* or *periodic B-splines*.

Uniform Quadratic B-Splines

The properties of B-splines listed in the previous section were intentionally written for arbitrary n and k . A general procedure for constructing the B-splines will be provided later, but first, let's consider another specific case with $k = 3$.

The B-spline $b_{2,3}$ is shown in Figure 15.19. It is made of quadratic pieces (degree 2), and has three of them. It is C^1 continuous and is nonzero only within the four knots that it spans. Notice that a quadratic B-spline is made of three pieces, one between knot 1 and 2, one between knot 2 and 3, and one between knot 3 and 4. In Section 15.6.3 we will see a general procedure for building these functions. For now, we simply examine these functions:

$$b_{i,3}(t) = \begin{cases} \frac{1}{2}u^2 & \text{if } i \leq t < i+1 \quad u = t - i, \\ -u^2 + u + \frac{1}{2} & \text{if } i+1 \leq t < i+2 \quad u = t - (i+1), \\ \frac{1}{2}(1-u)^2 & \text{if } i+2 \leq t < i+3 \quad u = t - (i+2), \\ 0 & \text{otherwise.} \end{cases} \quad (15.17)$$

In order to make the expressions simpler, we wrote the function for each part as if it applied over the range 0 to 1.

If we evaluate the overall function made from summing together the B-splines, at any time only k (3 in this case) of them are nonzero. One of them will be in the first part of Equation 15.17, one will be in the second part, and one will be in the third part. Therefore, we can think of any piece of the overall function as being

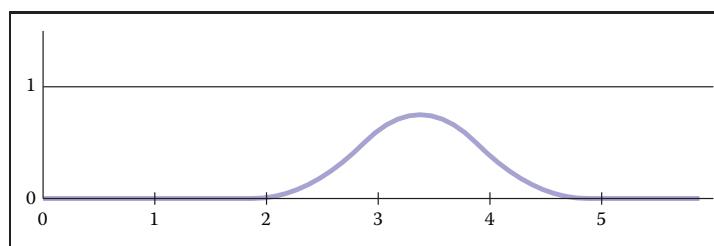


Figure 15.19. The B-spline $b_{2,3}$ with uniform knot spacing.

- 在其每个结之间，B样条是度 $d=k-1$ 的单多项式。因此，整体曲线（将这些求和在一起）也可以表示为任何相邻结之间的单个， d 度多项式。

在这个例子中，我们选择了均匀间隔的结。我们稍后将考虑具有非均匀间距的B样条。当结间距均匀时，每个B样条除了被移位之外是相同的。具有均匀结间距的B样条有时被称为均匀B样条或周期性B样条。

均匀二次B样条

上一节中列出的B样条的属性对于任意 n 和 k 有意为十。稍后将提供构建B样条的一般过程，但首先，让我们考虑 $k=3$ 的另一种特定情况。

B样条 $b_{2,3}$ 如图15.19所示。它是由二次件（度2），并有三个。它是 C^1 连续的，并且仅在它跨越的四个结内非零。请注意，二次B样条由三个部分组成，一个在结1和2之间，一个在结2和3之间，一个在结3和4之间。在第15.6.3节中，我们将看到构建这些功能的一般程序。现在，我们简单地检查这些函数：

$$b_{i,3}(t) = \begin{cases} \frac{1}{2}u^2 & \text{if } i \leq t < i+1 \quad u = t - i, \\ -u^2 + u + \frac{1}{2} & \text{if } i+1 \leq t < i+2 \quad u = t - (i+1), \\ \frac{1}{2}(1-u)^2 & \text{if } i+2 \leq t < i+3 \quad u = t - (i+2), \\ 0 & \text{otherwise.} \end{cases} \quad (15.17)$$

为了使表达式更简单，我们为每个部分编写了函数，就好像它在0到1的范围内应用一样。

如果我们评估由B样条求和而成的整体函数，则在任何时候，它们中只有 k （在本例中为3）非零。其中一个将在等式15.17的第一部分中，一个将在第二部分中，一个将在第三部分中。因此，我们可以认为整体功能的任何部分都是

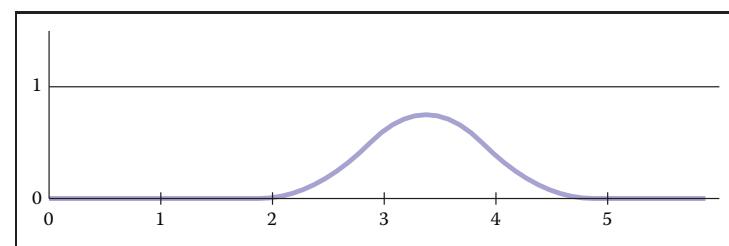


图15.19。该样条 $b_{2,3}$ 具有均匀的结间距。

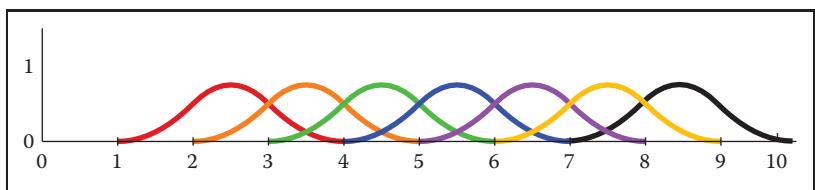


Figure 15.20. The set of seven B-splines with $k = 3$ and uniform knot spacing $[1, 2, 3, 4, 5, 6, 7, 8, 10]$.

made up of a degree $d = k - 1$ polynomial that depends on k coefficients. For the $k = 3$ case, we can write

$$\mathbf{f}(u) = \frac{1}{2}(1-u)^2 \mathbf{p}_i + (-u^2 + u + \frac{1}{2})\mathbf{p}_{i+1} + \frac{1}{2}u^2 \mathbf{p}_{i+2}$$

where $u = t - i$. This defines the piece of the overall function when $i \leq t < i+1$.

If we have a set of n points, we can use the B-splines to create a curve. If we have seven points, we will need a set of seven B-splines. A set of seven B-splines for $k = 3$ is shown in Figure 15.20. Notice that there are $n + k$ (10) knots, that the sum of the B-splines is 1 over the range k to $n + 1$ (knots 3 through 8). A curve specified using these B-splines and a set of points is shown in Figure 15.21.

Uniform Cubic B-Splines

Because cubic polynomials are so popular in computer graphics, the special case of B-splines with $k = 4$ is sufficiently important that we consider it before discussing the general case. A B-spline of third degree is defined by four cubic polynomial pieces. The general process by which these pieces are determined is

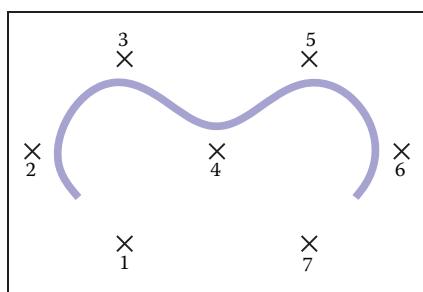


Figure 15.21. Curve made from seven quadratic ($k=3$) B-splines, using seven control points.

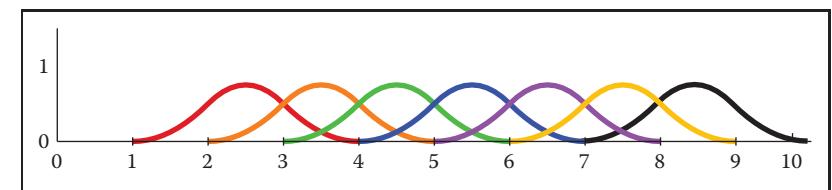


图15.20。具有 $k=3$ 和均匀结间距的七个B样条的集合[1 2 3 4 5 6 7 8 10].

由依赖于 k 个系数的度 $d=k-1$ 多项式组成。对于 $k=3$ 的情况，我们可以写

$$\mathbf{f}(u) = \frac{1}{2}(1-u)^2 \mathbf{p}_i + (-u^2 + u + \frac{1}{2})\mathbf{p}_{i+1} + \frac{1}{2}u^2 \mathbf{p}_{i+2}$$

其中 $u=t-i$ 。这定义了 $i \leq t < i+1$ 时整体函数的一块。

如果我们有一组 n 个点，我们可以使用B样条创建曲线。如果我们有七个点，我们将需要一组七个B样条。一组 $k=3$ 的七个B样条如图15.20所示。请注意，有 $n+k$ (10) 节，B样条的总和在 k 到 $n+1$ 范围内为1 (结3到8)。使用这些B样条和一组点指定的曲线如图15.21所示。

均匀立方B样条

因为三次多项式在计算机图形学中如此受欢迎，所以 $k=4$ 的B样条的特殊情况足够重要，我们在讨论一般情况之前考虑它。第三度的B样条由四个三次多项式块定义。确定这些部分的一般过程是

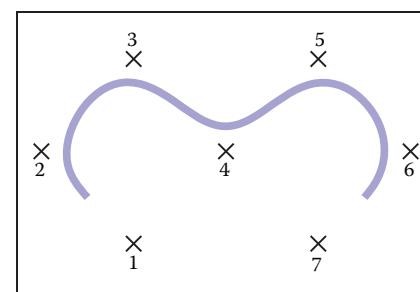
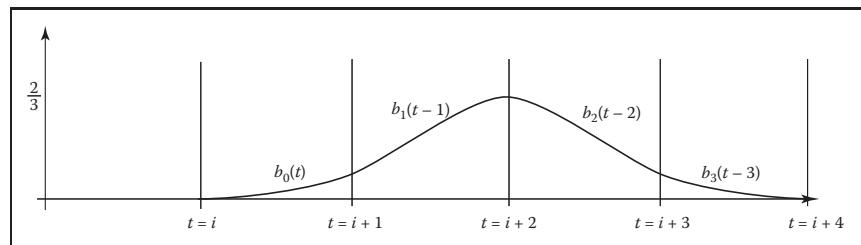


Figure 15.21. 曲线由七个二次 ($k=3$) B样条，使用七个控制点。

Figure 15.22. The cubic ($k = 4$) B-spline with uniform knots.

described later, but the result is

$$b_{i,4}(t) = \begin{cases} \frac{1}{6}u^3 & \text{if } i \leq t < i+1 \quad u = t - i, \\ \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1) & \text{if } i+1 \leq t < i+2 \quad u = t - (i+1), \\ \frac{1}{6}(3u^3 - 6u^2 + 4) & \text{if } i+2 \leq t < i+3 \quad u = t - (i+2), \\ \frac{1}{6}(-u^3 + 3u^2 - 3u + 1) & \text{if } i+3 \leq t < i+4 \quad u = t - (i+3), \\ 0 & \text{otherwise.} \end{cases} \quad (15.18)$$

This degree 3 B-spline is graphed for $i = 1$ in Figure 15.22.

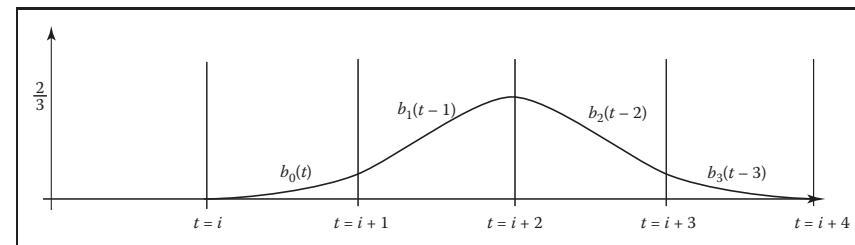
We can write the function for the overall curve between knots $i+3$ and $i+4$ as a function of the parameter u between 0 and 1 and the four control points that influence it:

$$\mathbf{f}(u) = \frac{1}{6}(-u^3 + 3u^2 - 3u + 1)\mathbf{p}_i + \frac{1}{6}(3u^3 - 6u^2 + 4)\mathbf{p}_{i+1} + \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1)\mathbf{p}_{i+2} + \frac{1}{6}u^3\mathbf{p}_{i+3}.$$

This can be rewritten using the matrix notation of the previous sections, giving a basis matrix for cubic B-splines of

$$\mathbf{M}_b = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}.$$

Unlike the matrices that were derived from constraints in Section 15.5, this matrix is created from the polynomials that are determined by the general B-spline procedure defined in the next section.

图15.22。具有均匀结的立方 ($k=4$) B样条。

后述，但结果是

$$b_{i,4}(t) = \begin{cases} \frac{1}{6}u^3 & \text{if } i \leq t < i+1 \quad u = t - i, \\ \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1) & \text{if } i+1 \leq t < i+2 \quad u = t - (i+1), \\ \frac{1}{6}(3u^3 - 6u^2 + 4) & \text{if } i+2 \leq t < i+3 \quad u = t - (i+2), \\ \frac{1}{6}(-u^3 + 3u^2 - 3u + 1) & \text{if } i+3 \leq t < i+4 \quad u = t - (i+3), \\ 0 & \text{otherwise.} \end{cases} \quad (15.18)$$

此度数3B样条曲线在图15.22中为*i=1*绘制。

我们可以将节点*i+3*和*i+4*之间的整体曲线的函数写为参数u在0和1之间以及影响它的四个控制点的函数：

$$\mathbf{f}(u) = \frac{1}{6}(-u^3 + 3u^2 - 3u + 1)\mathbf{p}_i + \frac{1}{6}(3u^3 - 6u^2 + 4)\mathbf{p}_{i+1} + \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1)\mathbf{p}_{i+2} + \frac{1}{6}u^3\mathbf{p}_{i+3}.$$

这可以使用前面部分的矩阵表示法重写，给出一个立方B样条的基矩阵

$$\mathbf{M}_b = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}.$$

与从第15.5节中的约束派生的矩阵不同，该矩阵是由下一节中定义的一般B样条过程确定的多项式创建的。

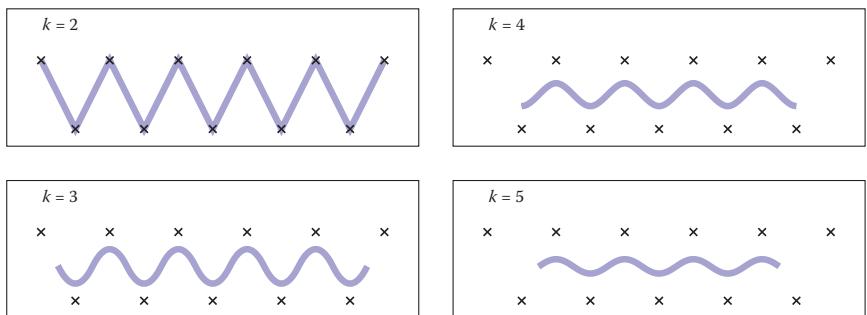


Figure 15.23. B-spline curves using the same uniform set of knots and the same control points, for various values of k . Note that as k increases, the valid parameter range for the curve shrinks.

15.6.3 Nonuniform B-Splines

One nice feature of B-splines is that they can be defined for any $k > 1$. So if we need a smoother curve, we can simply increase the value of k . This is illustrated in Figure 15.23.

So far, we have said that B-splines generalize to any $k > 1$ and any $n \geq d$. There is one last generalization to introduce before we show how to actually compute these B-splines. B-splines are defined for any non-decreasing knot vector.

For a given n and k , the set of B-splines (and the function created by their linear combination) has $n + k$ knots. We can write the value of these knots as a vector, that we will denote as t . For the uniform B-splines, the knot vector is $[1, 2, 3, \dots, n + k]$. However, B-splines can be generated for any knot vector of length $n + k$, providing the values are non-decreasing (e.g., $t_{i+1} \geq t_i$).

There are two main reasons why nonuniform knot spacing is useful: it gives us control over what parameter range of the overall function each coefficient affects, and it allows us to repeat knots (e.g., create knots with no spacing in between) in order to create functions with different properties around these points. The latter will be considered later in this section.

The ability to specify knot values for B-splines is similar to being able to specify the interpolation sites for interpolating spline curves. It allows us to associate curve features with parameter values. By specifying a nonuniform knot vector, we specify what parameter range each coefficient of a B-spline curve affects. Remember that B-spline i is nonzero only between knot i and knot $i + k$. Therefore, the coefficient associated with it only affects the curve between these parameter values.

One place where control over knot values is particularly useful is in inserting or deleting knots near the beginning of a sequence. To illustrate this, consider a

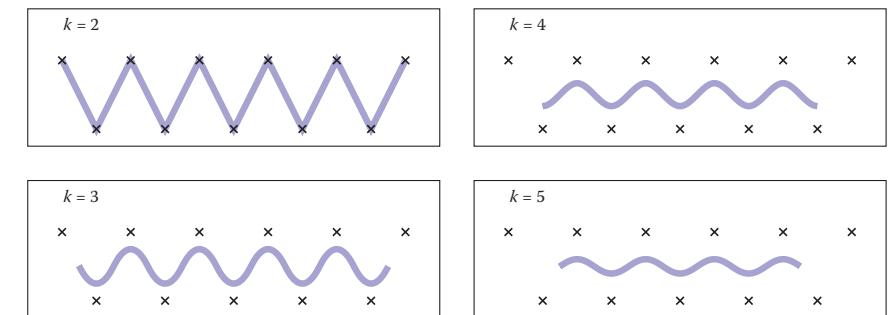


图15.23。B样条曲线使用相同的均匀结集和相同的控制点，用于k的各种值。请注意，随着k的增加，曲线的有效参数范围会缩小。

15.6.3 Nonuniform B-Splines

B样条的一个很好的特点是它们可以被定义为任何 $k>1$ 。因此，如果我们需要一个更平滑的曲线，我们可以简单地增加k的值。

到目前为止，我们已经说过B样条泛化为任何 $k>1$ 和任何 $n\geq d$ 。
在我们展示如何实际计算这些B样条之前，还有最后一个概括。对于任何非递减结向量定义B样条。对于给定的n和k，B样条的集合（以及由它们的线性组合创建的函数）具有 $n+k$ 个结。我们可以把这些结的值写成一个向量，我们将表示为t。对于均匀的B样条，结向量为 $[1 2 3 \dots, n+k]$ 。然而，可以为长度为 $n+k$ 的任何结向量生成B样条，提供值是非递减的（例如， $t_{i+1}\geq t_i$ ）。

非均匀结间距有用的主要原因有两个：它使我们能够控制每个系数影响的整体函数的参数范围，并且允许我们重复结（例如，创建之间没有间距的结），以便在这些点周围创建具有不同属性的函数。后者将在本节后面讨论。

为B样条线指定结值的能力类似于能够为插值样条曲线指定插值位点。它允许我们将曲线特征与参数值相关联。通过指定非均匀结矢量，我们指定B样条曲线的每个系数影响的参数范围。Re表示B样条线i仅在结i和结i+k之间非零，因此，与其相关的系数仅影响这些参数值之间的曲线。

控制结值特别有用的一个地方是在序列开头附近插入或删除结。为了说明这一点，考虑一个

curve defined using linear B-splines ($k = 2$) as discussed in Section 15.6.2. For $n = 4$, the uniform knot vector is $[1, 2, 3, 4, 5, 6]$. This curve is controlled by a set of four points and spans the parameter range $t = 2$ to $t = 5$. The “end” of the curve ($t = 5$) interpolates the last control point. If we insert a new point in the middle of the point set, we would need a longer knot vector. The locality properties of the B-splines prevent this insertion from affecting the values of the curve at the ends. The longer curve would still interpolate its last control point at its end. However, if we chose to keep the uniform knot spacing, the new knot vector would be $[1, 2, 3, 4, 5, 6, 7]$. The end of the curve would be at $t = 6$, and the parameter value at which the last control point is interpolated will be a different parameter value than before the insertion. With nonuniform knot spacing, we can use the knot vector $[1, 2, 3, 3.5, 4, 5, 6]$ so that the ends of the curve are unaffected by the change. The abilities to have nonuniform knot spacing makes the locality property of B-splines an algebraic property, as well as a geometric one.

We now introduce the general method for defining B-splines. Given values for the number of coefficients n , the B-spline parameter k , and the knot vector \mathbf{t} (which has length $n + k$), the following recursive equations define the B-splines:

$$b_{i,1,\mathbf{t}}(t) = \begin{cases} 1 & \text{if } \mathbf{t}_i \leq t < \mathbf{t}_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \quad (15.19)$$

$$b_{i,k,\mathbf{t}}(t) = \frac{\mathbf{t}_{i+k-1}-\mathbf{t}_i}{\mathbf{t}_{i+k}-\mathbf{t}_i} b_{i,k-1}(t) + \frac{\mathbf{t}_{i+k}-t}{\mathbf{t}_{i+k}-\mathbf{t}_{i+1}} b_{i+1,k-1}(t). \quad (15.20)$$

This equation is known as the *Cox-de Boor recurrence*. It may be used to compute specific values for specific B-splines. However, it is more often applied algebraically to derive equations such as Equation 15.17 or 15.18.

As an example, consider how we would have derived Equation 15.17. Using a uniform knot vector $[1, 2, 3, \dots]$, $t_i = i$, and the value $k = 3$ in Equation 15.20 yields

$$\begin{aligned} b_{i,3}(t) &= \frac{t-i}{(i+2)-i} b_{i,2} + \frac{(i+3)-t}{(i+3)-(i+1)} b_{i+1,2} \quad (15.21) \\ &= \frac{1}{2}(t-i)b_{i,2} + \frac{1}{2}(i+3-t)b_{i+1,2}. \end{aligned}$$

Continuing the recurrence, we must evaluate the recursive expressions:

$$\begin{aligned} b_{i,2}(t) &= \frac{t-i}{(i+2-1)-i} b_{i,1} + \frac{(i+2)-t}{(i+2)-(i+1)} b_{i+1,1} \\ &= (t-i)b_{i,1} + (i+2-t)b_{i+1,1}, \end{aligned}$$

使用线性B样条 ($k=2$) 定义的曲线, 如第15.6.2节所述。对于 $n=4$, 均匀结向量为 $[1 2 3 4 5 6]$ 。该曲线由一组四个点控制, 并跨越参数范围 $t=2$ 至 $t=5$ 。曲线 ($t=5$) 的“结束”内插最后一个控制点。如果我们在点集的中间插入一个新点, 我们将需要一个更长的结向量。B样条的局部性属性防止这种插入影响曲线两端的值。较长的曲线仍然会在其末端插入其最后一个控制点。但是, 如果我们选择保持均匀的结间距, 则新的结矢量将是 $[1 2 3 4 5 6 7]$ 。曲线的末端将在 $t=6$ 处, 并且内插最后一个控制点的参数值将是与插入之前不同的参数值。对于非均匀结间距, 我们可以使用结向量 $[1 2 3 3.5 4 5 6]$ 使得曲线的末端不受变化的影响。具有非均匀结间距的能力使B样条的局部属性成为代数属性, 也成为几何属性。

我们现在介绍定义B样条的一般方法。给定系数数 n 、B样条参数 k 和结向量 \mathbf{t} (长度为 $n+k$) 的值, 以下递归方程定义了B样条:

$$b_{i,1,\mathbf{t}}(t) = \begin{cases} 1 & \text{if } \mathbf{t}_i \leq t < \mathbf{t}_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \quad (15.19)$$

$$b_{i,k,\mathbf{t}}(t) = \frac{\mathbf{t}_{i+k-1}-\mathbf{t}_i}{\mathbf{t}_{i+k}-\mathbf{t}_i} b_{i,k-1}(t) + \frac{\mathbf{t}_{i+k}-t}{\mathbf{t}_{i+k}-\mathbf{t}_{i+1}} b_{i+1,k-1}(t). \quad (15.20)$$

这个方程被称为Cox-deBoor复发。它可用于计算特定B样条的特定值。然而, 它更经常地应用于brage来推导方程, 例如方程15.17或15.18。

举个例子 考虑我们如何推导出公式15.17. 使用均匀结向量 $[1 2 3 \dots]$, $t_i=i$, 并且等式15.20中的值 $k=3$ 。

$$\begin{aligned} b_{i,3}(t) &= \frac{t-i}{(i+2)-i} b_{i,2} + \frac{(i+3)-t}{(i+3)-(i+1)} b_{i+1,2} \quad (15.21) \\ &= \frac{1}{2}(t-i)b_{i,2} + \frac{1}{2}(i+3-t)b_{i+1,2}. \end{aligned}$$

继续重复, 我们必须评估递归表达式:

$$\begin{aligned} b_{i,2}(t) &= \frac{t-i}{(i+2-1)-i} b_{i,1} + \frac{(i+2)-t}{(i+2)-(i+1)} b_{i+1,1} \\ &= (t-i)b_{i,1} + (i+2-t)b_{i+1,1}, \end{aligned}$$

$$\begin{aligned}
 b_{i+1,2}(t) &= \frac{t - (i + 1)}{((i + 1) + 2 - 1) - (i + 1)} b_{i+1,1} \\
 &\quad + \frac{((i + 1) + 2) - t}{((i + 1) + 2) - ((i + 1) + 1)} b_{(i+1)+1,1} \\
 &= (t - i + 1)b_{i+1,1} + (i + 3 - t)b_{i+2,1}.
 \end{aligned}$$

Inserting these results into Equation 15.22 gives:

$$\begin{aligned}
 b_{i,3}(t) &= \frac{1}{2}(t - i)((t - i)b_{i,1} + (i + 2 - t)b_{i+1,1}) \\
 &\quad + \frac{1}{2}(i + 3 - t)(t - i + 1)b_{i+1,1} + (i + 3 - t)b_{i+2,1}.
 \end{aligned}$$

To see that this expression is equivalent to Equation 15.17, we note that each of the ($k = 1$) B-splines is like a switch, turning on only for a particular parameter range. For instance, $b_{i,1}$ is only nonzero between i and $i + 1$. So, if $i \leq t < i + 1$, only the first of the ($k = 1$) B-splines in the expression is nonzero, so

$$b_{i,3}(t) = \frac{1}{2}(t - i)^2 \text{ if } i \leq t < i + 1.$$

Similar manipulations give the other parts of Equation 15.17.

Repeated Knots and B-Spline Interpolation

While B-splines have many nice properties, functions defined using them generally do not interpolate the coefficients. This can be inconvenient if we are using them to define a curve that we want to interpolate a specific point. We give a brief overview of how to interpolate a specific point using B-splines here. A more complete discussion can be found in the books listed in the chapter notes.

One way to cause B-splines to interpolate their coefficients is to repeat knots. If all of the interior knots for a particular B-spline have the same value, then the overall function will interpolate this B-spline's coefficient. An example of this is shown in Figure 15.24.

Interpolation by repeated knots comes at a high cost: it removes the smoothness of the B-spline and the resulting overall function and represented curve. However, at the beginning and end of the spline, where continuity is not an issue, knot repetition is useful for creating *endpoint interpolating B-splines*. While the first (or last) knot's value is not important for interpolation, for simplicity, we make the first (or last) k knots have the same value to achieve interpolation.

Endpoint interpolating quadratic B-splines are shown in Figure 15.25. The first two and last two B-splines are different than the uniform ones. Their expres-

$$\begin{aligned}
 b_{i+1,2}(t) &= \frac{t - (i + 1)}{((i + 1) + 2 - 1) - (i + 1)} b_{i+1,1} \\
 &\quad + \frac{((i + 1) + 2) - t}{((i + 1) + 2) - ((i + 1) + 1)} b_{(i+1)+1,1} \\
 &= (t - i + 1)b_{i+1,1} + (i + 3 - t)b_{i+2,1}.
 \end{aligned}$$

将这些结果插入公式15.22给出:

$$\begin{aligned}
 b_{i,3}(t) &= \frac{1}{2}(t - i)((t - i)b_{i,1} + (i + 2 - t)b_{i+1,1}) \\
 &\quad + \frac{1}{2}(i + 3 - t)(t - i + 1)b_{i+1,1} + (i + 3 - t)b_{i+2,1}.
 \end{aligned}$$

为了看到这个表达式相当于公式15.17，我们注意到每个 ($k=1$) B样条就像一个开关，只在一个特定的参数范围内打开。例如， $b_{i,1}$ 仅在*i*和*i+1*之间非零。因此，如果*i*≤*t*<*i+1*，则表达式中只有 ($k=1$) 个B样条中的第一个非零，因此

$$b_{i,3}(t) = \frac{1}{2}(t - i)^2 \text{ if } i \leq t < i + 1.$$

类似的操作给出了等式15.17的其他部分。

重复结和B样条插值

虽然B样条具有许多不错的属性，但使用它们定义的函数generally不会插值系数。如果我们使用它们来定义要插值特定点的曲线，这可能会很不方便。我们在此简要概述了如何使用B样条插值特定点。更完整的讨论可以在章节注释中列出的书籍中找到。

导致B样条插值其系数的一种方法是重复结。

如果特定B样条的所有内部结具有相同的值，则整体函数将插值此B样条的系数。这方面的一个例子如图15.24所示。

通过重复结进行插值的代价很高：它消除了B样条的平滑ness以及由此产生的整体函数和表示曲线。然而，在样条线的开始和结束处，其中连续性不是issue，结重复对于创建端点内插B样条线是有用的。虽然第一个（或最后一个）结的值对于插值并不重要，但为了简单起见，我们使第一个（或最后一个） k 个结具有相同的值以实现插值。

端点插值二次B样条如图15.25所示。前两个和最后两个B样条与均匀样条不同。他们的表达

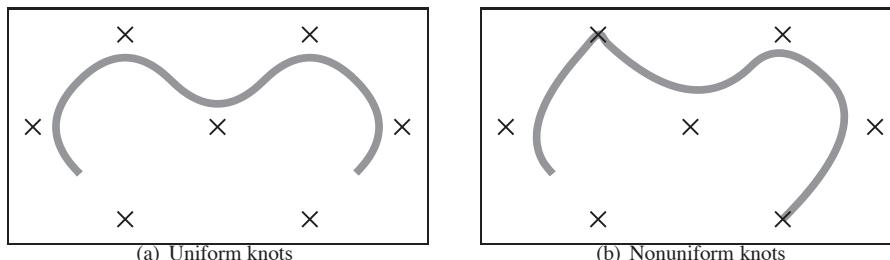


Figure 15.24. A curve parameterized by quadratic B-splines ($k = 3$) with seven control points. On the left, uniform knots vector $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ is used. On the right, the nonuniform knot spacing $[1, 2, 3, 4, 4, 6, 7, 8, 8, 10]$ is used. The duplication of the 4th and 8th knot means that all interior knots of the 3rd and 7th B-spline are equal, so the curve interpolates the control point associated with those points.

sions can be derived through the use of the Cox–de Boor recurrence:

$$b_{1,3,[0,0,0,1,2,\dots]}(t) = \begin{cases} (1-t)^2 & \text{if } 0 \leq t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

$$b_{2,3,[0,0,0,1,2,\dots]}(t) = \begin{cases} 2u - \frac{3}{2}u^2 & \text{if } 0 \leq t < 1 \quad u = t, \\ \frac{1}{2}(1-u)^2 & \text{if } 1 \leq t < 2 \quad u = t-1, \\ 0 & \text{otherwise.} \end{cases}$$

15.6.4 NURBS

Despite all of the generality B-splines provide, there are some functions that cannot be exactly represented using them. In particular, B-splines cannot represent conic sections. To represent such curves, a ratio of two polynomials is used. Nonuniform B-splines are used to represent both the numerator and the denom-

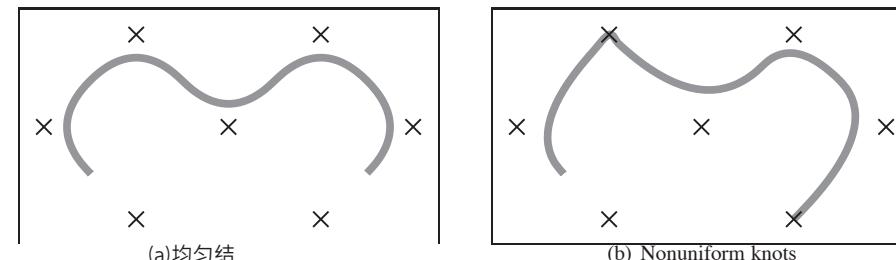


图15.24。由具有七个控制点的二次B样条($k=3$)参数化的曲线。在左侧, 使用uniformknotsvector[1 2 3 4 5 6 7 8 9 10]。在右侧, 使用非均匀结间距[1 2 3 4 4 6 7 8 8 10]。第4个和第8个结的重复意味着第3个和第7个B样条的所有内部结是相等的, 因此曲线内插与那些点相关联的控制点。

sions可以通过使用Cox deBoor复发来获得:

$$b_{1,3,[0,0,0,1,2,\dots]}(t) = \begin{cases} (1-t)^2 & \text{if } 0 \leq t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

$$b_{2,3,[0,0,0,1,2,\dots]}(t) = \begin{cases} 2u - \frac{3}{2}u^2 & \text{if } 0 \leq t < 1 \quad u = t, \\ \frac{1}{2}(1-u)^2 & \text{if } 1 \leq t < 2 \quad u = t-1, \\ 0 & \text{otherwise.} \end{cases}$$

15.6.4 NURBS

尽管B样条提供了所有的一般性, 但有一些函数不能用它们来精确表示。特别是, B样条不能表示圆锥曲线部分。为了表示这样的曲线, 使用两个多项式的比率。非均匀B样条用于表示分子和denom-

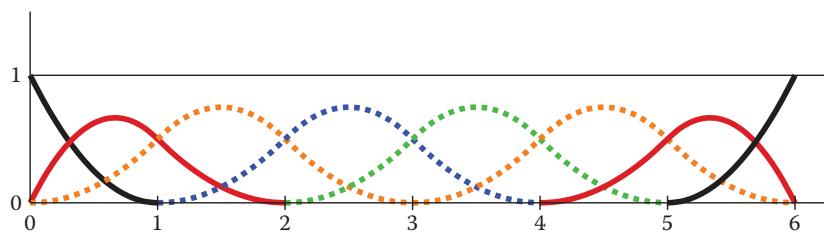


Figure 15.25. Endpoint interpolating quadratic ($k = 3$) B-splines, for $n = 8$. The knot vector is $[0, 0, 0, 1, 2, 3, 4, 5, 6, 6, 6]$. The first and last two B-splines are aperiodic, while the middle four (shown as dotted lines) are periodic and identical to the ones in Figure 15.20.

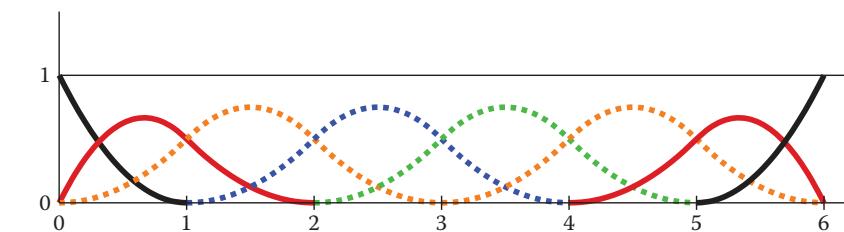


图15.25。端点插值二次($k=3$)B样条, 对于 $n=8$ 。向量为 $[0\ 0\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 6\ 6]$ 。第一个和最后两个B样条是非周期性的, 而中间的四个(如虚线所示)是周期性的, 与图15.20中的相同。

inator. The most general form of these are nonuniform rational B-splines, or NURBS for short.

NURBS associate a scalar weight h_i with every control point \mathbf{p}_i and use the same B-splines for both:

$$\mathbf{f}(u) = \frac{\sum_{i=1}^n h_i \mathbf{p}_i b_{i,k,t}}{\sum_{i=1}^n h_i b_{i,k,t}},$$

where $b_{i,k,t}$ are the B-splines with parameter k and knot vector t .

NURBS are very widely used to represent curves and surfaces in geometric modeling because of the amazing versatility they provide, in addition to the useful properties of B-splines.

15.7 Summary

In this chapter, we have discussed a number of representations for free-form curves. The most important ones for computer graphics are:

- Cardinal splines use a set of cubic pieces to interpolate control points. They are generally preferred to interpolating polynomials because they are local and easier to evaluate.
- Bézier curves approximate their control points and have many useful properties and associated algorithms. For this reason, they are popular in graphics applications.
- B-spline curves represent the curve as a linear combination of B-spline functions. They are general and have many useful properties such as being bounded by their convex hull and being variation diminishing. B-splines are often used when smooth curves are desired.

Notes

The problem of representing shapes mathematically is an entire field unto itself, generally known as geometric modeling. Representing curves is just the beginning and is generally a precursor to modeling surfaces and solids. A more thorough discussion of curves can be found in most geometric modeling texts, see for example *Geometric Modeling* (Mortenson, 1985) for a text that is accessible to computer graphics students. Many geometric modeling books specifically focus

inator。这些最一般的形式是非均匀有理B样条，或简称NURBS。

NURBS将标量权重 h_i 与每个控制点 \mathbf{p}_i 相关联，并对两者使用相同的B样条：

$$\mathbf{f}(u) = \frac{\sum_{i=1}^n h_i \mathbf{p}_i b_{i,k,t}}{\sum_{i=1}^n h_i b_{i,k,t}},$$

其中 $b_{i,k,t}$ 是参数 k 和结向量 t 的B样条。

NURBS在几何建模中被广泛用于表示曲线和曲面，因为除了B样条的有用属性外，它们还提供了惊人的多功能性。

15.7 Summary

在本章中，我们讨论了自由曲线的一些表示形式。计算机图形学中最重要的：

*基数样条使用一组立方块来插值控制点。它们通常优先于插值多项式，因为它们是局部的并且更容易评估。

*贝塞尔曲线近似其控制点，并具有许多有用的属性和相关算法。出于这个原因，它们在图形应用程序中很受欢迎。

*B样条曲线将曲线表示为B样条函数的线性组合。它们是一般的，具有许多有用的属性，例如由它们的凸包边界和变异递减。当需要平滑曲线时，通常使用B样条。

Notes

用数学方式表示形状的问题本身就是一个完整的领域，通常被称为几何建模。表示曲线只是一个开始，通常是曲面和实体建模的前兆。关于曲线的更彻底的讨论可以在大多数几何建模文本中找到，参见例如几何建模（Mortenson, 1985），以获取计算机图形学学生可以访问的文本。许多几何建模书籍都特别关注



on smooth curves and surfaces. Texts such as *An Introduction to Splines for Use in Computer Graphics* (Bartels, Beatty, & Barsky, 1987), *Curves and Surfaces for CAGD: A Practical Guide* (Farin, 2002) and *Geometric Modeling with Splines: An Introduction* (E. Cohen, Riesenfeld, & Elber, 2001) provide considerable detail about curve and surface representations. Other books focus on the mathematics of splines; *A Practical Guide to Splines* (De Boor, 2001) is a standard reference.

The history of the development of curve and surface representations is complex, see the chapter by Farin in *Handbook of Computer Aided Geometric Design* (Farin, Hoschek, & Kim, 2002) or the book on the subject *An Introduction to NURBS: With Historical Perspective* (D. F. Rogers, 2000) for a discussion. Many ideas were independently developed by multiple groups who approached the problems from different disciplines. Because of this, it can be difficult to attribute ideas to a single person or to point at the “original” sources. It has also led to a diversity of notation, terminology, and ways of introducing the concepts in the literature.

15.7.1 Exercises

For Exercises 1–4, find the constraint matrix, the basis matrix, and the basis functions. To invert the matrices you can use a program such as MATLAB or OCTAVE (a free MATLAB-like system).

1. A line segment: parameterized with p_0 located 25% of the way along the segment ($u = 0.25$), and p_1 located 75% of the way along the segment.
2. A quadratic: parameterized with p_0 as the position of the beginning point ($u = 0$), p_1 , the first derivative at the beginning point, and p_2 , the second derivative at the beginning point.
3. A cubic: its control points are equally spaced (p_0 has $u = 0$, p_1 has $u = 1/3$, p_2 has $u = 2/3$, and p_3 has $u = 1$).
4. A quintic: (a degree five polynomial, so the matrices will be 6×6) where p_0 is the beginning position, p_1 is the beginning derivative, p_2 is the middle ($u = 0.5$) position, p_3 is the first derivative at the middle, p_4 is the position at the end, and p_5 is the first derivative at the end.
5. The Lagrange form (Equation (15.12)) can be used to represent the interpolating cubic of Exercise 3. Use it at several different parameter values to confirm that it does produce the same results as the basis functions derived in Exercise 3.



在光滑的曲线和表面上。诸如计算机图形学中使用的样条简介 (Bartels, Beatty, & Barsky, 1987) , CAGD的曲线和曲面：实用指南 (Farin, 2002) 和用样条的几何建模：介绍 (E.Cohen, Riesenfeld, & Elber, 2001) 等文本提供了有关曲线和曲面表示的其他书籍专注于样条的数学;样条的实用指南 (DeBoor, 2001) 是标准参考。曲线和表面表示的发展历史是复杂的, 请参阅计算机辅助几何设计手册 (Farin, Hoschek, & Kim, 2002) 中的farin一章或关于该主题的书AnIntroductiontoNURBS: WithHistoricalPerspective (D.F.Rogers, 2000) 进行讨论。许多想法是由多个团体独立开发的, 他们从不同的学科来解决问题。因此, 很难将想法归因于一个人或指向“原始”来源。它还导致了符号, 术语和文献中引入概念的方式的多样性。

15.7.1 Exercises

对于练习1–4, 找到约束矩阵, 基矩阵和基函数。要反转矩阵, 您可以使用MATLAB或OCTAVE (一个免费的类似MATLAB的系统) 等程序。

- 1.线段：用位于线段25%的p0参数化 ($u=0$) 。25) , 而p1位于该段的75%。
- 2.二次型：以p0作为起始点 ($u=0$) 的位置参数化, p1是起始点的一阶导数, p2是起始点的二阶导数。
- 3.一个立方：它的控制点是等距的 (p0有u=0, p1有u=13, p2有u=23, p3有u=1) 。
- 4.一个五次多项式： (一个五度多项式, 所以矩阵将是 6×6) 其中p0是开始位置, p1是开始导数, p2是中间 ($u=0.5$) 位置, p3是中间的一阶导数, p4是末端的位置, p5是一阶导数。
- 5.拉格朗日形式 (方程 (15.12)) 可用于表示练习3的插值立方。在几个不同的参数值上使用它, 以确认它确实产生与练习3中导出的基函数相同的结果。

6. Devise an arc-length parameterization for the curve represented by the parametric function

$$f(u) = (u, u^2).$$

7. Given the four control points of a segment of a Hermite spline, compute the control points of an equivalent Bézier segment.

8. Use the de Casteljau algorithm to evaluate the position of the cubic Bézier curve with its control points at $(0,0)$, $(0,1)$, $(1,1)$ and $(1,0)$ for parameter values $u = 0.5$ and $u = 0.75$. Drawing a sketch will help you do this.

9. Use the Cox–de Boor recurrence to derive Equation (15.16).

6. 为 para 度量函数表示的曲线设计弧长参数化

$$f(u) = (u, u^2).$$

7. 给定 Hermite 样条线段的四个控制点，计算等效贝塞尔线段的控制点。

8. 使用 deCasteljau 算法评估三次贝塞尔曲线的位置，其控制点为 $(0, 0)$ 、 $(0, 1)$ 、 $(1, 1)$ 和 $(1, 0)$ 。对于参数值 $u=0.5$ 和 $u=0.75$ ，绘制草图将帮助您做到这一点。

9. 使用 Cox–deBoor 复发推导方程（15.16）。



Computer Animation

Animation is derived from the Latin *anima* and means the act, process, or result of imparting life, interest, spirit, motion, or activity. Motion is a defining property of life and much of the true art of animation is about how to tell a story, show emotion, or even express subtle details of human character through motion. A computer is a secondary tool for achieving these goals—it is a tool which a skillful animator can use to help get the result he wants faster and without concentrating on technicalities in which he is not interested. Animation without computers, which is now often called “traditional” animation, has a long and rich history of its own which is continuously being written by hundreds of people still active in this art. As in any established field, some time-tested rules have been crystallized which give general high-level guidance to how certain things should be done and what should be avoided. These principles of traditional animation apply equally to computer animation, and we will discuss some of them in this chapter.

The computer, however, is more than just a tool. In addition to making the animator’s main task less tedious, computers also add some truly unique abilities that were simply not available or were extremely difficult to obtain before. Modern modeling tools allow the relatively easy creation of detailed three-dimensional models, rendering algorithms can produce an impressive range of appearances, from fully photorealistic to highly stylized, powerful numerical simulation algorithms can help to produce desired physics-based motion for particularly hard to animate objects, and motion capture systems give the ability to



电脑动画

动画源自拉丁文*anima*, 意思是赋予生命, 兴趣, 精神, 运动或活动的行为, 过程或结果。运动是生命的一个定义属性, 真正的动画艺术大部分是关于如何通过运动讲述一个故事, 表达情感, 甚至表达人类性格的微妙细节。计算机是实现这些目标的辅助工具—它是一个熟练的动画师可以用来帮助更快地获得他想要的结果的工具, 而不会专注于他不感兴趣的技术细节。没有计算机的动画, 现在通常被称为“传统”动画, 有着悠久而丰富的历史, 数百名仍然活跃在这门艺术中的人正在不断地创作。与任何既定领域一样, 一些经过时间考验的规则已经具体化, 这些规则对应如何做某些事情以及应避免哪些事情提供了一般的高级指导。传统动画的这些原则同样适用于计算机动画, 我们将在本章中讨论其中的一些原则。

然而, 计算机不仅仅是一个工具。除了使动画师的主要任务不那么繁琐之外, 计算机还添加了一些真正独特的能力, 这些能力在以前根本无法获得或非常难以获得。现代建模工具允许相对容易地创建详细的三维模型, 渲染算法可以产生令人印象深刻的外观范围, 从完全真实到高度风格化, 强大的数值模拟算法可以帮助产生期望的基于物理的运动, 特别是难以动画的对象, 运动捕捉系统提供了

record and use real-life motion. These developments led to an exploding use of computer animation techniques in motion pictures and commercials, automotive design and architecture, medicine and scientific research, among many other areas. Completely new domains and applications have also appeared including fully computer-animated feature films, virtual/augmented reality systems, and, of course, computer games.

Other chapters of this book cover many of the developments mentioned above (for example, geometric modeling and rendering) more directly. Here, we will provide an overview only of techniques and algorithms directly used to create and manipulate motion. In particular, we will loosely distinguish and briefly describe four main computer animation approaches:

- **Keyframing** gives the most direct control to the animator who provides necessary data at some moments in time and the computer fills in the rest.
- **Procedural** animation involves specially designed, often empirical, mathematical functions and procedures whose output resembles some particular motion.
- **Physics-based** techniques solve differential equation of motion.
- **Motion capture** uses special equipment or techniques to record real-world motion and then transfers this motion into that of computer models.

We do not touch upon the artistic side of the field at all here. In general, we cannot possibly do more here than just scratch the surface of the fascinating subject of creating motion with a computer. We hope that readers truly interested in the subject will continue their journey well beyond the material of this chapter.

16.1 Principles of Animation

In his seminal 1987 SIGGRAPH paper (Lasseter, 1987), John Lasseter brought key principles developed as early as the 1930's by traditional animators of Walt Disney studios to the attention of the then-fledgling computer animation community. Twelve principles were mentioned: *squash and stretch*, *timing*, *anticipation*, *follow through and overlapping action*, *slow-in and slow-out*, *staging*, *arcs*, *secondary action*, *straight-ahead and pose-to-pose action*, *exaggeration*, *solid drawing skill*, and *appeal*. Almost two decades later, these time-tested rules, which can make a difference between a natural and entertaining animation and a mechanistic-looking and boring one, are as important as ever. For computer animation, in addition, it is very important to *balance* control and flexibility given to

记录和使用现实生活中的运动。这些发展导致计算机动画技术在电影和商业广告、汽车设计和建筑、医学和科学的研究等许多领域的爆炸式使用。全新的领域和应用也出现了，包括完全计算机动画的故事片、虚拟增强现实系统，当然还有计算机游戏。

本书的其他章节更直接地涵盖了上面提到的许多发展（例如，几何建模和渲染）。在这里，我们将仅概述直接用于创建和操作运动的技术和算法。特别是，我们将松散地区分并简要描述四种主要的计算机动画方法：

- *关键帧为动画师提供最直接的控制，动画师在某些时刻提供必要的数据，计算机填写其余部分。
- *程序动画涉及专门设计的，通常是经验的，数学函数和程序，其输出类似于某些特定的运动。
- *基于物理的技术解决微分方程运动。
- *运动捕捉使用特殊的设备或技术来记录真实世界的运动，然后将这种运动转移到计算机模型中。

我们在这里根本不涉及该领域的艺术方面。一般来说，我们不可能在这里做更多的事情，而不仅仅是用计算机创造运动的迷人主题的表面。我们希望读者真正感兴趣的主題将继续他们的旅程远远超出本章的材料。

16.1 动画原理

在他1987年开创性的SIGGRAPH论文（Lasseter, 1987年）中，John Lasseter将早在20世纪30年代由沃尔特迪士尼工作室的传统动画师开发的关键原则带到了当时初出茅庐的计算机动画中。其中提到了12个原则：挤压和伸展、时间、预期、贯穿和重叠动作、慢进和慢出、分期、弧线、次要动作、直进和摆姿势动作、夸张、扎实的绘画技巧和吸引力。近二十年后，这些经过时间考验的规则，可以在自然和娱乐的动画和机械外观和无聊的动画之间产生差异，与以往一样重要。对于计算机动画来说，此外，平衡控制和灵活性是非常重要的。

the animator with the full advantage of the computer's abilities. Although these principles are widely known, many factors affect how much attention is being paid to these rules in practice. While a character animator working on a feature film might spend many hours trying to follow some of these suggestions (for example, tweaking his timing to be just right), many game designers tend to believe that their time is better spent elsewhere.

16.1.1 Timing

Timing, or the speed of action, is at the heart of any animation. How fast things happen affects the meaning of action, emotional state, and even perceived weight of objects involved. Depending on its speed, the same action, a turn of a character's head from left to right, can mean anything from a reaction to being hit by a heavy object to slowly seeking a book on a bookshelf or stretching a neck muscle. It is very important to set timing appropriate for the specific action at hand. Action should occupy enough time to be noticed while avoiding too slow and potentially boring motions. For computer animation projects involving recorded sound, the sound provides a natural timing anchor to be followed. In fact, in most productions, the actor's voice is recorded first and the complete animation is then synchronized to this recording. Since large and heavy objects tend to move slower than small and light ones (with less acceleration, to be more precise), timing can be used to provide significant information about the weight of an object.

16.1.2 Action Layout

At any moment during an animation, it should be clear to the viewer what idea (action, mood, expression) is being presented. Good *staging*, or high-level planning of the action, should lead a viewer's eye to where the important action is currently concentrated, effectively telling him "look at this, and now, look at this" without using any words. Some familiarity with human perception can help us with this difficult task. Since human visual systems react mostly to relative changes rather than absolute values of stimuli, a sudden motion in a still environment or lack of motion in some part of a busy scene naturally draws attention. The same action presented so that the silhouette of the object is changing can often be much more noticeable compared with a frontal arrangement (see Figure 16.1 (bottom left)).

On a slightly lower level, each action can be split into three parts: *anticipation* (preparation for the action), the action itself, and *follow-through* (termination of the action). In many cases, the action itself is the shortest part and, in some sense,

具有计算机能力的充分优势的动画师。虽然这些原则广为人知，但许多因素影响了在实践中对这些规则的关注程度。虽然制作故事片的角色动画师可能会花费很多时间来尝试遵循其中的一些建议（例如，调整时机以恰到好处），但许多游戏设计师倾向于相信他们的时间

16.1.1 Timing

时间或动作速度是任何动画的核心。事情发生的速度会影响行动的意义，情绪状态，甚至所涉及物体的感知重量。根据它的速度，相同动作，一个角色的头部从左到右的转动，可能意味着任何事情，从反应到被重物击中，到在书架上慢慢寻找一本书或伸展颈部肌肉。为手头的具体行动设定适当的时间是非常重要的。动作应该占据足够的时间被注意，同时避免太慢和可能无聊的动作。对于涉及录制声音的计算机动画项目，声音提供了要遵循的自然定时锚。事实上，在大多数制作中，首先录制演员的声音，然后将完整的动画同步到此录制中。由于大型和重型物体的移动速度往往比小型和轻型物体慢（更精确地说，加速度更小），因此可以使用计时来提供有关物体重量的重要信息。

16.1.2 动作布局

在动画期间的任何时刻，观众都应该清楚地看到正在呈现的想法（动作，情绪，表达）。良好的舞台表演，或对行动的高层次规划，应该引导观众的眼睛到重要行动目前集中的地方，有效地告诉他“看看这个，现在，看看这个”，而不使用任何文字。对人类感知的一些熟悉可以帮助我们完成这项艰巨的任务。由于人类视觉系统的反应主要是相对变化而不是刺激的绝对值，因此静止环境中的突然运动或繁忙场景的某些部分的缺乏运动自然会引起注意。与正面排列相比，呈现的相同动作使物体的轮廓发生变化，通常会更加明显（见图16.1（左下））。在稍低的水平上，每个动作可以分为三个部分：预期（动作的准备），动作本身和后续（动作的终止）。在许多情况下，动作本身是最短的部分，从某种意义上说

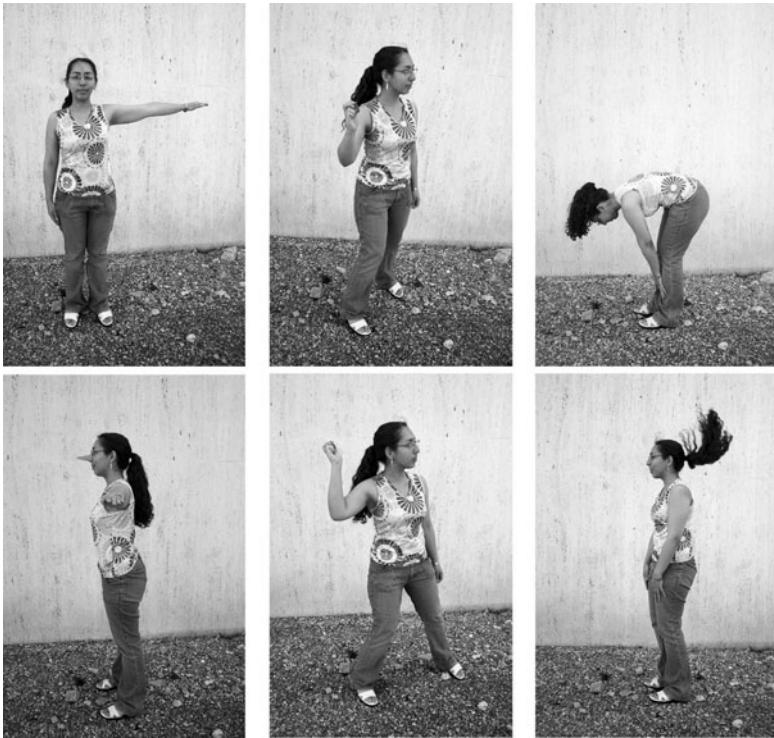
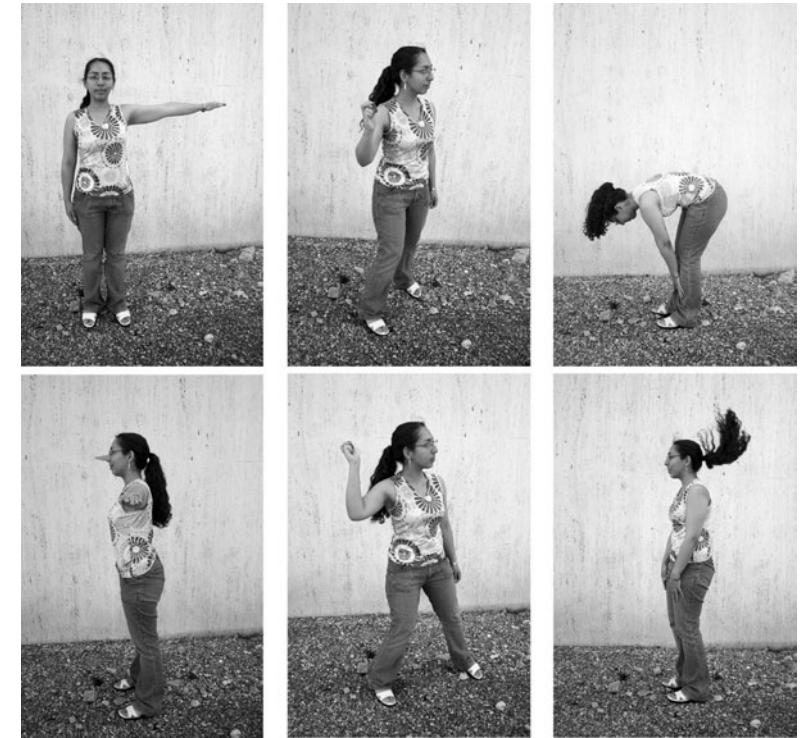


Figure 16.1. Action layout. Left: Staging action properly is crucial for bringing attention to currently important motion. The act of raising a hand would be prominent on the top but harder to notice on the bottom. A change in nose length, on the contrary, might be completely invisible in the first case. Note that this might be intentionally hidden, for example, to be suddenly revealed later. Neither arrangement is particularly good if both motions should be attended to. Middle: The amount of anticipation can tell much about the following action. The action which is about to follow (throwing a ball) is very short, but it is clear what is about to happen. The more wound up the character is, the faster the following action is perceived to be. Right: The follow-through phase is especially important for secondary appendages (hair) whose motion follows the leading part (head). The motion of the head is very simple, but leads to nontrivial follow-through behavior of the hair itself. It is impossible to create a natural animation without a follow-through phase and overlapping action in this case. *Figure courtesy Peter Shirley and Christina Villarruel.*

the least interesting. For example, kicking a football might involve extensive preparation on the part of the kicker and long “visual tracking” of the departing ball with ample opportunities to show the stress of the moment, emotional state of the kicker, and even the reaction to the expected result of the action. The action itself (motion of the leg to kick the ball) is rather plain and takes just a fraction of a second in this case.

The goal of anticipation is to prepare the viewer for what is about to happen. This becomes especially important if the action itself is very fast, greatly



动作布局。左：适当的分期行动对于引起人们对当前重要运动的关注至关重要。举手的行为在顶部很突出，但在底部很难注意到。相反，鼻子长度的变化在第一种情况下可能是完全看不见的。请注意，这可能是故意隐藏的，例如，稍后会突然显示。如果两项动议都要处理，这两项安排都不是特别好。中间：预期的数量可以告诉很多关于以下行动。即将发生的作用（扔球）很短，但很明显即将发生的事情。角色的清盘越多，下面的动作就越快。右：后续阶段对于运动跟随前导部分（头部）的次要附属物（头发）尤其重要。头部的运动非常简单，但导致头发本身的非平凡的后续行为。在这种情况下，如果没有后续阶段和重叠动作，就不可能创建自然动画。图礼貌彼得·雪莉和克里斯蒂娜·Villarruel。

最不有趣的。例如，踢足球可能涉及踢球者的广泛准备和离开球的长时间“视觉跟踪”，有足够的机会显示踢球者的压力，情绪状态，甚至对预期结果的反应。动作本身（腿部踢球的运动）相当简单，在这种情况下只需几秒钟。

期待的目标是让观众为即将发生的事情做好准备。这变得特别重要，如果行动本身是非常快的，大大

important, or extremely difficult. Creating a more extensive anticipation for such actions serves to underscore these properties and, in case of fast events, makes sure the action will not be missed (see Figure 16.1 (bottom center)).

In real life, the main action often causes one or more other *overlapping actions*. Different appendages or loose parts of the object typically drag behind the main leading section and keep moving for a while in the follow-through part of the main action as shown in Figure 16.1 (bottom right). Moreover, the next action often starts before the previous one is completely over. A player might start running while he is still tracking the ball he just kicked. Ignoring such natural flow is generally perceived as if there are pauses between actions and can result in robot-like mechanical motion. While overlapping is necessary to keep the motion natural, *secondary action* is often added by the animator to make motion more interesting and achieve realistic complexity of the animation. It is important not to allow secondary action to dominate the main action.

16.1.3 Animation Techniques

Several specific techniques can be used to make motion look more natural. The most important one is probably *squash and stretch* which suggests to change the shape of a moving object in a particular way as it moves. One would generally stretch an object in the direction of motion and squash it when a force is applied to it, as demonstrated in Figure 16.2 for a classic animation of a bouncing ball. It is important to preserve the total volume as this happens to avoid the illusion of growing or shrinking of the object. The greater the speed of motion (or the force), the more stretching (or squashing) is applied. Such deformations are used for several reasons. For very fast motion, an object can move between two sequential frames so quickly that there is no overlap between the object at the time of the current frame and at the time of the previous frame which can lead to strobing (a variant of aliasing). Having the object elongated in the direction of motion can ensure better overlap and helps the eye to fight this unpleasant effect. Stretching/squashing can also be used to show flexibility of the object with more deformation applied for more pliable materials. If the object is intended to appear as rigid, its shape is purposefully left the same when it moves.

Natural motion rarely happens along straight lines, so this should generally be avoided in animation and *arcs* should be used instead. Similarly, no real-world motion can instantly change its speed—this would require an infinite amount of force to be applied to an object. It is desirable to avoid such situations in animation as well. In particular, the motion should start and end gradually (*slow in and out*). While hand-drawn animation is sometimes done via *straight-ahead action*

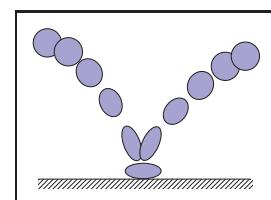


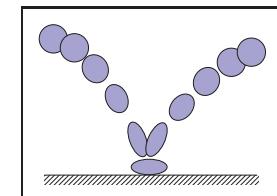
Figure 16.2. Classic example of applying the squash and stretch principle. Note that the volume of the bouncing ball should remain roughly the same throughout the animation.

重要, 或者极其困难。为这些操作创建更广泛的预期有助于强调这些属性, 并且在快速事件的情况下, 确保不会错过操作 (参见图16.1 (底部中心))。

在现实生活中, 主要动作经常引起一个或多个其他重叠的交流。对象的不同附属物或松散部分通常拖到主要前导部分后面, 并在主要动作的后续部分保持移动一段时间, 如图16.1 (右下) 所示。此外 下一次交流往往在前一次完全结束之前就开始了.一个球员可能会在他还在跟踪他刚刚踢的球时开始跑步。忽略这种自然流动通常被认为好像动作之间有停顿, 并可能导致类似机器人的机械运动。虽然重叠是保持运动自然所必需的, 但动画师通常会添加辅助动作, 以使运动更有趣并实现动画的逼真复杂性。重要的是不要让次要行动主导主要行动。

16.1.3 动画技术

几种特定的技术可以用来使运动看起来更自然。最重要的一个可能是squash and stretch, 这表明在移动时以特定方式改变移动物体的形状。人们通常会在运动方向上拉伸一个物体, 并在施加力时将其压扁, 如图16.2中弹跳球的经典动画所示。保持总体积很重要, 因为这是为了避免物体增长或缩小的错觉。运动速度 (或力) 越大, 施加的拉伸 (或压扁) 越多。使用这种变形有几个原因。对于非常快的运动, 对象可以在两个之间移动



经典的前充分应用壁球和伸展原则。
请注意, 反弹球的体积应该在整个过程中保持大致相同。

顺序帧的速度如此之快, 以至于对象在当前帧的时间和前一帧的时间之间没有重叠, 这可能导致频闪 (混叠的变体)。使物体在运动方向上伸长可以确保更好的重叠, 并帮助眼睛对抗这种不愉快的效果。拉伸压扁也可以用来显示物体的柔韧性, 更多的变形应用于更柔韧的材料。如果物体看起来是刚性的, 那么当它移动时, 它的形状是有目的地保持不变的。

自然运动很少沿着直线发生, 所以在动画中通常应该避免这种情况, 而应该使用弧线。同样, 任何真实世界的运动都不能立即改变其速度-这需要对物体施加无限量的力。在animation中也希望避免这种情况。特别是, 运动应该逐渐开始和结束 (慢进和慢出)。而手绘动画有时是通过直线前进的动作完成的

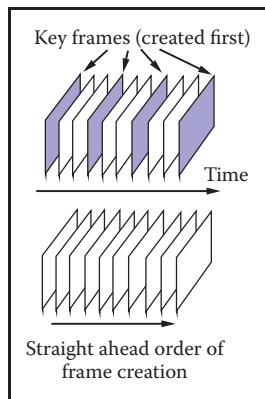


Figure 16.3. Keyframing (top) encourages detailed action planning while straight-ahead action (bottom) leads to a more spontaneous result.

with an animator starting at the first frame and drawing one frame after another in sequence until the end, *pose-to-pose action*, also known as *keyframing*, is much more suitable for computer animation. In this technique, animation is carefully planned through a series of relatively sparsely spaced key frames with the rest of the animation (in-between frames) filled in only after the keys are set (Figure 16.3). This allows more precise timing and allows the computer to take over the most tedious part of the process—the creation of the in-between frames—using algorithms presented in the next section.

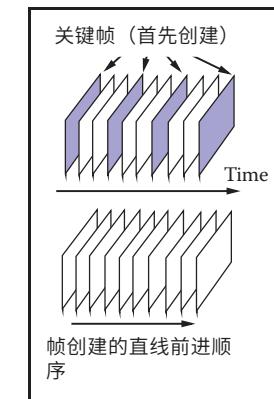
Almost any of the techniques outlined above can be used with some reasonable amount of *exaggeration* to achieve greater artistic effect or underscore some specific property of an action or a character. The ultimate goal is to achieve something the audience will want to see, something which is *appealing*. Extreme complexity or too much symmetry in a character or action tends to be less appealing. To create good results, a traditional animator needs *solid drawing skills*. Analogously, a computer animator should certainly understand computer graphics and have a solid knowledge of the tools he uses.

16.1.4 Animator Control vs. Automatic Methods

In traditional animation, the animator has complete control over all aspects of the production process and nothing prevents the final product to be as it was planned in every detail. The price paid for this flexibility is that every frame is created by hand, leading to an extremely time- and labor-consuming enterprise. In computer animation, there is a clear tradeoff between, on the one hand, giving an animator more direct control over the result, but asking him to contribute more work and, on the other hand, relying on more automatic techniques which might require setting just a few input parameters but offer little or no control over some of the properties of the result. A good algorithm should provide sufficient flexibility while asking an animator only the information which is intuitive, easy to provide, and which he himself feels is necessary for achieving the desired effect. While perfect compliance with this requirement is unlikely in practice since it would probably take something close to a mind-reading machine, we do encourage the reader to evaluate any computer-animation technique from the point of view of providing such *balance*.

16.2 Keyframing

The term *keyframing* can be misleading when applied to 3D computer animation since no actual completed frames (i.e., images) are typically involved. At any



Keyframing (顶部) 鼓励详细的行动计划，而直线前进的行动（底部）导致更自发的结果。

动画师从第一帧开始，依次绘制一帧又一帧直到结束，*pose-to-pose*动作，也称为关键帧，更适合计算机动画。在这种技术中，动画是精心设计的

计划通过一系列相对稀疏间隔的关键帧，其余的动画（在帧之间）只在关键帧设置后填充（图Figure 16.3）。这允许更精确的定时，并允许计算机接管过程中最繁琐的部分—创建中间帧—使用下一节中提出的算法。

几乎任何上面概述的技术都可以使用某种原因，可以使用大量的夸张来达到更大的艺术效果，或者强调一个动作或一个角色的某些特定属性。最终的目标是实现一些观众想要看到的东西，这是有吸引力的东西。在角色或动作中，极端的complexity或太多的对称性往往不那么吸引人。要创造良好的效果，传统的动画师需要扎实的绘图技巧。类比gousy，计算机动画师当然应该了解计算机图形学，并对他使用的工具有扎实的知识。

16.1.4 动画控制与自动方法

在传统动画中，动画师可以完全控制制作过程的各个方面，没有什么可以阻止最终产品在每个细节上都像计划的那样。为这种灵活性付出的代价是每个框架都是手工创建的，这导致了一个极其耗时的劳动消耗企业。在计算机动画中，一方面给动画师更直接地控制结果，另一方面要求他贡献更多的工作，另一方面依赖更多的自动技术，这些技术可能只需要设置几个输入参数，但对结果的某些属性提供很少或根本没有控制。一个好的算法应该提供足够的灵活性，同时只向动画师询问直观，易于提供的信息，并且他自己认为对于达到预期效果是必要的。虽然在实践中不太可能完全符合这一要求，因为它可能需要一些接近读心术的机器，但我们确实鼓励读者从提供这种平衡的角度来评估任何计算机动画技术。

16.2 Keyframing

当应用于3D计算机动画时，术语关键帧可能会产生误导，因为通常不涉及实际完成的帧（即图像）。在任何

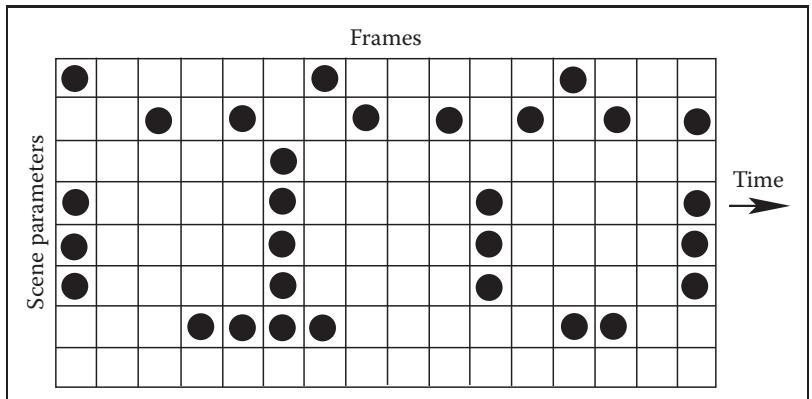


Figure 16.4. Different patterns of setting keys (black circles above) can be used simultaneously for the same scene. It is assumed that there are more frames before, as well as after, this portion.

given moment, a 3D scene being animated is specified by a set of numbers: the positions of centers of all objects, their RGB colors, the amount of scaling applied to each object in each axis, modeling transformations between different parts of a complex object, camera position and orientation, light sources intensity, etc. To animate a scene, some subset of these values have to change with time. One can, of course, directly set these values at every frame, but this will not be particularly efficient. Short of that, some number of important moments in time (key frames t_k) can be chosen along the timeline of animation for each of the parameters and values of this parameter (key values f_k) are set only for these selected frames. We will call a combination (t_k, f_k) of key frame and key value simply a key. Key frames do not have to be the same for different parameters, but it is often logical to set keys at least for some of them simultaneously. For example, key frames chosen for x -, y - and z -coordinates of a specific object might be set at exactly the same frames forming a single position vector key (t_k, \mathbf{p}_k) . These key frames, however, might be completely different from those chosen for the object's orientation or color. The closer key frames are to each other, the more control the animator has over the result; however the cost of doing more work of setting the keys has to be assessed. It is, therefore, typical to have large spacing between keys in parts of the animation which are relatively simple, concentrating them in intervals where complex action occurs, as shown in Figure 16.4.

Once the animator sets the key (t_k, f_k) , the system has to compute values of f for all other frames. Although we are ultimately interested only in a discrete set of values, it is convenient to treat this as a classical interpolation problem which fits a continuous *animation curve* $f(t)$ through a provided set of data points (Figure 16.5). Extensive discussion of curve-fitting algorithms can be found in Chap-

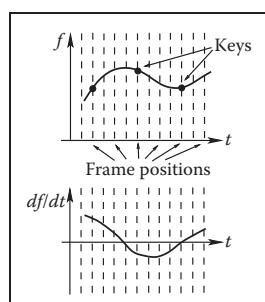


Figure 16.5. A continuous curve $f(t)$ is fit through the keys provided by the animator even though only values at frame positions are of interest. The derivative of this function gives the speed of parameter change and is at first determined automatically by the fitting procedure.

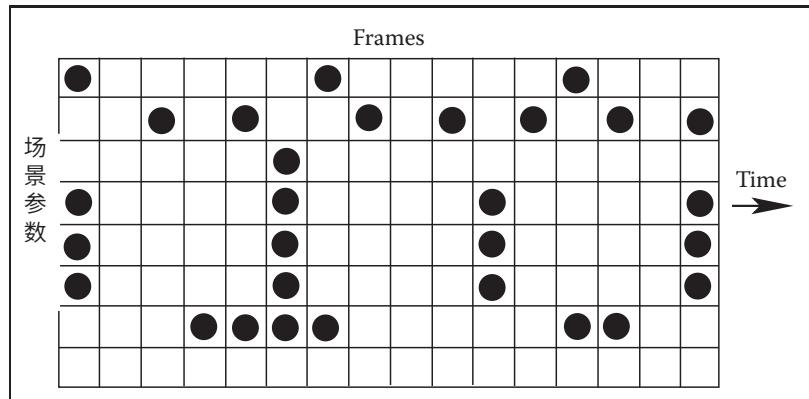
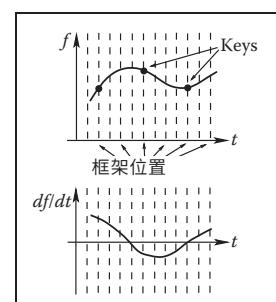


图16.4。不同模式的设置键（上面的黑色圆圈）可以同时用于同一场景。假设有更多的帧之前，以及之后，这部分。

给定时刻，动画的3d场景由一组数字指定：所有对象的中心位置，它们的RGB颜色，应用于每个轴中的每个对象的缩放量，复杂对象不同部分之间的建模转换，相机位置和方向，光源强度等。要使场景动画化，这些值的某些子集必须随时间而变化。当然，人们可以在每一帧直接设置这些值，但这不会特别有效。除此之外，在时间上的一些重要时刻（关键帧tk）可以沿着动画的时间线为每个参数选择，并且该参数的值（关键值fk）仅为这些选择的帧设置。我们将键帧和键值的组合 (tk, fk) 简单地称为键。对于不同的参数，密钥帧不必是相同的，但至少同时为其中一些密钥设置密钥通常是合乎逻辑的。例如，为特定对象的x, y和z坐标选择的关键帧可以设置在形成单个位置矢量键 (tk, pk) 的完全相同的帧上。这些关键

然而，框架可能与为对象的方向或颜色选择的框架完全不同。关键帧彼此之间的距离越近，动画师对结果的控制就越多；然而，必须评估更多设置关键帧的工作的成本。因此，动画中相对简单的部分键之间的间距通常很大，将它们集中在发生复杂动作的间隔中，如图16.4所示。

一旦动画师设置了键 (tk, fk) ，系统必须计算所有其他帧的 f 值。虽然我们最终只对一组离散的值感兴趣，但将其视为经典插值问题是方便的，该问题通过提供的一组数据点拟合连续动画曲线 $f(t)$ （Figure 16.5）。曲线拟合算法的广泛讨论可以在Chap中找到



一个连续的ous曲线 $f(t)$ 是通过由动画tor提供的键拟合，即使只有在帧位置的值是感兴趣的。该函数的导数给出了参数变化的速度，并且首先由fitting过程自动挖掘。

ter 15, and we will not repeat it here. Since the animator initially provides only the keys and not the derivative (tangent), methods which compute all necessary information directly from keys are preferable for animation. The speed of parameter change along the curve is given by the derivative of the curve with respect to time df/dt . Therefore, to avoid sudden jumps in velocity, C^1 continuity is typically necessary. A higher degree of continuity is typically not required from animation curves, since the second derivative, which corresponds to acceleration or applied force, can experience very sudden changes in real-world situations (ball hitting a solid wall), and higher derivatives do not directly correspond to any parameters of physical motion. These considerations make Catmull-Rom splines one of the best choices for initial animation curve creation.

Most animation systems give the animator the ability to perform interactive fine editing of this initial curve, including inserting more keys, adjusting existing keys, or modifying automatically computed tangents. Another useful technique which can help to tweak the shape of the curve is called TCB control (TCB stands for tension, continuity, and bias). The idea is to introduce three new parameters which can be used to modify the shape of the curve near a key through coordinated adjustment of incoming and outgoing tangents at this point. For keys uniformly spaced in time with distance Δt between them, the standard Catmull-Rom expression for incoming T_i^{in} and outgoing T_i^{out} tangents at an internal key (t_k, f_k) can be rewritten as

$$T_k^{in} = T_k^{out} = \frac{1}{2\Delta t}(f_{k+1} - f_k) + \frac{1}{2\Delta t}(f_k - f_{k-1}).$$

Modified tangents of a TCB spline are

$$T_k^{in} = \frac{(1-t)(1-c)(1+b)}{2\Delta t}(f_{k+1} - f_k) + \frac{(1-t)(1+c)(1-b)}{2\Delta t}(f_k - f_{k-1}),$$

$$T_k^{out} = \frac{(1-t)(1+c)(1+b)}{2\Delta t}(f_{k+1} - f_k) + \frac{(1-t)(1-c)(1-b)}{2\Delta t}(f_k - f_{k-1}).$$

The tension parameter t controls the sharpness of the curve near the key by scaling both incoming and outgoing tangents. Larger tangents (lower tension) lead to a flatter curve shape near the key. Bias b allows the animator to selectively increase the weight of a key's neighbors locally pulling the curve closer to a straight line connecting the key with its left (b near 1, "overshooting" the action) or right (b near -1, "undershooting" the action) neighbors. A nonzero value of continuity c makes incoming and outgoing tangents different allowing the animator to create kinks in the curve at the key value. Practically useful values of TCB parameters are typically confined to the interval $[-1; 1]$ with defaults $t = c = b = 0$ corresponding to the original Catmull-Rom spline. Examples of possible curve shape adjustments are shown in Figure 16.6.

15之三，在此不再赘述。由于动画师最初只提供键而不提供导数（切线），因此直接从键计算所有必要信息的方法对于动画来说是更好的。参数沿曲线变化的速度由曲线相对于时间 df/dt 的导数给出。因此，为了避免速度的突然跳跃， C^1 连续性通常是必要的。动画曲线通常不需要更高程度的连续性，因为对应于加速度或施加力的二阶导数可以在现实世界的情况下经历非常突然的变化（球击中实心墙），而更高的导数不直接对应于物理运动的任何参数。这些考虑使Catmull-Rom样条成为初始动画曲线创建的最佳选择之一。

大多数动画系统使动画师能够对此初始曲线进行交互式精细编辑，包括插入更多键、调整现有键或修改自动计算的切线。另一种有助于调整曲线形状的有用技术称为TCB控制（TCB代表张力、连续性和偏差）。我们的想法是引入三个新的参数，这些参数可以通过协调调整传入和传出切线来修改关键附近的曲线形状。对于在时间上均匀间隔的键，它们之间的距离 Δt ，标准的Catmull-Rom表达式用于输入的 T

可以改写为
和传出 T 出 内键处的切线 $(tk fk)$

$$T_k^{in} = T_k^{out} = \frac{1}{2\Delta t}(f_{k+1} - f_k) + \frac{1}{2\Delta t}(f_k - f_{k-1}).$$

TCB样条的修改切线为

$$T_k^{in} = \frac{(1-t)(1-c)(1+b)}{2\Delta t}(f_{k+1} - f_k) + \frac{(1-t)(1+c)(1-b)}{2\Delta t}(f_k - f_{k-1}),$$

$$T_k^{out} = \frac{(1-t)(1+c)(1+b)}{2\Delta t}(f_{k+1} - f_k) + \frac{(1-t)(1-c)(1-b)}{2\Delta t}(f_k - f_{k-1}).$$

张力参数 t 通过缩放传入和传出切线来控制键附近曲线的锐度。较大的切线（较低的张力）导致键附近更平坦的曲线形状。偏置 b 允许动画师有选择地增加键的邻居的权重，局部地将曲线拉近连接键与其左（ b 近 1，“超调”动作）或右（ b 近 -1，“超调”动作）邻居的直线。连续性 c 的非零值使传入和传出切线不同，允许动画师在关键值处在曲线中创建扭结。TCB 参数的实际有用值通常限于区间 $[-1; 1]$ ，默认值 $t=c=b=0$ 对应于原始 Catmull-Rom 样条。可能的曲线形状调整的例子如图 16.6 所示。

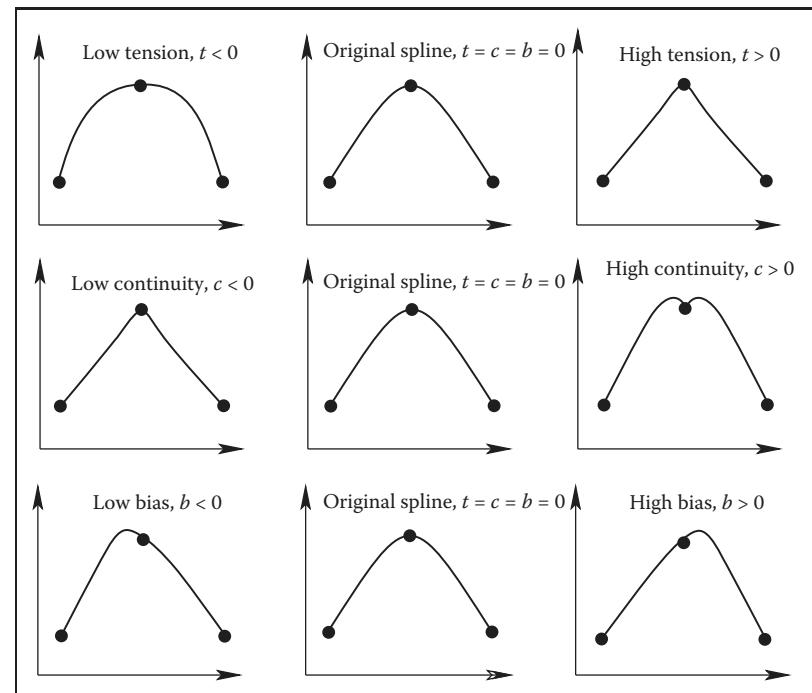
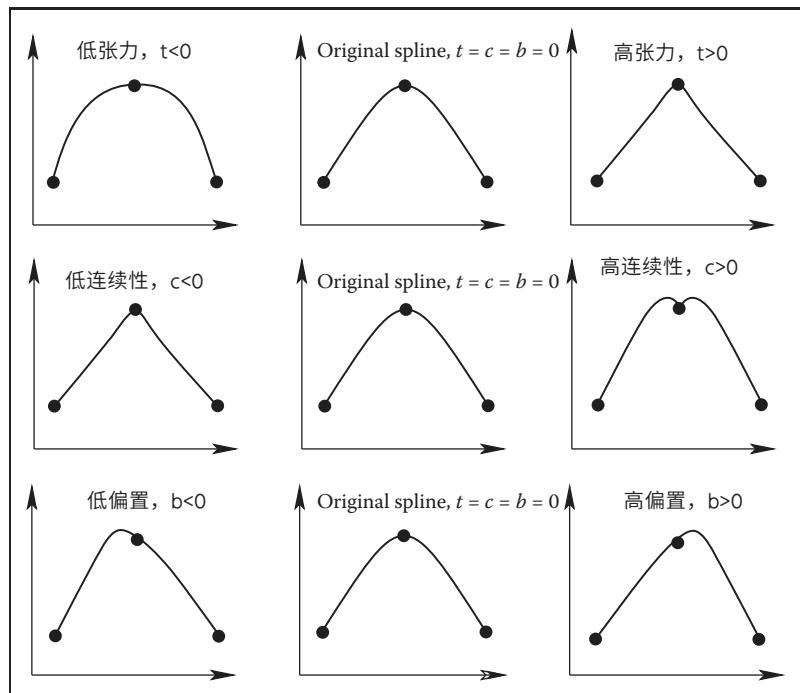


Figure 16.6. Editing the default interpolating spline (middle column) using TCB controls. Note that all keys remain at the same positions.

16.2.1 Motion Controls

So far, we have described how to control the shape of the animation curve through key positioning and fine tweaking of tangent values at the keys. This, however, is generally not sufficient when one would like to have control both over where the object is moving, i.e., its path, and how fast it moves along this path. Given a set of positions in space as keys, automatic curve-fitting techniques can fit a curve through them, but resulting motion is only constrained by forcing the object to arrive at a specified key position p_k at the corresponding key frame t_k , and nothing is directly said about the speed of motion between the keys. This can create problems. For example, if an object moves along the x -axis with velocity 11 meters per second for 1 second and then with 1 meter per second for 9 seconds, it will arrive at position $x = 20$ after 10 seconds thus satisfying animator's keys $(0,0)$ and $(10, 20)$. It is rather unlikely that this jerky motion was actually desired, and uniform motion with speed 2 meters/second is probably closer to what the animator wanted when setting these keys. Although typically not displaying



使用TCB控件编辑默认插值样条（中间列）。请注意，所有键保持在相同的位置。

16.2.1 运动控制

到目前为止，我们已经介绍了如何通过按键定位和微调按键处的切线值来控制动画曲线的形状。然而，当人们想要控制物体移动的位置（即其路径）以及它沿着这条路径移动的速度时，这通常是不够的。给定一组空间中的位置作为键，自动曲线拟合技术可以通过它们拟合一条曲线，但结果运动仅通过迫使对象到达相应键帧 t_k 处的指定键位置 p_k 来约束，并且没有直接说明键之间的运动速度。这可能会产生问题。例如，如果对象沿 x 轴以每秒11米的速度移动1秒，然后以每秒1米的速度移动9秒，则它将在10秒后到达位置 $x=20$ ，从而满足animator的键 $(0,0)$ 和 $(10, 20)$ 。实际上不太可能需要这种生涩的运动，并且速度为2米/秒的匀速运动可能更接近动画师在设置这些键时想要的。虽然通常不显示

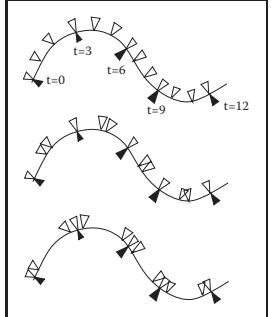


Figure 16.7. All three motions are along the same 2D path and satisfy the set of keys at the tips of the black triangles. The tips of the white triangles show object position at $\Delta t = 1$ intervals. Uniform speed of motion between the keys (top) might be closer to what the animator wanted, but automatic fitting procedures could result in either of the other two motions.

$$\mathbf{p}(t) = \mathbf{p}(u(s(t))).$$

Several standard functions can be used as the distance-time function $s(t)$. One of the simplest is the linear function corresponding to constant velocity: $s(t) = vt$ with $v = \text{const}$. Another common example is the motion with constant acceleration a (and initial speed v_0) which is described by the parabolic $s(t) = v_0t + at^2/2$. Since velocity is changing gradually here, this function can help to model desirable ease-in and ease-out behavior. More generally, the slope of $s(t)$ gives the velocity of motion with negative slope corresponding to the motion backwards along the curve. To achieve most flexibility, the ability to

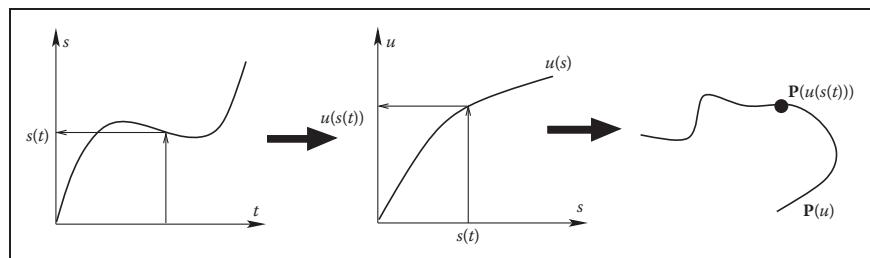
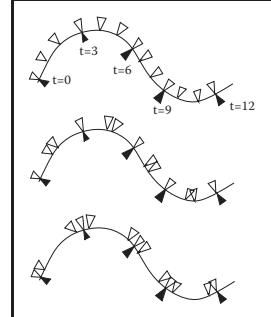


Figure 16.8. To get position in space at a given time t , one first utilizes user-specified motion control to obtain the distance along the curve $s(t)$ and then computes the corresponding curve parameter value $u(s(t))$. Previously fitted curve $\mathbf{P}(u)$ can now be used to find the position $\mathbf{P}(u(s(t)))$.



这种极端行为，由标准拟合procedures产生的多项式曲线确实表现出键之间的运动速度不均匀，如图16.7所示。虽然对于人类视觉系统不太擅长确定变化率（例如颜色甚至旋转速率）的不均匀性的某些参数，这是可以容忍的（在限制范围内），但我们必须对物体的位置p

所有三个motions都沿着相同的2D路径，并满足黑色三角形尖端的一组键。

白色三角形的尖端以 $\Delta t=1$ 的间隔显示物体位置。
键（顶部）之间的Uni形式运动速度可能更接近动画师想要的，但自动拟合procedures可能会导致其他两个运动中的任何一个。

$$\mathbf{p}(t) = \mathbf{p}(u(s(t))).$$

几个标准函数可以用作距离-时间函数 $s(t)$ 。
其中最简单的是对应于恒定速度的线性函数： $s(t) = vt$ with $v = \text{const}$ 。另一个常见的例子是具有constant加速度 a （和初始速度 v_0 ）的运动，其由抛物线 $s(t) = v_0t + at^2/2$ 描述。由于速度在这里是逐渐变化的，这个函数可以帮助建模理想的缓入和缓出行为。更一般地， $s(t)$ 的斜率给出了与沿曲线向后的运动相对应的负斜率的运动速度。为了达到最大的灵活性，

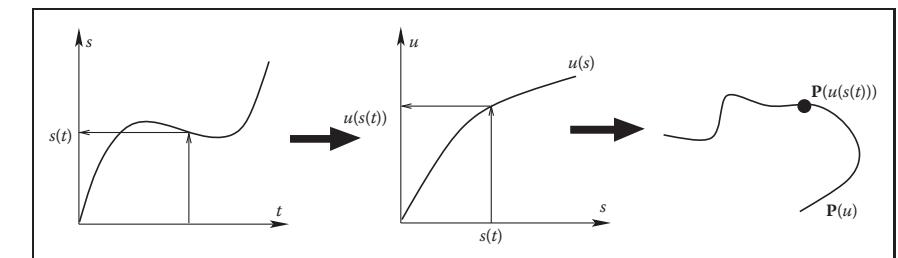


图16.8。为了在给定时间t获得空间中的位置，首先利用用户指定的运动控制来获得沿曲线 $s(t)$ 的距离，然后计算相应的曲线参数值 $u(s(t))$ 。先前拟合的曲线 $\mathbf{P}(u)$ 现在可用于找到位置 $\mathbf{P}(u(s(t)))$ 。



interactively edit $s(t)$ is typically provided to the animator by the animation system. The distance-time function is not the only way to control motion. In some cases it might be more convenient for the user to specify a velocity-time function $v(t)$ or even an acceleration-time function $a(t)$. Since these are correspondingly first and second derivatives of $s(t)$, to use these type of controls, the system first recovers the distance-time function by integrating the user input (twice in the case of $a(t)$).

The relationship between the curve parameter u and arc length s is established automatically by the system. In practice, the system first determines arc length dependence on parameter u (i.e., the inverse function $s(u)$). Using this function, for any given S it is possible to solve the equation $s(u) - S = 0$ with unknown u obtaining $u(S)$. For most curves, the function $s(u)$ cannot be expressed in closed analytic form and numerical integration is necessary (see Chapter 14). Standard numerical root-finding procedures (such as the Newton-Raphson method, for example) can then be directly used to solve the equation $s(u) - S = 0$ for u .

An alternative technique is to approximate the curve itself as a set of linear segments between points \mathbf{p}_i computed at some set of sufficiently densely spaced parameter values u_i . One then creates a table of approximate arc lengths

$$s(u_i) \approx \sum_{j=1}^i \|\mathbf{p}_j - \mathbf{p}_{j-1}\| = s(u_{i-1}) + \|\mathbf{p}_i - \mathbf{p}_{i-1}\|.$$

Since $s(u)$ is a non-decreasing function of u , one can then find the interval containing the value S by simple searching through the table (see Figure 16.9). Linear interpolation of the interval's u end values is then performed to finally find $u(S)$. If greater precision is necessary, a few steps of the Newton-Raphson algorithm with this value as the starting point can be applied.

16.2.2 Interpolating Rotation

The techniques presented above can be used to interpolate the keys set for most of the parameters describing the scene. Three-dimensional rotation is one important motion for which more specialized interpolation methods and representations are common. The reason for this is that applying standard techniques to 3D rotations often leads to serious practical problems. Rotation (a change in orientation of an object) is the only motion other than translation which leaves the shape of the object intact. It therefore plays a special role in animating rigid objects.

There are several ways to specify the orientation of an object. First, transformation matrices as described in Chapter 6 can be used. Unfortunately, naive

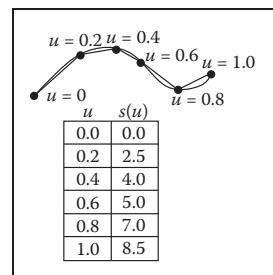


Figure 16.9. To create a tabular version of $s(u)$, the curve can be approximated by a number of line segments connecting points on the curve positioned at equal parameter increments. The table is searched to find the u -interval for a given S . For the curve above, for example, the value of u corresponding to the position of $S = 6.5$ lies between $u = 0.6$ and $u = 0.8$.



交互式编辑 $s(t)$ 通常由动画系统提供给动画师。距离-时间函数不是控制运动的唯一方法。在某些情况下，用户可能更方便地指定速度-时间函数 $v(t)$ 甚至加速度-时间函数 $a(t)$ 。由于这些相应地是 $s(t)$ 的一阶导数和二阶导数，为了使用这些类型的控件，系统首先通过积分用户输入（在 $a(t)$ 的情况下是两次）来恢复距离-时间函数。

曲线参数 u 与弧长 s 之间的关系由系统自动建立。在实践中，系统首先确定弧长依赖于参数 u (即反函数 $s(u)$)。使用该函数，对于任何给定的 S ，可以求解方程 $s(u) - S = 0$ ，未知 u 获得 $u(S)$ 。对于大多数曲线，函数 $s(u)$ 不能用封闭解析形式表示，并且需要数值积分（参见第 14 章）。然后可以直接使用标准数值寻根程序（例如 Newton-Raphson 方法，用于 \exp ）来求解 u 的方程 $s(u) - S = 0$ 。

一种替代技术是将曲线本身近似为在某组足够密集间隔的参数值 u_i 处计算的点 \mathbf{p}_i 之间的一组直线段。然后创建一个近似弧长表

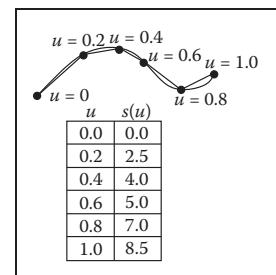
$$s(u_i) \approx \sum_{j=1}^i \|\mathbf{p}_j - \mathbf{p}_{j-1}\| = s(u_{i-1}) + \|\mathbf{p}_i - \mathbf{p}_{i-1}\|.$$

由于 $s(u)$ 是 u 的非递减函数，因此可以通过简单搜索表格找到包含 S 的区间（见图 16.9）。然后对区间的 u 端值进行线性插值以最终找到 $u(S)$ 。如果需要更高的精度，可以应用以该值为起点的 Newton-Raphson 算法的几个步骤。

16.2.2 插值旋转

上面呈现的技术可用于内插针对描述场景的大部分参数设置的键。三维旋转是一个重要的运动，其中更专门的插值方法和表示是常见的。其原因是，将标准技术应用于 3D 旋转通常会导致严重的实际问题。旋转（物体方向的变化）是除了平移之外的唯一运动，它使物体的形状保持不变。因此，它在动画刚性对象中起着特殊的作用。

有几种方法可以指定对象的方向。首先，可以使用第 6 章中描述的反式形成矩阵。不幸的是，天真



为了得到 $s(u)$ 的表格版本，曲线可以通过以相等的 parameter 增量定位的连接曲线上的点的若干线段来近似。

该表被搜索以找到给定 S 的 u 间隔。对于上面的曲线，例如，对于 $S=6.5$ 的 position 的 u 的值位于 $u=0.6$ 和 $u=0.8$ 之间。

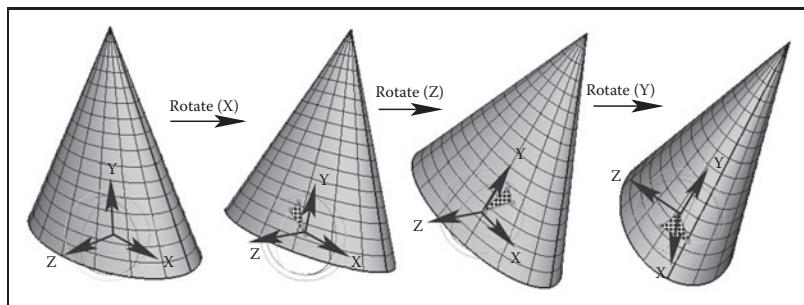


Figure 16.10. Three Euler angles can be used to specify arbitrary object orientation through a sequence of three rotations around coordinate axes embedded into the object (axis Y always points to the tip of the cone). Note that each rotation is given in a new coordinate system. Fixed angle representation is very similar, but the coordinate axes it uses are fixed in space and do not rotate with the object.

(element-by-element) interpolation of rotation matrices does not produce a correct result. For example, the matrix “halfway” between 2D clock- and counterclockwise 90 degree rotation is the null matrix:

$$\frac{1}{2} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

The correct result is, of course, the unit matrix corresponding to no rotation. Second, one can specify arbitrary orientation as a sequence of exactly three rotations around coordinate axes chosen in some specific order. These axes can be fixed in space (*fixed-angle* representation) or embedded into the object therefore changing after each rotation (*Euler-angle* representation as shown in Figure 16.10). These three angles of rotation can be animated directly through standard keyframing, but a subtle problem known as gimbal lock arises. Gimbal lock occurs if during rotation one of the three rotation axes is by accident aligned with another, thereby reducing by one the number of available degrees of freedom as shown in Figure 16.11 for a physical device. This effect is more common than one might think—a single 90 degree turn to the right (or left) can potentially put an object into a gimbal lock. Finally, any orientation can be specified by choosing an appropriate axis in space and angle of rotation around this axis. While animating in this representation is relatively straightforward, combining two rotations, i.e., finding the axis and angle corresponding to a sequence of two rotations both represented by axis and angle, is nontrivial. A special mathematical apparatus, *quaternions* has been developed to make this representation suitable both for combining several rotations into a single one and for animation.

Given a 3D vector $\mathbf{v} = (x, y, z)$ and a scalar s , a quaternion q is formed by combining the two into a four-component object: $q = [s \ x \ y \ z] = [s; \mathbf{v}]$. Several

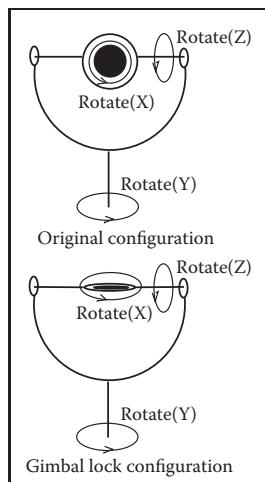
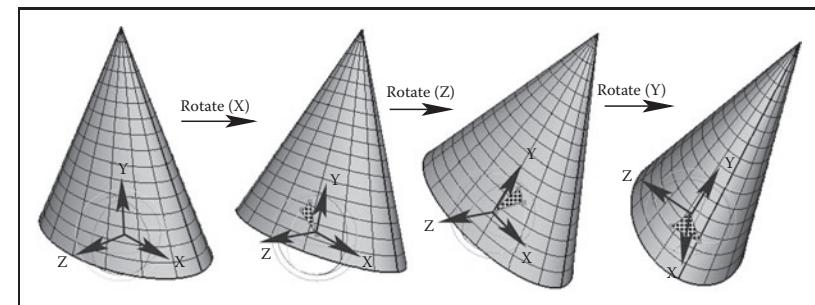


Figure 16.11. In this example, gimbal lock occurs when a 90 degree turn around axis Z is made. Both X and Y rotations are now performed around the same axis leading to the loss of one degree of freedom.



三个欧拉角可用于通过围绕嵌入到对象中的坐标轴（轴Y始终指向锥体的尖端）的三个旋转序列来指定任意对象方向。请注意，每个旋转都在一个新的坐标系中给出。固定角度表示非常相似，但它使用的坐标轴在空间上是固定的，不会随物体旋转。

（逐个元素）旋转矩阵的插值不会产生正确的结果。例如，2d时钟与逆时针90度旋转之间的矩阵“中途”是空矩阵：

$$\frac{1}{2} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

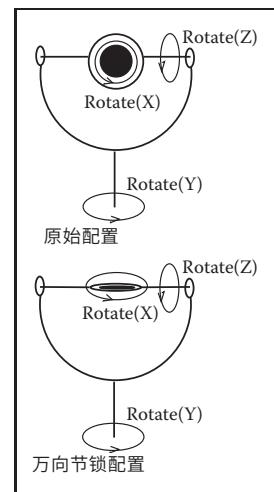


图16.11。在本次考试中，当绕轴Z进行90度转弯时发生万向节锁定。
现在，x和Y旋转都围绕同一轴进行，导致失去一个自由度。

正确的结果当然是无旋转对应的单位矩阵。其次，可以将任意方向指定为以特定顺序选择的坐标轴周围恰好三个旋转的序列。这些轴可以在空间中固定（固定角度表示）或嵌入到物体中，因此在每次旋转后都会发生变化（如图16.10所示的欧拉角度表示）。这三个角度的旋转可以直接通过标准的关键帧动画，但一个微妙的问题被称为万向节锁。如果在旋转过程中三个旋转轴中的一个意外地与另一个旋转轴对齐，则会发生万向节锁定，从而将物理设备的可用自由度数量减少一个，如图16.11所示。这种影响比人们可能更常见

想想-向右（或向左）单个90度转弯可能会将物体放入万向节锁中。最后，可以通过在空间中选择一个合适的轴以及绕该轴旋转的角度来指定任何方向。虽然此表示中的动画相对简单，但组合两个旋转，即找到对应于由轴和角度表示的两个旋转序列的轴和角度，并不是平凡的。一种特殊的数学装置，四元数已经被开发出来，以使这种表示既适用于将several旋转组合成单个旋转，也适用于动画。

给定一个3D向量 $\mathbf{v}=(x \ y \ z)$ 和一个标量 s ，通过将两者组合成一个四分量对象形成四元数 q : $q=[sxyz]=[s;\mathbf{v}]$ 。几个



new operations are then defined for quaternions. Quaternion addition simply sums scalar and vector parts separately:

$$q_1 + q_2 \equiv [s_1 + s_2; \mathbf{v}_1 + \mathbf{v}_2].$$

Multiplication by a scalar a gives a new quaternion

$$aq \equiv [as; a\mathbf{v}].$$

More complex quaternion multiplication is defined as

$$q_1 \cdot q_2 \equiv [s_1 s_2 - \mathbf{v}_1 \mathbf{v}_2; s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2],$$

where \times denotes a vector cross product. It is easy to see that, similar to matrices, quaternion multiplication is associative, but not commutative. We will be interested mostly in normalized quaternions—those for which the quaternion norm $|q| = \sqrt{s^2 + \mathbf{v}^2}$ is equal to one. One final definition we need is that of an inverse quaternion:

$$q^{-1} = (1/|q|)[s; -\mathbf{v}].$$

To represent a rotation by angle ϕ around an axis passing through the origin whose direction is given by the normalized vector \mathbf{n} , a normalized quaternion

$$q = [\cos(\phi/2); \sin(\phi/2)\mathbf{n}]$$

is formed. To rotate point \mathbf{p} , one turns it into the quaternion $q_p = [0; \mathbf{p}]$ and computes the quaternion product

$$q'_p = q \cdot q_p \cdot q^{-1}$$

which is guaranteed to have a zero scalar part and the rotated point as its vector part. Composite rotation is given simply by the product of quaternions representing each of the separate rotation steps. To animate with quaternions, one can treat them as points in a four-dimensional space and set keys directly in this space. To keep quaternions normalized, one should, strictly speaking, restrict interpolation procedures to a unit sphere (a 3D object) in this 4D space. However, a spherical version of even linear interpolation (often called *slerp*) already results in rather unpleasant math. Simple 4D linear interpolation followed by projection onto the unit sphere shown in Figure 16.12 is much simpler and often sufficient in practice. Smoother results can be obtained via repeated application of a linear interpolation procedure using the de Casteljau algorithm.

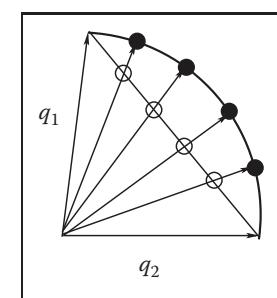


Figure 16.12. Interpolating quaternions should be done on the surface of a 3D unit sphere embedded in 4D space. However, much simpler interpolation along a 4D straight line (open circles) followed by re-projection of the results onto the sphere (black circles) is often sufficient.



然后为四元数定义新操作。四元数加法简单地将标量和矢量部分分开求和：

$$q_1 + q_2 \equiv [s_1 + s_2; \mathbf{v}_1 + \mathbf{v}_2].$$

乘以标量a给出了一个新的四元数

$$aq \equiv [as; a\mathbf{v}].$$

更复杂的四元数乘法定义为

$$q_1 \cdot q_2 \equiv [s_1 s_2 - \mathbf{v}_1 \mathbf{v}_2; s_1 \mathbf{v}_2 + s_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2],$$

其中 \times 表示向量叉积。很容易看出，与矩阵类似，四元数乘法是关联的，但不是交换的。我们将主要在归一化四元数中进行交互—那些四元数范数 $|q|=1$ 的四元数
 $s_2+v_2=1$ 。我们需要的最后一个定义是逆四元数：

$$q^{-1} = (1/|q|)[s; -\mathbf{v}].$$

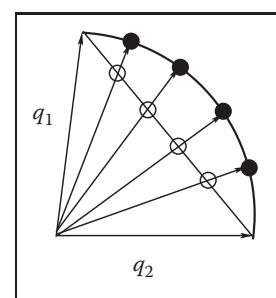
用角度 ϕ 来表示一个通过原点的轴的旋转，该原点的方向由归一化矢量 \mathbf{n} 给出，一个归一化四元数

$$q = [\cos(\phi/2); \sin(\phi/2)\mathbf{n}]$$

被形成。要旋转点 \mathbf{p} ，一个把它变成四元数 $q_p = [0; \mathbf{p}]$ 和计算四元数乘积

$$q'_p = q \cdot q_p \cdot q^{-1}$$

其保证具有零标量部分和旋转点作为其矢量部分。复合旋转简单地由表示每个单独旋转步骤的四元数的乘积给出。要使用四元数动画，可以将它们视为四维空间中的点，并直接在此空间中设置键。为了保持四元数标准化，严格来说，应该将插值过程限制在此4D空间中的单位球体（3D对象）。然而，即使是线性插值（通常称为slerp）的球形版本已经导致相当不愉快的数学。简单的4d线性插值，然后投影到图16.12所示的单位球体上，在实践中要简单得多，而且通常足够。通过使用deCasteljau算法重复应用线性插值程序，可以获得更平滑的结果。



插值四元数应该在嵌入4D空间的3D单位球体的表面上完成。然而，沿着一条4D直线（开放的圆圈）简单得多，然后将结果重新投影到球体（黑色圆圈）上就足够了。

16.3 Deformations

Although techniques for object deformation might be more properly treated as modeling tools, they are traditionally discussed together with animation methods. Probably the simplest example of an operation which changes object shape is a nonuniform scaling. More generally, some function can be applied to local coordinates of all points specifying the object (i.e., vertices of a triangular mesh or control polygon of a spline surface), repositioning these points and creating a new shape: $\mathbf{p}' = f(\mathbf{p}, \gamma)$ where γ is a vector of parameters used by the deformation function. Choosing different f (and combining them by applying one after another) can help to create very interesting deformations. Examples of useful simple functions include bend, twist, and taper which are shown in Figure 16.13. Animating shape change is very easy in this case by keyframing the parameters of the deformation function. Disadvantages of this technique include difficulty of choosing the mathematical function for some nonstandard deformations and the fact that the resulting deformation is *global* in the sense that the complete object, and not just some part of it, is reshaped.

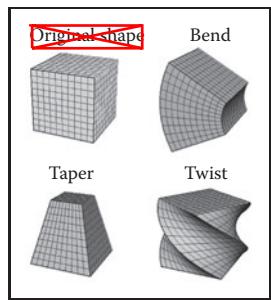


Figure 16.13. Popular examples of global deformations. Bending and twisting angles, as well as the degree of taper, can all be animated to achieve dynamic shape change.

To deform an object locally while providing more direct control over the result, one can choose a single vertex, move it to a new location and adjust vertices within some neighborhood to follow the seed vertex. The area affected by the deformation and the specific amount of displacement in different parts of the object are controlled by an attenuation function which decreases with distance (typically computed over the object's surface) to the seed vertex. Seed vertex motion can be keyframed to produce animated shape change.

A more general deformation technique is called free-form deformation (FFD) (Sederberg & Parry, 1986). A local (in most cases rectilinear) coordinate grid is first established to encapsulate the part of the object to be deformed, and coordinates (s, t, u) of all relevant points are computed with respect to this grid. The user then freely reshapes the grid of lattice points \mathbf{P}_{ijk} into a new distorted lattice \mathbf{P}'_{ijk} (Figure 16.14). The object is reconstructed using coordinates computed in the original undistorted grid in the trivariate analog of Bézier interpolants (see Chapter 15) with distorted lattice points \mathbf{P}'_{ijk} serving as control points in this expression:

$$P(s, u, t) = \sum_{i=0}^L \binom{i}{L} (1-s)^{L-i} s^i \sum_{j=0}^M \binom{j}{M} (1-t)^{M-j} t^j \sum_{k=0}^N \binom{k}{N} (1-u)^{N-k} u^k \mathbf{P}'_{ijk},$$

where L, M, N are maximum indices of lattice points in each dimension. In effect, the lattice serves as a low-resolution version of the object for the purpose of deformation, allowing for a smooth shape change of an arbitrarily complex ob-

虽然对象变形技术可能被更恰当地视为建模工具，但传统上它们与动画方法一起讨论。改变对象形状的操作最简单的例子可能是非均匀缩放。更一般地，一些函数可以应用于指定对象的所有点（即三角形网格的顶点或样条曲面的控制多边形）的局部坐标，重新定位这些点并创建新的形状： $\mathbf{p}' = f(\mathbf{p}, \gamma)$ ，选择不同的 f （并通过一个接一个地应用来组合它们）可以帮助创建非常有趣的变形。有用的简单函数的例子包括弯曲、扭曲和锥度如图16.13所示。在这种情况下，通过对变形函数的参数设置关键帧，动画形状更改非常容易。这种技术的缺点包括难以选择一些非标准变形的数学函数，以及由此产生的变形是全局的，因为完整的对象，而不仅仅是其中的一部分，被重塑。

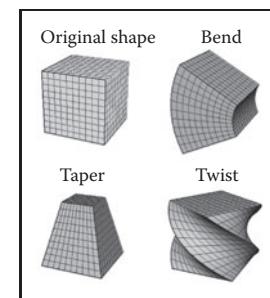


Figure 16.13. Popular examples of global deformations. Bending and twisting angles, as well as the degree of taper, can all be animated to achieve dynamic shape change.

To deform an object locally while providing more direct control over the result, one can choose a single vertex, move it to a new location and adjust vertices within some neighborhood to follow the seed vertex. The area affected by the deformation and the specific amount of displacement in different parts of the object are controlled by an attenuation function which decreases with distance (typically computed over the object's surface) to the seed vertex. Seed vertex motion can be keyframed to produce animated shape change.

A more general deformation technique is called free-form deformation (FFD) (Sederberg & Parry, 1986). A local (in most cases rectilinear) coordinate grid is first established to encapsulate the part of the object to be deformed, and coordinates (s, t, u) of all relevant points are computed with respect to this grid. The user then freely reshapes the grid of lattice points \mathbf{P}_{ijk} into a new distorted lattice \mathbf{P}'_{ijk} (Figure 16.14). The object is reconstructed using coordinates computed in the original undistorted grid in the trivariate analog of Bézier interpolants (see Chapter 15) with distorted lattice points \mathbf{P}'_{ijk} serving as control points in this expression:

$$P(s, u, t) = \sum_{i=0}^L \binom{i}{L} (1-s)^{L-i} s^i \sum_{j=0}^M \binom{j}{M} (1-t)^{M-j} t^j \sum_{k=0}^N \binom{k}{N} (1-u)^{N-k} u^k \mathbf{P}'_{ijk},$$

where L, M, N are maximum indices of lattice points in each dimension. In effect, the lattice serves as a low-resolution version of the object for the purpose of deformation, allowing for a smooth shape change of an arbitrarily complex ob-

ject through a relatively small number of intuitive adjustments. FFD lattices can themselves be treated as regular objects by the system and can be transformed, animated, and even further deformed if necessary, leading to corresponding changes in the object to which the lattice is attached. For example, moving a *deformation tool* consisting of the original lattice and distorted lattice representing a bulge across an object results in a bulge moving across the object.

16.4 Character Animation

Animation of articulated figures is most often performed through a combination of keyframing and specialized deformation techniques. The character model intended for animation typically consists of at least two main layers as shown in Figure 16.15. The motion of a highly detailed surface representing the outer shell or *skin* of the character is what the viewer will eventually see in the final product. The *skeleton* underneath it is a hierarchical structure (a tree) of joints which provides a kinematic model of the figure and is used exclusively for animation. In some cases, additional intermediate layer(s) roughly corresponding to muscles are inserted between the skeleton and the skin.

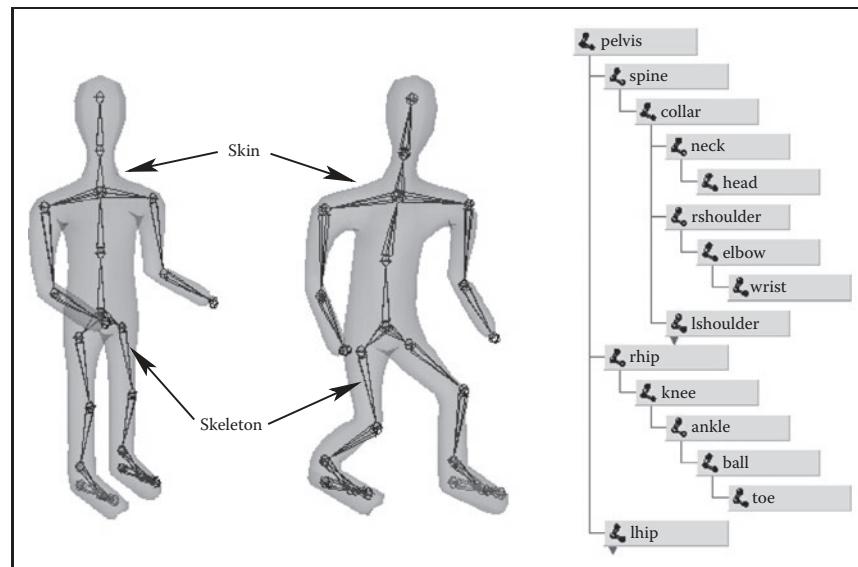


Figure 16.15. (Left) A hierarchy of joints, a skeleton, serves as a kinematic abstraction of the character; (middle) repositioning the skeleton deforms a separate skin object attached to it; (right) a tree data structure is used to represent the skeleton. For compactness, the internal structure of several nodes is hidden (they are identical to a corresponding sibling).

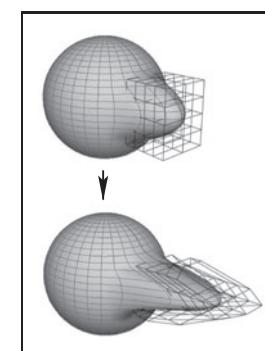
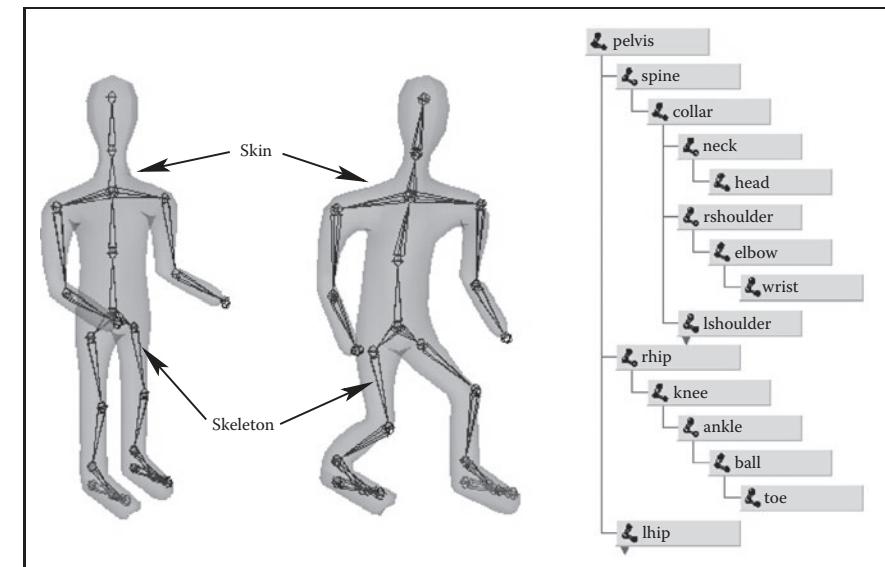


Figure 16.14. Adjusting the FFD lattice results in the deformation of the object.

ject通过相对少量的直观调整。FFD晶格本身可以被系统视为规则对象，并且可以进行变换，动画化，甚至在必要时进一步变形，从而导致晶格所附着的对象的相应变化。例如，移动由原始晶格和表示跨越对象的凸起的扭曲晶格组成的变形工具会导致凸起在对象上移动。

16.4 角色动画

关节图形的动画通常是通过关键帧和专门的变形技术的组合来执行的。用于动画的角色模型通常由至少两个主要层组成，如图16.15所示。一个高度详细的表面的运动代表了角色的外壳或皮肤，这是观众最终将看到的最终产品。它下面的骨架是一个关节的层次结构（树），它提供了一个图形的运动学模型，专门用于动画。在某些情况下，在骨架和皮肤之间插入大致对应于肌肉的附加中间层。



(左) 关节的层次结构，一个骨架，作为角色的运动学抽象；(中) 重新定位骨架变形附加到它的单独皮肤对象；(右) 树数据结构用于表示骨架。为了紧凑，隐藏了几个节点的内部结构（它们与相应的兄弟节点相同）。

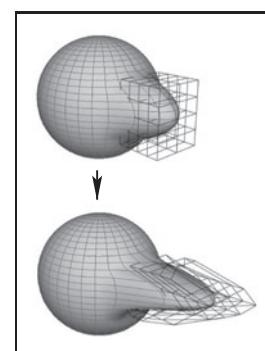


图16.14。调整FFD晶格会导致物体变形。

Each of the skeleton's joints acts as a parent for the hierarchy below it. The root represents the whole character and is positioned directly in the world coordinate system. If a local transformation matrix which relates a joint to its parent in the hierarchy is available, one can obtain a transformation which relates local space of any joint to the world system (i.e., the system of the root) by simply concatenating transformations along the path from the root to the joint. To evaluate the whole skeleton (i.e., find position and orientation of all joints), a depth-first traversal of the complete tree of joints is performed. A transformation stack is a natural data structure to help with this task. While traversing down the tree, the current composite matrix is pushed on the stack and a new one is created by multiplying the current matrix with the one stored at the joint. When backtracking to the parent, this extra transformation should be undone before another branch is visited; this is easily done by simply popping the stack. Although this general and simple technique for evaluating hierarchies is used throughout computer graphics, in animation (and robotics) it is given a special name—*forward kinematics* (FK). While general representations for all transformations can be used, it is common to use specialized sets of parameters, such as link lengths or joint angles, to specify skeletons. To animate with forward kinematics, rotational parameters of all joints are manipulated directly. The technique also allows the animator to change the distance between joints (link lengths), but one should be aware that this corresponds to limb stretching and can often look rather unnatural.

Forward kinematics requires the user to set parameters for all joints involved in the motion (Figure 16.16 (top)). Most of these joints, however, belong to in-

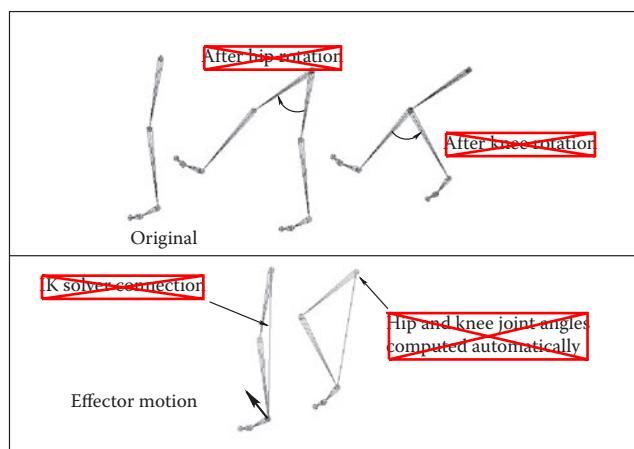


Figure 16.16. Forward kinematics (top) requires the animator to put all joints into correct position in inverse kinematic (bottom), parameters of some internal joints are computed based on desired end effector motion.

骨架的每个关节都充当其下方层次结构的父级。根代表整个角色，直接定位在世界coordinate系统中。如果在层次结构中有一个将关节与其父关节相关联的局部变换矩阵，则可以通过简单地沿着从根到关节的路径连接变换来获得一个将任何关节的局部空间与世界系统（即根的系统）相关联的变换。为了评估整个骨架（即找到所有关节的位置和方向），对完整的关节树进行深度优先遍历。转换堆栈是帮助完成此任务的自然数据结构。在向下遍历树的同时，当前的复合矩阵被推到堆栈上，并通过将当前矩阵与存储在关节处的矩阵混合来创建一个新的矩阵。当回溯到父级时，这个额外的转换应该在访问另一个分支之前撤销；这很容易通过简单地弹出堆栈来完成。虽然这种评估层次结构的一般和简单技术在整个计算机图形学中使用，但在动画（和机器人）中，它被赋予了一个特殊的名称—前向运动学（fk）。虽然可以使用所有转换的一般表示，但通常使用专门的参数集（如链接长度或关节角度）来指定骨架。要使用前向运动学动画，所有关节的旋转参数都被直接操纵。该技术还允许动画师改变关节之间的距离（链接长度），但人们应该知道，这与肢体伸展有关，并且通常看起来相当不自然。

前向运动学要求用户为参与运动的所有关节设置参数（图16.16（上））。然而，这些关节大多属于

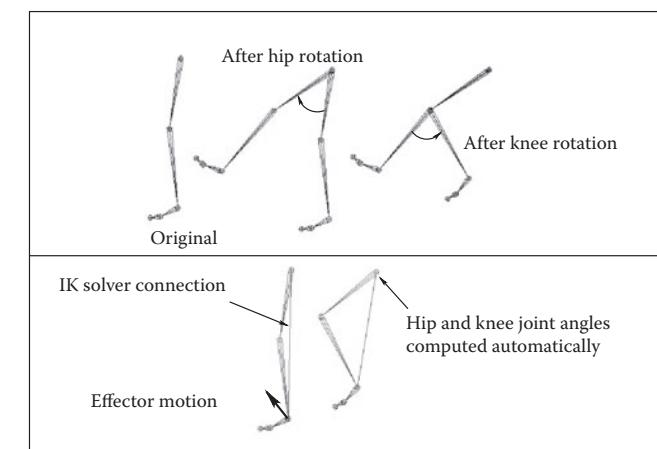


Figure 16.16. Forward kinematics (top) requires the animator to put all joints into correct position. In inverse kinematic (bottom), parameters of some internal joints are computed based on desired end effector motion.



ternal nodes of the hierarchy, and their motion is typically not something the animator wants to worry about. In most situations, the animator just wants them to move naturally “on their own,” and one is much more interested in specifying the behavior of the endpoint of a joint chain, which typically corresponds to something performing a specific action, such as an ankle or a tip of a finger. The animator would rather have parameters of all internal joints be determined from the motion of the end effector automatically by the system. *Inverse kinematics* (IK) allows us to do just that (see Figure 16.16 (bottom)).

Let \mathbf{x} be the position of the end effector and α be the vector of parameters needed to specify all internal joints along the chain from the root to the final joint. Sometimes the orientation of the final joint is also directly set by the animator, in which case we assume that the corresponding variables are included in the vector \mathbf{x} . For simplicity, however, we will write all specific expressions for the vector:

$$\mathbf{x} = (x_1, x_2, x_3)^T.$$

Since each of the variables in \mathbf{x} is a function of α , it can be written as a vector equation $\mathbf{x} = \mathbf{F}(\alpha)$. If we change the internal joint parameters by a small amount $\delta\alpha$, a resulting change $\delta\mathbf{x}$ in the position of the end effector can be approximately written as

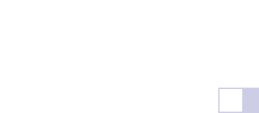
$$\delta\mathbf{x} = \frac{\partial \mathbf{F}}{\partial \alpha} \delta\alpha, \quad (16.1)$$

where $\frac{\partial \mathbf{F}}{\partial \alpha}$ is the matrix of partial derivatives called the Jacobian:

$$\frac{\partial \mathbf{F}}{\partial \alpha} = \begin{bmatrix} \frac{\partial f_1}{\partial \alpha_1} & \frac{\partial f_1}{\partial \alpha_2} & \dots & \frac{\partial f_1}{\partial \alpha_n} \\ \frac{\partial f_2}{\partial \alpha_1} & \frac{\partial f_2}{\partial \alpha_2} & \dots & \frac{\partial f_2}{\partial \alpha_n} \\ \frac{\partial f_3}{\partial \alpha_1} & \frac{\partial f_3}{\partial \alpha_2} & \dots & \frac{\partial f_3}{\partial \alpha_n} \end{bmatrix}.$$

At each moment in time, we know the desired position of the end effector (set by the animator) and, of course, the effector’s current position. Subtracting the two, we will get the desired adjustment $\delta\mathbf{x}$. Elements of the Jacobian matrix are related to changes in a coordinate of the end effector when a particular internal parameter is changed while others remain fixed (see Figure 16.17). These elements can be computed for any given skeleton configuration using geometric relationships. The only remaining unknowns in the system of equations (16.1) are the changes in internal parameters α . Once we solve for them, we update $\alpha = \alpha + \delta\alpha$ which gives all the necessary information for the FK procedure to reposition the skeleton.

Unfortunately, the system (16.1) cannot usually be solved analytically and, moreover, it is in most cases underconstrained, i.e., the number of unknown internal joint parameters α exceeds the number of variables in vector \mathbf{x} . This means that different motions of the skeleton can result in the same motion of the end



层次结构的三个节点，它们的运动通常不是动画师想要担心的。在大多数情况下，动画师只是希望它们“自己”自然移动，而人们更感兴趣的是指定关节链端点的行为，这通常对应于执行特定动作的东西，例如脚踝或手指尖。动画师宁愿让所有内部关节的参数由系统自动从末端执行器的运动中确定。逆运动学 (ik) 允许我们做到这一点（见图16.16（底部））。

设 \mathbf{x} 是末端执行器的位置， α 是指定从根部到最终关节沿着链的所有内部关节所需的参数的向量。有时最终关节的方向也由动画师直接设置，在这种情况下，我们假设相应的变量包含在向量 \mathbf{x} 中。然而，为了简单起见，我们将为向量编写所有特定的表达式：

$$\mathbf{x} = (x_1, x_2, x_3)^T.$$

由于 \mathbf{x} 中的每个变量都是 α 的函数，因此可以将其写为矢量方程 $\mathbf{x} = \mathbf{F}(\alpha)$ 。如果我们通过少量 $\delta\alpha$ 改变内部关节参数，则末端执行器位置的最终变化 $\delta\mathbf{x}$ 可近似写为

$$\delta\mathbf{x} = \frac{\partial \mathbf{F}}{\partial \alpha} \delta\alpha, \quad (16.1)$$

where $\Delta\alpha$ 是偏导数的矩阵，称为雅可比：

$$\frac{\partial \mathbf{F}}{\partial \alpha} = \begin{bmatrix} \frac{\partial f_1}{\partial \alpha_1} & \frac{\partial f_1}{\partial \alpha_2} & \dots & \frac{\partial f_1}{\partial \alpha_n} \\ \frac{\partial f_2}{\partial \alpha_1} & \frac{\partial f_2}{\partial \alpha_2} & \dots & \frac{\partial f_2}{\partial \alpha_n} \\ \frac{\partial f_3}{\partial \alpha_1} & \frac{\partial f_3}{\partial \alpha_2} & \dots & \frac{\partial f_3}{\partial \alpha_n} \end{bmatrix}.$$

在每个时刻，我们都应该知道末端执行器的期望位置（由动画师设置），当然还有执行器的当前位置。将两者相减，我们将得到所需的调整 $\delta\mathbf{x}$ 。雅可比矩阵的元素与特定内部参数改变而其他参数保持固定时末端执行器坐标的变化有关（见图16.17）。这些元素可以

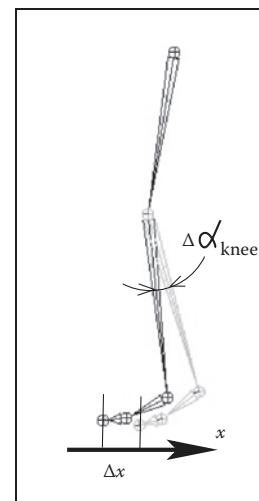


Figure 16.17. Partial derivative $\frac{\partial x}{\partial \alpha_{\text{knee}}}$ is given by the limit of $\Delta x / \Delta \alpha_{\text{knee}}$. Effector displacement is computed while all joints, except the knee, are kept fixed.

使用几何关系对任何给定的骨架配置进行计算。方程组 (16.1) 中唯一剩下的未知数是内部参数 α 的变化。一旦我们解决了它们，我们更新 $\alpha = \alpha + \delta\alpha$ ，这给出了 FK 过程重新定位骨架的所有必要信息。

不幸的是，系统 (16.1) 通常不能解析地求解，而且，它在大多数情况下是欠约束的，即未知的 n 个联合参数 α 的数量超过向量 \mathbf{x} 中的变量数量。这意味着骨架的不同运动可能导致末端的相同运动。

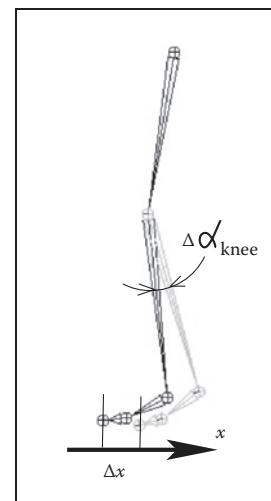


Figure 16.17. Partial derivative $\frac{\partial x}{\partial \alpha_{\text{knee}}}$ is given by the limit of当所有关节（膝关节除外）保持固定时，计算效应器 $\delta\mathbf{x}$ 放置。

effector. Some examples are shown on Figure 16.18. Many ways of obtaining specific solution for such systems are available, including those taking into account natural *constraints* needed for some real-life joints (bending a knee only in one direction, for example). One should also remember that the computed Jacobian matrix is valid only for one specific configuration, and it has to be updated as the skeleton moves. The complete IK framework is presented in Figure 16.19. Of course, the root joint for IK does not have to be the root of the whole hierarchy, and multiple IK solvers can be applied to independent parts of the skeleton. For example, one can use separate solvers for right and left feet and yet another one to help animate grasping with the right hand, each with its own root.

A combination of FK and IK approaches is typically used to animate the skeleton. Many common motions (walking or running cycles, grasping, reaching, etc.) exhibit well-known patterns of mutual joint motion making it possible to quickly

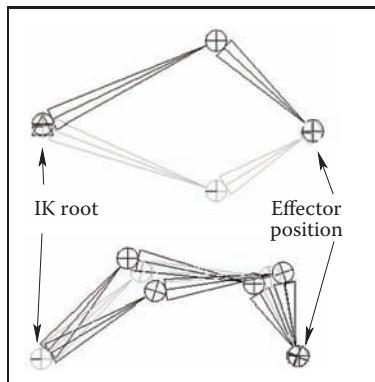


Figure 16.18. Multiple configurations of internal joints can result in the same effector position. (Top) disjoint “flipped” solutions; (bottom) a continuum of solutions.

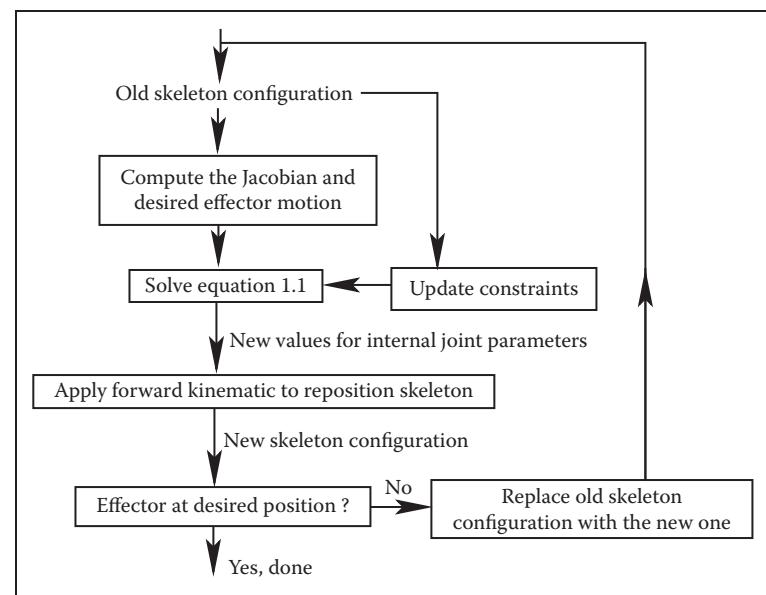
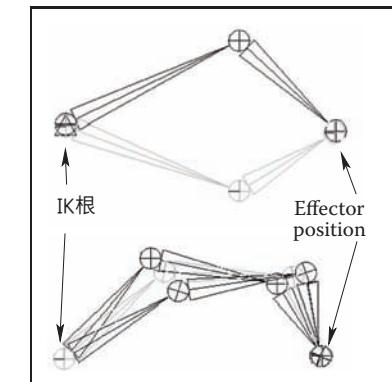


Figure 16.19. A diagram of the inverse kinematic algorithm.

效应器。一些例子如图16.18所示。有许多方法可以获得这种系统的具体解决方案，包括考虑到考虑一些现实生活中的关节所需的自然约束（例如，仅在一个方向上弯曲膝盖）。人们还应该记住，计算的Jacobian matrix只对一个特定的配置有效，并且它必须随着skeleton的移动而更新。完整的IK框架如图16.19所示。当然，用于IK的根关节不必是整个层次结构的根，并且多个IK求解器可以应用于骨架的独立部分。例如，一个可以为右脚和左脚使用单独的求解器，另一个可以帮助用右手抓握动画，每个都有自己的根。



内部接头的多个配置可以导致相同的效应器位置。（上）不相交的“翻转”解决方案；（下）解决方案的连续体。

FK和IK方法的组合通常用于动画skeleton。许多常见的动作（步行或跑步循环，抓握，伸手等）。表现出众所周知的相互关节运动模式，使快速运动成为可能

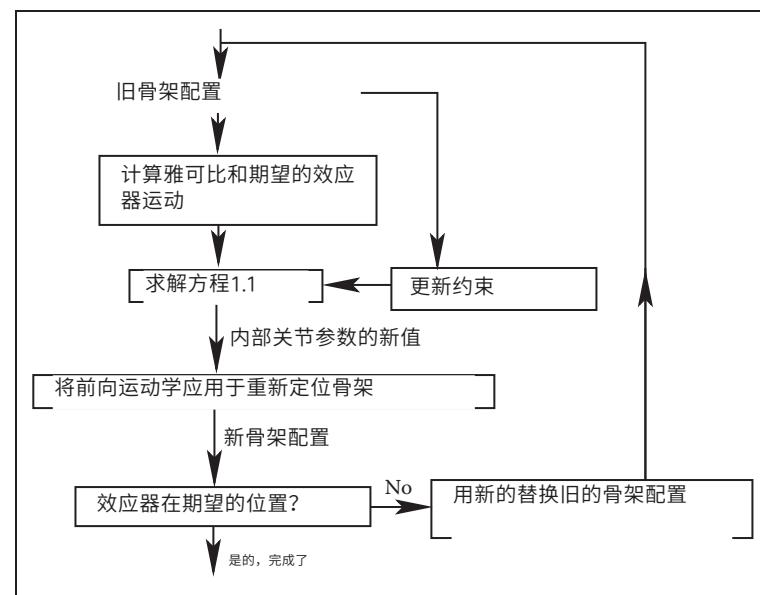


图16.19。逆运动学算法的图。

create naturally looking motion or even use a library of such “clips.” The animator then adjusts this generic result according to the physical parameters of the character and also to give it more individuality.

When a skeleton changes its position, it acts as a special type of deformer applied to the skin of the character. The motion is transferred to this surface by assigning each skin vertex one (*rigid skinning*) or more (*smooth skinning*) joints as drivers (see Figure 16.20). In the first case, a skin vertex is simply frozen into the local space of the corresponding joint, which can be the one nearest in space or one chosen directly by the user. The vertex then repeats whatever motion this joint experiences, and its position in world coordinates is determined by standard FK procedure. Although it is simple, rigid skinning makes it difficult to obtain sufficiently smooth skin deformation in areas near the joints or also for more subtle effects resembling breathing or muscle action. Additional specialized deformers called *flexors* can be used for this purpose. In smooth skinning, several joints can influence a skin vertex according to some weight assigned by the animator, providing more detailed control over the results. Displacement vectors, \mathbf{d}_i , suggested by different joints affecting a given skin vertex (each again computed with standard FK) are averaged according to their weights w_i to compute the final displacement of the vertex $\mathbf{d} = \sum w_i \mathbf{d}_i$. Normalized weights ($\sum w_i = 1$) are the most common but not fundamentally necessary. Setting smooth skinning weights to achieve the desired effect is not easy and requires significant skill from the animator.

16.4.1 Facial Animation

Skeletons are well suited for creating most motions of a character’s body, but they are not very convenient for realistic facial animation. The reason is that the skin of a human face is moved by muscles directly attached to it, contrary to other parts of the body where the primary objective of the muscles is to move the bones of the skeleton and any skin deformation is a secondary outcome. The result of this facial anatomical arrangement is a very rich set of dynamic facial expressions humans use as one of the main instruments of communication. We are all very well trained to recognize such facial variations and can easily notice any unnatural appearance. This not only puts special demands on the animator but also requires a high-resolution geometric model of the face and, if photorealism is desired, accurate skin reflection properties and textures.

While it is possible to set key poses of the face vertex-by-vertex and interpolate between them or directly simulate the behavior of the underlying muscle structure using physics-based techniques (see Section 16.5), more specialized

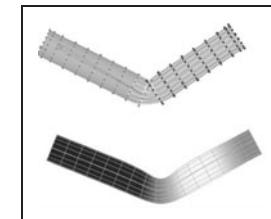


Figure 16.20. Top: Rigid skinning assigns skin vertices to a specific joint. Those belonging to the elbow joint are shown in black; Bottom: Soft skinning can blend the influence of several joints. Weights for the elbow joint are shown (lighter = greater weight). Note smoother skin deformation of the inner part of the skin near the joint.

创建自然外观的运动，甚至使用这样的“剪辑”库。“Animator”然后根据角色的物理参数调整这个通用结果，并赋予它更多的个性。

当骨架改变其位置时，它充当应用于角色皮肤的特殊类型的变形器。通过将每个蒙皮顶点分配一个（刚性蒙皮）或多个（平滑蒙皮）关节作为驱动器，运动被转移到该表面（见图16.20）。在第一种情况下，皮肤顶点被简单地冻结

入相应关节的局部空间，可以是空间最近的一个，也可以是用户直接选择的一个。然后，顶点重复此关节经历的任何运动，并且它在世界坐标中的位置由标准FK过程确定。虽然它很简单，但刚性换肤使得难以在关节附近区域获得足够光滑的皮肤变形，或者也难以获得类似呼吸或肌肉动作的更微妙效果。称为屈肌的额外专门变形器可用于此目的。在平滑蒙皮中，多个关节可以根据动画师分配的某些权重影响蒙皮顶点，从而提供对结果的更详细控制。由影响给定皮肤顶点的不同关节（每个关节再次用标准FK计算）建议的位移矢量 \mathbf{d}_i 根据它们的权重 w_i 进行平均，以计算顶点的最终位移 $\mathbf{d}=\sigma w_i \mathbf{d}_i$ 。归一化权值（ $\sigma w_i=1$ ）是最常见的，但不是根本必要的。设置平滑的蒙皮权重以达到预期的效果不容易，需要动画师的显着技巧。

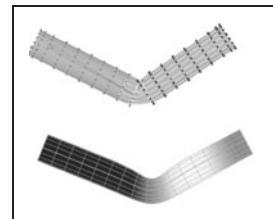


Figure 16.20. Top: 刚性蒙皮将蒙皮顶点分配给特定关节。属于肘关节的那些以黑色显示;Bottom: 柔软的剥皮可以融合几个关节的影响。肘关节的重量显示（更轻=更大的重量）。注意关节附近皮肤内部的皮肤变形。

16.4.1 面部动画

骨架非常适合创建角色身体的大部分动作，但对于逼真的面部动画来说并不是很方便。原因是人脸的皮肤是由直接附着在它上面的肌肉移动的，与身体的其他部位相反，肌肉的主要目标是移动骨骼的骨骼，任何皮肤变形都是次要的结果。这种面部解剖安排的结果是人类用作主要通信工具之一的一组非常丰富的动态面部表情。我们都训练有素，能够识别这种面部变化，并且很容易注意到任何不自然的外观。这不仅对动画师提出了特殊的要求，而且还需要高分辨率的面部几何模型，如果需要逼真的效果，还需要精确的皮肤反射属性和纹理。

虽然可以逐个顶点地设置面部的关键姿势，并在它们之间进行交互，或者使用基于物理的技术直接模拟底层muscle结构的行为（参见第16.5节），但更专业的

high-level approaches also exist. The static shape of a specific face can be characterized by a relatively small set of so-called *conformational parameters* (overall scale, distance from the eye to the forehead, length of the nose, width of the jaws, etc.) which are used to morph a generic face model into one with individual features. An additional set of *expressive parameters* can be used to describe the dynamic shape of the face for animation. Examples include rigid rotation of the head, how wide the eyes are open, movement of some feature point from its static position, etc. These are chosen so that most of the interesting expressions can be obtained through some combination of parameter adjustments, therefore, allowing a face to be animated via standard keyframing. To achieve a higher level of control, one can use expressive parameters to create a set of expressions corresponding to common emotions (neutral, sadness, happiness, anger, surprise, etc.) and then blend these key poses to obtain a “slightly sad” or “angrily surprised” face. Similar techniques can be used to perform lip-synch animation, but key poses in this case correspond to different phonemes. Instead of using a sequence of static expressions to describe a dynamic one, the Facial Action Coding System (FACS) (Eckman & Friesen, 1978) decomposes dynamic facial expressions directly into a sum of elementary motions called action units (AUs). The set of AUs is based on extensive psychological research and includes such movements as raising the inner brow, wrinkling the nose, stretching lips, etc. Combining AUs can be used to synthesize a necessary expression.

16.4.2 Motion Capture

Even with the help of the techniques described above, creating realistic-looking character animation from scratch remains a daunting task. It is therefore only natural that much attention is directed toward techniques which record an actor’s motion in the real world and then apply it to computer-generated characters. Two main classes of such *motion capture* (MC) techniques exist: electromagnetic and optical.

In electromagnetic motion capture, an electromagnetic sensor directly measures its position (and possibly orientation) in 3D, often providing the captured results in real time. Disadvantages of this technique include significant equipment cost, possible interference from nearby metal objects, and noticeable size of sensors and batteries which can be an obstacle in performing high-amplitude motions. In optical MC, small colored markers are used instead of active sensors making it a much less intrusive procedure. Figure 16.21 shows the operation of such a system. In the most basic arrangement, the motion is recorded by two calibrated video cameras, and simple triangulation is used to extract the marker’s 3D position. More advanced computer vision algorithms used for accurate tracking

高级别方法也存在。特定面部的静态形状可以通过相对较小的一组所谓的构象参数（总体尺度，从眼睛到前额的距离，鼻子的长度，颌骨的宽度等）来表征。用于将通用人脸模型变形为具有单个特征的人脸模型。一组额外的表现参数可用于描述用于动画的面部的动态形状。示例包括头部的刚性旋转、眼睛睁得多宽、某些特征点从其静态位置的移动等。选择这些是为了使大多数有趣的表达可以通过参数调整的组合获得，因此，允许通过标准关键帧动画人脸。为了达到更高的控制水平，人们可以使用表达参数来创建一组对应于常见情绪（中性，悲伤，快乐，愤怒，惊讶等）的表达。），然后混合这些关键姿势，以获得“略微悲伤”或“愤怒惊讶”的脸。类似的技术可用于执行唇同步动画，但是在这种情况下关键姿势对应于不同的音素。面部动作编码系统（FACS）（Eckman & Friesen, 1978）不是使用静态表情序列来描述动态表情，而是将动态面部表情直接分解为称为动作单元（AUs）的基本动作总和。这套运动是基于广泛的心理学研究，包括提高内眉，皱鼻子，伸展嘴唇等运动。结合AUs可用于合成必要的表达。

16.4.2 动作捕捉

即使借助上述技术，从头开始创建逼真的角色动画仍然是一项艰巨的任务。因此，很自然地，人们会把注意力集中在记录演员在现实世界中的动作，然后将其应用于计算机生成的角色的技术上。这种运动捕捉（MC）技术存在两大类：电磁和光学。

在电磁运动捕获中，电磁传感器直接在3D中确定其位置（可能还有方向），通常实时提供捕获的结果。这种技术的缺点包括显着的设备成本、可能来自附近金属物体的干扰以及传感器和电池的明显尺寸，这可能是执行高振幅运动的障碍。在光学MC中，使用小的彩色标记而不是有源传感器，使其成为一个更少侵入性的过程。图16.21显示了这样一个系统的操作。在最基本的布置中，运动由两个calibrated视频摄像机记录，并且使用简单的三角测量来提取标记的3D位置。更先进的计算机视觉算法用于精确跟踪



of multiple markers from video are computationally expensive, so, in most cases, such processing is done offline. Optical tracking is generally less robust than electromagnetic. Occlusion of a given marker in some frames, possible misidentification of markers, and noise in images are just a few of the common problem which have to be addressed. Introducing more cameras observing the motion from different directions improves both accuracy and robustness, but this approach is more expensive and it takes longer to process such data. Optical MC becomes more attractive as available computational power increases and better computer vision algorithms are developed. Because of low impact nature of markers, optical methods are suitable for delicate facial motion capture and can also be used with objects other than humans—for example, animals or even tree branches in the wind.

With several sensors or markers attached to a performer's body, a set of time-dependant 3D positions of some collection of points can be recorded. These tracking locations are commonly chosen near joints, but, of course, they still lie on skin surface and not at points where actual bones meet. Therefore, some additional care and a bit of extra processing is necessary to convert recorded positions into those of the physical skeleton joints. For example, putting two markers on opposite sides of the elbow or ankle allows the system to obtain better joint position by averaging locations of the two markers. Without such extra care, very noticeable artifacts can appear due to offset joint positions as well as inherent noise and insufficient measurement accuracy. Because of physical inaccuracy during motion, for example, character limbs can lose contact with objects they are supposed to touch during walking or grasping, problems like foot-sliding (skating) of the skeleton can occur. Most of these problems can be corrected by using inverse kinematics techniques which can explicitly force the required behavior of the limb's end.

Recovered joint positions can now be directly applied to the skeleton of a computer-generated character. This procedure assumes that the physical dimensions of the character are identical to those of the performer. Retargeting recorded motion to a different character and, more generally, editing MC data, requires significant care to satisfy necessary constraints (such as maintaining feet on the ground or not allowing an elbow to bend backwards) and preserve an overall natural appearance of the modified motion. Generally, the greater the desired change from the original, the less likely it will be possible to maintain the quality of the result. An interesting approach to the problem is to record a large collection of motions and stitch together short clips from this library to obtain desired movement. Although this topic is currently a very active research area, limited ability to adjust the recorded motion to the animator's needs remains one of the main disadvantages of motion capture technique.

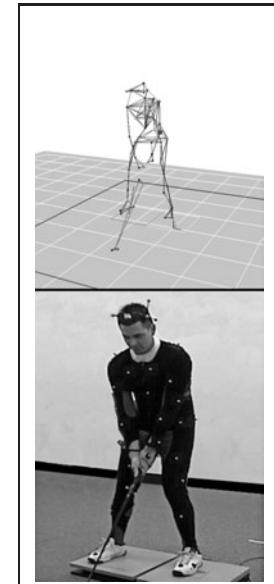


Figure 16.21. Optical motion capture: markers attached to a performer's body allow skeletal motion to be extracted. *Image courtesy of Motion Analysis Corp.*



来自视频的多个标记在计算上是昂贵的，因此，在大多数情况下，这样的处理是离线完成的。光学跟踪的鲁棒性通常不如

电磁。某些帧中给定标记物的遮挡、标记物可能出现的误码化以及图像中的噪声只是一些必须解决的常见问题。引入更多从不同方向观察运动的摄像机可以提高准确性和鲁棒性，但这种方法更昂贵，处理此类数据需要更长时间。随着可用计算能力的提高和更好的计算机视觉算法的开发，光学MC变得更具吸引力。由于标记的低影响性质，optical方法适用于精细的面部动作捕捉，也可用于人类以外的物体—例如，动物甚至是风中的树枝。

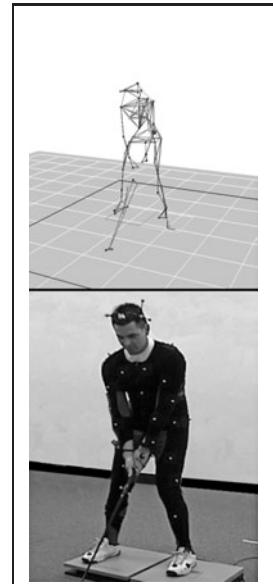


Figure 16.21. Optical motion capture: 贴在表演者身体上的标记允许骨骼运动被释放。图片由Motion分析公司提供。

通过将多个传感器或标记连接到表演者的身体，可以记录一些点集合的一组时间依赖性3D位置。这些跟踪位置通常选择在关节附近，但是，当然，它们仍然位于皮肤表面，而不是在实际骨骼相遇的地方。因此，需要一些额外的护理和一些额外的处理来将记录的位置转换为物理骨架关节的位置。例如，在肘部或踝部的两侧放置两个标记可以让系统通过平均两个标记的位置来获得更好的关节位置。如果没有这种额外的护理，由于关节位置偏移以及固有噪声和测量精度不足，可能会出现非常明显的伪像。由于在运动期间的物理不准确性，例如，角色肢体可以与它们在行走或抓握期间被准备触摸的物体失去接触，可以发生像骨架的脚滑 (skating)这样的问题。这些问题中的大多数可以通过使用可以明确地强制肢体末端的所需行为的运动学技术来纠正。

恢复的关节位置现在可以直接应用于计算机生成的角色的骨架。这个过程假设角色的物理二值与表演者的相同。将记录的运动重新定位到不同的角色，更一般地说，编辑MC数据，需要非常小心地满足必要的约束（例如保持脚在地面上或不允许肘部向后弯曲），并保持修改后的运动的整体natural外观。通常，从原始的期望变化越大，越不可能保持结果的质量。解决这个问题的一个有趣的方法是记录大量的运动，并从这个库中缝合短剪辑以获得所需的运动ment。虽然这个主题目前是一个非常活跃的研究领域，但根据动画师的需求调整记录的运动的能力有限仍然是运动捕捉技术的主要缺点之一。

16.5 Physics-Based Animation

The world around us is governed by physical laws, many of which can be formalized as sets of partial or, in some simpler cases, ordinary differential equations. One of the original applications of computers was (and remains) solving such equations. It is therefore only natural to attempt to use numerical techniques developed over the several past decades to obtain realistic motion for computer animation.

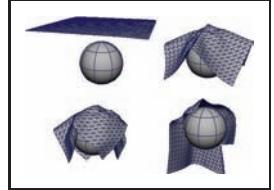


Figure 16.22. Realistic cloth simulation is often performed with physics-based methods. In this example, forces are due to collisions and gravity.

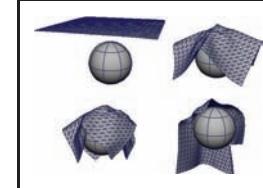
Because of its relative complexity and significant cost, physics-based animation is most commonly used in situations when other techniques are either unavailable or do not produce sufficiently realistic results. Prime examples include animation of fluids (which includes many gaseous phase phenomena described by the same equations—smoke, clouds, fire, etc.), cloth simulation (an example is shown in Figure 16.22), rigid body motion, and accurate deformation of elastic objects. Governing equations and details of commonly used numerical approaches are different in each of these cases, but many fundamental ideas and difficulties remain applicable across applications. Many methods for numerically solving ODEs and PDEs exist, but discussing them in details is far beyond the scope of this book. To give the reader a flavor of physics-based techniques and some of the issues involved, we will briefly mention here only the finite difference approach—one of the conceptually simplest and most popular families of algorithms which has been applied to most, if not all, differential equations encountered in animation.

The key idea of this approach is to replace a differential equation with its discrete analog—a difference equation. To do this, the continuous domain of interest is represented by a finite set of points at which the solution will be computed. In the simplest case, these are defined on a uniform rectangular grid as shown in Figure 16.23. Every derivative present in the original ODE or PDE is then replaced by its approximation through function values at grid points. One way of doing this is to subtract the function value at a given point from the function value for its neighboring point on the grid:

$$\frac{df(t)}{dt} \approx \frac{\Delta f}{\Delta t} = \frac{f(t + \Delta t) - f(t)}{\Delta t} \text{ or } \frac{\partial f(x, t)}{\partial x} \approx \frac{\Delta f}{\Delta x} = \frac{f(x + \Delta x, t) - f(x, t)}{\Delta x}. \quad (16.2)$$

These expressions are, of course, not the only way. One can, for example, use $f(t - \Delta t)$ instead of $f(t)$ above and divide by $2\Delta t$. For an equation containing a time derivative, it is now possible to propagate values of an unknown function forward in time in a sequence of Δt -size steps by solving the system of difference equations (one at each spatial location) for unknown $f(t + \Delta t)$. Some initial

我们周围的世界受物理定律的支配，其中许多可以被恶意化为部分集合，或者在一些更简单的情况下，普通微分等。计算机最初的应用之一是（现在仍然是）求解这样的方程。因此，尝试使用过去几十年发展起来的数值技术来获得计算机动画的真实运动是很自然的。



由于其相对复杂性和显着的成本，基于物理的animation最常用于其他技术不可用或无法产生足够逼真的结果的情况。主要的例子包括流体动画（包括许多由相同方程描述的气相现象—烟、云、火等）。），布料模拟（考试

在这个例子中，力是由碰撞和重力引起的。

如图16.22所示），刚体运动，弹性物体的精确变形。在每种情况下，常用数值方法的控制方程和细节都是不同的，但是许多基本的想法和困难仍然适用于不同的应用程序。存在许多数值求解ODEs和PDEs的方法，但详细讨论它们远远超出了本书的范围。为了让读者了解基于物理的技术和所涉及的一些问题，我们将在这里简要地提到有限差分方法——这是概念上最简单和最流行的算法家族之一，它已经应用于动画中反对的大多数（如果不是全部的话）微分方程。

这种方法的关键思想是用它的离散模拟替换微分方程—差分方程。为此，连续感兴趣的域由计算解的有限点集合表示。在最简单的情况下，这些定义在均匀的矩形网格上，如图16.23所示。然后，原始ODE或PDE中存在的每个导数都被其通过网格点处的函数值的近似值所取代。这样做的一种方法是从网格上其相邻点的函数值中减去给定点的函数值：

$\frac{df}{\Delta x}$ (16.2) 这些表达当然不是唯一的办法。人们可以，例如，用 $f(t + \Delta t)$ 替代上面的 $f(t)$ 并除以 $2\Delta t$ 。对于包含时间导数的方程，现在可以通过求解未知 $f(t + \Delta t)$ 的差分方程组（每个空间位置一个），以 Δt 大小的步骤序列在时间上向前传播未知函数的值。一些初始

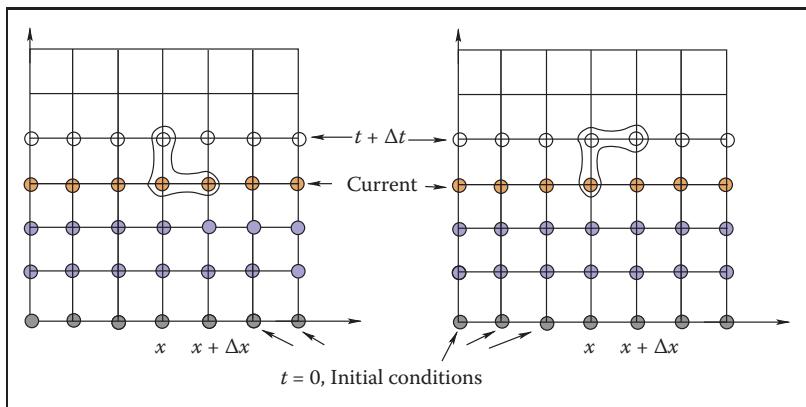


Figure 16.23. Two possible difference schemes for an equation involving derivatives $\partial f / \partial x$ and $\partial f / \partial t$. (Left) An explicit scheme expresses unknown values (open circles) only through known values at the current (orange circles) and possibly past (blue circles) time; (Right) Implicit schemes mix known and unknown values in a single equation making it necessary to solve all such equations as a system. For both schemes, information about values on the right boundary is needed to close the process.

conditions, i.e., values of the unknown function at $t = 0$, are necessary to start the process. Other information, such as values on the boundary of the domain, might also be required depending on the specific problem.

The computation of $f(t + \Delta t)$ can be done easily for so-called *explicit* schemes when all other values present are taken at the current time and the only unknown in the corresponding difference equation $f(t + \Delta t)$ is expressed through these known values. *Implicit* schemes mix values at current and future times and might use, for example,

$$\frac{f(x + \Delta x, t + \Delta t) - f(x, t + \Delta t)}{\Delta x}$$

as an approximation of $\frac{\partial f}{\partial x}$. In this case one has to solve a system of algebraic equations at each step.

The choice of difference scheme can dramatically affect all aspects of the algorithm. The most obvious among them is *accuracy*. In the limit $\Delta t \rightarrow 0$ or $\Delta x \rightarrow 0$, expressions of the type in Equation (16.2) are exact, but for finite step size some schemes allow better approximation of the derivative than others. *Stability* of a difference scheme is related to how fast numerical errors, which are always present in practice, can grow with time. For stable schemes this growth is bounded, while for unstable ones it is exponential and can quickly overwhelm the solution one seeks (see Figure 16.24). It is important to realize that while some inaccuracy in the solution is tolerable (and, in fact, accuracy demanded in physics and engineering is rarely needed for animation), an unstable result is completely

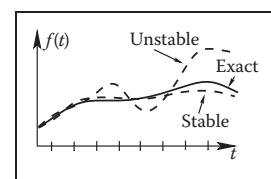


Figure 16.24. An unstable solution might follow the exact one initially, but can deviate arbitrarily far from it with time. Accuracy of a stable solution might still be insufficient for a specific application.

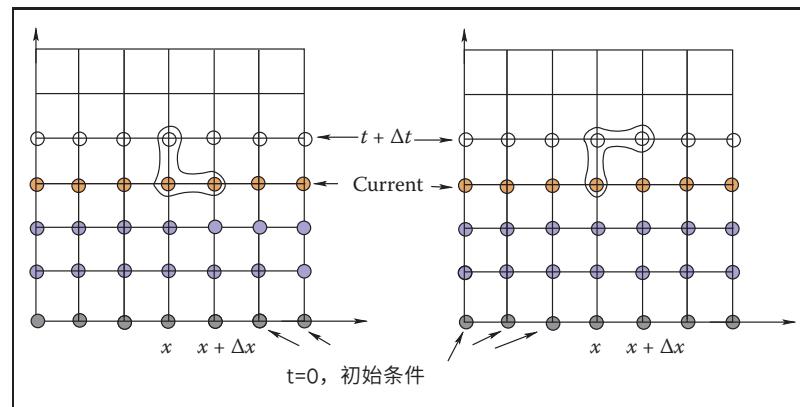


图16.23。 涉及导数 $\partial f / \partial x$ 和 $\partial f / \partial t$ 的方程的两个可能的差分方案。(左) 显式方案仅通过当前(橙色圆圈)和可能过去(蓝色圆圈)时间的已知值来表达未知值(开放圆圈);(右) 隐式方案将已知和未知值混合在一个方程中,因此必须对于这两种方案,都需要有关右边界上的值的信息来关闭过程。

条件,即未知函数在 $t=0$ 时的值,是启动该过程所必需的。其他信息(如域边界上的值)也可能需要,具体取决于特定问题。

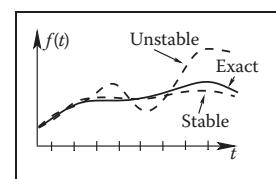
对于所谓的显式方案, $f(t + \Delta t)$ 的计算可以很容易地完成,当存在的所有其他值都在当前时间取时,并且相应的差分方程 $f(t + \Delta t)$ 中唯一的未知值通过这些已知值表示时。隐式方案在当前和将来混合值,并可能使用,例如

$$\frac{f(x + \Delta x, t + \Delta t) - f(x, t + \Delta t)}{\Delta x}$$

作为 ∂f 的近似值
 Δx . 在这种情况下,必须解决代数系统
每个步骤的方程。

差异方案的选择可以显著影响算法的各个方面。其中最明显的是准确性。在极限 $\Delta t \rightarrow 0$ 或 $\Delta x \rightarrow 0$ 中,方程(16.2)中的类型表达式是精确的,但对于有限步长,一些方案允许比其他方案更好地近似导数。差值方案的稳定性与实际中始终存在的数值误差随时间增长的速度有关。对于稳定的方案,这种增长是有界的,而对于不稳定的方案,它是指数的,可以迅速压倒人们寻求的解决方案(见图16.24)。重要的是要认识到,虽然有些

解决方案中的不准确性是可以容忍的(事实上,动画很少需要物理和工程中要求的精度),一个不稳定的结果是完全



一个不稳定的解决方案最初可能会遵循前行为,但随着时间的推移,devi可以任意地远离它。
对于特定的应用,稳定的解决方案的准确性可能仍然不够。

meaningless, and one should avoid using unstable schemes. Generally, explicit schemes are either unstable or can become unstable at larger step sizes while implicit ones are unconditionally stable. Implicit schemes allows greater step size (and, therefore, fewer steps) which is why they are popular despite the need to solve a system of algebraic equations at each step. Explicit schemes are attractive because of their simplicity if their stability conditions can be satisfied. Developing a good difference scheme and corresponding algorithm for a specific problem is not easy, and for most standard situations it is well advised to use an existing method. Ample literature discussing details of these techniques is available.

One should remember that, in many cases, just computing all necessary terms in the equation is a difficult and time-consuming task on its own. In rigid body or cloth simulation, for example, most of the forces acting on the system are due to collisions among objects. At each step during animation, one therefore has to solve a purely geometric, but very nontrivial, problem of collision detection. In such conditions, schemes which require fewer evaluations of such forces might provide significant computational savings.

Although the result of solving appropriate time-dependant equations gives very realistic motion, this approach has its limitations. First of all, it is very hard to control the result of physics-based animation. Fundamental mathematical properties of these equations state that once the initial conditions are set, the solution is uniquely defined. This does not leave much room for animator input and, if the result is not satisfactory for some reason, one has only a few options. They are mostly limited to adjusting initial condition used, changing physical properties of the system, or even modifying the equations themselves by introducing artificial terms intended to "drive" the solution in the direction the animator wants. Making such changes requires significant skill as well as understanding of the underlying physics and, ideally, numerical methods. Without this knowledge, the realism provided by physics-based animation can be destroyed or severe numerical problems might appear.

16.6 Procedural Techniques

Imagine that one could write (and implement on a computer) a mathematical function which outputs precisely the desired motion given some animator guidance. Physics-based techniques outlined above can be treated as a special case of such an approach when the "function" involved is the procedure to solve a particular differential equation and "guidance" is the set of initial and boundary conditions, extra equation terms, etc.

无意义，并且应该避免使用不稳定的方案。通常，显式方案要么不稳定，要么在较大的步长下变得不稳定，而隐式方案是无条件稳定的。隐式方案允许更大的步长（因此，更少的步骤），这就是为什么尽管需要在每个步骤求解代数方程组，但它们仍然很受欢迎。明确的方案是有吸引力的，因为它们的简单性，如果他们的稳定性条件可以满足。为特定问题开发一个好的差异方案和相应的算法并不容易，对于大多数标准情况，最好使用现有的方法。大量的文献讨论这些技术的细节是可用的。

人们应该记住，在许多情况下，仅仅计算方程中的所有必要项本身就是一项困难且耗时的任务。例如，在刚体或布料模拟中，作用在系统上的大部分力是由于物体之间的碰撞造成的。因此，在动画过程中的每一步，都必须解决一个纯粹几何但非常平凡的碰撞检测问题。在这种情况下，需要对这种力进行较少评估的方案可能会节省大量的计算费用。

虽然求解适当的时间依赖方程的结果给出了非常现实的运动，但这种方法有其局限性。首先，控制基于物理的动画的结果是非常困难的。这些方程的基本数学特性表明，一旦初始条件设置，solution是唯一定义的。这不会为动画师输入留下太多空间，如果由于某种原因导致结果不满意，则只有几个选项。它们主要限于调整使用的初始条件，改变系统的物理属性，甚至通过引入人工术语来修改方程本身，这些术语旨在将解"驱动"到动画师想要的方向。做出这样的改变需要显着的技能，以及对基础物理的理解，理想情况下，数值方法。如果没有这些知识，基于物理的动画提供的真实感可能会被破坏，或者可能会出现严重的数值失真。

16.6 程序技术

想象一下，人们可以编写（并在计算机上实现）一个数学函数，在动画师的指导下精确输出所需的运动。上面概述的基于物理的技术可以被视为这种方法的特例，当涉及的"函数"是求解特定微分方程的过程，"指导"是初始和边界条件的集合，额外的方程项等。

However, if we are only concerned with the final result, we do not have to follow a physics-based approach. For example, a simple constant amplitude wave on the surface of a lake can be directly created by applying the function $f(x, t) = A \cos(\omega t - kx + \phi)$ with constant frequency ω , wave vector k and phase ϕ to get displacement at the 2D point x at time t . A collection of such waves with random phases and appropriately chosen amplitudes, frequencies, and wave vectors can result in a very realistic animation of the surface of water without explicitly solving any fluid dynamics equations. It turns out that other rather simple mathematical functions can also create very interesting patterns or objects. Several such functions, most based on lattice noises, have been described in Section 11.5. Adding time dependance to these functions allows us to animate certain complex phenomena much easier and cheaper than with physics-based techniques while maintaining very high visual quality of the results. If $\text{noise}(x)$ is the underlying pattern-generating function, one can create a time-dependant variant of it by moving the argument position through the lattice. The simplest case is motion with constant speed: $\text{timenoise}(x, t) = \text{noise}(x + vt)$, but more complex motion through the lattice is, of course, also possible and, in fact, more common. One such path, a spiral, is shown in Figure 16.25. Another approach is to animate parameters used to generate the noise function. This is especially appropriate if the appearance changes significantly with time—a cloud becoming more turbulent, for example. In this way one can animate the dynamic process of formation of clouds using the function which generates static ones.

For some procedural techniques, time dependance is a more integral component. The simplest *cellular automata* operate on a 2D rectangular grid where a binary value is stored at each location (cell). To create a time varying pattern, some user-provided rules for modifying these values are repeatedly applied. Rules typically involve some set of conditions on the current value and that of the cell's neighbors. For example, the rules of the popular 2D *Game of Life* cellular automaton invented in 1970 by British mathematician John Conway are the following:

1. A dead cell (i.e., binary value at a given location is 0) with exactly three live neighbors becomes a live cell (i.e., its value set to 1).
2. A live cell with two or three live neighbors stays alive.
3. In all other cases, a cell dies or remains dead.

Once the rules are applied to all grid locations, a new pattern is created and a new evolution cycle can be started. Three sample snapshots of the live cell distribution at different times are shown in Figure 16.26. More sophisticated automata

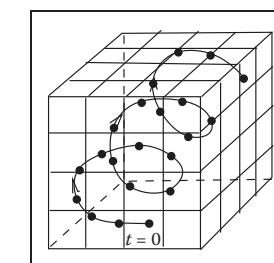


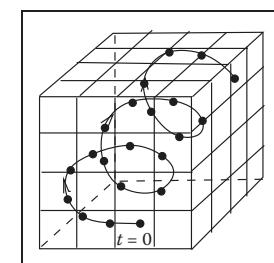
Figure 16.25. A path through the cube defining procedural noise is traversed to animate the resulting pattern.

但是,如果我们只关注最终结果,我们不必遵循基于物理的方法。例如,可以通过应用具有恒定频率 ω , 波矢 k 和相位 ϕ 的函数 $f(x, t) = \text{acos}(wt - kx + \phi)$ 来直接创建湖泊表面上的简单恒定振幅波,以获得在时间t的2D点 x 处这些具有随机相位和适当选择的振幅、频率和波矢量的波的集合可以产生一个非常真实的水面动画,并明确地求解任何流体动力学方程。事实证明,其他相当简单的数学函数也可以创建非常有趣的模式或对象。几种这样的函数 大多数是基于晶格噪声 已在第11.5节中描述.为这些函数添加时间依赖性使我们能够比使用基于物理的技术更容易和更便宜地动画某些复杂现象,同时保持非常高的结果视觉质量。如果噪声(x)是非干扰模式生成函数,则可以通过将参数位置移动到晶格中来创建它的时间依赖变体。最简单的情况是具有恒定速度的运动: $\text{timenoise}(x, t) = \text{noise}(x + vt)$,但是通过晶格的更复杂的运动当然也是可能的,实际上更常见。一个这样的路径,螺旋,如图16.25所示。另一种方法是对用于生成噪声函数的p a配子进行动画处理.如果外观随时间显着变化(例如,云变得更加湍流),这尤其合适。通过这种方式,可以使用生成静态云的函数来动画云形成的动态过程。

对于某些程序技术,时间依赖性是一个更重要的组成部分。最简单的元胞自动机在二维矩形网格上运行,其中在每个位置(单元格)存储二进制值。为了创建时变模式,会重复应用一些用户提供的用于修改这些值的规则。规则通常涉及当前值和小区邻居值的一些条件集。例如,英国数学家约翰康威在1970年发明的流行的2d生命元胞自动机的规则如下:

- 1.一个死细胞(即给定位置的二进制值为0)与三个活邻居成为一个活细胞(即其值设置为1)。
- 2.有两个或三个活邻居的活细胞保持活力。
- 3.在所有其他情况下,细胞死亡或仍然死亡。

一旦规则应用于所有网格位置,就会创建一个新的模式,并可以开始一个新的进化周期。不同时间的活细胞分布的三个样品快照如图16.26所示。更复杂的自动机



通过定义procedural噪声的立方体的路径被遍历以动画生成的模式。

simultaneously operate on several 3D grids of possibly floating point values and can be used for modeling dynamics of clouds and other gaseous phenomena or biological systems for which this apparatus was originally invented (note the terminology). Surprising pattern complexity can arise from just a few well-chosen rules, but how to write such rules to create the desired behavior is often not obvious. This is a common problem with procedural techniques: there is only limited, if any, guidance on how to create new procedures or even adjust parameters of existing ones. Therefore, a lot of tweaking and learning by trial-and-error ("by experience") is usually needed to unlock the full potential of procedural methods.

Another interesting approach which was also originally developed to describe biological objects is the technique called *L-systems* (after the name of their original inventor, Astrid Lindenmayer). This approach is based on *grammars* or sets of recursive rules for rewriting strings of symbols. There are two types of symbols: *terminal symbols* stand for elements of something we want to represent with a grammar. Depending on their meaning, grammars can describe structure of trees and bushes, buildings and whole cities, or programming and natural languages. In animation, L-systems are most popular for representing plants and corresponding terminals are instructions to the geometric modeling system: put a leaf (or a branch) at a current position—we will use the symbol @ and just draw a circle, move current position forward by some number of units (symbol f), turn current direction 60 degrees around world Z-axis (symbol +), pop (symbol [) or push (symbol]) current position/orientation, etc. Auxiliary *nonterminal symbols* (denoted by capital letters) have only semantic rather than any direct meaning. They are intended to be eventually rewritten through terminals. We start from the special nonterminal start symbol S and keep applying grammar rules to the current string in parallel, i.e., replace all nonterminals currently present to get the new

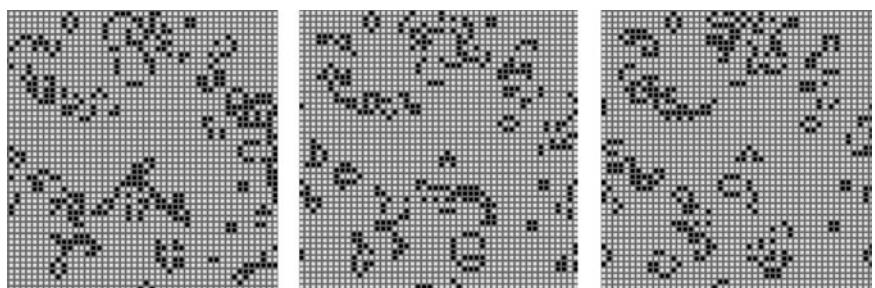


Figure 16.26. Several (non-consecutive) stages in the evolution of a *Game of Life* automaton. Live cells are shown in black. Stable objects, oscillators, traveling patterns, and many other interesting constructions can result from the application of very simple rules. *Figure created using a program by Alan Hensel.*

同时操作几个可能是浮点值的三维网格，并可用于模拟云和其他气体现象或生物系统的动力学，这是该装置最初发明的（注意术语）。令人惊讶的模式复杂性可能来自少数精心选择的规则，但如何编写此类规则以创建所需的行为通常并不明显。这是程序技术的一个常见问题：只有有限的指导，如果有的话，如何创建新的程序，甚至调整现有程序的参数。因此，通常需要通过反复试验（“经验”）进行大量调整和学习，以释放程序方法的全部潜力。另一个有趣的方法也最初被开发来描述生物对象是称为L-systems的技术（以其原始发明者Astrid Lindenmayer的名字命名）。这种方法基于语法或递归规则集，用于重写符号字符串。有两种类型的符号：终端符号代表我们想要用语法表示的元素。根据其含义，语法可以描述树木和灌木丛，建筑物和整个城市的结构，或编程和自然语言。在动画中，L-系统最常用于表示植物，相应的终端是几何建模系统的指令：在当前位置放置一个叶子（或一个分支）—我们将使用符号@，只画一个圆圈，将当前位置向前移动一定数量的单位（符号f），将当前方向绕世界Z轴（符号+）旋转60度，pop（符号[）或push（符号]）当前位置方向等。辅助非终结符号（用大写字母表示）只有语义而不是任何直接含义。它们旨在最终通过终端重写。我们从特殊的nonterminal开始符号S开始，并继续并行地对当前字符串应用语法规则，即替换当前存在的所有nonterminal以获得新的

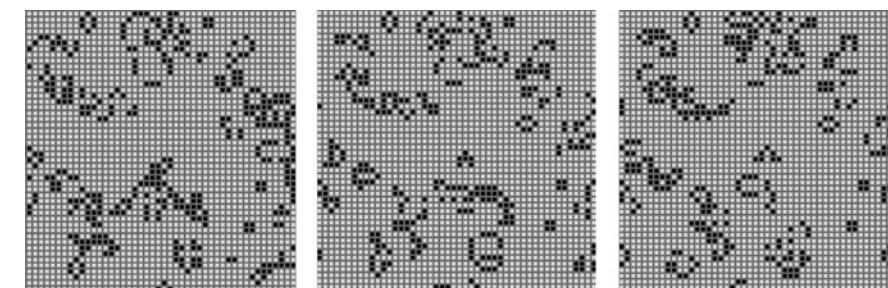


图16.26。 生命自动机游戏进化中的几个（非连续）阶段。活细胞以黑色显示。应用非常简单的规则可以产生稳定的对象、振荡器、行进模式和许多其他有趣的结构。使用alanHensel的程序创建的图。



string, until we end up with a string containing only terminals and no more substitution is therefore possible. This string of modeling instructions is then used to output the actual geometry. For example, a set of rules (productions)

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow [+B]fA \\ A &\rightarrow B \\ B &\rightarrow fB \\ B &\rightarrow f@ \end{aligned}$$

might result in the following sequence of rewriting steps demonstrated in Figure 16.27:

$$\begin{aligned} S &\mapsto A \mapsto [+B]fA \mapsto [+fB]f[+B]fA \mapsto \\ &\quad [+ff@]f[+fB]fB \mapsto [+ff@]f[+ff@]ff@. \end{aligned}$$

As shown above, there are typically many different productions for the same non-terminal allowing the generation of many different objects with the same grammar. The choice of which rule to apply can depend on which symbols are located next to the one being replaced (context-sensitivity) or can be performed at random with some assigned probability for each rule (stochastic L-systems). More complex rules can model interaction with the environment, such as pruning to a particular shape, and parameters can be associated with symbols to control geometric commands issued.

L-systems already capture plant topology changes with time: each intermediate string obtained in the rewriting process can be interpreted as a "younger" version of the plant (see Figure 16.27). For more significant changes, different productions can be in effect at different times allowing the structure of the plant to change significantly as it grows. A young tree, for example, produces a lot of new branches, while an older one branches only moderately.

Very realistic plant models have been created with L-systems. However, as with most procedural techniques, one needs some experience to meaningfully apply existing L-systems, and writing new grammars to capture some desired effect is certainly not easy.

16.7 Groups of Objects

To animate multiple objects one can, of course, simply apply standard techniques described in the chapter so far to each of them. This works reasonably well for a moderate number of independent objects whose desired motion is known in

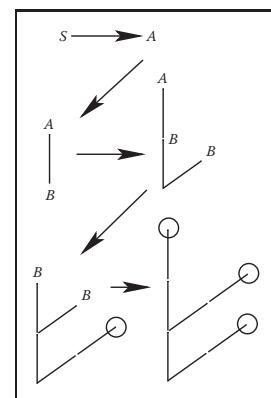


Figure 16.27. Consecutive derivation steps using a simple L-system. Capital letters denote nonterminals and illustrate positions at which corresponding nonterminal will be expanded. They are not part of the actual output.



字符串，直到我们最终得到一个只包含终端的字符串，因此不可能再进行替换。这串建模指令然后用于输出实际几何形状。例如，一组规则（制作）

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow [+B]fA \\ A &\rightarrow B \\ B &\rightarrow fB \\ B &\rightarrow f@ \end{aligned}$$

可能会导致以下重写步骤的顺序如图16.27所示：

$$\begin{aligned} S &\mapsto A \mapsto [+B]fA \mapsto [+fB]f[+B]fA \mapsto \\ &\quad [+ff@]f[+fB]fB \mapsto [+ff@]f[+ff@]ff@. \end{aligned}$$

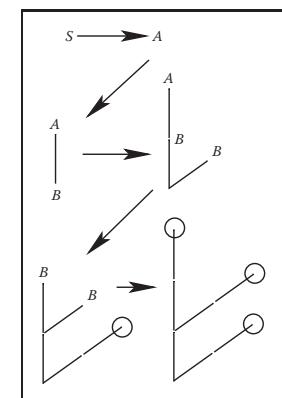
如上所示，对于相同的非终端通常有许多不同的生产，允许产生具有相同grammar的许多不同对象。应用哪个规则的选择可以取决于哪些符号位于被替换的符号旁边（上下文敏感性），或者可以在random中执行，每个规则都有一定的概率（随机L系统）。更复杂的规则可以建模与环境的交互，例如修剪到特定形状，并且参数可以与符号相关联，以控制发出的geo度量命令。

L-systems已经捕获植物拓扑随时间的变化：在重写过程中获得的每个intermediate字符串都可以被解释为植物的“年轻”版本（见图16.27）。对于更显着的变化，不同的生产可以在不同的时间生效，允许植物的结构随着它的生长而显着变化。例如，一棵年轻的树会产生很多新的树枝，而一棵较老的树只会适度地分支。

使用L-systems创建了非常逼真的工厂模型。然而，与大多数程序技术一样，人们需要一些经验才能有意义地应用现有的L系统，并且编写新的语法来捕捉一些期望的效果肯定是不容易的。

16.7 对象组

当然，要为多个对象制作动画，只需将本章中描述的标准技术应用于每个对象即可。这适用于中等数量的独立对象，其期望的运动是已知的



使用simpleL-系统的连续推导步骤。
大写字母表示nonterminals
和illustrate位置，在该位置将扩展correspondingnonterminal。它们不是实际输出的一部分。

advance. However, in many cases, some kind of coordinated action in a dynamic environment is necessary. If only a few objects are involved, the animator can use an artificial intelligence (AI)-based system to automatically determine immediate tasks for each object based on some high-level goal, plan necessary motion, and execute the plan. Many modern games use such *autonomous objects* to create smart monsters or player's collaborators.

Interestingly, as the number of objects in a group grows from just a few to several dozens, hundreds, and thousands, individual members of a group must have only very limited “intelligence” in order for the group as a whole to exhibit what looks like coordinated goal-driven motion. It turns out that this *flocking* is *emergent behavior* which can arise as a result of limited interaction of group members with just a few of their closest neighbors (Reynolds, 1987). Flocking should be familiar to anyone who has observed the fascinatingly synchronized motion of a flock of birds or a school of fish. The technique can also be used to control groups of animals moving over terrain or even a human crowd.

At any given moment, the motion of a member of a group, often called boid when applied to flocks, is the result of balancing several often contradictory tendencies, each of which suggests its own velocity vector (see Figure 16.28). First, there are external physical forces F acting on the boid, such as gravity or wind. New velocity due to those forces can be computed directly through Newton’s law as

$$\mathbf{v}_{new}^{physics} = \mathbf{v}_{old} + \mathbf{F}\Delta t/m.$$

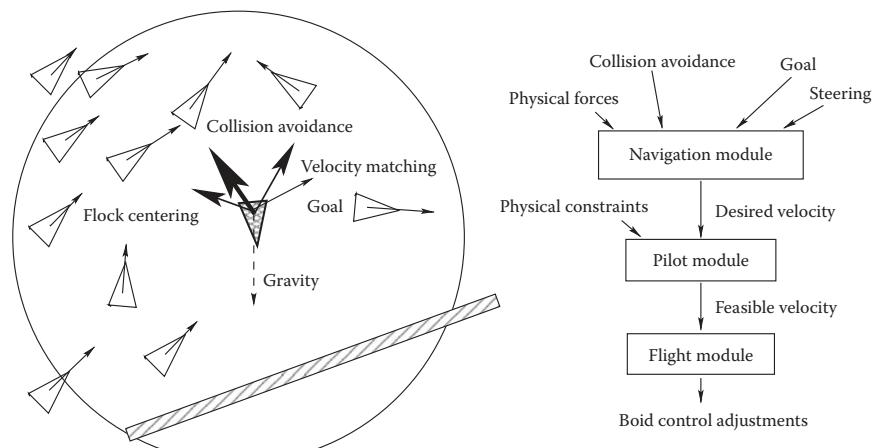


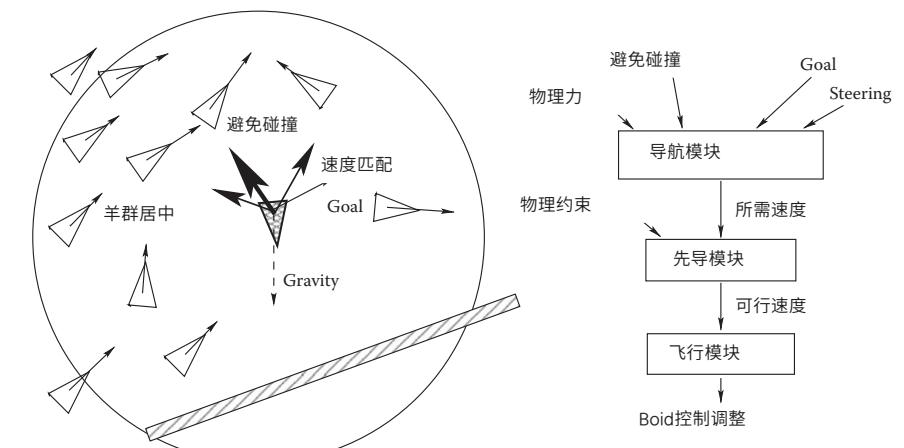
Figure 16.28. (Left) Individual flock member (boid) can experience several urges of different importance (shown by line thickness) which have to be negotiated into a single velocity vector. A boid is aware of only its limited neighborhood (circle). (Right) Boid control is commonly implemented as three separate modules.

前进。然而，在许多情况下，在动态环境中采取某种协调行动是必要的。如果仅涉及少数对象，则动画师可以使用基于人工智能(AI)的系统基于一些高级目标自动确定每个对象的即时任务，规划必要的运动，并执行计划。许多现代游戏使用这样的自主对象来创建智能怪物或玩家的合作者。

有趣的是，随着一个组中的对象数量从几个增长到几十个，几百个和数千个，一个组的单个成员必须只有非常有限的“智力”，才能使整个组表现出看起来像协调的目标驱动运动。事实证明，这种蜂拥是紧急行为，可能是由于小组成员与几个最亲密的邻居的有限互动而产生的结果（雷诺兹，1987）。对于任何观察过一群鸟或一群鱼的迷人同步运动的人来说，植绒应该是熟悉的。该技术还可用于控制在地形或甚至人类人群中移动的动物群。

在任何给定时刻，一个群体成员的运动，当应用于羊群时通常称为boid，是平衡几个经常相互矛盾的十个dencies的结果，每个dencies都暗示了自己速度矢量（见图16.28）。首先，有外部物理力 F 作用在boid上，例如重力或风。由于这些力引起的新速度可以通过牛顿定律直接计算为

$$v_{\text{物理新}} = \mathbf{v}_{old} + \mathbf{F}\Delta t/m.$$



(左) 单个羊群成员 (boid) 可以经历几个不同重要性的冲动 (由线厚度显示)，这些冲动必须协商成一个单一的速度向量。Boid只知道其有限的邻居 (圆圈)。(右) Boid控制通常作为三个独立的模块实现。



Second, a boid should react to global environment and to the behavior of other group members. Collision avoidance is one of the main results of such interaction. It is crucial for flocking that each group member has only limited field of view, and therefore is aware only of things happening within some neighborhood of its current position. To avoid objects in the environment, the simplest, if imperfect, strategy is to set up a limited extent repulsive force field around each such object. This will create a second desired velocity vector $v_{new}^{col_avoid}$, also given by Newton's law. Interaction with other group members can be modeled by simultaneously applying different steering behaviors resulting in several additional desired velocity vectors v_{new}^{steer} . Moving away from neighbors to avoid crowding, steering toward flock mates to ensure flock cohesion, and adjusting a boid's speed to align with average heading of neighbors are most common. Finally, some additional desired velocity vectors v_{new}^{goal} are usually applied to achieve needed global goals. These can be vectors along some path in space, following some specific designated leader of the flock, or simply representing migratory urge of a flock member.

Once all v_{new} are determined, the final desired vector is negotiated based on priorities among them. Collision avoidance and velocity matching typically have higher priority. Instead of simple averaging of desired velocity vectors which can lead to cancellation of urges and unnatural "moving nowhere" behavior, an acceleration allocation strategy is used. Some fixed total amount of acceleration is made available for a boid and fractions of it are being given to each urge in order of priority. If the total available acceleration runs out, some lower priority urges will have less effect on the motion or be completely ignored. The hope is that once the currently most important task (collision avoidance in most situations) is accomplished, other tasks can be taken care of in near future. It is also important to respect some physical limitations of real objects, for example, clamping too high accelerations or speeds to some realistic values. Depending on the internal complexity of the flock member, the final stage of animation might be to turn the negotiated velocity vector into a specific set of parameters (bird's wing positions, orientation of plane model in space, leg skeleton bone configuration) used to control a boid's motion. A diagram of a system implementing flocking is shown on Figure 16.28 (right).

A much simpler, but still very useful, version of group control is implemented by *particle systems* (Reeves, 1983). The number of particles in a system is typically much larger than number of boids in a flock and can be in the tens or hundreds of thousands, or even more. Moreover, the exact number of particles can fluctuate during animation with new particles being born and some of the old ones destroyed at each step. Particles are typically completely independent from



其次，boid应该对全球环境和其他组成员的行为做出反应。避免碰撞是这种相互作用的主要结果之一。对于蜂拥而至的是，每个小组成员只有有限的视野，因此只知道在其当前位置的某个附近发生的事情。为了避免环境中物体，最简单的（如果有影响的话）策略是在每个这样的物体周围设置有限程度的排斥力场。这将创建第二个期望的速度矢量 v_{col} 。

，也由牛顿定律给出。与其他组成员的交互可以通过多层应用不同的转向行为来建模，从而产生几个额外的期望速度矢量 v_{turn}

新的。远离邻居以避免拥挤，转向羊群伴侣以确保羊群凝聚力，以及调整boid的速度以与邻居的平均航向保持一致是最常见的。最后，一些additional期望的速度矢量 v_{target}

新的应用通常是为了实现所需的全局目标。这些可以是沿着空间某条路径的载体，跟随某个特定的羊群领导者，或者仅仅代表一个羊群成员的迁徙冲动。

一旦确定了所有 v_{new} ，则基于它们之间的优先级来协商最终的期望向量。避免碰撞和速度匹配通常具有更高的优先级。使用acceleration分配策略代替了可能导致消除冲动和不自然的"无处移动"行为的期望速度矢量的简单平均。为boid提供了一些固定的总加速度量，并且按优先顺序将其部分给予每个敦促。如果总的可用加速度用完，一些较低优先级的敦促将对运动产生较少的影响或被完全忽略。希望是，一旦完成了目前最重要的任务（在大多数情况下避免碰撞），就可以在不久的将来处理其他任务。同样重要的是要尊重真实物体的一些物理限制，例如，将过高的加速度或速度夹紧到一些现实值。根据flock成员的内部复杂性，动画的最后阶段可能是将协商的速度矢量转换为用于控制boid运动的一组特定参数（鸟的机翼位置，平面模型在空间中的方向，腿图16.28（右）显示了实现植绒的系统图）。

一个更简单，但仍然非常有用的群体控制版本是由粒子系统实现的（Reeves, 1983）。一个系统中粒子的数量通常比一群boids的数量大得多，可以是数十个或数千个，甚至更多。此外，在动画过程中，随着新粒子的诞生和一些旧粒子在每个步骤中被破坏，粒子的确切数量可能会波动。粒子通常完全独立于

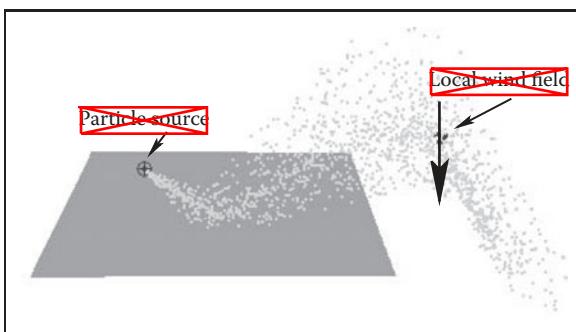


Figure 16.29. After being emitted by a directional source, particles collide with an object and then are blown down by a local wind field once they clear the obstacle.

each other, ignoring one's neighbors and interacting with the environment only by experiencing external forces and collisions with objects, *not* through collision avoidance as was the case for flocks. At each step during animation, the system first creates new particles with some initial parameters, terminates old ones, and then computes necessary forces and updates velocities and positions of the remaining particles according to Newton's law.

All parameters of a particle system (number of particles, particle life span, initial velocity, and location of a particle, etc.) are usually under the direct control of the animator. Prime applications of particle systems include modeling fireworks, explosions, spraying liquids, smoke and fire, or other fuzzy objects and phenomena with no sharp boundaries. To achieve a realistic appearance, it is important to introduce some randomness to all parameters, for example, having a random number of particles born (and destroyed) at each step with their velocities generated according to some distribution. In addition to setting appropriate initial parameters, controlling the motion of a particle system is commonly done by creating a specific force pattern in space—blowing a particle in a new direction once it reaches some specific location or adding a center of attraction, for example. One should remember that with all their advantages, simplicity of implementation and ease of control being the prime ones, particle systems typically do not provide the level of realism characteristic of true physics-based simulation of the same phenomena.

Notes

In this chapter we have concentrated on techniques used in 3D animation. There also exist a rich set of algorithms to help with 2D animation production and post-

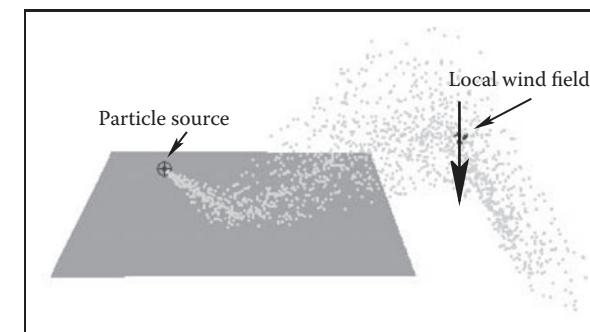


Figure 16.29. After being emitted by a directional source, particles collide with an object and then are blown down by a local wind field once they clear the obstacle.

each other, ignoring one's neighbors and interacting with the environment only by experiencing external forces and collisions with objects, *not* through collision avoidance as was the case for flocks. At each step during animation, the system first creates new particles with some initial parameters, terminates old ones, and then computes necessary forces and updates velocities and positions of the remaining particles according to Newton's law.

All parameters of a particle system (number of particles, particle life span, initial velocity, and location of a particle, etc.) are usually under the direct control of the animator. Prime applications of particle systems include modeling fireworks, explosions, spraying liquids, smoke and fire, or other fuzzy objects and phenomena with no sharp boundaries. To achieve a realistic appearance, it is important to introduce some randomness to all parameters, for example, having a random number of particles born (and destroyed) at each step with their velocities generated according to some distribution. In addition to setting appropriate initial parameters, controlling the motion of a particle system is commonly done by creating a specific force pattern in space—blowing a particle in a new direction once it reaches some specific location or adding a center of attraction, for example. One should remember that with all their advantages, simplicity of implementation and ease of control being the prime ones, particle systems typically do not provide the level of realism characteristic of true physics-based simulation of the same phenomena.

Notes

In this chapter we have concentrated on techniques used in 3D animation. There also exist a rich set of algorithms to help with 2D animation production and post-



processing of images created by computer graphics rendering systems. These include techniques for cleaning up scanned-in artist drawings, feature extraction, automatic 2D in-betweening, colorization, image warping, enhancement and compositing, and many others.

One of the most significant developments in the area of computer animation has been the increasing power and availability of sophisticated animation systems. While different in their specific set of features, internal structure, details of user interface, and price, most such systems include extensive support not only for animation, but also for modeling and rendering, turning them into complete production platforms. It is also common to use these systems to create still images. For example, many images for figures in this section were produced using Maya software generously donated by Alias.

Large-scale animation production is an extremely complex process which typically involves a combined effort by dozens of people with different backgrounds spread across many departments or even companies. To better coordinate this activity, a certain production pipeline is established which starts with a story and character sketches, proceeds to record necessary sound, build models, and rig characters for animation. Once actual animation commences, it is common to go back and revise the original designs, models, and rigs to fix any discovered motion and appearance problems. Setting up lighting and material properties is then necessary, after which it is possible to start rendering. In most sufficiently complex projects, extensive postprocessing and compositing stages bring together images from different sources and finalize the product.

We conclude this chapter by reminding the reader that in the field of computer animation, any technical sophistication is secondary to a good story, expressive characters, and other artistic factors, most of which are hard or simply impossible to quantify. It is safe to say that Snow White and her seven dwarfs will always share the screen with green ogres and donkeys, and most of the audience will be much more interested in the characters and the story rather than in which, if any, computers (and in what exact way) helped to create them.



计算机图形渲染系统创建的图像的处理。这些技术包括清理扫描艺术家绘画、特征提取、自动2d中间转换、彩色化、图像翘曲、增强和合成等技术。

计算机动画领域最重要的发展之一是复杂动画系统的能力和可用性不断提高。虽然它们的特定功能，内部结构，用户界面细节和价格有所不同，但大多数此类系统不仅包括对动画的广泛支持，还包括对建模和渲染的广泛支持，将它们使用这些系统来创建静止图像也是常见的。例如，本节中的许多数字图像都是使用Maya软件通过Alias慷慨捐赠的。

大型动画制作是一个极其复杂的过程，它通常涉及到数十个不同背景的人的共同努力，分布在许多部门甚至公司。为了更好地协调这一交流，建立了一个特定的生产管道，从故事和角色草图开始，继续录制必要的声音，建立模型，并为动画装备角色。实际动画开始后，通常会返回并修改原始设计、模型和钻机，以修复任何发现的运动和外观问题。设置光照和材质属性是必要的，之后就可以开始渲染了。在大多数足够复杂的项目中，广泛的后处理和合成阶段将来自不同来源的图像汇集在一起并最终确定产品。

我们通过提醒读者来结束本章，在计算机动画领域，任何技术复杂性都是次要的一个好故事，富有表现力的人物和其他艺术因素，其中大部分是难以或根本可以肯定地说，白雪公主和她的七个小矮人将永远与绿色食人魔和驴子共享屏幕，大多数观众将对角色和故事更感兴趣，而不是计算机（以及确切的方式）



Using Graphics Hardware

17.1 Hardware Overview

Throughout most of this book, the focus is on the fundamentals that underly computer graphics rather than on any specifics relating to the APIs or hardware on which the algorithms may be implemented. This chapter takes a slightly different route and blends the details of using graphics hardware with some of the practical issues associated with programming that hardware. The chapter is designed to be an introductory guide to graphics hardware and could be used as the basis for a set of weekly labs that investigate graphics hardware.

17.2 What Is Graphics Hardware

Graphics hardware describes the hardware components necessary to quickly render 3D objects as pixels on your computer's screen using specialized rasterization-based (and in some cases, ray-tracer-based) hardware architectures. The use of the term *graphics hardware* is meant to elicit a sense of the physical components necessary for performing a range of graphics computations. In other words, the hardware is the set of chipsets, transistors, buses, processors, and computing cores found on current video cards. As you will learn in this chapter, and eventually experience yourself, current graphics hardware is very good at processing



使用图形硬件

17.1 硬件概述

在本书的大部分内容中，重点是计算机图形学的基础知识，而不是与算法可以实现的API或硬件相关的任何细节。本章采用略有不同的路线，并将使用图形硬件的细节与与硬件编程相关的一些实际问题混合在一起。本章旨在作为图形硬件的入门指南，并可作为研究图形硬件的每周实验室的基础。

17.2 什么是图形硬件

图形硬件描述了使用专门的基于光栅化（在某些情况下，基于光线追踪器）的硬件体系结构将3d对象快速渲染为计算机屏幕上的像素所需的硬件组件。图形硬件一词的使用是为了让人们了解执行一系列图形计算所必需的物理组件。换句话说，硬件是当前视频卡上发现的芯片组，晶体管，总线，处理器和计算核心的集合。正如您将在本章中学习，并最终体验自己，当前的图形硬件非常擅长处理

descriptions of 3D objects and transforming those representations into the colored pixels that fill your monitor.

Real-Time Graphics: By real-time graphics, we generally mean that the graphics-related computations are being carried out fast enough that the results can be viewed immediately. Being able to conduct operations at 60Hz or higher is considered real time. Once the time to refresh the display (*frame rate*) drops below 15Hz, the speed is considered more interactive than it is real-time, but this distinction is not critical. Because the computations need to be fast, the equations used to render the graphics are often approximations to what could be done if more time were available.

Graphics hardware has certainly changed very *rapidly* over the last decade. Newer graphics hardware provides more parallel processing capabilities, as well as better support for specialized rendering. One explanation for the fast pace is the video game industry and its economic momentum. Essentially what this means is that each new graphics card provides better performance and processing capabilities. As a result, video games appear more visually realistic. The processors on graphics hardware, often called GPUs, or Graphics Processing Units, are highly parallel and afford thousands of concurrent threads of execution. The hardware is designed for throughput which allows larger numbers of pixels and vertices to be processed in shorter amounts of time. All of this parallelism is good for graphics algorithms, but other work has benefited from the parallel hardware. In addition to video games, GPUs are used to accelerate physics computations, develop real-time ray tracing codes, solve Navier-Stokes related equations for fluid flow simulations, and develop faster codes for understanding the climate (Purcell, Buck, Mark, & Hanrahan, 2002; S. G. Parker et al., 2010; Harris, 2004). Several APIs and SDKs have been developed that afford more direct general purpose computation, such as OpenCL and NVIDIA's CUDA. Hardware accelerated ray tracing APIs also exist to accelerate ray-object intersection (S. G. Parker et al., 2010). Similarly, the standard APIs that are used to program the graphics components of video games, such as OpenGL and DirectX, also allow mechanisms to leverage the graphics hardware's parallel capabilities. Many of these APIs change as new hardware is developed to support more sophisticated computations.

Graphics hardware is programmable. As a developer, you have control over much of the computations associated with processing geometry, vertices, and the fragments that eventually become pixels. Recent hardware changes as well as ongoing updates to the APIs, such as OpenGL or DirectX, support a completely programmable pipeline. These changes afford developers creative license to exploit the computation available on GPUs. Prior to this, fixed-function rasterization pipelines forced the computation to a specific style of vertex transformations, lighting, and fragment processing. The fixed functionality of the pipeline ensured that basic coloring, lighting, and texturing could occur very quickly. Whether it is a programmable interface, or fixed-function computation, the basic computations of the rasterization pipeline are similar, and follow the illustration in Figure 17.1. In the rasterization pipeline, vertices are transformed from local space to global space, and eventually into screen coordinates, after being transformed by the viewing and projection transformation matrices. The set of screen coordinates associated with a geometry's vertices are rasterized into fragments. The final stages of the pipeline process the fragments into pixels and can apply

Fragment: *Fragment* is a term that describes the information associated with a pixel prior to being processed in the final stages of the graphics pipeline. This definition includes much of the data that might be used to calculate the color of the pixel, such as the pixel's scene depth, texture coordinates, or stencil information.

3d对象的描述，并将这些表示转换为填充显示器的彩色像素。

Real-Time Graphics: By real-time graphics, we 一般是指与图形相关的computations正在进行足够快的结果

立即查看。能够以60hz或更高的频率进行操作被认为是实时的。

一旦刷新显示的时间（帧速率）下降到15hz以下，被认为比实时更具交互性，但这种差异并不重要。由于计算需要快速，用于渲染图形的方程通常是

如果有更多的时间，可以做些什么。

在过去的十年里 图形硬件的变化确实非常迅速。较新的图形硬件提供了更多的并行处理能力，以及更好地支持专门的渲染。快节奏的一个解释是视频游戏行业及其经济动力。从本质上讲，这意味着每个新的显卡都提供了更好的性能和处理能力。因此，视频游戏在视觉上显得更加逼真。图形硬件上的处理器（通常称为Gpu或图形处理单元）高度并行，可提供数千个并发执行线程。硬件专为吞吐量而设计，允许在更短的时间内处理更大数量的像素和顶点。所有这些并行性都有利于图形算法，但其他工作也受益于并行硬件。除了视频游戏之外，Gpu还用于加速物理计算，develop实时光线追踪代码，求解流体流动模拟的Navier-Stokes相关方程，并开发更快的代码以了解气候（Purcell, Buck, Mark, & Hanrahan, 2002; S.G.Parkeret, 2010; Harris, 2004）。已经开发了Several API和Sdk，可以提供更直接的通用计算，例如OpenCL和NVIDIA的CUDA。还存在硬件加速光线追踪Api来加速光线-物体交叉(S.G.Parkeretal 2010)。同样，用于对视频游戏的图形组件进行编程的标准Api（如OpenGL和DirectX）也允许机制利用图形硬件的并行功能。许多这些Api随着新硬件的开发而改变，以支持更复杂的计算。

图形硬件是可编程的。作为开发人员，您可以控制与处理几何体、顶点和最终成为像素的片段相关的大部分计算。最近的硬件变化以及

片段：片段是一个术语，描述在图形管道的最后阶段处理之前与像素相关的信息。此定义包括可能用于计算像素颜色的大部分数据，例如像素的场景深度、纹理坐标或模板信息。

对Api（如OpenGL或DirectX）的持续更新支持完全可编程的流水线。这些更改使开发人员提供了利用Gpu上可用计算的创造性许可。在此之前，固定函数光栅化管道将计算强制为特定样式的顶点转换、光照和片段处理。管道的固定功能确保了基本的着色、光照和纹理可以非常快地发生。无论是可编程接口，还是固定函数计算，光栅化流水线的基本计算都是相似的，如下图17.1所示。在光栅化管道中，顶点在被观看和投影变换矩阵变换后，从局部空间变换到全局空间，并最终转换为屏幕坐标。与几何顶点相关的屏幕坐标集被栅格化为片段。流水线的最后阶段将片段处理为像素，并且可以应用

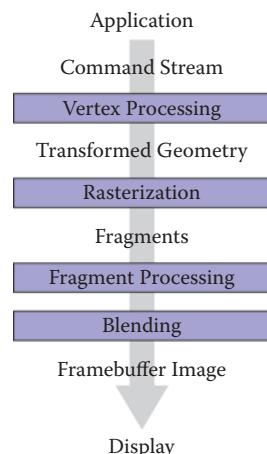


Figure 17.1. The basic graphics hardware pipeline consists of stages that transform 3D data into 2D screen objects ready for rasterizing and coloring by the pixel processing stages.

per-fragment texture lookups, lighting, and any necessary blending. In general, the pipeline lends itself to parallel execution and the GPU cores can be used to process both vertices and fragments concurrently. Additional details about the rasterization pipeline can be found in Chapter 8.

17.3 Heterogeneous Multiprocessing

When using graphics hardware, it is convenient to distinguish between the CPU and the GPU as separate computational entities. In this context, the term *host* is used to refer to the CPU including the threads and memory available to it. The term *device* is used to refer to the GPU, or the graphics processing units, and the threads and memory associated with it. This makes some sense because most graphics hardware is comprised of external hardware that is connected to the machine via the PCI bus. The hardware may also be soldered to the machine as a separate chipset. In this sense, the graphics hardware represents a specialized co-processor since both the CPU (and its cores) can be programmed, as can the GPU and its cores. All programs that utilize graphics hardware must first establish a mapping between the CPU and the GPU memory. This is a rather low-level detail that is necessary so that the graphics hardware driver residing within the operating system can interface between the hardware and the operating system and windowing system software. Recall that because the host (CPU) and the de-

Host: In a graphics hardware program, the host refers to the CPU components of the application.

Device: The GPU side of the graphics application, including the data and computation that are stored and executed on the GPU.



基本图形硬件流水线由将3D数据转换为2D屏幕对象的阶段组成，这些阶段准备由像素处理阶段进行光栅化和着色。

每个片段的纹理查找，照明和任何必要的混合。通常，流水线适合并行执行，GPU内核可用于同时处理顶点和片段。关于光栅化管道的其他细节可以在第8章中找到。

17.3 Heterogeneous Multiprocessing

当使用图形硬件时，在CPU和GPU之间分离为单独的计算实体是很方便的。在此上下文中，术语主机用于指CPU，包括其可用的线程和内存。该

术语设备用于指GPU，或图形处理单元，以及与之相关联的线程和内存。这是有道理的，因为大多数图形硬件由通过PCI总线连接到机器的外部硬件组成。硬件也可以作为单独的芯片组焊接到机器上。从这个意义上说，图形硬件代表了一个专门的

协处理器由于CPU（及其内核）都可以编程，GPU及其内核也是如此。所有使用图形硬件的程序都必须首先在CPU和GPU内存之间建立映射。这是一个相当低级的细节，这是必要的，以便驻留在操作系统内的图形硬件驱动程序可以在硬件和操作系统以及窗口系统软件之间进行接口。回想一下，因为主机（CPU）和de-

主机：在图形硬件中程序，主机指的是应用程序的CPU components。

图形应用程序的GPU端，包括在GPU上存储和执行的数据和computation。

vice (GPU) are separate, data must be communicated between the two systems. More formally, this mapping between the operating system, the hardware driver, the hardware, and the windowing system is known as the graphics *context*. The context is usually established through API calls to the windowing system. Details about establishing a context is outside the scope of this chapter, but many windowing system development libraries have ways to query the graphics hardware for various capabilities and establish the graphics context based on those requirements. Because setting up the context is windowing system dependent, it also means that such code is not likely to be cross-platform code. However, in practice, or at least when starting out, it is very unlikely that such low-level context setup code will be required since many higher level APIs exist to help people develop portable interactive applications.

Many of the frameworks for developing interactive applications support querying input devices such as the keyboard or mouse. Some frameworks provide access to the network, audio system, and other higher level system resources. In this regard, many of these APIs are the preferred way to develop graphics, and even game applications.

Cross-platform hardware acceleration is often achieved with the OpenGL API. OpenGL is an open industry standard graphics API that supports hardware acceleration on many types of graphics hardware. OpenGL represents one of the most common APIs for programming graphics hardware along with APIs such as DirectX. While OpenGL is available on many operating systems and hardware architectures, DirectX is specific to Microsoft-based systems. For the purposes of this chapter, hardware programming concepts and examples will be presented with OpenGL.

17.3.1 Programming with OpenGL

When you program with the OpenGL API, you are writing code for at least two processors: the CPU(s) and the GPU(s). OpenGL is implemented in a C-style API and all functions are prefixed with “gl” to indicate their inclusion with OpenGL. OpenGL function calls change the state of the graphics hardware and can be used to declare and define geometry, load vertex and fragment shaders, and determine how computation will occur as data passes through the hardware.

The variant of OpenGL that this chapter presents is the OpenGL 3.3 Core Profile version. While not the most recent version of OpenGL, the 3.3 version of OpenGL is in line with the future direction of OpenGL programming. These versions are focused on improving efficiency while also fully placing the programming of the pipeline within the hands of the developer. Many of the function

副 (GPU) 是分开的，数据必须在两个系统之间进行通信。更正式地说，操作系统、硬件驱动程序、硬件和窗口系统之间的这种映射称为图形上下文。上下文通常通过对窗口系统的API调用来建立。关于建立上下文的Details超出了本章的范围，但是许多窗口系统开发库都有办法查询图形硬件的各种功能，并根据这些要求建立图形上下文。因为设置上下文依赖于窗口系统，这也意味着此类代码不太可能是跨平台代码。然而，在实践中，或者至少在开始时，不太可能需要这样的低级con文本设置代码，因为存在许多更高级别的Api来帮助人们开发便携式交互应用程序。

许多用于开发交互式应用程序的框架支持查询输入设备，如键盘或鼠标。一些框架为网络、音频系统和其他更高级别的系统资源提供交流。在这方面，许多这些Api是开发图形，甚至游戏应用程序的首选方式。

跨平台硬件加速通常通过OpenGL API实现。

OpenGL是一个开放的行业标准图形API，支持多种类型的图形硬件上的硬件交流。OpenGL表示用于编程图形硬件以及DirectX等Api的最常见Api之一。虽然OpenGL可在许多操作系统和硬件体系结构上使用，但DirectX特定于基于Microsoft的系统。为了本章的目的，硬件编程概念和示例将与OpenGL一起呈现。

17.3.1 使用OpenGL编程

使用OpenGL API编程时，您至少要为两个处理器编写代码：CPU和GPU。OpenGL是在一个C风格的API中实现的，所有函数都以“gl”作为前缀，以表示它们包含在OpenGL中。OpenGL函数调用可更改图形硬件的状态，可用于声明和定义几何体、加载顶点和片段着色器，以及确定数据通过硬件时的计算方式。

本章介绍的OpenGL变体是OpenGL3.3核心配置文件版本。虽然不是最新版本的OpenGL，但3.3版本的OpenGL符合OpenGL编程的未来方向。这些版本的重点是提高效率，同时也将流水线的专业语法完全放在开发人员手中。许多功能



calls present in earlier versions of OpenGL are not present in these newer APIs. For instance, *immediate mode* rendering is deprecated. Immediate mode rendering was used to send data from the CPU memory to the graphics card memory as needed each frame and was often very inefficient, especially for larger models and complex scenes. The current API focuses on storing data on the graphics card before it is needed and instancing it at render time. As another example, OpenGL's matrix stacks have been deprecated as well, leaving the developer to use third-party matrix libraries (such as GLM) or their own classes to create the necessary matrices for viewing, projection, and transformation, as presented in Chapter 7. As a result, OpenGL's shader language (GLSL) has taken on larger roles as well, performing the necessary matrix transformations along with lighting and shading within the shaders. Because the fixed-function pipeline which performed per-vertex transformation and lighting is no longer present, programmers must develop all shaders themselves. The shading examples presented in this chapter will utilize the GLSL 3.3 Core Profile version shader specification. Future readers of this chapter will want to explore the current OpenGL and OpenGL Shading Language specifications for additional details on what these APIs and languages can support.

17.4 Graphics Hardware Programming: Buffers, State, and Shaders

Three concepts will help to understand contemporary graphics hardware programming. The first is the notion of a *data buffer*, which is quite simply, a linear allocation of memory on the device that can store various data on which the GPUs will operate. The second is the idea that the graphics card maintains a computational *state* that determines how computations associated with scene data and shaders will occur on the graphics hardware. Moreover, state can be communicated from the host to the device and even within the device between shaders. *Shaders* represent the mechanism by which computation occurs on the GPU related to per-vertex or per-fragment processing. This chapter will focus on vertex and fragment shaders, but specialized geometry and compute shaders also exist in the current versions of OpenGL. Shaders play a very important role in how modern graphics hardware functions.

17.4.1 Buffers

Buffers are the primary structure to store data on graphics hardware. They represent the graphics hardware's internal memory associated with everything from



在早期版本的OpenGL中存在的调用在这些较新的API中不存在。例如，不赞成成立即模式呈现。即时模式渲染用于根据每帧需要将数据从CPU内存发送到显卡内存，并且通常效率非常低，特别是对于较大的模型和复杂的场景。当前的API侧重于在需要之前将数据存储在图形卡上，并在渲染时对其进行实例化。再举一个例子，OpenGL的矩阵堆栈也被弃用了，开发人员只能使用第三方矩阵库（如GLM）或他们自己的类来创建必要的矩阵来查看、投影和转换，如第7章所述。因此，OpenGL的着色器语言（GLSL）也承担了更大的角色，在着色器中执行必要的矩阵转换以及照明和着色。由于执行每个顶点变换和光照的固定函数管道不再存在，程序员必须自己开发所有着色器。本章介绍的着色示例将使用GLSL 3.3 Core Profile版本着色器规范。本章的未来读者将希望探索当前的OpenGL和OpenGL着色语言规范，以获取有关这些API和语言可以支持哪些内容的更多详细信息。

17.4 图形硬件编程：缓冲区 状态和着色器

三个概念将有助于理解当代图形硬件编程。第一个是数据缓冲区的概念，它非常简单，是设备上的线性内存分配，可以存储GPU将在其上运行的各种数据。第二个想法是显卡保持计算状态，该状态决定了与场景数据和着色器相关的计算将如何在图形硬件上发生。而且，状态可以在着色器之间从主机传达到设备，甚至在设备内。着色器表示在GPU上发生与每顶点或每片段处理相关的计算的机制。本章将重点介绍顶点和片段着色器，但在当前版本的OpenGL中也存在专门的几何和计算着色器。着色器在现代图形硬件的功能中起着非常重要的作用。

17.4.1 Buffers

缓冲区是在图形硬件上存储数据的主要结构。它们代表图形硬件的内部存储器，与以下所有内容相关联

geometry, textures, and image plane data. With regard to the rasterization pipeline described in Chapter 8, the computations associated with hardware-accelerated rasterization read and write the various buffers on the GPU. From a programming standpoint, an application must initialize the buffers on the GPU that are needed for the application. This amounts to a host to device copy operation. At the end of various stages of execution, device to host copies can be performed as well to pull data from the GPU to the CPU memory. Additionally, mechanisms do exist in OpenGL's API that allow device memory to be mapped into host memory so that an application program can write directly to the buffers on the graphics card.

17.4.2 Display Buffer

In the graphics pipeline, the final set of pixel colors can be linked to the display, or they may be written to disk as a PNG image. The data associated with these pixels is generally a 2D array of color values. The data is inherently 2D, but it is efficiently represented on the GPU as a 1D linear array of memory. This array implements the *display buffer*, which eventually gets mapped to the window. Rendering images involves communicating the changes to the display buffer on the graphics hardware through the graphics API. At the end of the rasterization pipeline, the fragment processing and blending stages write data to the output display buffer memory. Meanwhile, the windowing system reads the contents of the display buffer to produce the raster images on the monitor's window.

17.4.3 Cycle of Refresh

Most applications prefer a double-buffered display state. What this means is that there are two buffers associated with a graphics window: the front buffer and the back buffer. The purpose of the double-buffered system is that the application can communicate changes to the back buffer (and thus, write changes to that buffer) while the front-buffer memory is used to drive the pixel colors on the window.

At the end of the rendering loop, the buffers are swapped through a pointer exchange. The front-buffer pointer points to the back buffer and the back-buffer pointer is then assigned to the previous front buffer. In this way, the windowing system will refresh the content of the window with the most up-to-date buffer. If the buffer pointer swap is synchronized with the windowing system's refresh of the entire display, the rendering will appear seamless. Otherwise, users may observe a tearing of the geometry on the actual display as changes to the scene's geometry and fragments are processed (and thus written to the display buffer) faster than the screen is refreshed.

几何、纹理和图像平面数据。关于第8章中描述的光栅化管道，与硬件加速光栅化相关的计算读取和写入GPU上的各种缓冲区。从编程的角度来看，应用程序必须初始化GPU上应用程序所需的缓冲区。这相当于主机到设备的复制操作。在执行的各个阶段结束时，也可以执行设备到主机的副本，以将数据从GPU拉到CPU内存。此外，OpenGL的API中确实存在一些机制，允许将设备内存映射到主机内存中，以便应用程序可以直接写入图形卡上的缓冲区。

17.4.2 显示缓冲区

在图形管道中，最终的一组像素颜色可以链接到显示器，或者它们可以作为PNG图像写入磁盘。与这些像素相关联的数据一般是颜色值的2D阵列。数据本质上是2D的，但它在GPU上有效地表示为1D线性内存阵列。此数组实现显示缓冲区，最终映射到窗口。渲染图像涉及通过图形API将更改传达到图形硬件上的显示缓冲区。在光栅化流水线的末端，片段处理和混合阶段将数据写入输出显示缓冲存储器。同时，窗口化系统读取显示缓冲区的内容，以产生监视器窗口上的光栅图像。

17.4.3 刷新周期

大多数应用程序更喜欢双缓冲显示状态。这意味着有两个缓冲区与图形窗口相关联：前缓冲区和后缓冲区。双缓冲系统的目的是应用程序可以将更改传达给后缓冲区（从而将更改写入该缓冲区），而前缓冲区内存用于驱动窗口上的像素颜色。

在渲染循环结束时，缓冲区通过指针交换进行交换。前缓冲区指针指向后缓冲区，然后将后缓冲区指针分配给前一个前缓冲区。这样，窗口系统将使用最新的缓冲区刷新窗口的内容。如果缓冲区指针交换与窗口系统对整个显示器的刷新同步，则呈现将呈现无缝。否则，用户可能会在实际显示器上观察到几何图形的撕裂，因为场景几何图形和片段的更改被处理（从而写入显示缓冲区）比屏幕刷新更快。



When the display is considered a memory buffer, one of the simplest operations on the display is essentially a memory setting (or copying) operation that zeros-out, or clears the memory to a default state. For a graphics program, this likely means clearing the background of the window to a specific color. To clear the background color (to black) in an OpenGL application, the following code can be used:

```
glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
glClear( GL_COLOR_BUFFER_BIT );
```

The first three arguments for the `glClearColor` function represent the *red*, *green*, and *blue* color components, specified within the range [0, 1]. The fourth argument represents opacity, or *alpha* value, ranging from 0.0 being completely transparent to 1.0 being completely opaque. The *alpha* value is used to determine transparency through various fragment blending operations in the final stages of the pipeline.

This operation only clears the color buffer. In addition to the color buffer, specified by `GL_COLOR_BUFFER_BIT`, being cleared to black in this case, graphics hardware also uses a depth buffer to represent the distance that fragments are relative to the camera (you may recall the discussion of the z-buffer algorithm in Chapter 8). Clearing the depth buffer is necessary to ensure operation of the z-buffer algorithm and allow correct hidden surface removal to occur. Clearing the depth buffer can be achieved by *or*'ing two bit field values together, as follows:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Within a basic interactive graphics application, this step of clearing is normally the first operation performed before any geometry or fragments are processed.

17.5 State Machine

By illustrating the buffer-clearing operation for the display's color and depth buffers, the idea of graphics hardware *state* is also introduced. The `glClearColor` function sets the default color values that are written to all the pixels within the color buffer when `glClear` is called. The clear call initializes the color component of the display buffer and can also reset the values of the depth buffer. If the clear color does not change within an application, the clear color need only be set once, and often this is done in the initialization of an OpenGL program. Each time that `glClear` is called it uses the previously set state of the clear color.



当显示器被认为是内存缓冲区时，显示器上最简单的操作之一本质上是内存设置（或复制）操作，该操作将内存清零或清除到默认状态。对于图形程序，这可能意味着将窗口的背景清除为特定颜色。要在OpenGL应用程序中清除背景颜色（到黑色），可以使用以下代码：

```
glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
glClear( GL_COLOR_BUFFER_BIT );
```

`GlClearColor`函数的前三个参数表示在[0 1]范围内指定的红色、绿色和蓝色分量。第四个参数表示不透明度或alpha值，范围为0。0对1完全透明。0完全不透明。Alpha值用于通过管道最后阶段的各种片段混合操作来确定透明度。

此操作仅清除颜色缓冲区。除了由GL颜色缓冲位指定的颜色缓冲区在这种情况下被清除为黑色外，图形ics硬件还使用深度缓冲区来表示片段相对于相机的距离（您可能还记得第8章中对z缓冲算法的讨论）。清除深度缓冲区是必要的，以确保z缓冲算法的操作，并允许正确的隐藏表面去除发生。清除深度缓冲器可以通过或'ing两个位域值一起来实现，如下：

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

在基本的交互式图形应用程序中，清除此步骤通常是在处理任何几何体或片段之前执行的第一个操作。

17.5 状态机

通过说明显示器的颜色和深度缓冲区的缓冲区清除操作
`GlClearColor`函数设置调用`glClear`时写入颜色缓冲区内所有像素的默认颜色值。清除调用`ini`对显示缓冲区的颜色分量进行排序，还可以重置深度缓冲区的值。如果清除颜色在应用程序中没有变化，则清除颜色只需要设置一次，通常这是在OpenGL程序的初始化中完成的。每次调用`glClear`时，它都会使用先前设置的清晰颜色状态。

Note also that the z-buffer algorithm state can be enabled and disabled as needed. The z-buffer algorithm is also known in OpenGL as the depth test. By enabling it, a fragment's depth value will be compared to the depth value currently stored in the depth buffer prior to writing any fragment colors to the color buffer. Sometimes, the depth test is not necessary and could potentially slow down an application. Disabling the depth test will prevent the z-buffer computation and change the behavior of the executable. Enabling the z-buffer test with OpenGL is done as follows:

```
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS);
```

The glEnable call turns on the depth test while the glDepthFunc call sets the mechanism for how the depth comparison is performed. In this case, the depth function is set to its default value of GL_LESS to show that other state variables exist and can be modified. The converse of the glEnable calls are glDisable calls.

The idea of state in OpenGL mimics the use of static variables in object-oriented classes. As needed, programmers enable, disable, and/or set the state of OpenGL variables that reside on the graphics card. These state then affect any succeeding computations on the hardware. In general, efficient OpenGL programs attempt to minimize state changes, enabling states that are needed, while disabling states that are not required for rendering.

17.6 Basic OpenGL Application Layout

A simple and basic OpenGL application has, at its heart, a display loop that is called either as fast as possible, or at a rate that coincides with the refresh rate of the monitor or display device. The example loop below uses the GLFW library, which supports OpenGL coding across multiple platforms.

```
while (!glfwWindowShouldClose(window)) {
{
    // OpenGL code is called here,
    // each time this loop is executed.
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Swap front and back buffers
    glfwSwapBuffers(window);

    // Poll for events
    glfwPollEvents();
}
```

另请注意，z-buffer算法状态可以根据需要启用和禁用。Z-buffer算法在OpenGL中也被称为深度测试。通过启用它，在将任何片段颜色写入颜色缓冲区之前，片段的深度值将与当前存储在深度缓冲区中的深度值进行比较。有时，深度测试不是必要的，可能会减慢应用程序的速度。禁用深度测试将阻止z缓冲区计算并更改可执行文件的行为。使用OpenGL启用z-buffer测试如下所示：

```
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS);
```

GLEnable调用打开深度测试，而glDepthFunc调用设置执行深度比较的机制。在这种情况下，深度函数被设置为其默认值GLLESS，以表明其他状态变量存在并且可以修改。GLEnable调用的相反是glDisable调用。

OpenGL中的状态思想模仿了面向对象类中静态变量的使用。根据需要，程序员启用，禁用和/或设置驻留在显卡上的OpenGL变量的状态。然后，这些状态会影响硬件上的任何后续计算。通常，高效的OpenGL程序会尝试最大限度地减少状态更改，启用需要的状态，同时禁用渲染不需要的状态。

17.6 基本OpenGL应用程序布局

一个简单而基本的OpenGL应用程序的核心是一个显示循环，该循环被称为尽可能快，或者以与监视器或显示设备的刷新率一致的速率。下面的示例循环使用GLFW库，该库支持跨多个平台的OpenGL编码。

```
while (!glfwWindowShouldClose(window)) {
{
    // 这里调用OpenGL代码
    // 每次执行此循环。
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // 交换前后缓冲区glfwSwapBuffers(窗口)；

    // 事件投票glfwPollEvents();
};
```

```
if (glfwGetKey( window, GLFW_KEY_ESCAPE ) == GLFW_PRESS)
    glfwSetWindowShouldClose(window, 1);
}
```

The loop is tightly constrained to operate only while the window is open. This example loop resets the color buffer values and also resets the z-buffer depth values in the graphics hardware memory based on previously set (or default) values. Input devices, such as keyboards, mouse, network, or some other interaction mechanism are processed at the end of the loop to change the state of data structures associated with the program. The call to `glfwSwapBuffers` synchronizes the graphics context with the display refresh, performing the pointer swap between the front and back buffers so that the updated graphics state is displayed on the user's screen. The call to swap the buffers occurs after all graphics calls have been issued.

While conceptually separate, the depth and color buffers are often collectively called the *framebuffer*. By clearing the contents of the framebuffer, the application can proceed with additional OpenGL calls to push geometry and fragments through the graphics pipeline. The framebuffer is directly related to the size of the window that has been opened to contain the graphics context. The window, or *viewport*, dimensions are needed by OpenGL to construct the M_{vp} matrix (from Chapter 7) within the hardware. This is accomplished through the following code, demonstrated again with the GLFW toolkit, which provides functions for querying the requested window (or framebuffer) dimensions:

```
int nx, ny;
glfwGetFramebufferSize(window, &nx, &ny);
glViewport(0, 0, nx, ny);
```

In this example, `glViewport` sets the OpenGL state for the window dimension using `nx` and `ny` for the width and height of the window and the *viewport* being specified to start at the origin.

Technically, OpenGL writes to the framebuffer memory as a result of operations that rasterize geometry, and process fragments. These writes happen before the pixels are displayed on the user's monitor.

17.7 Geometry

Similar to the idea of a display buffer, geometry is also specified using arrays to store vertex data and other vertex attributes, such as vertex colors, normals, or texture coordinates needed for shading. The concept of buffers will be used to

```
如果(glfwGetKey(window GLFW_KEY_ESCAPE)==GLFW_PRESS)
    glfwSetWindowShouldClose(window, 1);
}
```

循环被严格限制为仅在窗口打开时操作。

此示例循环重置颜色缓冲区值，并且还基于先前设置的（或默认）`value`重置图形硬件存储器中的z缓冲区深度值。输入设备，例如键盘、鼠标、网络或一些其他交互机制在循环结束时被处理，以改变与程序相关联的数据结构的状态。对`glfwSwapBuffers`的调用使图形上下文与显示刷新同步，在前后缓冲区之间执行指针交换，以便在用户的屏幕上显示更新的图形状态。交换缓冲区的调用发生在发出所有图形调用之后。

虽然在概念上是分开的，但深度缓冲区和颜色缓冲区通常统称为帧缓冲区。通过清除帧缓冲区的内容，应用程序可以继续进行额外的OpenGL调用，通过图形管道推送几何体和片段。帧缓冲区与已打开以包含图形上下文的窗口的大小直接相关。窗口或视口尺寸是OpenGL在硬件中构建mvp矩阵（从第7章开始）所需要的。这是通过以下代码实现的，GLFW工具包再次演示，它提供了查询所请求的窗口（或帧缓冲区）维度的函数：

```
int nx, ny;
glfwGetFramebufferSize(window, &nx, &ny);
glViewport(0, 0, nx, ny);
```

在此示例中，`glViewport`使用`nx`和`ny`为窗口的宽度和高度以及指定从原点开始的视口设置窗口维度的OpenGL状态。

从技术上讲，OpenGL写入帧缓冲内存是由于operations对几何进行光栅化和处理片段。这些写入发生在像素显示在用户的显示器上之前。

17.7 Geometry

与显示缓冲区的想法类似，几何也是使用数组来指定的，以存储顶点数据和其他顶点属性，如顶点颜色、法线或着色所需的纹理坐标。缓冲区的概念将用于

allocate storage on the graphics hardware, transferring data from the host to the device.

17.7.1 Describing Geometry for the Hardware

Primitives: The three primitives (points, lines, triangles, and quads) are really the only primitives available! Even when creating spline-based surfaces, such as NURBS, the surfaces are tessellated into triangle primitives by the graphics hardware.

Point Rendering: Point and line primitives may initially appear to be limited in use, but researchers have used points to render very complex geometry (Rusinkiewicz & Levoy, 2000; Dachsbacher, Vogelsgang, & Stamminger, 2003).

One of the challenges with graphics hardware programming is the management of the 3D data and its transfer to and from the memory of the graphics hardware. Most graphics hardware work with specific sets of geometric primitives. The different primitive types leverage primitive complexity for processing speed on the graphics hardware. Simpler primitives can sometimes be processed very fast. The caveat is that the primitive types need to be general purpose so as to model a wide range of geometry from very simple to very complex. On typical graphics hardware, the primitive types are limited to one or more of the following:

- **points**—single vertices used to represent points or particle systems;
- **lines**—pairs of vertices used to represent lines, silhouettes, or edge-highlighting;
- **triangles**—triangles, triangle strips, indexed triangles, indexed triangle strips, quadrilaterals, or triangle meshes approximating geometric surfaces.

These three primitive types form the basic building blocks for most geometry that can be defined. An example of a triangle mesh rendered with OpenGL is shown in Figure 17.2.

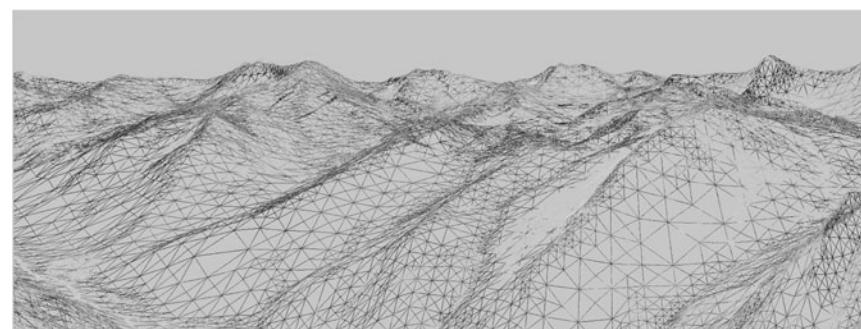


Figure 17.2. How your geometry is organized will affect the performance of your application. This wireframe depiction of the Little Cottonwood Canyon terrain dataset shows tens of thousands of triangles organized as a triangle mesh running at real-time rates. *The image is rendered using the VTerrain Project terrain system courtesy of Ben Discue.*

在图形硬件上分配存储，将数据从主机传输到设备。

17.7.1 描述硬件的几何

Primitives: (点, 线, 三角形和四边形) 真的是唯一可用的基元! 即使在创建基于样条的surface (如NURBS) 时, 图形硬件也会将曲面细分为三角形基元。

点和线基元可能最初但重新搜索者已经使用点 geometry (Rusinkiewicz & Levoy, 2000; Dachsbacher, Vogelsgang, & Stamminger, 2003).

图形硬件编程面临的挑战之一是3d数据的管理及其与图形硬件内存之间的传输。大多数图形硬件使用特定的几何图元集。该

不同的基元类型利用基元复杂性来提高图形硬件的处理速度。更简单的基元有时可以非常快地处理。需要注意的是, 基本类型需要是通用的, 以便对从非常简单到非常复杂的各种几何进行建模。在典型的图形硬件上, 基元类型仅限于以下一种或多种:

- ***点**—用于表示点或粒子系统的单个顶点;
- ***线条**—用于表示线条、轮廓或边缘照明的顶点对;
- ***三角形**—三角形、三角形条带、索引三角形、索引三角形条带、四边形或近似几何曲面的三角形网格。

这三种基本类型构成了可以定义的大多数几何的基本构建块。使用OpenGL渲染的三角形网格示例如图17.2所示。

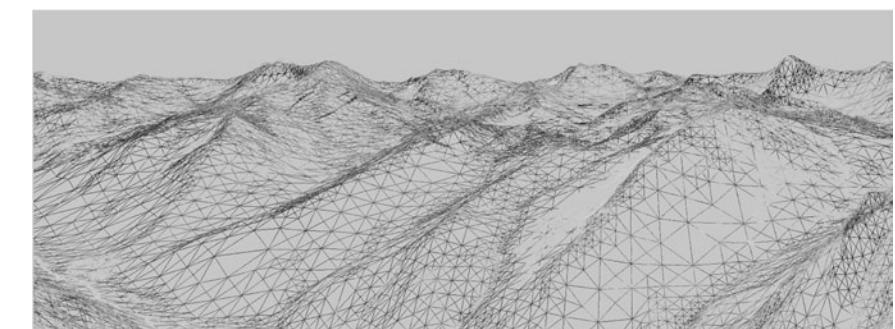


Figure 17.2. 几何体的组织方式将影响应用程序的性能。此线框描绘了LittleCottonwoodCanyonterrain数据集, 显示了数万个三角形, 这些三角形组织为以实时速率运行的三角形网格。图像使用BenDisco提供的VTerrainProjectterrain系统渲染。



17.8 A First Look at Shaders

Modern versions of OpenGL require that shaders be used to process vertices and fragments. As such, no primitives can be rendered without at least one vertex shader to process the incoming primitive vertices and another shader to process the rasterized fragments. Advanced shader types exist within OpenGL and the OpenGL Shading Language: *geometry shaders* and *compute shaders*. Geometry shaders are designed to process primitives, potentially creating additional primitives, and can support geometric instancing operations. Compute shaders are designed for performing general computation on the GPU, and can be linked into the set of shaders necessary for a specific application. For more information on geometry and compute shaders, the reader is referred the OpenGL specification documents and other resources.

17.8.1 Vertex Shader Example

Vertex shaders provide control over how vertices are transformed and often help prepare data for use in fragment shaders. In addition to standard transformations and potential per-vertex lighting operations, vertex shaders could be used to perform general computation on the GPU. For instance, if the vertices represent particles and the particle motion can be (simply) modeled within the vertex shader computations, the CPU can mostly be removed from performing those computations. The ability to perform computations on the vertices already stored in the graphics hardware memory is a potential performance gain. While this approach is useful in some situations, advanced general computation may be more appropriately coded with compute shaders.

In Chapter 7, the viewport matrix M_{vp} was introduced. It transforms the canonical view volume coordinates to screen coordinates. Within the canonical view volume, coordinates exist in the range of $[-1, 1]$. Anything outside of this range is clipped. If we make an initial assumption that the geometry exists within this range and the z-value is ignored, we can create a very simple vertex shader. This vertex shader passes the vertex positions through to the rasterization stage, where the final viewport transformation will occur. Note that because of this simplification, there are no projection, viewing, or model transforms that will be applied to the incoming vertices. This is initially cumbersome for creating anything except very simple scenes, but will help introduce the concepts of shaders and allow you to render an initial triangle to the screen. The *passthrough* vertex shader follows:



17.8 初看着色器

OpenGL的现代版本要求使用着色器来处理顶点和片段。因此，在没有至少一个顶点着色器来处理传入的基元顶点和另一个着色器来处理光栅化片段的情况下，不能呈现任何基元。OpenGL和OpenGL着色语言中存在高级着色器类型：几何着色器和计算着色器。几何着色器旨在处理图元，可能会创建其他图元，并且可以支持几何实例化操作。计算着色器设计用于在GPU上执行一般计算，并且可以链接到特定应用程序所需的着色器集。有关几何和计算着色器的更多信息，请参阅OpenGL规范文档和其他资源。

17.8.1 顶点着色器示例

顶点着色器提供对顶点转换方式的控制，并且通常有助于准备数据以用于片段着色器。除了标准的变换和潜在的每顶点照明操作外，顶点着色器还可用于在GPU上执行一般计算。例如，如果顶点表示粒子，并且粒子运动可以（简单地）在顶点着色器计算中建模，那么CPU基本上可以从执行这些计算中删除。对已经存储在图形硬件存储器中的顶点执行计算的能力是潜在的性能增益。虽然这种方法在某些情况下很有用，但高级通用计算可能会使用计算着色器进行更恰当的编码。

在第7章中，介绍了视口矩阵mvp。它将规范视图体积坐标转换为屏幕坐标。在规范视图体积内，坐标存在于 $[-1, 1]$ 的范围内。任何超出这个范围的东西都被剪掉了。如果我们做一个初始假设，即几何存在于这个范围内，并且z值被忽略，我们可以创建一个非常简单的顶点着色器。此顶点着色器将顶点位置传递到光栅化阶段，在那里将发生最终的视口转换。请注意，由于这种简化，没有将应用于传入顶点的投影、查看或模型变换。这对于创建除了非常简单的场景之外的任何东西来说最初都很麻烦，但将有助于引入着色器的概念，并允许您将初始三角形渲染到屏幕上。直通顶点着色器如下：

```
#version 330 core

layout(location=0) in vec3 in_Position;
void main(void)
{
    gl_Position = vec4(in_Position, 1.0);
}
```

This vertex shader does only one thing. It passes the incoming vertex position out as the `gl_Position` that OpenGL uses to rasterize fragments. Note that `gl_Position` is a built-in, reserved variable that signifies one of the key outputs required from a vertex shader. Also note the `version` string in the first line. In this case, the string instructs the GLSL compiler that version 3.3 of the GLSL Core profile is to be used to compile the shading language.

Vertex and fragment shaders are SIMD operations that respectively operate on all the vertices or fragments being processed in the pipeline. Additional data can be communicated from the host to the shaders executing on the device by using input, output, or uniform variables. Data that is passed into a shader is prefixed with the keyword `in`. The location of that data as it relates to specific vertex attributes or fragment output indices is also specified directly in the shader. Thus,

```
layout(location=0) in vec3 in_Position;
```

specifies that `in_Position` is an input variable that is of type `vec3`. The source of that data is the attribute index 0 that is associated with the geometry. The name of this variable is determined by the programmer, and the link between the incoming geometry and the shader occurs while setting up the vertex data on the device. The GLSL contains a nice variety of types useful to graphics programs, including `vec2`, `vec3`, `vec4`, `mat2`, `mat3`, and `mat4` to name a few. Standard types such as `int` or `float` also exist. In shader programming, vectors, such as `vec4` hold 4-components corresponding to the `x`, `y`, `z`, and `w` components of a homogeneous coordinate, or the `r`, `g`, `b`, and `a` components of a RGBA tuple. The labels for the types can be interchanged as needed (and even repeated) in what is called *swizzling* (e.g., `in_Position.zyxa`). Moreover, the component-wise labels are overloaded and can be used appropriately to provide context.

All shaders must have a `main` function that performs the primary computation across all inputs. In this example, the `main` function simply copies the input vertex position (`in_Position`), which is of type `vec3` into the built-in vertex shader output variable, which is of type `vec4`. Note that many of the built-in types have constructors that are useful for conversions such as the one presented here to convert the incoming vertex position's `vec3` type into `gl_Position`'s `vec4`

```
#version 330 core

布局 (位置=0) 在 vec3 in_Position; void main (void)
{
    gl_Position = vec4 (in_Position, 1.0);
}
```

这个顶点着色器只做一件事。它将传入的顶点位置作为OpenGL用于栅格化片段的gl位置传递出去。请注意，`glPosition`是一个内置的保留变量，表示顶点着色器所需的关键输出之一。还要注意第一行中的版本字符串。在这种情况下，该字符串指示GLSL编译器将使用GLSL核心配置文件的3.3版本来编译着色语言。

顶点着色器和片段着色器是SIMD操作，它们分别对管道中正在处理的所有顶点或片段进行操作。可以使用输入、输出或统一变量将其他数据从主机传输到在设备上执行的着色器。传入着色器的数据以关键字`in`作为前缀。该数据与特定顶点属性或片段输出索引相关的位置也直接在着色器中指定。因此

```
layout(location=0) in vec3 in_Position;
```

`指定InPosition`是类型为`vec3`的输入变量。该数据的来源是与几何关联的属性索引0。此变量的名称由程序员确定，传入几何体和着色器之间的链接发生在设备上设置顶点数据时。GLSL包含了很多对图形程序有用的类型，包括`vec2`、`vec3`、`vec4`、`mat2`、`mat3`和`mat4`等等。也存在`int`或`float`等标准类型。在着色器编程中，矢量（如`vec4`）包含与齐次坐标的`x`、`y`、`z`和`w`分量相对应的4分量，或RGBA元组的`r`、`g`、`b`和`a`分量。类型的标签可以根据需要互换（甚至重复），在所谓的`swizzling`（例如，在位置）中。`zyxa`）。此外，组件标签是重载的，可以适当地用于提供上下文。

所有着色器都必须有一个主函数，该函数在所有输入中执行主计算。在此示例中，`main`函数只需将类型为`vec3`的输入顶点位置（`inPosition`）复制到类型为`vec4`的内置顶点着色器输出变量中。请注意，许多内置类型都具有可用于转换的构造函数，例如此处介绍的将传入顶点位置的`vec3`类型转换为`glPosition`的`vec4`的构造函数



type. Homogeneous coordinates are used with OpenGL, so 1.0 is specified as the fourth coordinate to indicate that the vector is a position.

17.8.2 Fragment Shader Example

If the simplest vertex shader simply passes clip coordinates through, the simplest fragment shader sets the color of the fragment to a constant value.

```
#version 330 core
layout(location=0) out vec4 out_FragmentColor;
void main(void)
{
    out_FragmentColor = vec4(0.49, 0.87, 0.59, 1.0);
}
```

In this example, all fragments will be set to a light shade of green. One key difference is the use of the `out` keyword. In general, the keywords `in` and `out` in shader programs indicate the flow of data into, and out of, shaders. While the vertex shader received incoming vertices and output them to a built-in variable, the fragment shader declares its outgoing value which is written out to the color buffer:

```
layout(location=0) out vec4 out_FragmentColor;
```

The output variable `out_FragmentColor` is again user defined. The location of the output is color buffer index 0. Fragment shaders can output to multiple buffers, but this is an advanced topic left to the reader that will be needed if OpenGL's framebuffer objects are investigated. The use of the `layout` and `location` keywords makes an explicit connection between the application's geometric data in the vertex shader and the output color buffers in the fragment shader.

17.8.3 Loading, Compiling, and Using Shaders

Shader programs are transferred onto the graphics hardware in the form of character strings. They must then be compiled and linked. Furthermore, shaders are coupled together into shader programs so that vertex and fragment processing occur in a consistent manner. A developer can activate a shader that has been successfully compiled and linked into a shader program as needed, while also deactivating shaders when not required. While the detailed process of creating,



类型。齐次坐标与OpenGL一起使用，所以1.0被指定为第四坐标，以指示向量是一个位置。

17.8.2 片段着色器示例

如果最简单的顶点着色器只是简单地传递剪辑坐标，则最简单的片段着色器将片段的颜色设置为常量值。

```
#version 330 core
layout(location=0) out vec4 out_FragmentColor;
void main(void){
    out_FragmentColor = vec4(0.49, 0.87, 0.59, 1.0);
}
```

在此示例中，所有片段都将设置为浅绿色。一个关键的区别是使用`out`关键字。通常，着色器程序中的关键字`in`和`out`指示数据流入和流出着色器。当顶点着色器接收传入顶点并将其输出到内置变量时，片段着色器声明其输出值，并将其写入颜色缓冲区：

```
layout(location=0) out vec4 out_FragmentColor;
```

输出变量`outFragmentColor`再次是用户定义的。输出的位置是颜色缓冲区索引0。片段着色器可以输出到多个缓冲区，但如果研究OpenGL的帧缓冲对象，这是一个留给读者的高级主题。`Layout`和`location`关键字的使用在顶点着色器中的应用程序几何数据和片段着色器中的输出颜色缓冲区之间建立了显式连接。

17.8.3 加载、编译和使用着色器

着色器程序以字符串的形式传输到图形硬件上。然后必须对它们进行编译和链接。此外，着色器被耦合到着色器程序中，以便顶点和片段处理以一致的方式发生。开发人员可以根据需要激活已成功编译并链接到着色器程序的着色器，同时在不需要时停用着色器。而创建的详细过程

loading, compiling, and linking shader programs is not provided in this chapter, the following OpenGL functions will be helpful in creating shaders:

- `glCreateShader` creates a handle to a shader on the hardware.
- `glShaderSource` loads the character strings into the graphics hardware memory.
- `glCompileShader` performs the actual compilation of the shader within the hardware.

The functions above need to be called for each shader. So, for the simple pass-through shaders, each of those functions would be called for both the vertex shader code and the fragment shader code provided. At the end of the compilation phase, compilation status and any errors can be queried using additional OpenGL commands.

After both shader codes are loaded and compiled, they can be linked into a shader program. The shader program is what is used to affect rendering of geometry.

- `glCreateProgram` creates a program object that will contain the previously compiled shaders.
- `glAttachShader` attaches a shader to the shader program object. In the simple example, this function will be called for both the compiled vertex shader and the compiled fragment shader objects.
- `glLinkProgram` links the shaders internally after all shaders have been attached to the program object.
- `glUseProgram` binds the shader program for use on the graphics hardware. As shaders are needed, the program handles are bound using this function. When no shaders are needed, they can be unbound by using the shader program handle 0 as an argument to this function.

17.9 Vertex Buffer Objects

Vertices are stored on the graphics hardware using buffers, known as *vertex buffer objects*. In addition to vertices, any additional *vertex attributes*, such as colors, normal vectors, or texture coordinates, will also be specified using vertex buffer objects.

本章不提供加载、编译和链接着色器程序，以下OpenGL函数将有助于创建着色器：

*`glCreateShader`在硬件上创建着色器的句柄。

*`glShaderSource`将字符串加载到图形硬件内存中。

*`glCompileShader`执行硬件内着色器的实际编译。

上面的函数需要为每个着色器调用。因此，对于简单的通过着色器，将为顶点着色器代码和提供的片段着色器代码调用这些函数中的每一个。在编译阶段结束时，可以使用其他OpenGL命令查询编译状态和任何错误。

加载和编译两个着色器代码后，它们可以链接到着色器程序中。着色器程序是用来影响几何体渲染的程序。

*`glCreateProgram`创建一个程序对象，该对象将包含先前编译的着色器。

*`glAttachShader`将着色器附加到着色器程序对象。在简单的示例中，将为编译的顶点着色器和编译的片段着色器对象调用此函数。

*`glLinkProgram`在将所有着色器附加到程序对象后在内部链接着色器。

*`glUseProgram`绑定着色器程序，以便在图形硬件上使用。由于需要着色器，程序句柄使用此函数绑定。当不需要着色器时，可以通过使用着色器程序句柄0为此函数的参数来解除它们的绑定。

17.9 顶点缓冲区对象

顶点使用缓冲区（称为顶点缓冲区对象）存储在图形硬件上。除了顶点之外，还将使用顶点缓冲区对象指定任何其他顶点属性，例如颜色、法向量或纹理坐标。

First, let's focus on specifying the geometric primitive themselves. This starts by allocating the vertices associated with the primitive within the host memory of the application. The most general way to do this is to define an array on the host to contain the vertices needed for the primitive. For instance, a single triangle, fully contained within the canonical volume, could be defined statically on the host as follows:

```
GLfloat vertices[] = {-0.5f, -0.5f, 0.0f, 0.5f, -0.5f, 0.0f, 0.0f, 0.5f, 0.0f};
```

If the simple passthrough shaders are used for this triangle, then all vertices will be rendered. Although the triangle is placed on the $z = 0$ plane, the z coordinates for this example do not really matter since they are essentially dropped in the final transformation into screen coordinates. Another thing to note is the use of the type `GLfloat` in these examples. Just as the GLSL language has specialized types, OpenGL has related type which generally can intermix well with the standard types (like `float`). For precision, the OpenGL types will be used when necessary.

Before the vertices can be processed, a vertex buffer is first created on the device to store the vertices. The vertices on the host are then transferred to the device. After this, the vertex buffer can be referenced as needed to draw the array of vertices stored in the buffer. Moreover, after the initial transfer of vertex data, no additional copying of data across the host to device bus need occur, especially if the geometry remains static across rendering loop updates. Any host memory can also be deleted if it was dynamically allocated.

Vertex buffer objects, often called VBOs, represent the primary mechanism with modern OpenGL to store vertex and vertex attributes in the graphics memory. For efficiency purposes, the initial setup of a VBO and the transfer of vertex-related data mostly happens prior to entering the display loop. As an example, to create a VBO for this triangle, the following code could be used:

```
GLuint triangleVBO[1];
glGenBuffers(1, triangleVBO);
 glBindBuffer(GL_ARRAY_BUFFER, triangleVBO[0]);
 glBufferData(GL_ARRAY_BUFFER, 9 * sizeof(GLfloat), vertices, GL_STATIC_DRAW);
 glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Three OpenGL calls are required to create and allocate the vertex buffer object. The first, `glGenBuffers` creates a handle that can be used to refer to the VBO once it is stored on the device. Multiple handles to VBOs (stored in arrays) can be created in a single `glGenBuffers` call, as illustrated but not utilized here. Note that when a buffer object is generated, the actual allocation of space on the device is not yet performed.

With OpenGL, *objects*, such as vertex buffer objects, are primary targets for computation and processing. Objects must be bound to a known OpenGL state

首先, 让我们专注于指定几何图元本身。这通过在应用程序的主机内存内分配与基元相关联的顶点开始。执行此操作的最一般方法是在主机上定义一个数组, 以包含基元所需的顶点。例如, 完全包含在规范卷中的单个三角形可以在主机上静态定义, 如下所示:

```
GLfloat vertices[] = {-0.5f, -0.5f, 0.0f, 0.5f, -0.5f, 0.0f, 0.0f, 0.5f, 0.0f};
```

如果简单的直通着色器用于此三角形, 则所有顶点都将被渲染。虽然三角形被放置在 $z=0$ 平面上, 但此示例的 z 坐标并不重要, 因为它们在最终转换为屏幕坐标时基本上被丢弃。另外需要注意的是类型的使用

`GLfloat`在这些例子中。就像GLSL语言有专门的类型一样, OpenGL也有相关的类型, 通常可以与标准类型(如`float`)很好地混合。为了精确, 必要时将使用OpenGL类型。在处理顶点之前, 首先在设备上创建顶点缓冲区以存储顶点。然后将主机上的顶点传输到设备。在此之后, 可以根据需要引用顶点缓冲区来绘制存储在缓冲区中的顶点数组。而且, 在顶点数据的初始传输之后, 不需要发生跨主机到设备总线的数据的额外复制, 特别是如果几何体在渲染循环更新期间保持静态。如果是动态分配的, 也可以删除任何主机内存。

OpenGL Coordinate System: The coordinate system used by OpenGL is identical to that presented in this book. It is a right-handed coordinate system with $+x$ to the right, $+y$ up, and $+z$ away from the screen (or window). Thus, $-z$ points into the monitor.

OpenGL Coordinate
OpenGL所使用的坐标系
与本书中所介绍的相同。

它是一个
右手坐标系, $+x$ 向右, $+y$ 向上, $+z$ 远离屏幕(或
窗口)。因此 $-z$ 指向监
视器。

顶点缓冲区对象(通常称为VBOs)表示现代OpenGL在图形存储器中存储顶点和顶点属性的主要机制。为了提高效率, VBO的初始设置和顶点相关数据的传输主要发生在进入显示循环之前。例如, 要为此三角形创建VBO, 可以使用以下代码:

```
GLuint triangleVBO[1];
 glGenBuffers(1, triangleVBO);
 glBindBuffer(GL_ARRAY_BUFFER, triangleVBO[0]);
 glBufferData(GL_ARRAY_BUFFER, 9 * sizeof(GLfloat), vertices, GL_STATIC_DRAW);
 glBindBuffer(GL_ARRAY_BUFFER, 0);
```

创建和分配顶点缓冲区对象需要三次OpenGL调用。首先, `glGenBuffers`创建一个句柄, 一旦VBO存储在设备上, 就可以使用该句柄来引用它。可以在单个`glGenBuffers`调用中创建VBOs的多个句柄(存储在数组中), 如图所示, 但此处未使用。请注意, 当生成缓冲区对象时, 尚未执行设备上空间的实际分配。

使用OpenGL, 对象(如顶点缓冲区对象)是计算和处理的主要目标。对象必须绑定到已知的OpenGL状态

when used and unbound when not in use. Examples of OpenGL's use of objects include the vertex buffer objects, framebuffer objects, texture objects, and shader programs, to name a few. In the current example, the `GL_ARRAY_BUFFER` state of OpenGL is bound to the triangle VBO handle that was generated previously. This essentially makes the triangle VBO the active vertex buffer object. Any operations that affect vertex buffers that follow the `glBindBuffer(GL_ARRAY_BUFFER, triangleVBO[0])` command will use the triangle data in the VBO either by reading the data or writing to it.

Vertex data is copied from the host (the `vertices` array) to the device (currently bound `GL_ARRAY_BUFFER`) using the

```
glBufferData(GL_ARRAY_BUFFER, 9 * sizeof(GLfloat), vertices, GL_STATIC_DRAW);
```

call. The arguments represent the type of target, the size in bytes of the buffer to be copied, the pointer to the host buffer, and an enumerated type that indicates how the buffer will be used. In the current example, the target is `GL_ARRAY_BUFFER`, the size of the data is `9 * sizeof(GLfloat)`, and the last argument is `GL_STATIC_DRAW` indicating to OpenGL that the vertices will not change over the course of the rendering. Finally, when the VBO no longer needs to be an active target for reading or writing, it is unbound with the `glBindBuffer(GL_ARRAY_BUFFER, 0)` call. In general, binding any of OpenGL's objects or buffers to handle 0, unbinds, or disables that buffer from affecting subsequent functionality.

17.10 Vertex Array Objects

While vertex buffer objects are the storage containers for vertices (and vertex attributes), *vertex array objects* represent OpenGL's mechanism to bundle vertex buffers together into a consistent vertex state that can be communicated and linked with shaders in the graphics hardware. Recall that the fixed function pipeline of the past no longer exists and therefore, per-vertex state, such as normals or even vertex colors, must be stored in hardware buffers and then referenced in shaders, using input variables (e.g., `in`).

As with vertex buffer objects, vertex array objects, or VAOs, must be created and allocated with any necessary state being set while the vertex array object is bound. For instance, the following code shows how to create a VAO to contain the triangle VBO previously defined:

```
GLuint VAO;
 glGenVertexArrays(1, &VAO);
 glBindVertexArray(VAO);
```

使用时和不使用时解绑。OpenGL使用对象的示例包括顶点缓冲区对象、帧缓冲区对象、纹理对象和着色器程序，仅举几例。在当前示例中，OpenGL的GL数组缓冲区状态绑定到之前生成的三角形VBO句柄。这实质上使三角形VBO成为活动顶点缓冲对象。影响遵循`glBindBuffer(GL_ARRAY_BUFFER vbo[0])`命令的顶点缓冲区的任何操作都将通过读取数据或写入VBO中使用三角形数据。

顶点数据从主机（顶点数组）复制到设备（currently绑定GL数组缓冲区）使用

```
glBufferData(GL_ARRAY_BUFFER, 9 * sizeof(GLfloat), vertices, GL_STATIC_DRAW);
```

打电话。参数表示目标的类型、要复制的缓冲区的字节大小、指向主机缓冲区的指针以及指示如何使用缓冲区的枚举类型。在当前示例中，目标是GL数组缓冲区，数据的大小是`9 * sizeof(GLfloat)`，最后一个参数是GL静态绘制，向OpenGL指示顶点不会在渲染过程中发生变化。最后，当VBO不再需要作为读取或写入的活动目标时，它与`glBindBuffer(GL_ARRAY_BUFFER 0)`调用解除绑定。通常，绑定OpenGL的任何对象或缓冲区来处理0、取消绑定或禁止该缓冲区影响后续功能。

17.10 顶点数组对象

虽然顶点缓冲区对象是顶点（和顶点属性）的存储容器，但顶点数组对象表示OpenGL的机制，将顶点缓冲区捆绑在一起，形成一致的顶点状态，可以与图形硬件中的着色器通信和链接。回想一下，过去的固定函数管道不再存在，因此，每个顶点状态，如法线甚至顶点颜色，必须存储在硬件缓冲区中，然后使用输入变量（例如`in`）在着色器中引用。

与顶点缓冲区对象一样，必须在绑定顶点数组对象时创建和分配任何必要的状态。例如，以下代码演示如何创建VAO以包含先前定义的三角形VBO：

```
GLuint VAO;
 glGenVertexArrays(1, &VAO);
 glBindVertexArray(VAO);
```

```
glEnableVertexAttribArray(0);
 glBindBuffer(GL_ARRAY_BUFFER, triangleVBO[0]);
 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), 0);

 glBindVertexArray(0);
```

When defining a vertex array object, specific vertex buffer objects can be bound to specific vertex attributes (or inputs) in shader code. Recall the use of

```
layout(location=0) in vec3 in_Position
```

in the passthrough vertex shader. This syntax indicate that the shader variable will receive its data from attribute index 0 in the bound vertex array object. In host code, the mapping is created using the

```
glEnableVertexAttribArray(0);
 glBindBuffer(GL_ARRAY_BUFFER, triangleVBO[0]);
 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), 0);
```

calls. The first call enables the vertex attribute index (in this case, 0). The next two calls connect the previously defined vertex buffer object that holds the vertices to the vertex attribute itself. Because `glVertexAttribPointer` utilizes the currently bound VBO, it is important that the `glBindBuffer` is issued before assigning the vertex attribute pointer. These function calls create a mapping that binds the vertices in our vertex buffer to the `in_Position` variable within the vertex shader. The `glVertexAttribPointer` calls seems complicated but it basically sets attribute index 0 to hold three components (e.g., x, y, z) of `GLfloats` (the 2nd and 3rd arguments) that are not normalized (the fourth argument). The fifth argument instructs OpenGL that three float values separate the starts of each vertex set. In other words, the vertices are tightly packed in the memory, one after the other. The final argument is a pointer to the data, but because a vertex buffer has been bound prior to this call, the data will be associated with the vertex buffer.

The previous steps that initialize and construct the vertex array object, the vertex buffer objects, and the shaders should all be executed prior to entering the display loop. All memory from the vertex buffer will have been transferred to the GPU and the vertex array objects will make the connection between the data and shader input variable indexes. In the display loop, the following calls will trigger the processing of the vertex array object:

```
glBindVertexArray(VAO);
 glDrawArrays(GL_TRIANGLES, 0, 3);
 glBindVertexArray(0);
```

```
glEnableVertexAttribArray(0);
 glBindBuffer(GL_ARRAY_BUFFER, triangleVBO[0]);
 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), 0);

 glBindVertexArray(0);
```

定义顶点数组对象时，可以将特定顶点缓冲区对象绑定到着色器代码中的特定顶点属性（或输入）。回想一下

```
layout(location=0) in vec3 in_Position
```

在直通顶点着色器中。此语法指示着色器变量将从绑定顶点数组对象中的属性索引0接收其数据。在主机代码中，映射是使用

```
glEnableVertexAttribArray(0);
 glBindBuffer(GL_ARRAY_BUFFER, triangleVBO[0]);
 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), 0);
```

电话。第一次调用启用顶点属性索引（在本例中为0）。接下来的两个调用将先前定义的顶点缓冲区对象连接到顶点属性本身。因为`glVertexAttribPointer`利用当前绑定的VBO，所以在分配顶点属性指针之前发出`glBindBuffer`是很重要的。这些函数调用创建一个映射，将顶点缓冲区中的顶点绑定到顶点着色器中的`inPosition`变量。`glVertexAttribPointer`调用似乎很复杂，但它基本上将属性索引0设置为保存`GLfloats`（第二个和第三个参数）的三个组件（例如， x, y, z ），这些组件未归一化（第四个argument）。第五个参数指示OpenGL三个float值分隔每个顶点集的开始。换句话说，顶点紧密配合在内存中，一个接一个。最后一个参数是指向数据的指针，但由于在这个调用之前已经绑定了顶点缓冲区，数据将与顶点缓冲区相关联。

前面初始化和构造顶点数组对象、顶点缓冲区对象和着色器的步骤都应该在进入显示循环之前执行。顶点缓冲区中的所有内存都将被传输到GPU，顶点数组对象将使数据和着色器输入变量索引之间的连接。在显示循环中，以下调用将触发顶点数组对象的处理：

```
glBindVertexArray(VAO);
 glDrawArrays(GL_TRIANGLES, 0, 3);
 glBindVertexArray(0);
```

Note again, that a bind call makes the vertex array object active. The call to `glDrawArrays` initiates the pipeline for this geometry, describing that the geometry should be interpreted as a series of triangle primitives starting at offset 0 and only rendering three of the indices. In this example, there are only three elements in the array and the primitive is a triangle, so a single triangle will be rendered.

Combining all of these steps, the assembled code for the triangle would resemble the following, assuming that shader and vertex data loading are contained in external functions:

```
// Set the viewport once
int nx, ny;
glfwGetFramebufferSize(window, &nx, &ny);
glViewport(0, 0, nx, ny);

// Set clear color state
glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );

// Create the Shader programs, VBO, and VAO
GLuint shaderID = loadPassthroughShader();
GLuint VAO = loadVertexData();

while ( !glfwWindowShouldClose(window) ) {
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glUseProgram( shaderID );

    glBindVertexArray(VAO);
    glDrawArrays(GL_TRIANGLES, 0, 3);
    glBindVertexArray(0);

    glUseProgram( 0 );

    // Swap front and back buffers
    glfwSwapBuffers(window);

    // Poll for events
    glfwPollEvents();
    if (glfwGetKey( window, GLFW_KEY_ESCAPE ) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, 1);
}
```



Figure 17.3. The canonical triangle rendered using the simple vertex and fragment shaders.

Figure 17.3 shows the result of using the shaders and vertex state to render the canonical view volume triangle.

请再次注意，绑定调用使顶点数组对象处于活动状态。对`glDrawArrays`的调用将启动此几何体的管道，描述几何体应被解释为从偏移量0开始的一系列三角形基元，并且只呈现三个索引。在此示例中，数组中只有三个元素，并且基元是三角形，因此将呈现单个三角形。

结合所有这些步骤，三角形的组装代码将类似于以下内容，假设着色器和顶点数据加载包含在外部函数中：

设置一次视口 `int nx ny; glfwGetFramebufferSize(window &nx &ny); glViewport(0 0 nx ny);`

设置清晰的颜色状态 `glClearColor(0.0f 0.0f 0.0f 1.0f);`

创建着色程序、VBO和VAO `GLuint shaderID = loadPassthroughShader(); GLuint VAO = loadVertexData();`

while (!glfwWindowShouldClose(window)) {

{

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glUseProgram(shaderID);

glBindVertexArray(VAO);
 glDrawArrays(GL_TRIANGLES, 0, 3);
 glBindVertexArray(0);

glUseProgram(0);

// 交换前后缓冲区 `glfwSwapBuffers(窗口);`

轮询事件 `glfwPollEvents(); if(glfwGetKey(window GLFW_KEY_ESCAPE)==GLFW_PRESS)`

`glfwSetWindowShouldClose(window, 1);`



使用简单的顶点和片段着色器渲染的canonical三角形。

图17.3显示了使用着色器和顶点状态渲染规范视图体积三角形的结果。



17.11 Transformation Matrices

Current versions of OpenGL have removed the matrix stacks that were once used to reference the projection and modelview matrices from the hardware. Because these matrix stacks no longer exist, the programmer must write matrix code that can be transferred to vertex shaders where the transformations will occur. That initially may seem challenging. However, several libraries and toolkits have been developed to assist with cross-platform development of OpenGL code. One of these libraries, GLM, or OpenGL Mathematics, has been developed to track the OpenGL and GLSL specifications closely so that interoperation between GLM and the hardware will work seamlessly.

17.11.1 GLM

GLM provides several basic math types useful to computer graphics. For our purposes, we will focus on just a few types and a handful of functions that make use of matrix transforms within the shaders easy. A few types that will be used include the following:

- `glm::vec3`—a compact array of 3 floats that can be accessed using the same component-wise access found in the shaders;
- `glm::vec4`—a compact array of 4 floats that can be accessed using the same component-wise access found in the shaders;
- `glm::mat4`—a 4×4 matrix storage represented as 16 floats. The matrix is stored in column-major format.

Similarly, GLM provides functions for creating the projection matrices, M_{orth} and M_p , as well as functions for generating the view matrix, M_{cam} :

- `glm::ortho` creates a 4×4 orthographic projection matrix.
- `glm::perspective` creates the 4×4 perspective matrix.
- `glm::lookAt` creates the 4×4 homogeneous transform that translates and orients the camera.



17.11 变换矩阵

当前版本的OpenGL已经从硬件中删除了曾经用于引用投影和modelview矩阵的矩阵堆栈。由于这些矩阵堆栈不再存在，因此程序员必须编写矩阵代码，这些代码可以传输到将发生转换的顶点着色器。这一开始可能看起来很有挑战性。但是，已经开发了几个库和工具包来协助OpenGL代码的跨平台开发。其中一个库，GLM，或OpenGL数学，已经被开发来密切跟踪OpenGL和GLSL规范，以便GLM和硬件之间的互操作将无缝地工作。

17.11.1 GLM

GLM提供了几种对计算机图形学有用的基本数学类型。为了我们的目的，我们将专注于几个类型和一些函数，使使用矩阵变换在着色器容易。将使用的几种类型包括以下内容：

*`glm::vec3`—一个由3个浮点数组成的紧凑数组，可以使用着色器中相同的组件访问访问；

*`glm::vec4`—一个由4个浮点数组成的紧凑数组，可以使用着色器中相同的组件访问访问；

*`glm::mat4`—表示为16个浮点数的 4×4 矩阵存储。矩阵以列主格式存储。
◦

同样，GLM提供了用于创建投影矩阵的函数，`Morth`和`Mp`，以及用于生成视图矩阵`Mcam`的函数：

*`glm::ortho`创建 4×4 正投影矩阵。

*`glm::perspective`创建 4×4 透视矩阵。

*`glm::lookAt`创建 4×4 齐次变换，用于平移和定向摄像机。

17.11.2 Using an Orthographic Projection

A simple extension to the previous example would be to place the triangle vertices into a more flexible coordinate system and render the scene using an orthographic projection. The vertices in the previous example could become:

```
GLfloat vertices[] = {-3.0f, -3.0f, 0.0f, 3.0f, -3.0f, 0.0f, 0.0f, 3.0f, 0.0f};
```

Using GLM, an orthographic projection can be created easily on the host. For instance,

```
glm::mat4 projMatrix = glm::ortho(-5.0f, 5.0f, -5.0, 5.0, -10.0f, 10.0f);
```

The projection matrix can then be applied to each vertex transforming it into clip coordinates. The vertex shader will be modified to perform this operation:

$$\mathbf{v}_{\text{canon}} = \mathbf{M}_{\text{orth}} \mathbf{v}.$$

This computation will occur in a modified vertex shader that uses *uniform* variables to communicate data from the host to the device. Uniform variables represent static data that is invariant across the execution of a shader program. The data is the same for all elements and remains static. However, uniform variables can be modified by an application between executions of a shader. This is the primary mechanism that data within the host application can communicate changes to shader computations. Uniform data often represent the graphics state associated with an application. For instance, the projection, view, or model matrices can be set and accessed through uniform variables. Information about light sources within a scene may also be obtained through uniform variables.

Modifying the vertex shader requires adding a uniform variable to hold the projection matrix. We can use GLSL's `mat4` type to store this data. The projection matrix can then be used naturally to transform the incoming vertices into the canonical coordinate system:

```
#version 330 core

layout(location=0) in vec3 in_Position;
uniform mat4 projMatrix;

void main(void)
{
    gl_Position = projMatrix * vec4(in_Position, 1.0);
}
```

17.11.2 使用正投影

上一个示例的一个简单扩展是将三角形顶点放置到更灵活的坐标系中，并使用正投影渲染场景。前面示例中的顶点可能会变成：

```
GLfloat vertices[] = {-3.0f, -3.0f, 0.0f, 3.0f, -3.0f, 0.0f, 0.0f, 3.0f, 0.0f};
```

使用GLM，可以在主机上轻松创建正投影。例如

```
glm::mat4 projMatrix = glm::ortho(-5.0f, 5.0f, -5.0, 5.0, -10.0f, 10.0f);
```

然后可以将投影矩阵应用于将其转换为剪辑坐标的每个顶点。顶点着色器将被修改以执行此操作：

$$\mathbf{v}_{\text{佳能}} = \mathbf{M} \mathbf{v}.$$

此计算将发生在修改后的顶点着色器中，该着色器使用均匀可变性将数据从主机传输到设备。统一变量重发在着色器程序执行过程中不变的静态数据。所有元素的数据都是相同的，并且保持静态。但是，应用程序可以在着色器的执行之间修改统一变量。这是宿主应用程序内的数据可以将更改传达给着色器计算的pri机制。统一数据通常表示与应用程序相关的图形状态。例如，可以通过统一变量设置和访问投影、视图或模型矩阵。关于场景内光源的信息也可以通过均匀变量获得。

修改顶点着色器需要添加一个统一变量来保存投影矩阵。我们可以使用GLSL的`mat4`类型来存储这些数据。然后可以自然地使用投影矩阵将传入的顶点转换到规范坐标系中：

```
#version 330 core

布局 (位置=0) 在 vec3in_Position;uniformmat4proj
Matrix;

void main(void)
{
    gl_Position = projMatrix * vec4(in_Position, 1.0);
}
```

The application code need only transfer the uniform variable from the host memory (a GLM mat4) into the device's shader program (a GLSL mat4). This is easy enough, but requires that the host side of the application acquire a handle to the uniform variable after the shader program has been linked. For instance, to obtain a handle to the `projMatrix` variable, the following call would be issued once, after shader program linking is complete:

```
GLint pMatID = glGetUniformLocation(shaderProgram, "projMatrix");
```

The first argument is the shader program object handle and the second argument is the character string of the variable name in the shader. The id can then be used with a variety of OpenGL `glUniform` function call to transfer the memory on the host into the device. However, shader programs must first be bound prior to setting the value related to a uniform variable. Also, because GLM is used to store the projection matrix on the host, a GLM helper function will be used to obtain a pointer to the underlying matrix, and allow the copy to proceed.

```
glUseProgram( shaderID );
glUniformMatrix4fv(pMatID, 1, GL_FALSE, glm::value_ptr(projMatrix));
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, 3);
glBindVertexArray(0);

glUseProgram( 0 );
```

Notice the form that `glUniform` takes. The function name ends with characters that help define how it is used. In this case, a single 4×4 matrix of floats is being transferred into the uniform variable. The `v` indicates that an array contains the data, rather than passing by value. The third argument lets OpenGL know whether the matrix should be transposed (a potentially handy feature), and the last argument is a pointer to the memory where the matrix resides.

By this section of the chapter, you should have a sense for the role that shaders and vertex data play in rendering objects with OpenGL. Shaders, in particular, form a very important role in modern OpenGL. The remaining sections will further explore the role of shaders in rendering scenes, attempting to build upon the role that shaders play in other rendering styles presented in this book.

17.12 Shading with Per-Vertex Attributes

The previous examples specified a single triangle with no additional data. Vertex attributes, such as normal vectors, texture coordinates, or even colors, can be

应用程序代码只需要将统一变量从主机内存 (GLMmat4) 传输到设备的着色器程序 (GLSLmat4) 中。这很容易，但要求应用程序的主机端在着色器程序链接后获取统一变量的句柄。例如，要获取`projMatrix`变量的句柄，在着色器程序链接完成后，将发出一次以下调用：

```
GLint pMatID = glGetUniformLocation(shaderProgram, "projMatrix");
```

第一个参数是着色器程序对象句柄，第二个参数是着色器中变量名称的字符串。然后，该id可以与各种OpenGLUniform函数调用一起使用，以将主机上的内存传输到设备中。但是，在设置与统一变量相关的值之前，必须首先绑定着色器程序。此外，由于GLM用于在主机上存储投影矩阵，因此将使用GLM辅助函数来获取指向基础矩阵的指针，并允许复制继续进行。

```
glUseProgram( shaderID );
glUniformMatrix4fv(pMatID, 1, GL_FALSE, glm::value_ptr(projMatrix));
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, 3);
glBindVertexArray(0);

glUseProgram( 0 );
```

请注意`glUniform`采用的形式。函数名以字符结尾，这些字符有助于定义函数的使用方式。在这种情况下一个 4×4 的浮点矩阵被转移到统一变量中。`V`表示数组包含数据，而不是按值传递。第三个参数让OpenGL知道矩阵是否应该被转置（一个可能方便的特性），最后一个参数是指向矩阵所在的内存的指针。

通过本章的这一部分，您应该了解着色器和顶点数据在使用OpenGL渲染对象时所扮演的角色。特别是着色器，在现代OpenGL中起着非常重要的作用。其余的部分将进一步探讨着色器在渲染场景中的作用，试图建立在着色器在本书介绍的其他渲染风格中所扮演的角色之上。

17.12 具有每顶点属性的着色

前面的示例指定了没有附加数据的单个三角形。顶点属性，如法向量，纹理坐标，甚至颜色，可以是

interleaved with the vertex data in a vertex buffer. The memory layout is straightforward. Below, the color of each vertex is set after each vertex in the array. Three components are used to represent the red, green, and blue channels. Allocating the vertex buffer is identical with the exception being that the size of the array is now 18 GLfloats instead of 9.

```
GLfloat vertexData[] = {0.0f, 3.0f, 0.0f, 1.0f, 1.0f, 0.0f, -3.0f,
-3.0f, 0.0f, 0.0f, 1.0f, 1.0f, 3.0f, -3.0f, 0.0f, 1.0f, 0.0f, 1.0f};
```

The vertex array object specification is different. Because the color data is interleaved between vertices, the vertex attribute pointers must stride across the data appropriately. The second vertex attribute index must also be enabled. Building off the previous examples, we construct the new VAO as follows:

```
glBindBuffer(GL_ARRAY_BUFFER, m_triangleVBO[0]);

 glEnableVertexAttribArray(0);
 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat),
 0);

 glEnableVertexAttribArray(1);
 glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat),
 (const GLvoid *)12);
```

A single VBO is used and bound prior to setting the attributes since both vertex and color data are contained within the VBO. The first vertex attribute is enabled at index 0, which will represent the vertices in the shader. Note that the stride (the 5th argument) is different as the vertices are separated by six floats (e.g., the x, y, z of the vertex followed by the r, g, b of the color). The second vertex attribute index is enabled and will represent the vertex color attributes in the shader at location 1. It has the same stride, but the last argument now represents the pointer offset for the start of the first color value. While 12 is used in the above example, this is identical to stating $3 * \text{sizeof(GLfloat)}$. In other words, we need to jump across the three floats representing the vertex x, y, z values to locate the first color attribute in the array.

The shaders for this example are only slightly modified. The primary differences in the vertex shader (shown below) are (1) the second attribute, color, is at location 1 and (2) vColor is an output variable that is set in the main body of the vertex shader.

```
#version 330 core

layout(location=0) in vec3 in_Position;
layout(location=1) in vec3 in_Color;
```

与顶点缓冲区中的顶点数据交错。内存布局非常简单。下面，每个顶点的颜色设置在数组中的每个顶点之后。三个分量用于表示红色、绿色和蓝色通道。分配顶点缓冲区是相同的，例外是数组的大小现在是18GLfloats而不是9。

```
GLfloat vertexData[] = {0.0f, 3.0f, 0.0f, 1.0f, 1.0f, 0.0f, -3.0f,
-3.0f, 0.0f, 0.0f, 1.0f, 1.0f, 3.0f, -3.0f, 0.0f, 1.0f, 0.0f, 1.0f};
```

顶点数组对象规范不同。因为颜色数据在顶点之间是交错的，所以顶点属性指针必须适当地跨数据。还必须启用第二个顶点属性索引。在上面的例子的基础上，我们构建了新的VAO，如下所示：

```
glBindBuffer(GL_ARRAY_BUFFER, m_triangleVBO[0]);

 glEnableVertexAttribArray(0);
 glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat),
 0);

 glEnableVertexAttribArray(1);
 glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat),
 (const GLvoid *)12);
```

在设置属性之前使用和绑定单个VBO，因为顶点和颜色数据都包含在VBO中。第一个顶点属性在索引0处启用，它将表示着色器中的顶点。请注意，步幅（第5个参数）是不同的，因为顶点由六个浮点数分隔（例如，顶点的 x, y, z 后跟颜色的 r, g, b ）。第二个顶点属性索引已启用，将表示位置1处着色器中的顶点颜色属性。它具有相同的步幅，但最后一个参数现在表示第一个颜色值开始的指针偏移量。虽然在上面的例子中使用了12，但这与说明 $3 * \text{sizeof(GLfloat)}$ 相同。换句话说，我们需要跳过表示顶点 x, y, z 值的三个浮点数来定位数组中的第一个颜色属性。

此示例的着色器仅稍作修改。顶点着色器（如下所示）中的主要区别是（1）第二个属性color位于位置1，（2）vColor是在顶点着色器主体中设置的输出变量。

```
#version 330 core

vec3 in_Position中的layout(location=0);vec3 in_Color
中的layout(location=1);
```

```
out vec3 vColor;
uniform mat4 projMatrix;
void main(void)
{
    vColor = in_Color;
    gl_Position = projMatrix * vec4(in_Position, 1.0);
}
```

Recall that the keywords `in` and `out` refer to the flow of data between shaders. Data that flows out of the vertex shader becomes input data in the connected fragment shader, provided that the variable names match up. Moreover, `out` variables that are passed to fragment shaders are interpolated across the fragments using barycentric interpolation. Some modification of the interpolation can be achieved with additional keywords, but this detail will be left to the reader. In this example, three vertices are specified, each with a specific color value. Within the fragment shader, the colors will be interpolated across the face of the triangle.

The fragment shader changes are simple. The `vColor` variable that was set and passed `out` of the vertex shader now becomes an `in` variable. As fragments are processed, the `vColor` `vec3` will contain the correctly interpolated values based on the location of the fragment within the triangle.

```
#version 330 core
layout(location=0) out vec4 fragmentColor;
in vec3 vColor;
void main(void)
{
    fragmentColor = vec4(vColor, 1.0);
}
```

The image that results from running this shader with the triangle data is shown in Figure 17.4.

17.12.1 Structs of Vertex Data

The previous example illustrates the interleaving of data in an array. Vertex buffers can be used in a variety of ways, including separate vertex buffers for different model attributes. Interleaving data has advantages as the attributes associated with a vertex are near the vertex in memory and can likely take advantage

```
out vec3 vColor;
uniform mat4 projMatrix;
void main(void)
{
    vColor = in_Color;
    gl_Position = projMatrix * vec4(in_Position, 1.0);
}
```

回想一下，关键字`in`和`out`指的是着色器之间的数据流。从顶点着色器流出的数据将成为连接的片段着色器中的输入数据，前提是变量名称匹配。此外，传递给片段着色器的`out`变量将使用重心插值跨片段进行插值。插值的一些修改可以用额外的关键字来实现，但是这个细节将留给读者。在此示例中，指定了三个顶点，每个顶点都具有特定的颜色值。在片段着色器中，颜色将在三角形的面上插值。

片段着色器的更改很简单。设置并传递出顶点着色器的`vColor`变量现在变成了`in`变量。在处理片段时，`vColor``vec3`将根据片段在三角形中的位置包含正确插值的值。

```
#version 330 core
layout(location=0) out vec4 fragmentColor;
in vec3 vColor;
void main(void)
{
    fragmentColor = vec4(vColor, 1.0);
}
```

使用三角形数据运行此着色器所产生的图像如下所示
Figure 17.4.

17.12.1 顶点数据的结构

前面的示例说明了阵列中数据的交织。顶点缓冲区可以以多种方式使用，包括用于不同模型属性的单独顶点缓冲区。交错数据具有优势，因为与顶点相关联的属性位于内存中的顶点附近，并且可能会利用

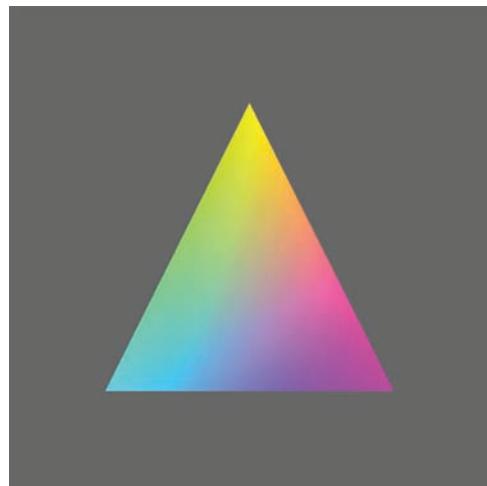


Figure 17.4. Setting the colors of each vertex in the vertex shader and passing the data to the fragment shader results in barycentric interpolation of the colors.

of memory locality when operating in the shaders. While the use of these interleaved arrays is straightforward, it can become cumbersome to manage large models in this way, especially as data structures are used for building robust (and sustainable) software infrastructure for graphics (see Chapter 12). It is rather simple to store vertex data as vectors of structs that contain the vertex and any related attributes. When done this way, the structure need only be mapped into the vertex buffer. For instance, the following structure contains the vertex position and vertex color, using GLM's `vec3` type:

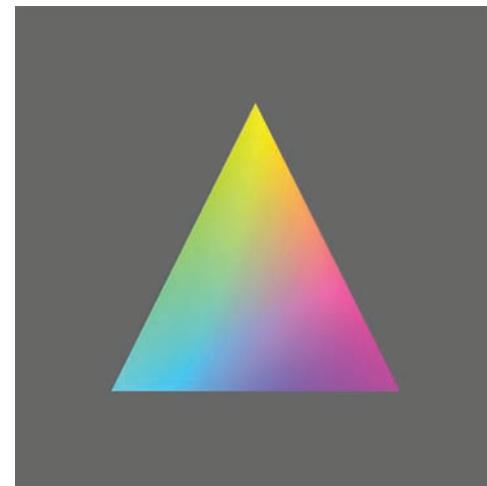
```
struct vertexData
{
    glm::vec3 pos;
    glm::vec3 color;
};

std::vector<vertexData> modelData;
```

The STL vector will hold all vertices related to all the triangles in the model. We will continue to use the same layout for triangles as in previous examples, which is a basic triangle strip. Every three vertices represents a triangle in the list. There are other data organizations that can be used with OpenGL, and Chapter 12 presents other options for organizing data more efficiently.

Once the data is loaded into the vector, the same calls used before load the data into the vertex buffer object:

```
int numBytes = modelData.size() * sizeof(vertexData);
```



在顶点着色器中设置每个顶点的颜色并将数据传递到片段着色器会导致颜色的重心插值。

在着色器中操作时的内存局部性。虽然这些交错阵列的使用很简单，但以这种方式管理大型模型会变得很麻烦，特别是当数据结构用于为图形构建健壮（和可持续）的软件基础设施时（见第12章）。将顶点数据存储为包含顶点和任何相关属性的结构的向量是相当简单的。这样做时，结构只需要映射到顶点缓冲区。例如，下面的结构包含顶点位置和顶点颜色，使用GLM的`vec3`类型：

```
struct vertexData
{
    glm::vec3 pos;
    glm::vec3 color;
};

std::vector<vertexData> modelData;
```

STL向量将保存与模型中所有三角形相关的所有顶点。我们将继续使用与前面示例相同的三角形布局，这是一个基本的三角形条带。每三个顶点代表列表中的一个三角形。还有其他数据组织可以与OpenGL一起使用，第12章介绍了更有效地组织数据的其他选项。

将数据加载到向量后，将数据加载到顶点缓冲区对象之前使用的相同调用：

```
int numBytes = modelData.size() * sizeof(vertexData);
```



```
glBufferData(GL_ARRAY_BUFFER, numBytes, modelData.data(), GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

STL vectors store data contiguously. The `vertexData` struct used above is represented by a flat memory layout (it does not contain pointers to other data elements) and is contiguous. However, the STL vector is an abstraction and the pointer that references the underlying memory must be queried using the `data()` member. That pointer is provided to the call to `glBufferData`. Attribute assignment in the vertex array object is identical as the locality of the vertex attributes remains the same.

17.13 Shading in the Fragment Processor

The graphics pipeline chapter (Chapter 8) and the surface shading chapter (Chapter 10) do a nice job of describing and illustrating the effects of per-vertex and per-fragment shading as they relate to rasterization and shading in general. With modern graphics hardware, applying shading algorithms in the fragment processor produces better visual results and more accurately approximates lighting. Shading that is computed on a per-vertex basis is often subject to visual artifacts related to the underlying geometry tessellation. In particular, per-vertex based shading often fails to approximate the appropriate intensities across the face of the triangle since the lighting is only being calculated at each vertex. For example, when the distance to the light source is small, as compared with the size of the face being shaded, the illumination on the face will be incorrect. Figure 17.5 illustrates this situation. The center of the triangle will not be illuminated brightly, despite being very close to the light source, since the lighting on the vertices, which are far from the light source, are used to interpolate the shading across the face. Of course, increasing the tessellation of the geometry can improve the visuals. However, this solution is of limited use in real-time graphics as the added geometry required for more accurate illumination can result in slower rendering.

Fragment shaders operate on the fragments that emerge from rasterization after vertices have been transformed and clipped. Generally speaking, fragment shaders must output a value that is written to a framebuffer. Often times, this is the color of the pixel. If the depth test is enabled, the fragment's depth value will be used to control whether the color and its depth are written to the framebuffer memory. The data that fragment shaders use for computation comes from various sources:

- **Built-in OpenGL variables.** These variables are provided by the system. Examples of fragment shader variables include `gl_FragCoord` or



```
glBufferData(GL_ARRAY_BUFFER, numBytes, modelData.data(), GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

STL向量连续存储数据。上面使用的`vertexData`结构由平面内存布局表示（它不包含指向其他数据元素的指针）并且是连续的。但是，STL向量是一个抽象，必须使用`data()`成员查询引用基础内存的指针。该指针提供给`glBufferData`的调用。顶点数组对象中的属性分配相同，因为顶点属性的位置保持不变。

17.13 片段处理器中的着色

图形管道章节（第8章）和表面着色章节（第10章）很好地描述和说明了每个顶点和完善着色的效果，因为它们与一般的光栅化和着色有关。使用现代图形硬件，在片段处理器中应用着色算法可以产生更好的视觉效果，并更准确地近似照明。基于每个顶点计算的着色通常会受到与基础几何曲面细分相关的视觉伪影的影响。特别是，基于每个顶点的阴影—

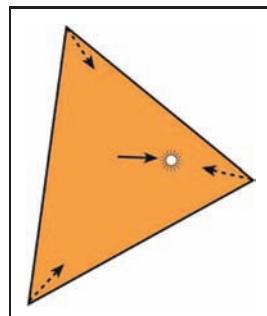
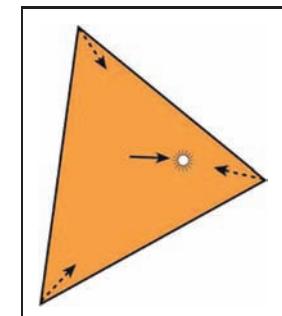


Figure 17.5. The distance to the light source is small relative to the size of the triangle.

由于光照只在每个顶点计算，因此ten无法近似三角形面上的适当强度。例如，当到光源的距离较小时，与被遮光的面部的尺寸相比，面部上的照明将是不正确的。图17.5说明了这种情况。三角形的中心不会被照亮，尽管非常靠近光源，因为顶点上的照明（远离光源）被用来在面上插值阴影。当然，在折痕几何的曲面细分可以提高视觉效果。然而，这种解决方案在实时图形中使用有限，因为更精确照明所需的添加几何体会导致渲染速度变慢。



到光源的距离相对于三角形的尺寸较小。

片段着色器对顶点经过转换和剪切后从光栅化中出现的片段进行操作。一般来说，片段着色器必须输出写入帧缓冲区的值。很多时候，这是像素的颜色。如果启用深度测试，则片段的深度值将用于控制是否将颜色及其深度写入帧缓冲存储器。片段着色器用于计算的数据来自各种来源：

- 内置OpenGL变量。这些变量由系统提供。片段着色器变量的示例包括`glFragCoord`或

`gl_FrontFacing`. These variables can change based on revisions to OpenGL and GLSL, so it is advised that you check the specification for the version of OpenGL and GLSL that you are targeting.

- **Uniform variables.** Uniform variables are transferred from the host to the device and can change as needed based on user input or changing simulation state in the application. These variables are declared and defined by the programmer for use within both vertex and fragment shaders. The projection matrix in the previous vertex shader examples was communicated to the shader via a uniform variable. If needed, the same uniform variable names can be used within both vertex and fragment shaders.
- **Input variables.** *Input* variables are specified in the fragment shader with the prefixed keyword `in`. Recall that data can flow into and out of shaders. Vertex shaders can output data to the next shader stage using the `out` keyword (e.g., `out vec3 vColor`, in a previous example). The outputs are linked to inputs when the next stage uses an `in` keyword followed by the same type and name qualifiers (e.g., `in vec3 vColor` in the previous example's corresponding fragment shader).

Any data that is passed to a fragment shader through the *in-out* linking mechanism will vary on a per-fragment basis using barycentric interpolation. The interpolation is computed outside of the shader by the graphics hardware. Within this infrastructure, fragment shaders can be used to perform per-fragment shading algorithms that evaluate specific equations across the face of the triangle. Vertex shaders provide support computations, transforming vertices and staging intermediate per-vertex values that will be interpolated for the fragment code.

The following shader program code implements per-fragment, Blinn-Phong shading. It brings together much of what has been presented in this chapter thus far and binds it to the shader descriptions from Chapter 4. An interleaved vertex buffer is used to contain the vertex position and normal vectors. These values manifest in the vertex shader as vertex array attributes for index 0 and index 1. The shading computations that occur in the fragment shader code are performed in camera coordinates (sometimes referred to as eye-space).

17.13.1 Blinn-Phong Shader Program: Vertex Shader

The vertex shader stage of our program is used to transform the incoming vertices using the M_{model} and M_{cam} matrices into camera coordinates. It also uses the

`glFrontFacing`。这些变量可以根据对OpenGL和GLSL的修订而更改，因此建议您检查所针对的OpenGL和GLSL版本的规范。

*统一变量。统一变量从主机传输到设备，并且可以根据用户输入或应用中变化的模拟状态根据需要改变。这些变量由程序员声明和定义，以便在顶点着色器和片段着色器中使用。前面顶点着色器示例中的投影矩阵通过统一变量传达给着色器。如果需要，可以在顶点着色器和片段着色器中使用相同的统一变量名称。

*输入变量。输入变量在片段着色器中指定，带有前缀关键字`in`。回想一下，数据可以流入和流出着色器。顶点着色器可以使用`out`关键字将数据输出到下一个着色器阶段（例如，在上一个示例中，`out vec3 vColor`）。当下一阶段使用`in`关键字后跟相同的类型和名称限定符（例如，在前一个示例的相应片段着色器中的`vec3 vColor`中）时，输出链接到输入。

通过`inout`链接机制传递到片段着色器的任何数据将使用重心插值在每个片段的基础上变化。`Interpolation`由图形硬件在着色器之外计算。在此基础架构中，片段着色器可用于执行每个片段着色算法，以评估三角形面上的特定方程。顶点着色器提供支持计算，转换顶点和暂存将为片段代码插值的每个顶点值。

下面的着色器程序代码实现了每个片段，Blinn-Phong着色。它汇集了本章迄今为止所介绍的大部分内容，并将其绑定到第4章的着色器描述。交错顶点缓冲区用于包含顶点位置和法向量。这些值在顶点着色器中表现为索引0和索引1的顶点数组属性。片段着色器代码中发生的着色计算是在相机坐标（有时称为眼睛空间）中执行的。

17.13.1 Blinn-Phong Shader Program: Vertex Shader

我们程序的顶点着色器阶段用于使用 M 模型和 M_{cam} 矩阵将传入顶点转换为相机坐标。它还使用

normal matrix, $(M^{-1})^T$, to appropriately transform the incoming normal vector attribute. The vertex shader outputs three variables to the fragment stage:

- **normal.** The vertex's normal vector as transformed into the camera coordinate system.
- **h.** The half-vector needed for Blinn-Phong shading.
- **l.** The light direction transformed into the camera coordinate system.

Each of these variables will then be available for fragment computation, after applying barycentric interpolation across the three vertices in the triangle.

A single point light is used with this shader program. The light position and intensity is communicated to both the vertex and fragment shaders using a uniform variable. The light data is declared using GLSL's struct qualifier, which allows variables to be grouped together in meaningful ways. Although not presented here, GLSL supports arrays and for-loop control structures, so additional lights could easily be added to this example.

All matrices are also provided to the vertex shader using uniform variables. For now, we will imagine that the model (or local transform) matrix will be set to the identity matrix. In the following section, more detail will be provided to expand on how the model matrix can be specified on the host using GLM.

```
#version 330 core

//  
// Blinn-Phong Vertex Shader  
//  
  
layout(location=0) in vec3 in_Position;  
layout(location=1) in vec3 in_Normal;  
  
out vec4 normal;  
out vec3 half;  
out vec3 lightdir;  
  
struct LightData {  
    vec3 position;  
    vec3 intensity;  
};  
uniform LightData light;  
  
uniform mat4 projMatrix;  
uniform mat4 viewMatrix;  
uniform mat4 modelMatrix;  
uniform mat4 normalMatrix;
```

法向矩阵, $(M^{-1})^T$, 来适当地变换传入的法向矢量属性。顶点着色器将三个变量输出到片段阶段:

- *正常。将顶点的法向量转换为相机坐标系。
- *h. Blinn-Phong阴影所需的半矢量。
- *升。变换到摄像机坐标系中的光方向。

然后, 在三角形中的三个顶点应用重心插值后, 这些变量中的每一个都可用于片段计算。

此着色器程序使用单点光源。使用uniform变量将光线位置和强度传达给顶点着色器和片段着色器。轻数据是使用GLSL的structqualifer声明的, 它允许变量以有意义的方式组合在一起。虽然这里没有预先说明, 但GLSL支持阵列和for-loop控制结构, 因此可以很容易地在这个例子中添加额外的灯光。

所有矩阵也使用统一变量提供给顶点着色器。

现在, 我们将设想模型 (或局部变换) 矩阵将设置为缩进矩阵。在下一节中, 将提供更多详细信息, 以扩展如何使用GLM在主机上指定模型矩阵。

```
#version 330 core

//  
// Blinn-Phong Vertex Shader  
//  
  
vec3in_Position中的layout(location=0);vec3in_Norm  
al中的layout(location=1);  
  
out vec4 normal;  
out vec3 half;  
out vec3 lightdir;  
  
struct LightData {  
    vec3 position;  
    vec3 intensity;  
};  
uniform LightData light;  
  
uniform mat4 projMatrix;  
uniform mat4 viewMatrix;  
uniform mat4 modelMatrix;  
uniform mat4 normalMatrix;
```

```

void main(void)
{
    // Calculate lighting in eye space: transform the local
    // position to world and then camera coordinates.
    vec4 pos = viewMatrix * modelMatrix * vec4(in_Position, 1.0);
    vec4 lightPos = viewMatrix * vec4(light.position, 1.0);

    normal = normalMatrix * vec4(in_Normal, 0.0);

    vec3 v = normalize(-pos.xyz);
    lightdir = normalize(lightPos.xyz - pos.xyz);
    half = normalize(v + lightdir);

    gl_Position = projMatrix * pos;
}

```

The vertex shader's main function first transforms the position and light position into camera coordinates using `vec4` types to correspond with the 4×4 matrices of GLSL's `mat4`. We then transform the normal vector and store it in the `out vec4 normal` variable. The view (or eye) vector and light direction vector are then calculated, which leads to the computation of the half vector needed for Blinn-Phong shading. The final computation completes the calculation of

$$\mathbf{v}_{\text{canon}} = \mathbf{M}_{\text{proj}} \mathbf{M}_{\text{cam}} \mathbf{M}_{\text{model}} \mathbf{v}$$

by applying the projection matrix. It then sets the canonical coordinates of the vertex to the built-in GLSL vertex shader output variable `gl_Position`. After this, the vertex is in clip-coordinates and is ready for rasterization.

17.13.2 Blinn-Phong Shader Program: Fragment Shader

The fragment shader computes the Blinn-Phong shading model. It receives barycentric interpolated values for the vertex normal, half vector, and light direction. Note that these variables are specified using the `in` keyword as they come *in* from the vertex processing stage. The light data is also shared with the fragment shader using the same uniform specification that was used in the vertex shader. The matrices are not required so no uniform matrix variables are declared. The material properties for the geometric model are communicated through uniform variables to specify k_a , k_d , k_s , I_a , and p . Together, the data allow the fragment shader to compute Equation 4.3:

$$L = k_a I_a + k_d I_{\max}(0, \mathbf{n} \cdot \mathbf{l}) + k_s I_{\max}(0, \mathbf{n} \cdot \mathbf{h})^p$$

at each fragment.

```

void main(void)
{
    // 计算眼睛空间中的照明：将局部位置转换为世界，然后转换为相机坐标。vec4pos=
    // viewMatrix*modelMatrix*vec4 (in_Position, 1.0);vec4lightPos=viewMatrix*vec
    // 4 (light. 位, 1.0);

    normal = normalMatrix * vec4(in_Normal, 0.0);

    vec3v=normalize(-pos.xyz);lightdir=normalize (lightPos. xyz
    pos. xyz);half=normalize(v+lightdir);

    gl_Position = projMatrix * pos;
}

```

顶点着色器的主函数首先使用`vec4`类型将位置和光线位置转换为相机坐标，以与GLSL的`mat4`的 4×4 矩阵相对应。然后，我们变换法向量并将其存储在`out vec4`法向量中。然后计算视图（或眼睛）矢量和光方向矢量，这导致计算Blinn-Phong阴影所需的半矢量。最终计算完成计算

$$\mathbf{v}_{\text{canon}} = \mathbf{M}_{\text{proj}} \mathbf{M}_{\text{cam}} \mathbf{M}_{\text{model}} \mathbf{v}$$

通过应用投影矩阵。然后，它将顶点的规范坐标设置为内置的GLSL顶点着色器输出变量`glPosition`。在这之后，顶点是在剪辑坐标和准备栅格化。

17.13.2 Blinn-Phong Shader Program: Fragment Shader

片段着色器计算Blinn-Phong着色模型。它接收顶点法线、半矢量和光线方向的重心插值值。请注意，这些变量是使用`in`关键字指定的，因为它们来自顶点处理阶段。光线数据也使用顶点着色器中使用的相同统一规范与片段着色器共享。矩阵不是必需的，因此没有声明均匀矩阵变量。几何模型的材料属性通过统一变量传达，以指定 k_a 、 k_d 、 k_s 、 I_a 和 p 。这些数据一起允许片段着色器计算公式4.3：

$$L = k_a I_a + k_d I_{\max}(0, \mathbf{n} \cdot \mathbf{l}) + k_s I_{\max}(0, \mathbf{n} \cdot \mathbf{h})^p$$

在每个片段处。

```
#version 330 core
//
// Blinn-Phong Fragment Shader
//

in vec4 normal;
in vec3 half;
in vec3 lightdir;

layout(location=0) out vec4 fragmentColor;

struct LightData {
    vec3 position;
    vec3 intensity;
};
uniform LightData light;

uniform vec3 Ia;
uniform vec3 ka, kd, ks;
uniform float phongExp;

void main(void)
{
    vec3 n = normalize(normal.xyz);
    vec3 h = normalize(half);
    vec3 l = normalize(lightdir);

    vec3 intensity = ka * Ia
        + kd * light.intensity * max( 0.0, dot(n, l) )
        + ks * light.intensity
            * pow( max( 0.0, dot(n, h) ), phongExp );
    fragmentColor = vec4( intensity, 1.0 );
}
```

The fragment shader writes the computed intensity to the fragment color output buffer. Figure 17.6 illustrates several examples that show the effect of per-fragment shading across varying degrees of tessellation on a geometric model. This fragment shader introduces the use of structures for holding uniform variables. It should be noted that they are user-defined structures, and in this example, the LightData type holds only the light position and its intensity. In host code, the uniform variables in structures are referenced using the fully qualified variable name when requesting the handle to the uniform variable, as in:

```
lightPosID = shader.createUniform( "light.position" );
lightIntensityID = shader.createUniform( "light.intensity" );
```

```
#version 330 core
//
// Blinn-Phong Fragment Shader
//

in vec4 normal;
in vec3 half;
in vec3 lightdir;

layout(location=0) out vec4 fragmentColor;

struct LightData {
    vec3 position;
    vec3 intensity;
};
uniform LightData light;

uniform vec3 Ia;
uniform vec3 ka, kd, ks;
uniform float phongExp;

void main(void)
{
    vec3 n = normalize(normal.xyz);
    vec3 h = normalize(half);
    vec3 l = normalize(lightdir);

    vec3强度=ka*Ia
        +kd*光。强度*最大(0.0 点(n l))+ks*光.强度
            *pow(最大(0.0 点(n h)) phongExp);
    fragmentColor = vec4( intensity, 1.0 );
}
```

片段着色器将计算的强度写入片段颜色输出缓冲区。图17.6展示了几个示例，这些示例显示了几何模型上不同程度曲面细分的完美着色效果。这个片段着色器介绍了使用结构来保存统一变量。应该注意的是，它们是用户定义的结构，并且在本示例中，LightData类型仅保存光线位置及其强度。在主机代码中，当请求统一变量的句柄时，使用全限定变量名引用结构中的统一变量，如：

```
lightPosID = shader.createUniform( "light.position" );
lightIntensityID = shader.createUniform( "light.intensity" );
```

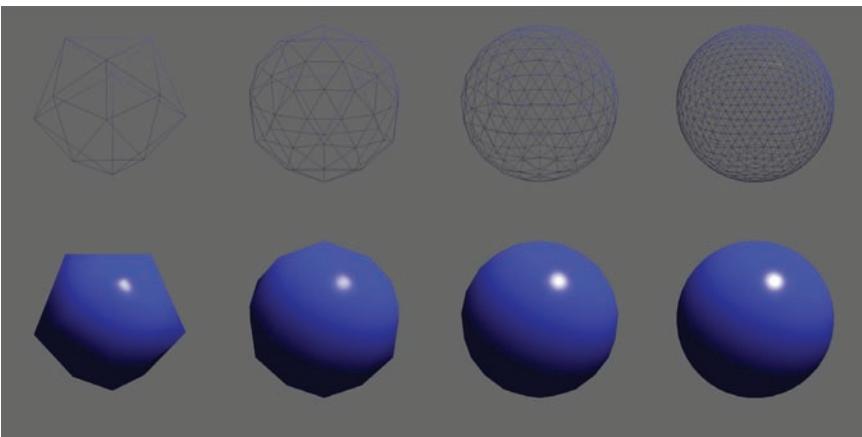


Figure 17.6. Per-fragment shading applied across increasing tessellation of a subdivision sphere. The specular highlight is apparent with lower tessellations.

17.13.3 A Normal Shader

Once you have a working shader program, such as the Blinn-Phong one presented here, it is easy to expand your ideas and develop new shaders. It may also be helpful to develop a set of very specific shaders for debugging. One such shader is the normal shader program. Normal shading is often helpful to understand whether the incoming geometry is organized correctly or whether the computations are correct. In this example, the vertex shader remains the same. Only the fragment shader changes:

```
#version 330 core

in vec4 normal;

layout(location=0) out vec4 fragmentColor;

void main(void)
{
    // Notice the use of swizzling here to access
    // only the xyz values to convert the normal vec4
    // into a vec3 type!
    vec3 intensity = normalize(normal.xyz) * 0.5 + 0.5;
    fragmentColor = vec4( intensity, 1.0 );
}
```

Whichever shaders you start building, be sure to comment them! The GLSL specification allows comments to be included in shader code, so leave yourself some details that can guide you later.

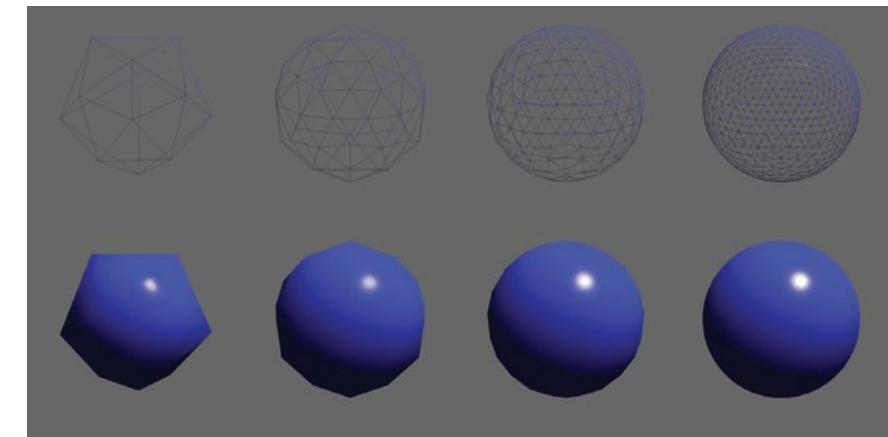


Figure 17.6. 每个片段着色应用于细分球体的增加曲面细分。镜面高光是明显的与较低的镶嵌。

17.13.3 A Normal Shader

一旦你有了一个有效的着色器程序，比如这里介绍的Blinn-Phong一个，就很容易扩展你的想法和开发新的着色器。开发一组用于调试的非常特定的着色器也可能会有所帮助。一个这样的着色器是正常的着色器程序。正常着色通常有助于了解传入几何结构是否组织正确或计算是否正确。在此示例中，顶点着色器保持不变。只有片段着色器更改：

```
#version 330 core

in vec4 normal;

layout(location=0) out vec4 fragmentColor;

void main(void)
{
    // 注意，这里使用swizzling只访问xyz值，将正常的vec4转换为vec3
    // 类型！ vec3强度=normalize(正常。xyz)*0.5+0.5;fragmentColor=vec4(i
    // ntensity 1.0);
}
```

无论您开始构建哪个着色器，请务必评论它们！GLSL规范允许在着色器代码中包含注释，因此请为自己留下一些可以在以后指导您的细节。

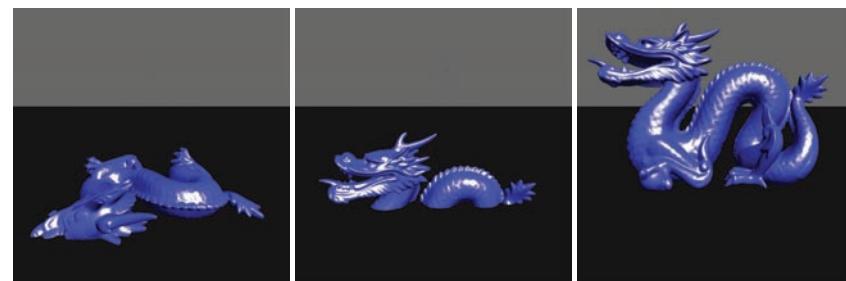


Figure 17.7. Images are described from left to right. The default local orientation of the dragon, lying on its side. After a -90 degree rotation about \vec{X} , the dragon is upright but still centered about the origin. Finally, after applying a translation of 1.0 in \vec{Y} , the dragon is ready for instancing.

17.14 Meshes and Instancing

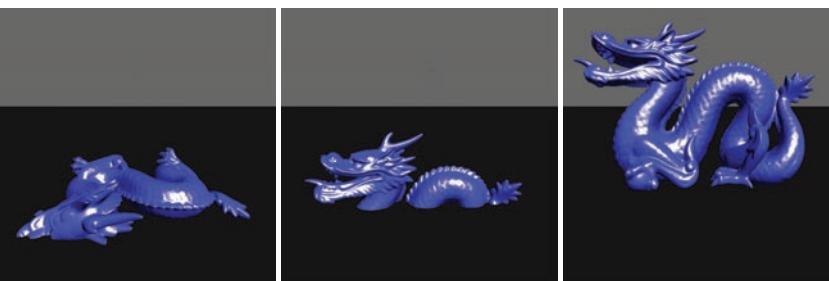
Once basic shaders are working, it's interesting to start creating more complex scenes. Some 3D model files are simple to load and others require more effort. One simple 3D object file representation is the OBJ format. OBJ is a widely used format and several codes are available to load these types of files. The array of structs mechanism presented earlier works well for containing the OBJ data on the host. It can then easily be transferred into a VBO and vertex array objects.

Many 3D models are defined in their own local coordinate systems and need various transformations to align them with the OpenGL coordinate system. For instance, when the Stanford Dragon's OBJ file is loaded into the OpenGL coordinate system, it appears lying on its side at the origin. Using GLM, we can create the model transformations to place objects within our scenes. For the dragon model, this means rotating -90 degrees about \vec{X} , and then translating up in \vec{Y} . The effective model transform becomes

$$\mathbf{M}_{\text{model}} = \mathbf{M}_{\text{translate}} \mathbf{M}_{\text{rotX}},$$

and the dragon is presented upright and above the ground plane, as shown in Figure 17.7. To do this we utilize several functions from GLM for generating local model transforms:

- `glm::translate` creates a translation matrix.
- `glm::rotate` creates a rotation matrix, specified in either degrees or radians about a specific axis.
- `glm::scale` creates a scale matrix.



图像从左到右描述。龙的默认局部方向，躺在它的一边。在大约 X 旋转 -90 度后，龙是直立的，但仍然以原点为中心。最后，在应用 Y 中的翻译之后。在 Y 中，龙已经准备好实例化了。

17.14 网格和实例化

一旦基本着色器工作，开始创建更复杂的场景是很有趣的。一些3D模型文件很容易加载，而另一些则需要更多的努力。一个简单的3D对象文件表示形式是OBJ格式。OBJ是一种广泛使用的格式，有几种代码可用于加载这些类型的文件。前面介绍的结构体数组机制可以很好地包含主机上的OBJ数据。然后，它可以很容易地被转移到一个VBO和顶点数组对象。

许多3d模型都是在自己的局部坐标系中定义的，需要进行各种变换才能将它们与OpenGL坐标系对齐。例如，当斯坦福龙的OBJ文件加载到OpenGL协调系统时，它出现在原点的侧面。使用GLM，我们可以创建模型转换以将对象放置在场景中。对于dragon模型，这意味着旋转-大约 $X90$ 度，然后在 Y 中向上平移。有效模型变换变为

$$\mathbf{M}_{\text{模型}} = \mathbf{M}_{\text{翻译}} \mathbf{m}_{\text{rotX}}$$

龙呈直立并在地平面上方，如图17.7所示。为此，我们使用GLM的几个函数来生成局部模型转换：

*`glm::translate` 创建翻译矩阵。

*`glm::rotate` 创建一个旋转矩阵，以特定轴的度数或弧度指定。

*`glm::scale` 创建一个缩放矩阵。

We can apply these functions to create the model transforms and pass the model matrix to the shader using uniform variables. The Blinn-Phong vertex shader contains instructions that apply the local transform to the incoming vertex. The following code shows how the dragon model is rendered:

```

glUseProgram( BlinnPhongShaderID );

// Describe the Local Transform Matrix
glm::mat4 modelMatrix = glm::mat4(1.0); // Identity Matrix
modelMatrix = glm::translate(modelMatrix, glm::vec3(0.0f, 1.0f, ←
    0.0f));
float rot = (-90.0f / 180.0f) * M_PI;
modelMatrix = glm::rotate(modelMatrix, rot, glm::vec3(1, 0, 0));

// Set the Normal Matrix
glm::mat4 normalMatrix = glm::transpose( glm::inverse( viewMatrix←
    * modelMatrix ) );

// Pass the matrices to the GPU memory
glUniformMatrix4fv(nMatID, 1, GL_FALSE, glm::value_ptr(←
    normalMatrix));
glUniformMatrix4fv(pMatID, 1, GL_FALSE, glm::value_ptr(projMatrix←
    ));
glUniformMatrix4fv(vMatID, 1, GL_FALSE, glm::value_ptr(viewMatrix←
    ));
glUniformMatrix4fv(mMatID, 1, GL_FALSE, glm::value_ptr(←
    modelMatrix));

// Set material for this object
glm::vec3 kd( 0.2, 0.2, 1.0 );
glm::vec3 ka = kd * 0.15f;
glm::vec3 ks( 1.0, 1.0, 1.0 );
float phongExp = 32.0;

glUniform3fv(kaID, 1, glm::value_ptr(ka));
glUniform3fv(kdID, 1, glm::value_ptr(kd));
glUniform3fv(ksID, 1, glm::value_ptr(ks));
glUniform1f(phongExpID, phongExp);

// Process the object and note that modelData.size() holds
// the number of vertices, not the number of triangles!
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, modelData.size());
glBindVertexArray(0);

glUseProgram( 0 );

```

我们可以应用这些函数来创建模型变换，并使用统一变量将模型矩阵传递给着色器。Blinn-Phong顶点着色器包含将局部变换应用于传入顶点的指令。下面的代码演示了如何渲染dragon模型：

```

glUseProgram( BlinnPhongShaderID );

描述局部变换矩阵glm::mat4modelMatrix=glm::mat4(1.0);单位矩阵modelMatrix=glm::tran
    slate(modelMatrix glm::vec3(0.0f 1.0f
    0.0f));
floatrot=(-90.0f180.0f)*M_PI;modelMatrix=glm::rotate(modelMatrix rot glm::vec3(1 0
    0));
设置正态矩阵glm::mat4normalMatrix=glm::transpose(glm::inverse(viewMatrix

将矩阵传递到GPU内存glUniformMatrix4fv(nMatID 1 GL_FALSE glm::value_ptr
(
    glUniformMatrix4fv(pMatID, 1, GL_FALSE, glm::value_ptr(projMatrix←
        ));
    glUniformMatrix4fv(vMatID, 1, GL_FALSE, glm::value_ptr(viewMatrix←
        ));
    glUniformMatrix4fv(mMatID, 1, GL_FALSE, glm::value_ptr(←
        modelMatrix));

设置此对象的材质glm::vec3kd(0.2 0.2 1.0);
glm::vec3ka=kd*0.15f;glm::vec3ks(1.0 1.0
1.0);floatphongExp=32.0;

glUniform3fv(kaID, 1, glm::value_ptr(ka));
glUniform3fv(kdID, 1, glm::value_ptr(kd));
glUniform3fv(ksID, 1, glm::value_ptr(ks));
glUniform1f(phongExpID, phongExp);

处理对象并注意modelData。size()保存顶点的数量，而不是三角形的数量！glBi
ndVertexArray (VAO) ;glDrawArrays (GL_TRIANGLES, 0, modelData。大小
());glBindVertexArray(0);

glUseProgram( 0 );

```



17.14.1 Instancing Models

Instancing with OpenGL is implemented differently than instancing with the ray tracer. With the ray tracer, rays are inversely transformed into the local space of the object using the model transform matrix. With OpenGL, instancing is performed by loading a single copy of the object as a vertex array object (with associated vertex buffer objects), and then reusing the geometry as needed. Like the ray tracer, only a single object is loaded into memory, but many may be rendered.

Modern OpenGL nicely supports this style of instancing because vertex shaders can (and must) compute the necessary transformations to transform vertices into clip coordinates. By writing generalized shaders that embed these transformations, such as presented with the Blinn-Phong vertex shader, models can be re-rendered with the same underlying local geometry. Different material types and transforms can be queried from higher-level class structures to populate the uniform variables passed from host to device each frame. Animations and interactive control are also easily created as the model transforms can change over time across the the display loop iteration. Figures 17.8 and 17.9 use the memory footprint of one dragon, yet render three different dragon models to the screen.

17.15 Texture Objects

Textures are an effective means to manipulate visual effects with OpenGL shaders. They are used extensively with many hardware-based graphics algorithms and OpenGL supports them natively with *Texture objects*. Like the previous OpenGL concepts, texture objects must be allocated and initialized by copying data on the host to the GPU memory and setting OpenGL state. Texture coordinates are often integrated into the vertex buffer objects and passed as vertex attributes to shader programs. Fragment shaders typically perform the texture lookup function using interpolated texture coordinate passed from the vertex shaders.

Textures are rather simple to add to your code if you already have working shader and vertex array objects. The standard OpenGL techniques for creating objects on the hardware are used with textures. However, the source of the texture data must first be determined. Data can either be loaded from a file (e.g., PNG, JPG, EXR, or HDR image file formats) or generated procedurally on the host (and even on the GPU). After the data is loaded into host memory, the data is copied to GPU memory, and optionally, OpenGL state associated with textures can be set. OpenGL texture data is loaded as a linear buffer of memory containing the data used for textures. Texture lookups on the hardware can be 1D, 2D, or 3D queries. Regardless of the texture dimension query, the data is loaded onto the



17.14.1 Instancing Models

使用OpenGL实例化的实现方式与使用光线跟踪器实例化的实现方式不同。使用射线追踪器，射线使用模型变换矩阵逆变换到对象的局部空间。使用OpenGL，通过将对象的单个副本加载为顶点数组对象（具有关联的顶点缓冲区对象），然后根据需要重用几何来执行实例化。与光线追踪器一样，只有一个对象被加载到内存中，但许多对象可能被渲染。

现代OpenGL很好地支持这种实例化风格，因为顶点着色器可以（并且必须）计算将顶点转换为剪辑坐标的必要转换。通过编写嵌入这些转换的广义着色器（如Blinn-Phong顶点着色器），可以使用相同的底层局部几何重新渲染模型。可以从更高级别的类结构中查询不同的材质类型和变换，以填充每帧从主机传递到设备的统一变量。动画和交互式控制也很容易创建，因为模型转换可以随时间变化的显示循环迭代。图17.8和17.9使用了一条龙的内存占用，但在屏幕上呈现了三个不同的龙模型。

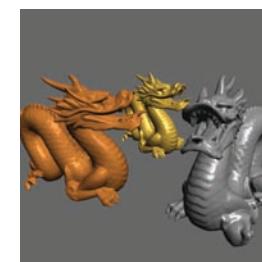


Figure 17.8. The results of running the Blinn-Phong shader program on the three dragons using uniform variables to specify material properties and transformations.

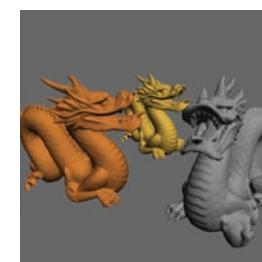
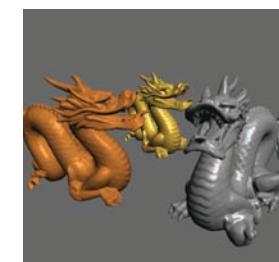


Figure 17.9. Setting the uniform variable $k_s = (0, 0, 0)$ in the Blinn-Phong shader program produces Lambertian shading.

17.15 纹理对象

纹理是使用OpenGL着色器操作视觉效果的有效手段。它们被广泛用于许多基于硬件的图形算法，OpenGL在纹理对象中原生支持它们。与之前的OpenGL概念一样，必须通过将主机上的数据复制到GPU内存并设置OpenGL状态来分配和初始化纹理对象。纹理坐标通常集成到顶点缓冲区对象中，并作为顶点属性传递给着色器程序。片段着色器通常使用从顶点着色器传递的插值纹理坐标来执行纹理查找功能。



使用uniformvariables在三条龙上运行Blinn-Phong着色器程序以指定材质道具和变换的结果。

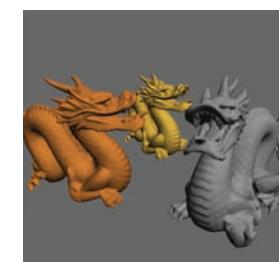


Figure 17.9. Set-BlinnPhong着色器程序produces中的均匀变量ks=(0 0 0)

如果您已经有工作着色器和顶点数组对象，则将纹理添加到代码中相当简单。用于在硬件上创建对象的标准OpenGL技术与纹理一起使用。但是，必须首先确定纹理数据的来源。数据可以从文件（例如PNG、JPG、EXR或HDR图像文件格式）加载，也可以在主机上（甚至在GPU上）以程序方式生成。在将数据加载到主机内存之后，数据被复制到GPU内存，并且可选地，可以设置与纹理相关联的OpenGL状态。OpenGL纹理数据作为内存的线性缓冲区加载，其中包含用于纹理的数据。硬件上的纹理查找可以是一维、二维或三维查询。无论纹理维度查询如何，数据都会加载到

memory in the same way, using linearly allocated memory on the host. In the following example, the process of loading data from an image file (or generating it procedurally) is left to the reader, but variable names are provided that match what might be present if an image is loaded (e.g., imgData, imgWidth, imgHeight).

```
float *imgData = new float[ imgHeight * imgWidth * 3 ];
...
GLuint texID;
 glGenTextures(1, &texID);
 glBindTexture(GL_TEXTURE_2D, texID);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, imgWidth, imgHeight, 0,
             GL_RGB, GL_FLOAT, imgData);
 glBindTexture(GL_TEXTURE_2D, 0);

delete [] imgData;
```

The example presented here highlights how to set up and use basic 2D OpenGL textures with shader programs. The process for creating OpenGL objects should be familiar by now. A handle (or ID) must be generated on the device to refer to the texture object (e.g., in this case, `texID`). The id is then bound to allow any subsequent texture state operations to affect the state of the texture. A fairly extensive set of OpenGL texture state and parameters exist that affect texture coordinate interpretation and texture lookup filtering. Various texture targets exist with graphics hardware. In this case, the texture target is specified as `GL_TEXTURE_2D` and will appear as the first argument in the texture-related functions. For OpenGL this particular texture target implies that texture coordinates will be specified in a device normalized manner (i.e., in the range of [0, 1]). Moreover, texture data must be allocated so that the width and height dimensions are powers of two (e.g., 512×512 , 1024×512 , etc.). Texture parameters are set for the currently bound texture by calling `glTexParameter`. This signature for this function takes on a variety of forms depending on the types of data being set. In this case, texture coordinates will be clamped by the hardware to the explicit range [0, 1]. The minifying and magnifying filters of OpenGL texture objects are set to use linear filtering (rather than nearest neighbor - `GL_NEAREST`) automatically when performing texture lookups. Chapter 11 provides substantial details on texturing, including details about the filtering that can occur with texture lookups. Graphics hardware can perform many of these operations automatically by setting the associated texture state.

Finally, the call to `glTexImage2D` performs the host to device copy for the texture. There are several arguments to this function, but the overall operation is to

内存以同样的方式，在主机上使用线性分配的内存。在以下示例中，从图像文件加载数据（或以程序方式生成数据）的过程留给读取器，但提供的变量名称与加载图像时可能存在的内容相匹配（例如，`imgData`, `imgWidth`, `imgHeight`）。

```
float*imgData=newfloat[imgHeight*imgWidth*3];..GLuinttexID;glGenTextures(1,&texID);glBindTexture(GL_TEXTURE_2D,texID);glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,imgWidth,imgHeight,0,GL_RGB,GL_FLOAT,imgData);glBindTexture(GL_TEXTURE_2D,0);

delete [] imgData;
```

这里介绍的示例重点介绍了如何设置和使用基本的2dOpenGL纹理与着色程序。创建OpenGL对象的过程现在应该很熟悉了。必须在设备上生成句柄（或ID）以重新绑定到纹理对象（例如，在这种情况下，`texID`）。然后，该id被绑定以允许任何后续纹理状态操作影响纹理的状态。存在一组相当广泛的OpenGL纹理状态和参数，它们影响纹理坐标解释和纹理查找过滤。图形硬件存在各种纹理tar。在这种情况下，纹理目标被指定为GL纹理2D，并将作为纹理相关函数中的第一个参数出现。对于OpenGL来说，这个特定的纹理目标意味着纹理坐标将以设备标准化的方式指定（即在[0 1]的范围内）。此外，必须分配纹理数据，使宽度和高度尺寸为2的幂（例如， 512×512 , 1024×512 等）。通过调用`glTexParameter`为当前绑定的纹理设置纹理参数。此函数的签名具有多种形式，具体取决于所设置的数据类型。在这种情况下，纹理坐标将被硬件钳制到显式范围[0 1]。OpenGL纹理对象的缩小和放大过滤器设置为在执行纹理查找时自动使用线性过滤（而不是最近邻GL最近）。第11章提供了关于纹理化的大量细节，包括关于纹理查找可能发生的过滤的细节。图形硬件可以通过设置关联的纹理状态自动执行许多这些操作。

最后，对`glTexImage2D`的调用执行纹理的主机到设备复制。这个函数有几个参数，但总体操作是

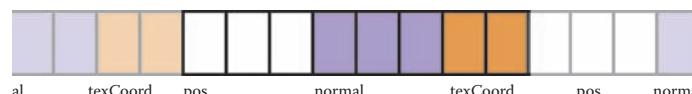


Figure 17.10. Data layout after adding the texture coordinate to the vertex buffer. Each block represents a GLfloat, which is 4 bytes. The position is encoded as a white block, the normals as purple, and the texture coordinates as orange.

allocate space on the graphics card (e.g., *imageWidth X imgHeight*) of three floats (7th and 8th arguments: GL_RGB and GL_FLOAT) and copy the linear texture data to the hardware (e.g., *imgData* pointer). The remaining arguments deal with setting the mipmap level of detail (2nd argument), specifying the internal format (e.g., 3rd argument's GL_RGB) and whether the texture has a border or not (6th argument). When learning OpenGL textures it is safe to keep these as the defaults listed here. However, the reader is advised to learn more about mipmaps and the potential internal formats of textures as more advanced graphics processing is required.

Texture object allocation and initialization happens with the code above. Additional modifications must be made to vertex buffers and vertex array objects to link in the correct texture coordinates with the geometric description. Following the previous examples, the storage for texture coordinates is a straightforward modification to the vertex data structure:

```
struct vertexData
{
    glm::vec3 pos;
    glm::vec3 normal;
    glm::vec2 texCoord;
};
```

As a result, the vertex buffer object will increase in size and the interleaving of texture coordinates will require a change to the stride in the vertex attribute specification for the vertex array objects. Figure 17.10 illustrates the basic interleaving of data within the vertex buffer.

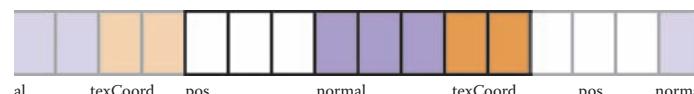
```
glBindBuffer(GL_ARRAY_BUFFER, m_triangleVBO[0]);

glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), 0);

glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (const GLvoid *)12);

glEnableVertexAttribArray(2);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (const GLvoid *)24);

glBindVertexArray(0);
```



将纹理坐标添加到顶点缓冲区后的数据布局。每个块代表一个GLfloat，它是4个字节。位置编码为白色块，法线为紫色，纹理坐标为橙色。

在三个浮点数（第7和第8个参数：GLRGB和GLfloat）的图形卡（例如，*imageWidthXimgHeight*）上分配空间，并将线性纹理数据复制到硬件（例如，*imgData*指针）。其余的参数处理设置mipmap细节级别（第二个参数），指定内部格式（例如，第三个参数的GLRGB）以及纹理是否有边框（第六个参数）。学习OpenGL纹理时，可以安全地保留这些作为此处列出的默认值。但是，建议读者更多地了解mipmap和纹理的潜在内部格式，因为需要更高级的图形处理。

纹理对象分配和初始化与上面的代码发生。必须对顶点缓冲区和顶点数组对象进行特殊修改，以便将正确的纹理坐标与几何描述联系起来。按照前面的示例，纹理坐标的存储是对顶点数据结构的直接修改：

```
struct vertexData
{
    glm::vec3 pos;
    glm::vec3 normal;
    glm::vec2 texCoord;
};
```

因此，顶点缓冲区对象的大小将增加，纹理坐标的交错将需要更改顶点数组对象的顶点属性规范中的步幅。图17.10说明了顶点缓冲区内数据的基本交织。

```
glBindBuffer(GL_ARRAY_BUFFER, m_triangleVBO[0]);

glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), 0);

glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (const GLvoid *)12);

glEnableVertexAttribArray(2);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (const GLvoid *)24);

glBindVertexArray(0);
```

With the code snippet above, the texture coordinates are placed at vertex attribute location 2. Note the change in size of the texture coordinate's size (e.g., 2nd argument of `glVertexAttribPointer` is 2 for texture coordinates to coincide with the `vec2` type in the structure). At this point, all initialization will have been completed for the texture object.

The texture object must be enabled (or bound) prior to rendering the vertex array object with your shaders. In general, graphics hardware allows the use of multiple texture objects when executing a shader program. In this way, shader programs can apply sophisticated texturing and visual effects. Thus, to bind a texture for use with a shader, it must be associated to one of potentially many *texture units*. Texture units represent the mechanism by which shaders can use multiple textures. In the sample below, only one texture is used so texture unit 0 will be made active and bound to our texture.

The function that activates a texture unit is `glActiveTexture`. Its only argument is the texture unit to make active. It is set to `GL_TEXTURE0` below, but it could be `GL_TEXTURE1` or `GL_TEXTURE2`, for instance, if multiple textures were needed in the shader. Once a texture unit is made active, a texture object can be bound to it using the `glBindTexture` call.

```
glUseProgram(shaderID);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texID);
glUniform1i(texUnitID, 0);

 glBindVertexArray(VAO);
 glDrawArrays(GL_TRIANGLES, 0, 3);
 glBindVertexArray(0);

 glBindTexture(GL_TEXTURE_2D, 0);

glUseProgram(0);
```

Most of the code above should be logical extensions to what you've developed thus far. Note the call to `glUniform` prior to rendering the vertex array object. In modern graphics hardware programming, shaders perform the work of texture lookups and blending, and therefore, must have data about which texture units hold the textures used in the shader. The active texture units are supplied to shaders using uniform variables. In this case, 0 is set to indicate that the texture lookups will come from texture unit 0. This will be expanded upon in the following section.

使用上面的代码片段，纹理坐标放置在顶点属性位置2。请注意纹理坐标大小的变化（例如，对于纹理坐标与结构中的`vec2`类型一致，`glVertexAttribPointer`的第二个argument为2）。此时，纹理对象的所有初始化都将完成。

在使用着色器渲染顶点数组对象之前，必须启用（或绑定）纹理对象。一般而言，图形硬件允许在执行着色器程序时使用多个纹理对象。通过这种方式，着色器程序可以应用复杂的纹理和视觉效果。因此，要绑定纹理以与着色器一起使用，必须将其关联到潜在的许多纹理单元之一。纹理单位表示着色器可以使用多个纹理的机制。在下面的示例中，只使用一个纹理，因此纹理单元0将被激活并绑定到我们的纹理。

激活纹理单元的函数是`glActiveTexture`。它唯一的参数是使活动的纹理单元。它在下面设置为`GL_TEXTURE0`，但它可以是`GL_TEXTURE1`或`GL_TEXTURE2`，例如，如果着色器中需要多个纹理。一旦一个纹理单元被激活，一个纹理对象可以绑定到它使用`glBindTexture`调用。

```
glUseProgram(shaderID);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texID);
glUniform1i(texUnitID, 0);

 glBindVertexArray(VAO);
 glDrawArrays(GL_TRIANGLES, 0, 3);
 glBindVertexArray(0);

 glBindTexture(GL_TEXTURE_2D, 0);

glUseProgram(0);
```

上面的大多数代码应该是对您迄今为止开发的内容的逻辑扩展。请注意在渲染顶点数组对象之前对`glUniform`的调用。在现代图形硬件编程中，着色器执行纹理查找和混合的工作，因此，必须有关于哪些纹理单元保存着色器中使用的纹理的数据。使用统一变量将活动纹理单元提供给着色器。在这种情况下，0被设置为表示纹理查找将来自纹理单元0。这将在下一节中展开。



17.15.1 Texture Lookup in Shaders

Shader programs perform the lookup and any blending that may be required. The bulk of that computation typically goes into the fragment shader, but the vertex shader often stages the fragment computation by passing the texture coordinate out to the fragment shader. In this way, the texture coordinates will be interpolated and afford per-fragment lookup of texture data.

Simple changes are required to use texture data in shader programs. Using the Blinn-Phong vertex shader provided previously, only three changes are needed:

1. The texture coordinates are a per-vertex attribute stored within the vertex array object. They are associated with vertex attribute index 2 (or location 2).

```
layout(location=2) in vec2 in_TexCoord;
```

2. The fragment shader will perform the texture lookup and will need an interpolated texture coordinate. This variable will be added as an output variable that gets passed to the fragment shader.

```
out vec2 tCoord;
```

3. Copy the incoming vertex attribute to the output variable in the main function.

```
// Pass the texture coordinate to the fragment shader
tCoord = in_TexCoord;
```

The fragment shader also requires simple changes. First, the incoming interpolated texture coordinates passed from the vertex shader must be declared. Also recall that a uniform variable should store the texture unit to which the texture is bound. This must be communicated to the shader as a *sampler* type. Samplers are a shading language type that allows the lookup of data from a single texture object. In this example, only one sampler is required, but in shaders in which multiple texture lookups are used, multiple sampler variables will be used. There are also multiple sampler types depending upon the type of texture object. In the example presented here, a GL_TEXTURE_2D type was used to create the texture state. The associated sampler within the fragment shader is of type sampler2D. The following two variable declarations must be added to the fragment shader:

```
in vec2 tCoord;
uniform sampler2D textureUnit;
```



17.15.1 着色器中的纹理查找

着色器程序执行查找和可能需要的任何混合。该计算的大部分通常进入片段着色器，但顶点着色器通常通过将纹理坐标传递给片段着色器来执行片段计算。通过这种方式，纹理坐标将被插值并提供纹理数据的每个片段查找。

在着色器程序中使用纹理数据需要进行简单的更改。使用之前提供的Blinn-Phong顶点着色器，只需要三个更改：

1. 纹理坐标是存储在顶点数组对象中的每个顶点属性。它们与顶点属性索引2（或位置2）相关联。

```
layout(location=2) in vec2 in_TexCoord;
```

2. 片段着色器将执行纹理查找，并且需要插值纹理坐标。此变量将作为传递给片段着色器的输出变量添加。

```
out vec2 tCoord;
```

3. 将传入顶点属性复制到主函数中的输出变量。

```
将纹理坐标传递给片段着色器tCoord=in_TexCoord;
```

片段着色器还需要简单的更改。首先，必须声明从顶点着色器传递的传入插值纹理坐标。还要记住，统一变量应存储纹理绑定到的纹理单元。这必须作为采样器类型传达给着色器。采样器是一种着色语言类型，允许从单个纹理对象中查找数据。在此示例中，只需要一个采样器，但在使用多个纹理查找的着色器中，将使用多个采样器变量。根据纹理对象的类型，还有多种采样器类型。在这里介绍的示例中，使用GL纹理2D类型来创建纹理状态。片段着色器内的关联采样器类型为sampler2D。必须将以下两个变量声明添加到片段着色器：

```
in vec2 tCoord;
uniform sampler2D textureUnit;
```

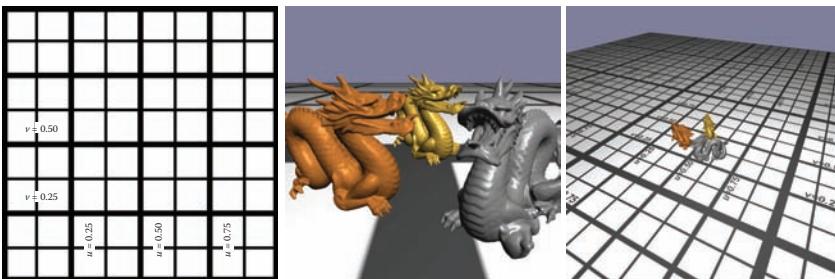


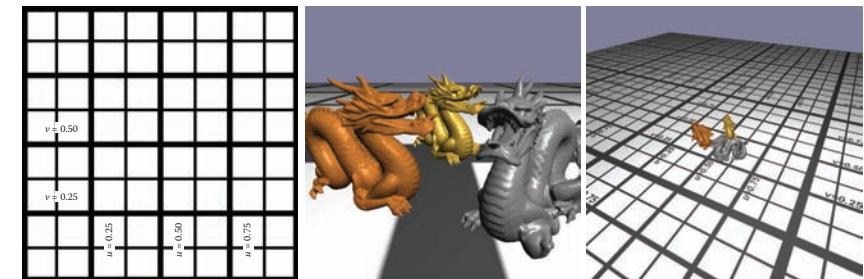
Figure 17.11. The left-most image shows the texture, a 1024×1024 pixel image. The middle image shows the scene with the texture applied using texture coordinates in the range of $[0, 1]$ so that only one image is tiled onto the ground plane. The right-most image modifies the texture parameters so that `GL_REPEAT` is used for `GL_TEXTURE_WRAP_S` and `GL_TEXTURE_WRAP_T` and the texture coordinate range from $[0, 5]$. The result is a tiled texture repeat five times in both texture dimensions.

The final modification goes into the main function of the fragment shader code. The texture is sampled using the GLSL `texture` lookup function and (in this case), replaces the diffuse coefficient of the geometry. The first argument to `texture` takes the sampler type which holds the texture unit to which the texture is bound. The second argument is the texture coordinate. The function returns a `vec4` type. In the code snippet below, no alpha values are utilized in the final computation so the resulting texture lookup value is component-wise selected to only the RGB components. The diffuse coefficient from the texture lookup is set to a `vec3` type that is used in the illumination equation.

```
vec3 kdTexel = texture(textureUnit, tCoord).rgb;
vec3 intensity = ka * Ia + kdTexel * light.intensity
    * max( 0.0, dot(n, l) ) + ks * light.intensity
    * pow( max( 0.0, dot(n, h) ), phongExp );
```

Figure 17.11 illustrates the results of using these shader modifications. The right-most image in the figure extends the example code by enabling texture tiling with the OpenGL state. Note that these changes are only done in host code and the shaders do not change. To enable this tiling, which allows for texture coordinates outside of the device normalized ranges, the texture parameters for `GL_TEXTURE_WRAP_S` and `GL_TEXTURE_WRAP_T` are changed from `GL_CLAMP` to `GL_REPEAT`. Additionally, the host code that sets the texture coordinates now ranges from $[0, 5]$.

As a side note, another texture target that may be useful for various applications is the `GL_TEXTURE_RECTANGLE`. Texture rectangle are unique texture objects that are not constrained with the power-of-two width and height image requirements and use non-normalized texture coordinates. Furthermore, they do



最左边的图像显示纹理，一个 1024×1024 像素的图像。中间的图像显示了使用 $[0, 1]$ 范围内的纹理坐标应用纹理的场景，因此只有一个图像被平铺到地平面上。最右边的图像修改纹理参数，使GL重复用于GL纹理包裹S和GL纹理包裹T，纹理坐标范围从 $[0, 5]$ 。结果是平铺纹理在两个纹理维度上重复五次。

最后的修改进入片段着色器代码的主要功能。使用GLSL纹理查找函数对纹理进行采样，并（在本例中）替换几何的漫射系数。纹理的第一个参数采用采样器类型，该采样器类型保存纹理绑定到的纹理单元。第二个参数是纹理坐标。函数返回`vec4`类型。在下面的代码片段中，在最终计算中没有使用alpha值，因此生成的纹理查找值将按组件选择为仅RGB组件。纹理查找中的漫射系数设置为照明方程中使用的`vec3`类型。

```
vec3 kdTexel=纹理 (textureUnit, tCoord) .rgb;强度=ka*Ia+kdT
exel*光。强度
* max( 0.0, dot(n, l) ) + ks * light.intensity
* pow( max( 0.0, dot(n, h) ), phongExp );
```

图17.11说明了使用这些着色器修改的结果。图中最右边的图像通过启用OpenGL状态的纹理平铺来扩展示例代码。请注意，这些更改仅在主机代码中完成，着色器不会更改。为了启用此平铺，允许纹理坐标超出设备标准化范围，GL纹理包裹S和GL纹理包裹T的纹理参数从GL钳位更改为GL重复。此外，设置纹理坐标的主机代码现在的范围为 $[0, 5]$ 。

作为一个侧面说明，另一个可能对各种应用有用的纹理目标是GL纹理矩形。纹理矩形是独特的纹理对象，不受宽度和高度图像要求的二次方约束，并使用非规范化的纹理坐标。此外，它们确实如此

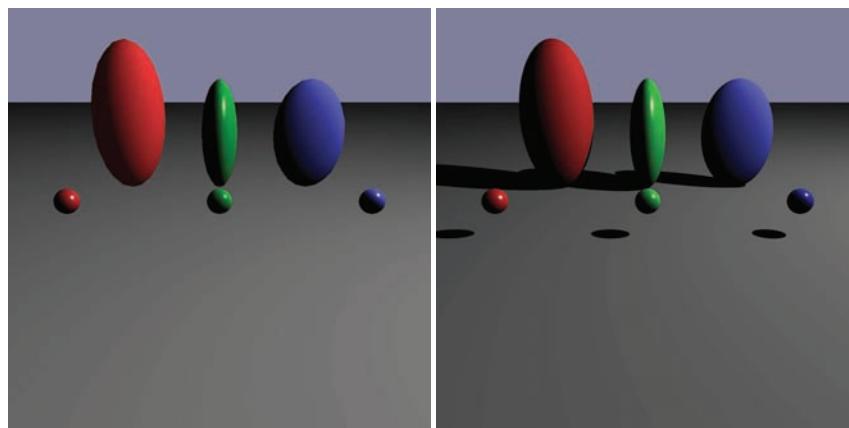
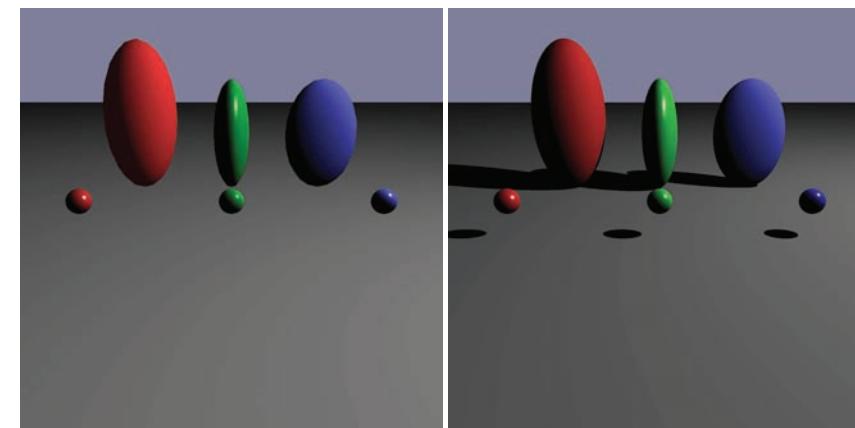


Figure 17.12. On the left, a single tessellated sphere is instanced six times using different model transforms to create this scene using the per-fragment shader program. The image on the right is rendered using a basic Whitted ray tracer. Notice the effect that shadows have on the perception of the scene. Per-fragment shading allows the specular highlight to be similar in both rendering styles.

not allow repeated tiling. If texture rectangles are used, shaders must reference them using the special sampler type: `sampler2DRect`.

17.16 Object-Oriented Design for Graphics Hardware Programming

As your familiarity with OpenGL increases, it becomes wise to encapsulate most of what is described in this chapter into class structures that can contain the model specific data and afford rendering of a variety of objects within the scene. For instance, in Figure 17.12, a single sphere is instanced six times to create the three ellipsoids and three spheres. Each model uses the same underlying geometry yet has different material properties and model transforms. If you've followed through the book and implemented the ray tracer, as detailed in Chapter 4, then it is likely that your implementation is based on a solid object-oriented design. That design can be leveraged to make developing a graphics hardware program with OpenGL easier. A typical ray tracer software architecture will include several classes that map directly into graphics hardware as well as software rasterization applications. The abstract base classes in the ray tracer that represent surfaces, materials, lights, shaders, and cameras can be adapted to initialize the graphics hardware state, update that state, and if necessary render the class data to the framebuffer. The interfaces to these virtual functions will likely need to be



在左侧，使用不同的模型变换实例化单个tessellated球体六次，以使用每片段着色器程序创建此场景。右边的图像是用一个基本的白色射线跟踪器渲染的。请注意阴影对场景感知的影响。每片段着色允许镜面高光在两种渲染样式中相似。

不允许重复平铺。如果使用纹理矩形，着色器必须使用特殊的采样器类型：`sampler2DRect`引用它们。

17.16 面向对象的设计 图形硬件编程

随着您对OpenGL的熟悉程度的增加，将本章中描述的大部分内容封装到类结构中变得明智，这些类结构可以包含特定于模型的数据，并提供场景中各种对象的渲染。例如，在图17.12中，一个球体被实例化六次，以创建三个椭球体和三个球体。每个模型使用相同的基础几何，但具有不同的材料属性和模型变换。如果您已经阅读了这本书并实现了射线跟踪器，如第4章所详述，那么您的实现很可能是基于可靠的面向对象设计的。这种设计可以用来使用OpenGL开发图形硬件程序更容易。典型的raytracer软件架构将包括几个直接映射到图形硬件以及软件光栅化应用程序的类。光线追踪器中表示曲面、材质、光源、着色器和摄像机的抽象基类可以调整为初始化图形硬件状态、更新该状态，并在必要时将类数据呈现到帧缓冲区。这些虚拟函数的接口可能需要

adapted to your specific implementation, but a first pass that extends the surface class design might resemble the following:

```
class surface
{
    virtual bool initializeOpenGL()
    virtual bool renderOpenGL(glm::mat4& Mp, glm::mat4& Mcam)
}
```

Passing the projection and view matrices to the render functions affords an indication for how these matrices are managed. These matrices would come from the camera classes which may be manipulated by interpreting keyboard, mouse, or joystick input. The initialization functions (at least for the surface derivatives) would contain the vertex buffer object and vertex array object allocation and initialization code. Aside from initiating the draw arrays for any vertex array objects, the render function would also need to activate shader programs and pass in the necessary matrices into the shaders, as illustrated previously in the dragon model example. As you work to integrate the image-order and object-order (hardware and software) algorithms into the same underlying data framework, a few software design challenges will pop up, mostly related to data access and organization. However, this is a highly useful exercise to become adept at software engineering for graphics programming and eventually gain solid experience hybridizing your rendering algorithms.

17.17 Continued Learning

This chapter was designed to provide an introductory glimpse into graphics hardware programming, influenced by the OpenGL API. There are many directions that your continued learning could go. Many topics, such as framebuffer objects, render to texture, environment mapping, geometry shaders, compute shaders, and advanced illumination shaders were not covered. These areas represent the next stages in learning about graphics hardware, but even within the areas covered, there are many directions that one could go to develop stronger graphics hardware understanding. Graphics hardware programming will continue to evolve and change. Interested readers should expect these changes and look to the specification documents for OpenGL and the OpenGL Shading Language for many more details about what OpenGL is capable of doing and how the hardware relates to those computations.

适用于您的特定实现，但扩展表面类设计的第一遍可能类似于以下内容：

```
类表面虚bool initializeOpenGL()虚bool renderOpenGL(g  
lm::mat4& Mp glm::mat4& Mcam)
```

将投影和视图矩阵传递给渲染函数可以间接地管理这些矩阵。这些矩阵来自相机类，可以通过解释键盘、鼠标或操纵杆输入来操作。初始化函数（至少对于曲面导数）将包含顶点缓冲区对象和顶点数组对象分配和初始化代码。除了为任何顶点数组对象启动绘制数组之外，渲染函数还需要激活着色器程序并将必要的矩阵传递到着色器中，如之前在dragon模型示例中所示。当您将图像顺序和对象顺序（硬件和软件）算法集成到同一个底层数据框架中时，会出现一些软件设计挑战，主要与数据访问和组织有关。然而，这是一个非常有用练习，成为熟练的软件工程图形编程，并最终获得坚实的经验混合您的渲染算法。

17.17 持续学习

本章旨在介绍受OpenGL API影响的图形硬件编程。有很多方向可以让你继续学习。许多主题，如帧缓冲对象，渲染到纹理，环境映射，几何着色器，计算着色器和高级照明着色器没有涵盖。这些领域代表了学习图形硬件的下一个阶段，但即使在所涵盖的领域内，也有许多方向可以发展更强的图形硬件理解。图形硬件编程将继续发展和变化。有兴趣的读者应该期待这些变化，并查看OpenGL和OpenGL着色语言的规范文档，了解更多关于OpenGL能够做什么以及硬件如何与这些计算相关的细节。



Frequently Asked Questions

- How do I debug shader programs?

On most platforms, debugging both vertex shaders and fragment shaders is not simple. However, more and more support is available through various drivers, operating system extensions, and IDEs to provide pertinent information to the developer. It still can be challenging, so use the shaders to visually debug your code. If nothing comes up on the screen, try rendering the normal vectors, the half vector, or anything that give you a sense for where the error might be (or not be). Figure 17.13 illustrates a normal shader in operation. If images do appear on your window, make sure they are what you expect (refer to Figure 17.14)!

Notes

There are many good resources available to learn more about the technical details involved with programming graphics hardware. A good starting point might be the OpenGL and GLSL specification documents. They are available for free online at the opengl.org website. These documents will provide complete details for all the different and emerging versions of OpenGL.

Exercises

The sections of this chapter are roughly organized to step students through the process of creating a modern OpenGL application. Some extra effort will be required to understand the details relating to setting up windows and OpenGL contexts. However, it should be possible to follow the sections for a set of weekly one hour labs:

1. **Lab 1: Basic code setup for OpenGL applications.** This includes installing the necessary drivers and related software such as GLM and GLFW. Students can then write code to open a window and clear the color buffers.
2. **Lab 2: Creating a shader.** Since a rudimentary shader is necessary to visualize the output in modern OpenGL, starting with efforts to create a very basic shader will go a long way. In this lab, or labs, students could build (or use provided) classes to load, compile, and link shaders into shader programs.

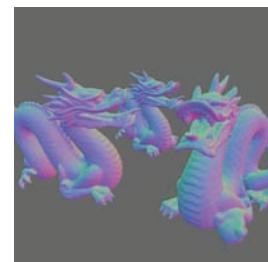


Figure 17.13. Applying the normal shader to a complex model for debugging purposes.

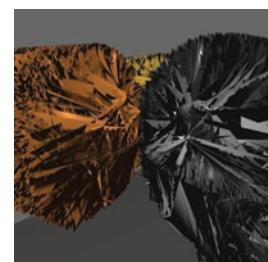


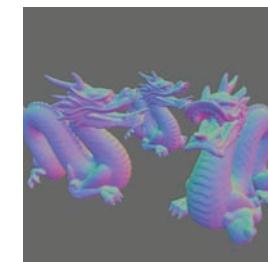
Figure 17.14. Visual debugging is important! Can you figure out what is wrong from the image or where to start debugging? When the incorrect stride is applied to the vertex array object, rendering goes awry.



常见问题

- 如何调试着色器程序？

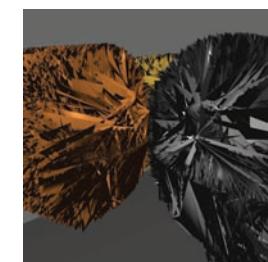
在大多数平台上，调试顶点着色器和片段着色器并不简单。但是，越来越多的支持可以通过各种驱动程序、操作系统扩展和IDE来向开发人员提供相关信息。这仍然是一个挑战，所以使用着色器直观地调试您的代码。如果屏幕上没有显示任何内容，请尝试渲染法向量，半向量或任何可以让您了解错误可能在哪里（或不是）的东西。图17.13显示了一个正常的着色器在运行。如果图像确实出现在您的窗口上，请确保它们是您所期望的（请参阅图17.14）！



将正常着色器应用于complex模型以调试pur姿势。

Notes

有很多很好的资源可以用来了解更多关于编程图形硬件所涉及的技术细节。一个很好的起点可能是OpenGL和GLSL规范文档。他们可以在网上免费获得opengl.org网站。这些文档将为OpenGL的所有不同和新兴版本提供完整的详细信息。



视觉窃听很重要！你能从图像中找出什么是错误的，或者从哪里开始去窃听吗？当不正确的步幅应用于顶点数组对象时，渲染会出错。

Exercises

本章的各个部分大致是为了引导学生完成创建现代OpenGL应用程序的过程。要了解有关设置windows和OpenGL上下文的详细信息，需要付出一些额外的努力。但是，应该可以遵循一组每周一小时的实验室的部分：

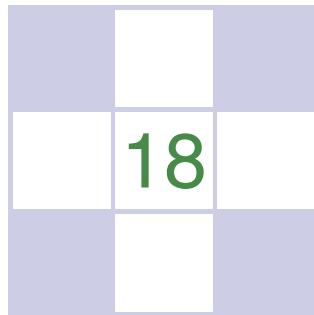
1. 实验室1：OpenGL应用程序的基本代码设置。这包括安装必要的驱动程序和相关软件，如GLM和GLFW。然后，学生可以编写代码来打开窗口并清除颜色缓冲区。
2. 实验室2：创建着色器。由于在现代OpenGL中可视化输出需要一个基本的着色器，因此从创建一个非常基本的着色器开始将有很长的路要走。在这个实验室或实验室中，学生可以构建（或使用提供的）类来加载，编译和链接着色器到着色器程序中。

3. Lab 3: Create a clip coordinate triangle and shade. Using the shader classes from the previous lab, students will add the passthrough shader and create simple geometry to render.
4. Lab 4: Introduce GLM. Start using GLM to generate projection matrices and viewing matrices for viewing more generalized, yet simple, scenes.
5. Lab 5: Use GLM for local transformations. Students can expand their working shader program to use local transforms, perhaps applying animations based on changing transforms.
6. Lab 6: Shader development. Develop the Lambertian or Blinn-Phong shaders.
7. Lab 7: Work with materials. Students can explore additional material properties and rendering styles with different shader programs.
8. Lab 8: Load 3D models. Using code to load OBJ files, students can further explore the capabilities of their graphics hardware including the limits of hardware processing for real-time applications.
9. Lab 9: Textures. Using PNG (or other formats), students can load images onto the hardware and practice a variety of texture-mapping strategies.
10. Lab 10: Integration with rendering code. If scene files are used to describe scenes for the ray tracer (or rasterizer), students' OpenGL code can be integrated into a complete rendering framework using common structures and classes to build a complete system.

This list is only a guide. In labs for my computer graphics course, students are provided material to get them started on the week's idea. After they get the basic idea working, the lab is completed once they add their spin or a creative exploration of the idea to their code. As students get familiar with graphics hardware programming, they can explore additional areas of interest, such as textures, render to texture, or more advanced shaders and graphics algorithms.

3. 实验室3：创建一个剪辑坐标三角形和阴影。使用之前实验室中的着色器类，学生将添加直通着色器并创建要渲染的简单几何体。
4. 实验室4：介绍GLM。开始使用GLM生成投影矩阵和查看矩阵，以查看更广义但简单的场景。
5. 实验室5：使用GLM进行局部转换。学生可以扩展他们的工作着色器程序以使用局部变换，也许根据变化的变换应用动画。
6. 实验室6：着色器开发。开发Lambertian或Blinn-Phong着色器。
7. 实验室7：使用材料。学生可以使用不同的着色器程序探索其他材质属性和渲染样式。
8. 实验室8：加载3D模型。使用代码加载OBJ文件，学生可以进一步探索其图形硬件的功能，包括实时应用程序的硬件处理限制。
9. 实验室9：纹理。使用PNG（或其他格式），学生可以将图像加载到硬件上并练习各种纹理映射策略。
10. 实验室10：与渲染代码集成。如果场景文件用于描述光线跟踪器（或光栅化器）的场景，学生的OpenGL代码可以使用通用结构和类集成到一个完整的渲染框架中，以构建一个完整的系统。

此列表只是一个指南。在我的计算机图形学课程的实验室里，学生们被提供材料，让他们开始本周的想法。在他们得到基本的想法工作后，一旦他们添加他们的旋转或创造性的探索想法到他们的代码，实验室就完成了。随着学生熟悉图形硬件编程，他们可以探索其他感兴趣的领域，如纹理、渲染到纹理或更高级的着色器和图形算法。



Light

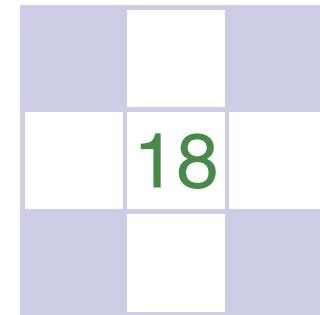
In this chapter, we discuss the practical issues of measuring light, usually called *radiometry*. The terms that arise in radiometry may at first seem strange and have terminology and notation that may be hard to keep straight. However, because radiometry is so fundamental to computer graphics, it is worth studying radiometry until it sinks in. This chapter also covers *photometry*, which takes radiometric quantities and scales them to estimate how much “useful” light is present. For example, a green light may seem twice as bright as a blue light of the same power because the eye is more sensitive to green light. Photometry attempts to quantify such distinctions.

18.1 Radiometry

Although we can define radiometric units in many systems, we use *SI* (International System of Units) units. Familiar SI units include the metric units of *meter* (*m*) and *gram* (*g*). Light is fundamentally a propagating form of energy, so it is useful to define the SI unit of energy, which is the *joule* (*J*).

18.1.1 Photons

To aid our intuition, we will describe radiometry in terms of collections of large numbers of *photons*, and this section establishes what is meant by a photon in this



Light

在本章中，我们讨论了测量光的实际问题，通常称为辐射测量。放射计量学中出现的术语起初可能看起来很奇怪，并且具有可能难以保持直截了当的术语和符号。然而，因为辐射测量是计算机图形学的基础，所以值得研究辐射测量直到它下沉。本章还介绍光度法，它采用辐射量并对其进行缩放，以估计存在多少“有用”光。例如，绿光可能看起来是相同功率的蓝光的两倍，因为眼睛对绿光更敏感。光度法试图量化这种差异。

18.1 Radiometry

虽然我们可以在许多系统中定义辐射单位，但我们使用SI（国际单位制）单位。熟悉的SI单位包括米（m）和克（g）的公制单位。光基本上是能量的传播形式，因此定义能量的SI单位是有用的，即焦耳（J）。

18.1.1 Photons

为了帮助我们的直觉，我们将用大量光子的集合来描述辐射测量，这一节确定了光子在这方面的含义

context. For the purposes of this chapter, a photon is a quantum of light that has a position, direction of propagation, and a wavelength λ . Somewhat strangely, the SI unit used for wavelength is *nanometer (nm)*. This is mainly for historical reasons, and $1 \text{ nm} = 10^{-9} \text{ m}$. Another unit, the *angstrom*, is sometimes used, and one nanometer is ten angstroms. A photon also has a speed c that depends only on the refractive index n of the medium through which it propagates. Sometimes the frequency $f = c/\lambda$ is also used for light. This is convenient because unlike λ and c , f does not change when the photon refracts into a medium with a new refractive index. Another invariant measure is the amount of energy q carried by a photon, which is given by the following relationship:

$$q = hf = \frac{hc}{\lambda}, \quad (18.1)$$

where $h = 6.63 \times 10^{-34} \text{ Js}$ is Plank's Constant. Although these quantities can be measured in any unit system, we will use SI units whenever possible.

18.1.2 Spectral Energy

If we have a large collection of photons, their total energy Q can be computed by summing the energy q_i of each photon. A reasonable question to ask is "How is the energy distributed across wavelengths?" An easy way to answer this is to partition the photons into bins, essentially histogramming them. We then have an energy associated with an interval. For example, we can count all the energy between $\lambda = 500 \text{ nm}$ and $\lambda = 600 \text{ nm}$ and have it turn out to be 10.2 J, and this might be denoted $q[500, 600] = 10.2$. If we divided the wavelength interval into two 50 nm intervals, we might find that $q[500, 550] = 5.2$ and $q[550, 600] = 5.0$. This tells us there was a little more energy in the short wavelength half of the interval [500, 600]. If we divide into 25 nm bins, we might find $q[500, 525] = 2.5$, and so on. The nice thing about the system is that it is straightforward. The bad thing about it is that the choice of the interval size determines the number.

A more commonly used system is to divide the energy by the size of the interval. So instead of $q[500, 600] = 10.2$ we would have

$$Q_\lambda[500, 600] = \frac{10.2}{100} = 0.12 \text{ J}(\text{nm})^{-1}.$$

This approach is nice, because the size of the interval has much less impact on the overall size of the numbers. An immediate idea would be to drive the interval size $\Delta\lambda$ to zero. This could be awkward, because for a sufficiently small $\Delta\lambda$, Q_λ will either be zero or huge depending on whether there is a single photon or no

上下文。在本章中，光子是具有位置、传播方向和波长 λ 的光量子。有些奇怪的是，用于波长的SI单位是纳米 (nm)。这主要是出于历史原因， $1 \text{ nm} = 10^{-9} \text{ m}$ 。有时使用另一个单位，埃，一纳米是十埃。光子还具有仅取决于其传播通过的介质的折射率 n 的速度 c 。有时频率 $f=c/\lambda$ 也用于光。这是方便的，因为与 λ 和 c 不同，当光子折射到具有新折射率的介质中时， f 不会改变。另一个不变度量是光子携带的能量 q 的量，其由以下关系式给出：

$$q = hf = \frac{hc}{\lambda}, \quad (18.1)$$

其中 $h=6.63 \times 10^{-34} \text{ Js}$ 是普朗克常数。虽然这些量可以在任何单位系统中测量，但我们将尽可能使用SI单位。

18.1.2 光谱能量

如果我们有一个大的光子集合，它们的总能量 Q 可以通过求和每个光子的能量 q_i 来计算。要问的一个合理的问题是"能量如何跨波长分布？"回答这个问题的一个简单方法是将光子分成垃圾箱，基本上将它们组织起来。然后我们有一个与间隔相关的能量。例如，我们可以计算 $\lambda=500 \text{ nm}$ 和 $\lambda=600 \text{ nm}$ 之间的所有能量，结果为10.2 J，这可能表示为 $q[500, 600]=10.2$ 。如果我们将波长间隔分成两个50 nm间隔，我们可能会发现 $q[500, 550]=5.0$ 和 $q[550, 600]=5.0$ 。这告诉我们在间隔的短波长一半[500, 600]中有更多的能量。如果我们分成25 nm的bin，我们可能会发现 $q[500, 525]=2.5$ ，以此类推。该系统的好处是它很简单。它的坏处是间隔大小的选择决定了数量。

更常用的系统是将能量除以间隔的大小。所以，而不是 $q[500, 600]=10.2$ ，我们会有

$$Q_\lambda[500, 600] = \frac{10.2}{100} = 0.12 \text{ J}(\text{nm})^{-1}.$$

这种方法很好，因为间隔的大小对数字的整体大小的影响要小得多。一个直接的想法是将间隔大小 $\Delta\lambda$ 驱动为零。这可能是尴尬的，因为对于足够小的 $\Delta\lambda$ ， Q_λ 将是零或巨大的，这取决于是否存在单个光子或没有

photon in the interval. There are two schools of thought to solve that dilemma. The first is to assume that $\Delta\lambda$ is small, but not so small that the quantum nature of light comes into play. The second is to assume that the light is a continuum rather than individual photons, so a true derivative $dQ/d\lambda$ is appropriate. Both ways of thinking about it are appropriate and lead to the same computational machinery. In practice, it seems that most people who measure light prefer small, but finite, intervals, because that is what they can measure in the lab. Most people who do theory or computation prefer infinitesimal intervals, because that makes the machinery of calculus available.

The quantity Q_λ is called *spectral energy*, and it is an *intensive* quantity as opposed to an *extensive* quantity such as energy, length, or mass. Intensive quantities can be thought of as density functions that tell the density of an extensive quantity at an infinitesimal point. For example, the energy Q at a specific wavelength is probably zero, but the spectral energy (energy density) Q_λ is a meaningful quantity. A probably more familiar example is that the population of a country may be 25 million, but the population at a point in that country is meaningless. However, the population *density* measured in people per square meter is meaningful, provided it is measured over large enough areas. Much like with photons, population density works best if we pretend that we can view population as a continuum where population density never becomes granular even when the area is small.

We will follow the convention of graphics where spectral energy is almost always used, and energy is rarely used. This results in a proliferation of λ subscripts if “proper” notation is used. Instead, we will drop the subscript and use Q to denote spectral energy. This can result in some confusion when people outside of graphics read graphics papers, so be aware of this standards issue. Your intuition about spectral energy might be aided by imagining a measurement device with a sensor that measures light energy Δq . If you place a colored filter in front of the sensor that allows only light in the interval $[\lambda - \Delta\lambda/2, \lambda + \Delta\lambda/2]$, then the spectral energy at λ is $Q = \Delta q / \Delta\lambda$.

18.1.3 Power

It is useful to estimate a rate of energy production for light sources. This rate is called *power*, and it is measured in *watts*, W , which is another name for *joules per second*. This is easiest to understand in a *steady state*, but because power is an intensive quantity (a density over time), it is well defined even when energy production is varying over time. The units of power may be more familiar, e.g., a 100-watt light bulb. Such bulbs draw approximately 100 J of energy each second. The power of the light produced will actually be less than 100 W because of

间隔中的光子。有两种思想流派来解决这一困境。首先是假设 $\Delta\lambda$ 很小，但不是那么小，以至于光的量子性质发挥作用。第二种是假设光是连续体而不是单个光子，因此真正的导数 $dQ/d\lambda$ 是合适的。这两种思考方式都是适当的，并导致相同的计算机器。在实践中，似乎大多数测量光线的人更喜欢小但有限的间隔，因为这是他们可以在实验室中测量的。大多数做理论或计算的人更喜欢无穷小的间隔，因为这使得微积分的机器可用。

数量 Q_λ 称为光谱能量，它是一个密集的数量，如对广泛的量，如能量，长度或质量。密集量可以被认为是密度函数，它告诉无限小点上广泛量的密度。例如，在特定波长处的能量 Q 可能为零，但是光谱能量(能量密度) q_λ 是有意义的量。一个可能更熟悉的例子是，一个国家的人口可能是2500万，但该国某个点的人口毫无意义。如果在足够大的面积上进行测量，那么以每平方米的人口密度是有意义的。就像光子一样，如果我们假装可以将人口视为一个连续体，即使面积很小，人口密度也不会变成颗粒状，那么人口密度就会效果最好。

我们将遵循图形的惯例，其中光谱能量几乎是很少使用的，而能量很少使用。如果使用“适当的”符号，这会导致下标的激增。相反，我们将删除下标并使用 Q 来记下光谱能量。当图形以外的人阅读图形文件时，这可能会导致一些混乱，所以要注意这个标准问题。您对光谱能量的直觉可能会通过想象具有测量光能量 Δq 的传感器的测量设备来帮助。如果在传感器前面放置一个只允许间隔 $[\lambda - \Delta\lambda/2, \lambda + \Delta\lambda/2]$ 中的光的彩色滤光片，那么 λ 处的光谱能量是 $Q = \Delta q / \Delta\lambda$ 。

18.1.3 Power

估计光源的能量产生速率是有用的。这个速率被称为功率，它以瓦特， W 测量，这是每秒焦耳的另一个名称。这在稳定状态下最容易理解，但由于功率是一个密集的数量（随时间变化的密度），因此即使能源产量随时间变化，它也是很好的定义。功率单位可能更熟悉，例如100瓦的灯泡。这种灯泡每秒消耗大约100J的能量。产生的光的功率实际上将小于100W，因为

heat loss, etc., but we can still use this example to help understand more about photons. For example, we can get a feel for how many photons are produced in a second by a 100 W light. Suppose the average photon produced has the energy of a $\lambda = 500 \text{ nm}$ photon. The frequency of such a photon is

$$f = \frac{c}{\lambda} = \frac{3 \times 10^8 \text{ ms}^{-1}}{500 \times 10^{-9} \text{ m}} = 6 \times 10^{14} \text{ s}^{-1}.$$

The energy of that photon is $hf \approx 4 \times 10^{-19} \text{ J}$. That means a staggering 10^{20} photons are produced each second, even if the bulb is not very efficient. This explains why simulating a camera with a fast shutter speed and directly simulated photons is an inefficient choice for producing images.

As with energy, we are really interested in *spectral power* measured in $\text{W}(\text{nm})^{-1}$. Again, although the formal standard symbol for spectral power is Φ_λ , we will use Φ with no subscript for convenience and consistency with most of the graphics literature. One thing to note is that the spectral power for a light source is usually a smaller number than the power. For example, if a light emits a power of 100 W evenly distributed over wavelengths 400 nm to 800 nm, then the spectral power will be $100 \text{ W}/400 \text{ nm} = 0.25 \text{ W}(\text{nm})^{-1}$. This is something to keep in mind if you set the spectral power of light sources by hand for debugging purposes.

The measurement device for spectral energy in the last section could be modified by taking a reading with a shutter that is open for a time interval Δt centered at time t . The spectral power would then be $\Phi = \Delta q / (\Delta t \Delta \lambda)$.

18.1.4 Irradiance

The quantity *irradiance* arises naturally if you ask the question “How much light hits this point?” Of course the answer is “none,” and again we must use a density function. If the point is on a surface, it is natural to use area to define our density function. We modify the device from the last section to have a finite ΔA area sensor that is smaller than the light field being measured. The spectral irradiance H is just the power per unit area $\Delta \Phi / \Delta A$. Fully expanded this is

$$H = \frac{\Delta q}{\Delta A \Delta t \Delta \lambda}. \quad (18.2)$$

Thus, the full units of irradiance are $\text{J m}^{-2} \text{s}^{-1} (\text{nm})^{-1}$. Note that the SI units for radiance include inverse-meter-squared for area and inverse-nanometer for wavelength. This seeming inconsistency (using both nanometer and meter) arises because of the natural units for area and visible light wavelengths.

热损失等。, 但我们仍然可以使用这个例子来帮助更多地了解光子。例如, 我们可以得到一个100w光在一秒钟内产生多少光子的感觉。假设产生的平均光子具有 $\lambda=500\text{nm}$ 光子的能量。这种光子的频率是

$$f = \frac{c}{\lambda} = \frac{3 \times 10^8 \text{ ms}^{-1}}{500 \times 10^{-9} \text{ m}} = 6 \times 10^{14} \text{ s}^{-1}.$$

该光子的能量为 $hf\approx4\times10^{-19}\text{J}$ 。这意味着惊人的 10^{20} 光子产生的每一秒, 即使灯泡是不是很有效率。这就解释了为什么用快速快门速度和直接模拟光子来模拟相机是产生图像的低效选择。

与能量一样, 我们对测量的光谱功率非常感兴趣。 $\text{W}(\text{nm})^{-1}$ 同样, 虽然光谱功率的正式标准符号是 Φ_λ , 但为了方便和与大多数图形文献保持一致, 我们将使用没有下标的 Φ 。需要注意的一点是, 光源的光谱功率通常比功率小。例如, 如果光在400nm到800nm的波长上均匀分布100W的功率, 那么光谱功率将为 $100\text{W}/400\text{nm}=0.25\text{W}(\text{nm})^{-1}$ 。如果您为调试目的手动设置光源的光谱功率, 请记住这一点。

最后一节光谱能量的测量装置可以通过以时间 t 为中心打开一个时间间隔 Δt 的快门进行修改。然后光谱功率为 $\Phi=\Delta q/(\Delta t \Delta \lambda)$ 。

18.1.4 Irradiance

如果你问这样一个问题：“多少光击中这一点？”答案当然是“无”，我们必须再次使用密度函数。如果点在表面上，使用面积来定义我们的密度函数是很自然的。我们从最后一节修改设备，使其具有小于被测量光场的有限 Δa 面积传感器。光谱辐照度 H 只是单位面积 $\Delta \Phi / \Delta A$ 的功率。完全扩展这是

$$H = \frac{\Delta q}{\Delta A \Delta t \Delta \lambda}. \quad (18.2)$$

因此, 完整的辐照度单位为 $\text{J m}^{-2} \text{s}^{-1} (\text{nm})^{-1}$ 。请注意, 辐射的SI单位包括面积的反米平方单位和波长的反纳米单位。这种看起来不一致(同时使用纳米和米)是因为面积和可见光波长的自然单位。

When the light is leaving a surface, e.g., when it is reflected, the same quantity as irradiance is called *radiant exitance*, E . It is useful to have different words for incident and exitant light, because the same point has potentially different irradiance and radiant exitance.

18.1.5 Radiance

Although irradiance tells us how much light is arriving at a point, it tells us little about the direction that light comes from. To measure something analogous to what we see with our eyes, we need to be able to associate “how much light” with a specific direction. We can imagine a simple device to measure such a quantity (Figure 18.1). We use a small irradiance meter and add a conical “baffler” which limits light hitting the counter to a range of angles with solid angle $\Delta\sigma$. The response of the detector is as follows:

$$\begin{aligned} \text{response} &= \frac{\Delta H}{\Delta\sigma} \\ &= \frac{\Delta q}{\Delta A \Delta\sigma \Delta t \Delta\lambda}. \end{aligned}$$

This is the spectral *radiance* of light traveling in space. Again, we will drop the “spectral” in our discussion and assume that it is implicit.

Radiance is what we are usually computing in graphics programs. A wonderful property of radiance is that it does not vary along a line in space. To see why this is true, examine the two radiance detectors both looking at a surface

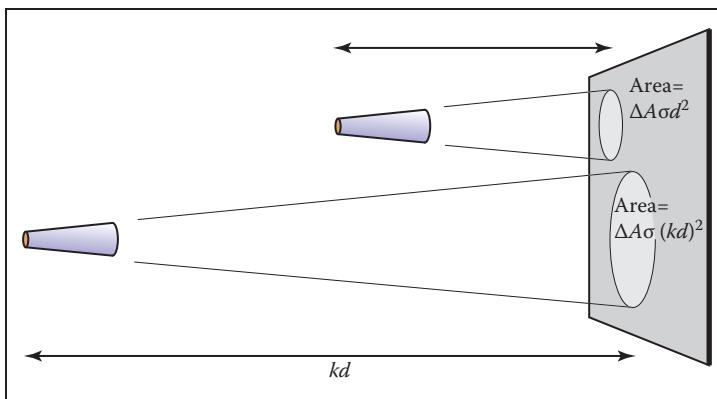


Figure 18.2. The signal a radiance detector receives does not depend on the distance to the surface being measured. This figure assumes the detectors are pointing at areas on the surface that are emitting light in the same way.

当光离开一个表面时，例如当它被反射时，与辐照度相同的量被称为辐射出射率。对入射光和出射光有不同的词汇是有用的，因为同一点具有潜在的不同的辐照度和辐射出射率。

18.1.5 Radiance

虽然辐照度告诉我们有多少光到达一个点，但它很少告诉我们光来自的方向。为了测量类似于我们用眼睛看到的东西，我们需要能够将“多少光”与特定方向相关联。我们可以想象一个简单的设备来测量这样的数量（图18.1）。我们使用一个小型辐照度计并添加一个圆锥形的“baffler”，它将光线击中计数器限制在具有实心角 $\Delta\sigma$ 的角度范围内。探测器的响应如下：

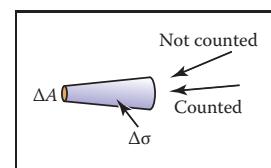


Figure 18.1. By adding a blinder that shows only a small solid angle $\Delta\sigma$ to the irradiance detector, we measure radiance.

$$\begin{aligned} \text{response} &= \frac{\Delta H}{\Delta\sigma} \\ &= \frac{\Delta q}{\Delta A \Delta\sigma \Delta t \Delta\lambda}. \end{aligned}$$

这是光在空间中行进的光谱辐射。同样，我们将在讨论中放弃“光谱”，并假设它是隐含的。

光辉是我们通常在图形程序中计算的。光辉的一个值得嘲笑的特性是它不会沿着空间的一条线变化。要了解为什么这是真的，检查两个辐射探测器都在看一个表面

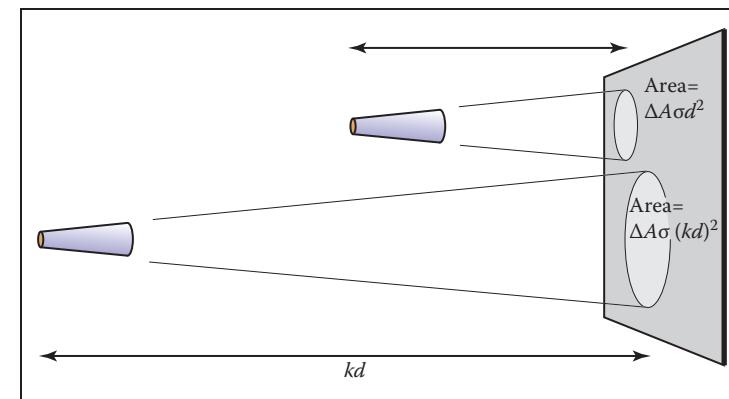
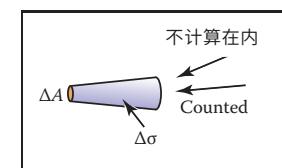


图18.2。辐射探测器接收的信号不依赖于到被测量表面的距离。该图假设探测器指向表面上以相同方式发光的区域。



通过向irradiance探测器添加仅显示小固体角 $\Delta\sigma$ 的blinder，我们测量radiance。

as shown in Figure 18.2. Assume the lines the detectors are looking along are close enough together that the surface is emitting/reflecting light “the same” in both of the areas being measured. Because the area of the surface being sampled is proportional to squared distance, and because the light reaching the detector is *inversely* proportional to squared distance, the two detectors should have the same reading.

It is useful to measure the radiance hitting a surface. We can think of placing the cone baffle from the radiance detector at a point on the surface and measuring the irradiance H on the surface originating from directions within the cone (Figure 18.3). Note that the surface “detector” is not aligned with the cone. For this reason we need to add a cosine correction term to our definition of radiance:

$$\begin{aligned} \text{response} &= \frac{\Delta H}{\Delta \sigma \cos \theta} \\ &= \frac{\Delta q}{\Delta A \cos \theta \Delta \sigma \Delta t \Delta \lambda}. \end{aligned}$$

As with irradiance and radiant exitance, it is useful to distinguish between radiance incident at a point on a surface and exitant from that point. Terms for these concepts sometimes used in the graphics literature are *surface radiance* L_s for the radiance of (leaving) a surface, and *field radiance* L_f for the radiance incident at a surface. Both require the cosine term, because they both correspond to the configuration in Figure 18.3:

$$\begin{aligned} L_s &= \frac{\Delta E}{\Delta \sigma \cos \theta} \\ L_f &= \frac{\Delta H}{\Delta \sigma \cos \theta}. \end{aligned}$$

Radiance and Other Radiometric Quantities

If we have a surface whose field radiance is L_f , then we can derive all of the other radiometric quantities from it. This is one reason radiance is considered the “fundamental” radiometric quantity. For example, the irradiance can be expressed as

$$H = \int_{\text{all } \mathbf{k}} L_f(\mathbf{k}) \cos \theta d\sigma.$$

This formula has several notational conventions that are common in graphics that make such formulae opaque to readers not familiar with them (Figure 18.4). First, \mathbf{k} is an incident direction and can be thought of as a unit vector, a direction, or a (θ, ϕ) pair in spherical coordinates with respect to the surface normal. The direction has a differential solid angle $d\sigma$ associated with it. The field radiance is potentially different for every direction, so we write it as a function $L(\mathbf{k})$.

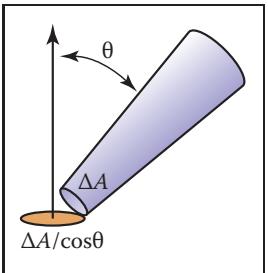


Figure 18.3. The irradiance at the surface as masked by the cone is smaller than that measured at the detector by a cosine factor.

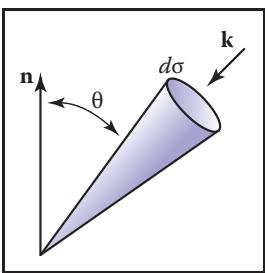


Figure 18.4. The direction \mathbf{k} has a differential solid angle $d\sigma$ associated with it.

如图18.2所示。假设探测器所看到的线足够接近，以至于表面在被测量的两个区域发射反射光“相同”。因为被采样的表面的面积与平方距离成正比，并且因为到达检测器的光与平方距离成反比，所以两个检测器应该具有相同的读数。

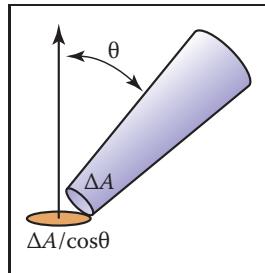


图18.3。被锥体掩盖的表面处的辐照度小于由余弦因子在检测器处测量的辐照度。

测量撞击表面的辐射是很有用的。我们可以考虑将辐射探测器上的锥形挡板放在表面上的一个点上，并从锥形内的方向测量表面上的辐照度 H （图18.3）。请注意，表面“检测器”未与锥体对齐。因此，我们需要在辐射的定义中添加一个余弦校正项：

$$\begin{aligned} \text{response} &= \frac{\Delta H}{\Delta \sigma \cos \theta} \\ &= \frac{\Delta q}{\Delta A \cos \theta \Delta \sigma \Delta t \Delta \lambda}. \end{aligned}$$

与辐照度和辐射出射率一样，区分入射在表面上某一点的辐射和从该点出射的辐射是有用的。这些条款

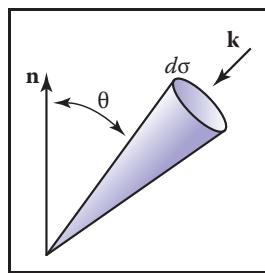
有时在图形文献中使用的概念是表面辐射 L_s 表示（离开）表面的辐射，而场辐射 L_f 表示入射到表面的辐射。两者都需要余弦项，因为它们都对应于图18.3中的配置：

$$\begin{aligned} L_s &= \frac{\Delta E}{\Delta \sigma \cos \theta} \\ L_f &= \frac{\Delta H}{\Delta \sigma \cos \theta}. \end{aligned}$$

辐射度及其他辐射量

如果我们有一个场辐射为 L_f 的表面，那么我们可以从它推导出所有其他辐射量。这就是辐射被认为是“基本”辐射量的原因之一。例如，辐照度可以表示为

$$H = \int_{\text{all } \mathbf{k}} L_f(\mathbf{k}) \cos \theta d\sigma.$$



方向 \mathbf{k} 具有与其相关联的微分固体角 $d\sigma$ 。

这个公式有几个符号约定，这些约定在图形中很常见，使这些公式对不熟悉的读者不透明（图18.4）。首先， \mathbf{k} 是一个入射方向，可以被认为是一个单位矢量，一个方向，或一个 (θ, ϕ) 对在球面坐标相对于表面法线。方向具有与其相关联的微分固体角 $d\sigma$ 。每个方向的场辐射都可能不同，因此我们将其写为函数 $L(\mathbf{k})$ 。

As an example, we can compute the irradiance H at a surface that has constant field radiance L_f in all directions. To integrate, we use a classic spherical coordinate system and recall that the differential solid angle is

$$d\sigma \equiv \sin \theta \, d\theta \, d\phi,$$

so the irradiance is

$$\begin{aligned} H &= \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\frac{\pi}{2}} L_f \cos \theta \sin \theta \, d\theta \, d\phi \\ &= \pi L_f. \end{aligned}$$

This relation shows us our first occurrence of a potentially surprising constant π . These factors of π occur frequently in radiometry and are an artifact of how we chose to measure solid angles, i.e., the area of a unit sphere is a multiple of π rather than a multiple of one.

Similarly, we can find the power hitting a surface by integrating the irradiance across the surface area:

$$\Phi = \int_{\text{all } \mathbf{x}} H(\mathbf{x}) \, dA,$$

where \mathbf{x} is a point on the surface, and dA is the differential area associated with that point. Note that we don't have special terms or symbols for incoming versus outgoing power. That distinction does not seem to come up enough to have encouraged the distinction.

18.1.6 BRDF

Because we are interested in surface appearance, we would like to characterize how a surface reflects light. At an intuitive level, for any incident light coming from direction \mathbf{k}_i , there is some fraction scattered in a small solid angle near the outgoing direction \mathbf{k}_o . There are many ways we could formalize such a concept, and not surprisingly, the standard way to do so is inspired by building a simple measurement device. Such a device is shown in Figure 18.5, where a small light source is positioned in direction \mathbf{k}_i as seen from a point on a surface, and a detector is placed in direction \mathbf{k}_o . For every directional pair $(\mathbf{k}_i, \mathbf{k}_o)$, we take a reading with the detector.

Now we just have to decide how to measure the strength of the light source and make our reflection function independent of this strength. For example, if we replaced the light with a brighter light, we would not want to think of the surface as reflecting light differently. We could place a radiance meter at the point being

例如，我们可以计算在所有方向上具有恒定场辐射 L_f 的表面处的辐照度 H 。为了整合，我们使用经典的球面坐标系统，并回忆起微分固体角是

$$d\sigma \equiv \sin \theta \, d\theta \, d\phi,$$

所以辐照度是

$$\begin{aligned} H &= \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\frac{\pi}{2}} L_f \cos \theta \sin \theta \, d\theta \, d\phi \\ &= \pi L_f. \end{aligned}$$

这种关系向我们展示了我们第一次出现一个可能令人惊讶的常数 π 。 π 的这些因素在辐射测量中经常出现，并且是我们如何选择测量固体角度的伪影，即单位球体的面积是 π 的倍数而不是 1 的倍数。

同样，我们可以通过积分整个表面区域的辐照度来找到击中表面的功率：

$$\Phi = \int_{\text{all } \mathbf{x}} H(\mathbf{x}) \, dA,$$

其中 \mathbf{x} 是表面上的点， dA 是与该点相关的差分区域。请注意，我们没有传入和传出功率的特殊术语或符号。这种区别似乎不足以鼓励这种区别。

18.1.6 BRDF

因为我们对表面外观感兴趣，所以我们想描述表面如何反射光线。在直观的水平上，对于来自方向 \mathbf{k}_i 的任何入射光，在出射方向 \mathbf{k}_o 附近以小的实心角散射一些分数。我们有很多方法可以将这样一个概念形式化，毫不奇怪，这样做的标准方法是通过构建一个简单的测量设备来启发的。这种装置如图 18.5 所示，其中一个小光源被定位在方向 \mathbf{k}_i 上，如从一个表面上的一个点看到的，并且一个 detector 被放置在方向 \mathbf{k}_o 上。对于每个方向对 $(\mathbf{k}_i, \mathbf{k}_o)$ ，我们使用探测器进行读数。

现在我们只需要决定如何测量光源的强度，并使我们的反射功能独立于这个强度。例如，如果我们用更亮的光替换光，我们不希望将表面视为不同的反射光。我们可以在点上放置一个发光计

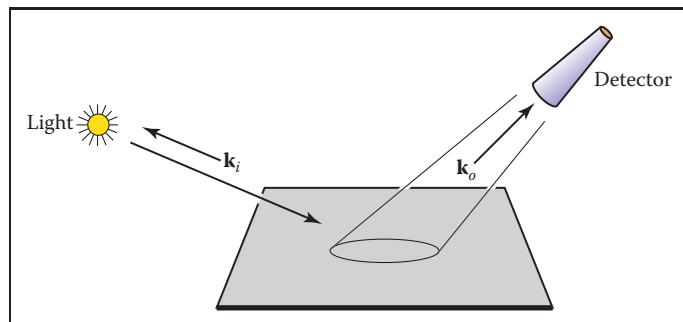


Figure 18.5. A simple measurement device for directional reflectance. The positions of light and detector are moved to each possible pair of directions. Note that both \mathbf{k}_i and \mathbf{k}_o point away from the surface to allow reciprocity.

illuminated to measure the light. However, for this to get an accurate reading that would not depend on the $\Delta\sigma$ of the detector, we would need the light to subtend a solid angle bigger than $\Delta\sigma$. Unfortunately, the measurement taken by our roving radiance detector in direction \mathbf{k}_o will also count light that comes from points outside the new detector's cone. So this does not seem like a practical solution.

Alternatively, we can place an irradiance meter at the point on the surface being measured. This will take a reading that does not depend strongly on subtleties of the light source geometry. This suggests characterizing reflectance as a ratio:

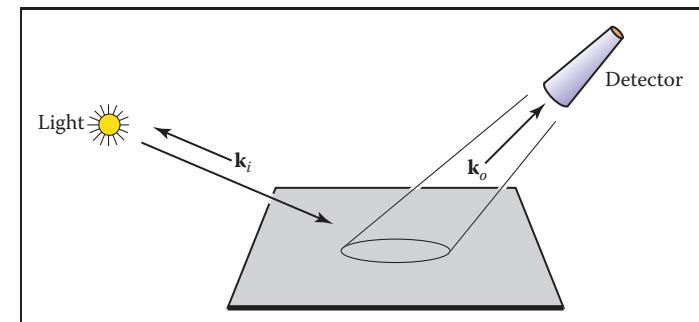
$$\rho = \frac{L_s}{H},$$

where this fraction ρ will vary with incident and exitant directions \mathbf{k}_i and \mathbf{k}_o , H is the irradiance for light position \mathbf{k}_i , and L_s is the surface radiance measured in direction \mathbf{k}_o . If we take such a measurement for all direction pairs, we end up with a 4D function $\rho(\mathbf{k}_i, \mathbf{k}_o)$. This function is called the *bidirectional reflectance distribution function* (BRDF). The BRDF is all we need to know to characterize the directional properties of how a surface reflects light.

Directional Hemispherical Reflectance

Given a BRDF, it is straightforward to ask, "What fraction of incident light is reflected?" However, the answer is not so easy; the fraction reflected depends on the directional distribution of incoming light. For this reason, we typically only set a fraction reflected for a fixed incident direction \mathbf{k}_i . This fraction is called the *directional hemispherical reflectance*. This fraction, $R(\mathbf{k}_i)$ is defined by

$$R(\mathbf{k}_i) = \frac{\text{power in all outgoing directions } \mathbf{k}_o}{\text{power in a beam from direction } \mathbf{k}_i}.$$



定向反射率的简单测量装置。光和检测器的位置移动到每个可能的方向对。请注意， \mathbf{k}_i 和 \mathbf{k}_o 都指向远离表面以允许互惠。

照明以测量光。然而，为了获得不依赖于探测器 $\Delta\sigma$ 的准确读数，我们需要光子化一个大于 $\Delta\sigma$ 的固体角。不幸的是，我们的粗纱辐射探测器在方向 \mathbf{k}_o 上的测量也将计算来自新探测器锥体之外的点的光。所以这似乎不是一个实际的解决方案。或者，我们可以在被测量表面的点上放置一个辐照度计。这将需要一个不强烈依赖于光源几何的微妙之处的读数。这表明将反射率表征为比率：

$$\rho = \frac{L_s}{H},$$

其中该分数 ρ 将随入射和出射方向 \mathbf{k}_i 和 \mathbf{k}_o 而变化， H 是光位置 \mathbf{k}_i 的辐照度， L_s 是在方向 \mathbf{k}_o 上测量的表面辐射度。如果我们对所有方向对进行这样的测量，我们最终得到一个4d函数 $\rho(\mathbf{k}_i, \mathbf{k}_o)$ 。该函数称为双向反射率分布函数(BRDF)。我们只需要知道BRDF来表征表面如何反射光的方向属性。

定向半球反射率

给出一个BRDF，可以很直接地问："什么部分的入射光被反射？"然而，答案并不那么容易：反射的分数取决于入射光的方向分布。由于这个原因，我们通常只设置一个固定的入射方向 \mathbf{k}_i 的分数。该分数称为定向半球反射率。该分数， $R(\mathbf{k}_i)$ 由下式定义

$$R(\mathbf{k}_i) = \frac{\text{所有输出方向的功率 } \mathbf{k}_o}{\text{来自方向 } \mathbf{k}_i \text{ 的光束中的功率}}.$$

Note that this quantity is between zero and one for reasons of energy conservation. If we allow the incident power Φ_i to hit on a small area ΔA , then the irradiance is $\Phi_i/\Delta A$. Also, the ratio of the incoming power is just the ratio of the radiant exitance to irradiance:

$$R(\mathbf{k}_i) = \frac{E}{H}.$$

The radiance in a particular direction resulting from this power is by the definition of BRDF:

$$\begin{aligned} L(\mathbf{k}_o) &= H\rho(\mathbf{k}_i, \mathbf{k}_o) \\ &= \frac{\Phi_i}{\Delta A}. \end{aligned}$$

And from the definition of radiance, we also have

$$L(\mathbf{k}_o) = \frac{\Delta E}{\Delta\sigma_o \cos\theta_o},$$

where E is the radiant exitance of the small patch in direction \mathbf{k}_o . Using these two definitions for radiance we get

$$H\rho(\mathbf{k}_i, \mathbf{k}_o) = \frac{\Delta E}{\Delta\sigma_o \cos\theta_o}.$$

Rearranging terms, we get

$$\frac{\Delta E}{H} = \rho(\mathbf{k}_i, \mathbf{k}_o)\Delta\sigma_o \cos\theta_o.$$

This is just the small contribution to E/H that is reflected near the particular \mathbf{k}_o .

To find the total $R(\mathbf{k}_i)$, we sum over all outgoing \mathbf{k}_o . In integral form this is

$$R(\mathbf{k}_i) = \int_{\text{all } \mathbf{k}_o} \rho(\mathbf{k}_i, \mathbf{k}_o) \cos\theta_o d\sigma_o.$$

Ideal Diffuse BRDF

An idealized diffuse surface is called *Lambertian*. Such surfaces are impossible in nature for thermodynamic reasons, but mathematically they do conserve energy. The Lambertian BRDF has ρ equal to a constant for all angles. This means the surface will have the same radiance for all viewing angles, and this radiance will be proportional to the irradiance.

If we compute $R(\mathbf{k}_i)$ for a Lambertian surface with $\rho = C$ we get

$$\begin{aligned} R(\mathbf{k}_i) &= \int_{\text{all } \mathbf{k}_o} C \cos\theta_o d\sigma_o \\ &= \int_{\phi_o=0}^{2\pi} \int_{\theta_o=0}^{\pi/2} C \cos\theta_o \sin\theta_o d\theta_o d\phi_o \\ &= \pi C. \end{aligned}$$

请注意，出于节能的原因，此数量介于零和一之间。如果我们允许入射功率 Φ_i 击中一个小区域 ΔA ，那么辐照度为 $\Phi_i/\Delta A$ 。此外，输入功率的比率只是辐射输出与辐照度的比率：

$$R(\mathbf{k}_i) = \frac{E}{H}.$$

由该功率产生的特定方向的辐射是由BRDF的定义：

$$\begin{aligned} L(\mathbf{k}_o) &= H\rho(\mathbf{k}_i, \mathbf{k}_o) \\ &= \frac{\Phi_i}{\Delta A}. \end{aligned}$$

从光辉的定义来看，我们也有

$$L(\mathbf{k}_o) = \frac{\Delta E}{\Delta\sigma_o \cos\theta_o},$$

其中 E 是小贴片在方向 \mathbf{k}_o 上的辐射力。使用这两个定义的光芒，我们得到

$$H\rho(\mathbf{k}_i, \mathbf{k}_o) = \frac{\Delta E}{\Delta\sigma_o \cos\theta_o}.$$

重新安排条款，我们得到

$$\frac{\Delta E}{H} = \rho(\mathbf{k}_i, \mathbf{k}_o)\Delta\sigma_o \cos\theta_o.$$

这只是在特定 \mathbf{k}_o 附近反映的对 E/H 的小贡献。

为了找到总 $R(\mathbf{k}_i)$ ，我们对所有输出的 \mathbf{k}_o 进行求和。在整体形式这是

$$R(\mathbf{k}_i) = \int_{\text{all } \mathbf{k}_o} \rho(\mathbf{k}_i, \mathbf{k}_o) \cos\theta_o d\sigma_o.$$

理想的弥漫性BRDF

理想化的漫反射表面称为Lambertian。由于热力学原因，这些表面在自然界中是不可能的，但在数学上它们确实节省了能量。朗伯的BRDF对于所有角度具有等于常数的 ρ 。这意味着表面对于所有视角将具有相同的辐射度，并且该辐射度将与辐照度成比例。

如果我们用 $\rho=C$ 计算朗伯表面的 $R(\mathbf{k}_i)$ ，我们得到

$$\begin{aligned} R(\mathbf{k}_i) &= \int_{\text{all } \mathbf{k}_o} C \cos\theta_o d\sigma_o \\ &= \int_{\phi_o=0}^{2\pi} \int_{\theta_o=0}^{\pi/2} C \cos\theta_o \sin\theta_o d\theta_o d\phi_o \\ &= \pi C. \end{aligned}$$

Thus, for a perfectly reflecting Lambertian surface ($R = 1$), we have $\rho = 1/\pi$, and for a Lambertian surface where $R(\mathbf{k}_i) = r$, we have

$$\rho(\mathbf{k}_i, \mathbf{k}_o) = \frac{r}{\pi}.$$

This is another example where the use of a steradian for the solid angle determines the normalizing constant and thus introduces factors of π .

18.2 Transport Equation

With the definition of BRDF, we can describe the radiance of a surface in terms of the incoming radiance from all different directions. Because in computer graphics we can use idealized mathematics that might be impractical to instantiate in the lab, we can also write the BRDF in terms of radiance only. If we take a small part of the light with solid angle $\Delta\sigma_i$ with radiance L_i and “measure” the reflected radiance in direction \mathbf{k}_o due to this small piece of the light, we can compute a BRDF (Figure 18.6). The irradiance due to the small piece of light is $H = L_i \cos \theta_i \Delta\sigma_i$. Thus the BRDF is

$$\rho = \frac{L_o}{L_i \cos \theta_i \Delta\sigma_i}.$$

This form can be useful in some situations. Rearranging terms, we can write down the part of the radiance that is due to light coming from direction \mathbf{k}_i :

$$\Delta L_o = \rho(\mathbf{k}_i, \mathbf{k}_o) L_i \cos \theta_i \Delta\sigma_i.$$

If there is light coming from many directions $L_i(\mathbf{k}_i)$, we can sum all of them. In integral form, with notation for surface and field radiance, this is

$$L_s(\mathbf{k}_o) = \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i.$$

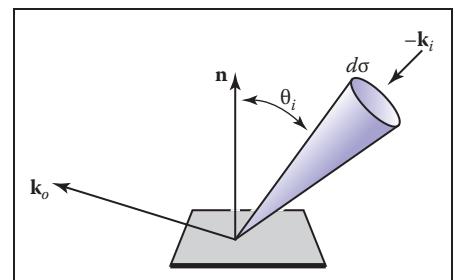


Figure 18.6. The geometry for the transport equation in its directional form.

因此，对于完全反射的朗伯表面 ($R=1$)，我们有 $\rho=1/\pi$ ，对于 $R(\mathbf{k}_i)=r$ 的朗伯表面，我们有

$$\rho(\mathbf{k}_i, \mathbf{k}_o) = \frac{r}{\pi}.$$

这是另一个例子，其中固体角使用steradian确定归一化常数并因此引入 π 的因子。

18.2 运输方程

通过BRDF的定义，我们可以根据来自所有不同方向的传入辐射来描述表面的辐射。因为在计算机图形学中，我们可以使用理想化的数学，在实验室中实例化可能是不切实际的，所以我们也只根据亮度来编写BRDF。如果我们用固体角 $\Delta\sigma_i$ 的一小部分光和辐射 L_i ，并“测量”由于这一小块光在方向 \mathbf{k}_o 上的反射辐射，我们可以计算BRDF（图18.6）。由于小片光的辐照度为 $H=L_i \cos \theta_i \Delta\sigma_i$ 。因此，BRDF是

$$\rho = \frac{L_o}{L_i \cos \theta_i \Delta\sigma_i}.$$

这种形式在某些情况下很有用。重新排列术语，我们可以记下由于来自方向 \mathbf{k}_i 的光而产生的辐射部分：

$$\Delta L_o = \rho(\mathbf{k}_i, \mathbf{k}_o) L_i \cos \theta_i \Delta\sigma_i.$$

如果有来自多个方向的光 $L_i(\mathbf{k}_i)$ ，我们可以对它们进行求和。在积分形式中，用表面和场辐射的符号表示，这是

$$L_s(\mathbf{k}_o) = \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i.$$

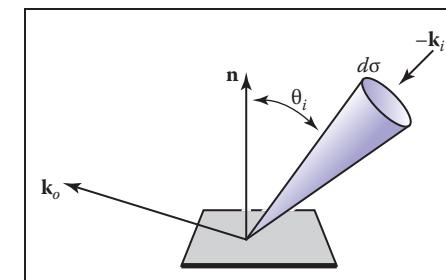


图18.6。运输方程的几何形状以其方向形式。



This is often called the *rendering equation* in computer graphics (Immel, Cohen, & Greenberg, 1986).

Sometimes it is useful to write the transport equation in terms of surface radiances only (Kajiya, 1986). Note, that in a closed environment, the field radiance $L_f(\mathbf{k}_i)$ comes from some surface with surface radiance $L_s(-\mathbf{k}_i) = L_f(\mathbf{k}_i)$ (Figure 18.7). The solid angle subtended by the point \mathbf{x}' in the figure is given by

$$\Delta\sigma_i = \frac{\Delta A' \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2},$$

where $\Delta A'$ the area we associate with \mathbf{x}' . Substituting for $\Delta\sigma_i$ in terms of $\Delta A'$ suggests the following transport equation:

$$L_s(\mathbf{x}, \mathbf{k}_o) = \int_{\text{all } \mathbf{x}' \text{ visible to } \mathbf{x}} \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_s(\mathbf{x}', \mathbf{x} - \mathbf{x}') \cos\theta_i \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2} dA'.$$

Note that we are using a non-normalized vector $\mathbf{x} - \mathbf{x}'$ to indicate the direction from \mathbf{x}' to \mathbf{x} . Also note that we are writing L_s as a function of position and direction.

The only problem with this new transport equation is that the domain of integration is awkward. If we introduce a visibility function, we can trade off complexity in the domain with complexity in the integrand:

$$L_s(\mathbf{x}, \mathbf{k}_o) = \int_{\text{all } \mathbf{x}'} \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_s(\mathbf{x}', \mathbf{x} - \mathbf{x}') v(\mathbf{x}, \mathbf{x}') \cos\theta_i \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2} dA',$$

where

$$v(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & \text{if } \mathbf{x} \text{ and } \mathbf{x}' \text{ are mutually visible,} \\ 0 & \text{otherwise.} \end{cases}$$

18.3 Photometry

For every spectral radiometric quantity there is a related *photometric quantity* that measures how much of that quantity is “useful” to a human observer. Given a spectral radiometric quantity $f_r(\lambda)$, the related photometric quantity f_p is

$$f_p = 683 \frac{\text{lm}}{\text{W}} \int_{\lambda=380 \text{ nm}}^{800 \text{ nm}} \bar{y}(\lambda) f_r(\lambda) d\lambda,$$

where \bar{y} is the *luminous efficiency function* of the human visual system. This function is zero outside the limits of integration above, so the limits could be 0 and ∞ and f_p would not change. The luminous efficiency function will be

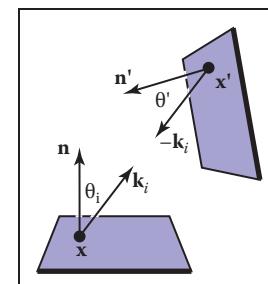


Figure 18.7. The light coming into one point comes from another point.



这通常被称为计算机图形学中的渲染方程 (Immell, Cohen, & Greenberg, 1986)。

有时只用表面辐射能写输运方程是有用的 (Kajiya, 1986)。请注意，在封闭环境中，场辐射 $L_f(\mathbf{k}_i)$ 来自某些表面，表面辐射 $L_s(-\mathbf{k}_i) = L_f(\mathbf{k}_i)$ (Figure 18.7)。由图中的点 \mathbf{x}' subtended 的实心角由下式给出

$$\Delta\sigma_i = \frac{\Delta A' \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2},$$

其中 $\Delta A'$ 我们与 \mathbf{x} 相关联的区域。用 $\Delta\sigma_i$ 替代 $\Delta A'$ 表示以下输运方程：

$$L_s(\mathbf{x}, \mathbf{k}_o) = \int_{\text{所有 } \mathbf{x}' \text{ 对 } \mathbf{x} \text{ 可见}} \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_s(\mathbf{x}', \mathbf{x} - \mathbf{x}') \cos\theta_i \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2} dA'.$$

请注意，我们使用非归一化向量 $\mathbf{x} - \mathbf{x}'$ 来指示从 \mathbf{x}' 到 \mathbf{x} 的方向。另请注意，我们正在编写 L_s 作为位置和方向的函数。

这个新的传输方程的唯一问题是 integration 的域是尴尬的。如果我们引入可见性函数，我们可以权衡域中的共性与被积体中的复杂性：

$$L_s(\mathbf{x}, \mathbf{k}_o) = \int_{\text{all } \mathbf{x}'} \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_s(\mathbf{x}', \mathbf{x} - \mathbf{x}') v(\mathbf{x}, \mathbf{x}') \cos\theta_i \cos\theta'}{\|\mathbf{x} - \mathbf{x}'\|^2} dA',$$

where

$$v(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & \text{如果 } \mathbf{x} \text{ 和 } \mathbf{x}' \text{ 是相互可见的} \\ 0 & \text{otherwise.} \end{cases}$$

18.3 Photometry

对于每个光谱辐射量，都有一个相关的光度量来衡量这个量对人类观察者“有用”的程度。给定光谱辐射量 $f_r(\lambda)$ ，相关光度量 f_p 为

$$f_p = 683 \frac{\text{lm}}{\text{W}} \int_{\lambda=380 \text{ nm}}^{800 \text{ nm}} \bar{y}(\lambda) f_r(\lambda) d\lambda,$$

其中 \bar{y} 为人类视觉系统的发光效率函数。此函数在上述积分限制之外为零，因此限制可能为 0, ∞ 和 f_p 不会改变。发光效率函数将是

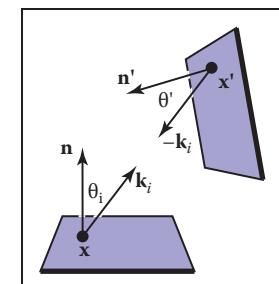


图18.7。进入一个点的光来自另一个点。

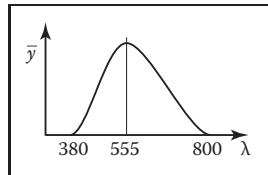


Figure 18.8. The luminous efficiency function versus wavelength (nm).

The photometric quantity that is most commonly used in graphics is *luminance*, the photometric analog of radiance:

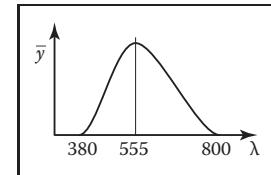
$$Y = 683 \frac{\text{lm}}{\text{W}} \int_{\lambda=380 \text{ nm}}^{800 \text{ nm}} \bar{y}(\lambda) L(\lambda) d\lambda.$$

The symbol Y for luminance comes from colorimetry. Most other fields use the symbol L ; we will not follow that convention because it is too confusing to use L for both luminance and spectral radiance. Luminance gives one a general idea of how “bright” something is independent of the adaptation of the viewer. Note that the black paper under noonday sun is subjectively darker than the lower luminance white paper under moonlight; reading too much into luminance is dangerous, but it is a very useful quantity for getting a quantitative feel for relative perceivable light output. The unit lm stands for *lumens*. Note that most light bulbs are rated in terms of the power they consume in watts, and the useful light they produce in lumens. More efficient bulbs produce more of their light where \bar{y} is large and thus produce more lumens per watt. A “perfect” light would convert all power into 555 nm light and would produce 683 lumens per watt. The units of luminance are thus $(\text{lm}/\text{W})(\text{W}/(\text{m}^2\text{sr})) = \text{lm}/(\text{m}^2\text{sr})$. The quantity one lumen per steradian is defined to be one *candela* (cd), so luminance is usually described in units cd/m^2 .

Frequently Asked Questions

- What is “intensity”?

The term *intensity* is used in a variety of contexts and its use varies with both era and discipline. In practice, it is no longer meaningful as a specific radiometric quantity, but it is useful for intuitive discussion. Most papers that use it do so in place of radiance.



的亮度效率函数versus波长(nm)。

在第19章中有更详细的讨论，但我们在讨论它的一般属性。主导常数是使定义与历史绝对光度量一致。

发光效率函数对所有波长都不同等敏感（图18.8）。对于低于380nm（紫外线范围）的波长，光对人类是不可见的，因此 \bar{y} 值为零。从380nm开始逐渐增加，直到 $\lambda=555\text{nm}$ 处达到峰值。这是一个纯粹的绿色光。然后，它逐渐减小，直到它在800nm处到达红外区域的边界。

在图形中最常用的光度量是亮度，亮度的光度模拟：

$$Y = 683 \frac{\text{lm}}{\text{W}} \int_{\lambda=380 \text{ nm}}^{800 \text{ nm}} \bar{y}(\lambda) L(\lambda) d\lambda.$$

亮度的符号 Y 来自比色法。大多数其他字段使用符号 L ；我们不会遵循该约定，因为将 L 用于亮度和光谱辐射太令人困惑。亮度给人一个关于“明亮”的东西是如何独立于观众的适应的一般概念。请注意，正午阳光下的黑纸主观上比月光下亮度较低的白纸暗；读太多的亮度是危险的，但它是一个非常有用的数量，以获得相对可感知的光输出的定量感觉。单位 lm 代表流明。请注意，大多数灯泡的额定功率是以瓦特为单位的，以及它们产生的有用光以流明为单位的。更高效的灯泡在 y 较大的地方产生更多的光，因此每瓦产生更多的流明。“完美”光将所有功率转换为555纳米光，每瓦产生683流明。亮度的单位因此是 $(\text{lm}/\text{W})(\text{W}/(\text{m}^2\text{sr}))=\text{lm}/(\text{m}^2\text{sr})$ 。每个steradian一个流明的数量被定义为一个坎德拉（ cd ），因此亮度通常以单位 cd/m^2 描述。

常见问题

- 什么是“强度”？

强度一词在各种上下文中使用，其使用随时代和学科而变化。在实践中，它作为特定的辐射量不再有意义，但它对于直观的讨论是有用的。大多数使用它的论文都是这样做的。



- What is “radiosity”?

The term *radiosity* is used in place of radiant exitance in some fields. It is also sometimes used to describe world-space light transport algorithms.

Notes

A common radiometric quantity not described in this chapter is *radiant intensity* (I), which is the spectral power per steradian emitted from an infinitesimal point source. It should usually be avoided in graphics programs because point sources cause implementation problems. A more rigorous treatment of radiometry can be found in *Analytic Methods for Simulated Light Transport* (Arvo, 1995). The radiometric and photometric terms in this chapter are from the *Illumination Engineering Society*'s standard that is increasingly used by all fields of science and engineering (American National Standard Institute, 1986). A broader discussion of radiometric and appearance standards can be found in *Principles of Digital Image Synthesis* (Glassner, 1995).

Exercises

1. For a diffuse surface with outgoing radiance L , what is the radiant exitance?
2. What is the total power exiting a diffuse surface with an area of 4 m^2 and a radiance of L ?
3. If a fluorescent light and an incandescent light both consume 20 watts of power, why is the fluorescent light usually preferred?



- 什么是“辐射度”?

在某些领域中 辐射度一词用来代替辐射.它有时也用于描述世界空间光传输算法。

Notes

本章中没有描述的一个常见辐射量是辐射强度 (I)，它是从无穷小点源发出的每个甾烷的光谱功率。在图形程序中通常应该避免，因为点源会导致实现问题。在模拟光传输的分析方法中可以找到更严格的辐射测量处理 (Arvo, 1995)。本章中的辐射和光度术语来自照明工程协会的标准，该标准越来越多地被所有科学和工程领域使用 (美国国家标准研究所, 1986)。有关辐射和外观标准的更广泛讨论可以在*Principles Of Digital Image Synthesis* (Glassner, 1995) 中找到。

Exercises

1. 对于具有出射辐射L的漫射表面，辐射辐射是什么？
2. 离开面积为 4m^2 且辐射度为L的漫射表面的总功率是多少？
3. 如果荧光灯和白炽灯都消耗20瓦的功率，为什么荧光灯通常是首选？



Erik Reinhard and Garrett Johnson

Color

Photons are the carriers of optical information. They propagate through media taking on properties associated with waves. At surface boundaries they interact with matter, behaving more as particles. They can also be absorbed by the retina, where the information they carry is transcoded into electrical signals that are subsequently processed by the brain. It is only there that a sensation of color is generated.

As a consequence, the study of color in all its guises touches upon several different fields: physics for the propagation of light through space, chemistry for its interaction with matter, and neuroscience and psychology for aspects relating to perception and cognition of color (Reinhard et al., 2008).

In computer graphics, we traditionally take a simplified view of how light propagates through space. Photons travel along straight paths until they hit a surface boundary and are then reflected according to a reflection function of some sort. A single photon will carry a certain amount of energy, which is represented by its wavelength. Thus, a photon will have only one wavelength. The relationship between its wavelength λ and the amount of energy it carries (ΔE) is given by

$$\lambda \Delta E = 1239.9,$$

where ΔE is measured in electron volts (eV).

In computer graphics, it is not very efficient to simulate single photons; instead large collections of them are simulated at the same time. If we take a very large number of photons, each carrying a possibly different amount of energy,



埃里克·莱因哈德和加勒特·约翰逊

Color

光子是光学信息的载体。它们通过具有与波相关属性的介质传播。在表面边界，它们与物质相互作用，表现得更像粒子。它们也可以被视网膜吸收，在那里它们携带的信息被转码成电信号，随后由大脑处理。只有在那里，才会产生色彩的感觉。

因此，对各种形式的颜色的研究涉及几个不同的领域：光通过空间传播的物理学，与物质相互作用的化学，以及与颜色感知和认知相关的神经科学和心理学 (Reinhard et al. 2008).

在计算机图形学中，我们传统上采取光如何通过空间传播的简化视图。光子沿着直线路径行进，直到它们到达表面边界，然后根据某种反射函数进行反射。单个光子将携带一定量的能量，其由其波长表示。因此，一个光子将只有一个波长。其波长 λ 与其携带的能量量 (ΔE) 之间的关系由下式给出

$$\lambda \Delta E = 1239.9,$$

其中 ΔE 以电子伏特 (eV) 测量。

在计算机图形学中，模拟单个光子的效率不是很高；相反，它们的大集合是在同一时间模拟的。如果我们需要大量的光子，每个光子携带的能量可能不同

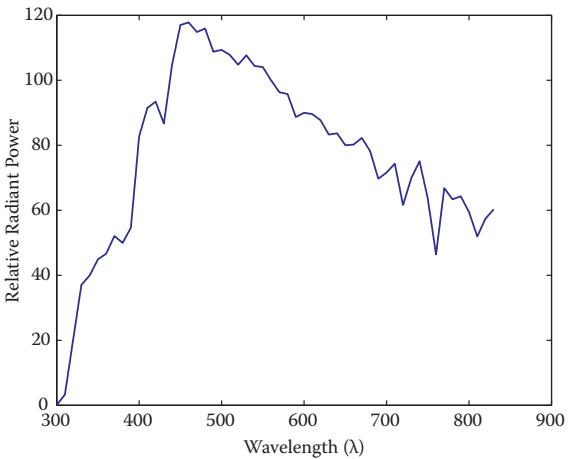


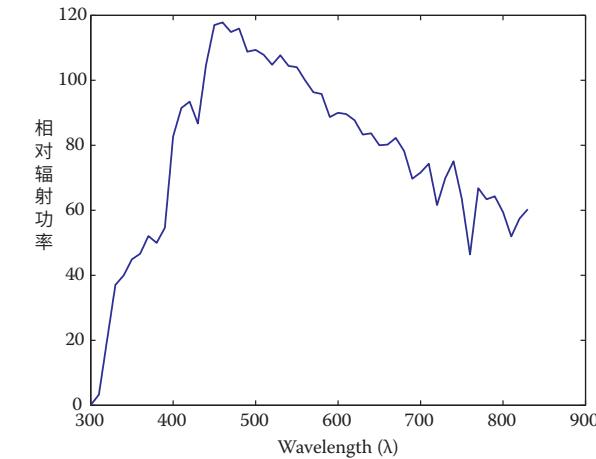
Figure 19.1. A spectrum describes how much energy is available at each wavelength λ , here measured as relative radiant power. This specific spectrum represents average daylight.

then together they represent a spectrum. A spectrum can be thought of as a graph where the number of photons is plotted against wavelength. Because two photons of the same wavelength carry twice as much energy as a single photon of that wavelength, this graph can also be seen as a plot of energy against wavelength. An example of a spectrum is shown in Figure 19.1. The range of wavelengths to which humans are sensitive is roughly between 380 and 800 nanometers (nm).

When simulating light, it would therefore be possible to trace rays that each carry a spectrum. A renderer that accomplishes this is normally called a *spectral renderer*. From preceding chapters, it should be clear that we are not normally going through the expense of building spectral renderers. Instead, we replace spectra with representations that typically use red, green, and blue components. The reason that this is possible at all has to do with human vision and will be discussed later in this chapter.

Simulating light by tracing rays takes care of the physics of light, although it should be noted that several properties of light, including, for instance, polarization, diffraction, and interference, are not modeled in this manner.

At surface boundaries, we normally model what happens with light by means of a reflectance function. These functions can be measured directly by means of *gonioreflectometers*, leading to a large amount of tabulated data, which can be more compactly represented by various different functions. Nonetheless, these reflectance functions are empirical in nature, i.e., they abstract away the chemistry that happens when a photon is absorbed and re-emitted by an electron. Thus, reflectance functions are useful for modeling in computer graphics, but do not



光谱描述了在每个波长 λ 处有多少能量可用，这里测量为相对辐射功率。该特定光谱代表平均日光。

然后它们一起代表一个光谱。光谱可以被认为是一个图表，其中光子的数量是根据波长绘制的。由于相同波长的两个光子携带的能量是该波长的单个光子的两倍，因此该图也可以看作是能量与波长的关系图。图19.1显示了光谱的一个例子。人类敏感的波长范围大致在380到800纳米 (nm) 之间。

因此，在模拟光线时，可以跟踪每个携带光谱的光线。完成此操作的渲染器通常称为光谱渲染器。从前面的章节中可以清楚地看出，我们通常不会花费构建光谱渲染器的费用。相反，我们将光谱替换为通常使用红色，绿色和蓝色分量的表示。这是可能的原因与人类的视觉有关，将在本章后面讨论。

通过跟踪射线模拟光需要注意光的物理特性，尽管应该注意的是，光的几个属性，包括，例如，极化，衍射和干涉，不是以这种方式建模的。

在表面边界，我们通常通过反射函数对光发生的情况进行建模。这些功能可以借助于测角仪直接测量，从而产生大量的表格数据，这些数据可以通过各种不同的功能更紧凑地表示。尽管如此，这些反射函数本质上是经验性的，即它们抽象出当光子被电子吸收并重新发射时发生的化学反应。因此，反射率函数对于计算机图形学中的建模是有用的，但不



offer an explanation as to why certain wavelengths of light are absorbed and others are reflected. We can therefore not use reflectance functions to explain why the light reflected off a banana has a spectral composition that appears to us as yellow. For that, we would have to study molecular orbital theory, a topic beyond the scope of this book.

Finally, when light reaches the retina, it is transcoded into electrical signals that are propagated to the brain. A large part of the brain is devoted to processing visual signals, part of which gives rise to the sensation of color. Thus, even if we know the spectrum of light that is reflected off a banana, we do not know yet why humans associate the term “yellow” with it. Moreover, as we will find out in the remainder of this chapter, our perception of color is vastly more complicated than it would seem at first glance. It changes with illumination, varies between observers, and varies within an observer over time.

In other words, the spectrum of light coming off a banana is perceived in the context of an environment. To predict how an observer perceives a “banana spectrum” requires knowledge of the environment that contains the banana as well as the observer’s environment. In many instances, these two environments are the same. However, when we are displaying a photograph of a banana on a monitor, then these two environments will be different. As human visual perception depends on the environment the observer is in, it may perceive the banana in the photograph differently from how an observer directly looking at the banana would perceive it. This has a significant impact on how we should deal with color and illustrates the complexities associated with color.

To emphasize the crucial role that human vision plays, we only have to look at the definition of color: “Color is the aspect of visual perception by which an observer may distinguish differences between two structure-free fields of view of the same size and shape, such as may be caused by differences in the spectral composition of the radiant energy concerned in the observation” (Wyszecki & Stiles, 2000). In essence, without a human observer there is no color.

Luckily, much of what we know about color can be quantified, so that we can carry out computations to correct for the idiosyncrasies of human vision and thereby display images that will appear to observers the way the designer of those images intended. This chapter contains the theory and mathematics required to do so.

19.1 Colorimetry

Colorimetry is the science of color measurement and description. Since color is ultimately a human response, color measurement should begin with human



提供解释为什么某些波长的光被吸收而其他波长被反射。因此，我们不能使用反射函数来解释为什么香蕉反射的光具有在我们看来为黄色的光谱组成。为此，我们将不得不研究分子轨道理论，这是一个超出本书范围的主题。

最后，当光线到达视网膜时，它被转码为传播到大脑的电信号。大脑的很大一部分致力于处理视觉信号，其中一部分产生了色彩的感觉。因此，即使我们知道香蕉反射的光谱，我们也不知道为什么人类将“黄色”一词与它联系起来。此外，正如我们将在本章的其余部分中发现的那样，我们对颜色的感知比乍看起来要复杂得多。它随着照明而变化，在观察者之间变化，并且随着时间的推移在观察者内变化。

换句话说，香蕉发出的光的光谱是在环境的背景下感知的。要预测观察者如何感知“香蕉规格”，需要了解包含香蕉的环境以及观察者的环境。在许多情况下，这两个环境是相同的。但是，当我们在monitor上展示香蕉的照片时，这两个环境会有所不同。由于人类的视觉感知取决于观察者所处的环境，因此它对照片中的香蕉的感知可能与直接看香蕉的观察者对香蕉的感知不同。这对我们应该如何处理颜色产生了重大影响，并说明了与颜色相关的复杂性。

为了强调人类视觉所起的关键作用，我们只需要看看颜色的定义：“颜色是视觉感知的方面，观察者可以通过它来区分两个相同大小和形状的无结构视野之间的差异，例如可能是由观察中有关辐射能量光谱组成的差异引起的” (Wyszecki & Stiles, 2000)。从本质上讲，没有人类观察者就没有颜色。

幸运的是，我们对颜色的大部分了解都可以量化，这样我们就可以进行计算来纠正人类视觉的特性，从而显示出观察者所看到的图像，这些图像的设计者所希望的方式。本章包含这样做所需的理论和数学。

19.1 Colorimetry

比色法是颜色测量和描述的科学。由于颜色最终是人类的反应，因此颜色测量应该从人类开始

observation. The photodetectors in the human retina consist of rods and cones. The rods are highly sensitive and come into play in low-light conditions. Under normal lighting conditions, the cones are operational, mediating human vision. There are three cone types and together they are primarily responsible for color vision.

Although it may be possible to directly record the electrical output of cones while some visual stimulus is being presented, such a procedure would be invasive, while at the same time ignoring the sometimes substantial differences between observers. Moreover, much of the measurement of color was developed well before such direct recording techniques were available.

The alternative is to measure color by means of measuring the human response to patches of color. This leads to color matching experiments, which will be described later in this section. Carrying out these experiments have resulted in several standardized observers, which can be thought of as statistical approximations of actual human observers. First, however, we need to describe some of the assumptions underlying the possibility of color matching, which are summarized by Grassmann's laws.

19.1.1 Grassmann's Laws

Given that humans have three different cone types, the experimental laws of color matching can be summed up as the trichromatic generalization (Wyszecki & Stiles, 2000), which states that any color stimulus can be matched completely with an additive mixture of three appropriately modulated color sources. This feature of color is often used in practice, for instance by televisions and monitors which reproduce many different colors by adding a mixture of red, green, and blue light for each pixel. It is also the reason that renderers can be built using only three values to describe each color.

The trichromatic generalization allows us to make color matches between any given stimulus and an additive mixture of three other color stimuli. Hermann Grassmann was the first to describe the algebraic rules to which color matching adheres. They are known as Grassmann's laws of additive color matching (Grassmann, 1853) and are the following:

- **Symmetry law.** If color stimulus A matches color stimulus B , then B matches A .
- **Transitive law.** If A matches B and B matches C , then A matches C .
- **Proportionality law.** If A matches B , then αA matches αB , where α is a positive scale factor.

观察。人视网膜中的光电探测器由视杆和视锥组成。棒是高度敏感的并在低光条件下发挥作用。在正常照明条件下，视锥细胞是可操作的，介导人类视觉。有三种锥体类型，它们一起主要负责色觉。

虽然有可能直接记录锥体的电输出，而一些视觉刺激正在呈现，这样的程序将是invasive，而在同一时间忽略有时实质性的差异是补间观察者。此外，在这种直接记录技术出现之前，很多颜色测量都是发展起来的。

另一种方法是通过测量人类对色块的再吸收来测量颜色。这导致颜色匹配实验，这将在本节后面描述。进行这些实验产生了几个标准化的观察者，这些观察者可以被认为是实际人类观察者的统计近似值。然而，首先，我们需要描述色彩匹配可能性的一些假设，这些假设由格拉斯曼定律总结。

19.1.1 Grassmann's Laws

鉴于人类有三种不同的锥体类型，颜色匹配的实验规律可以总结为三色泛化 (Wyszecki & Stiles, 2000)，其中指出任何颜色刺激都可以与三种适当调制的颜色源的添加剂混这种颜色特征通常在实践中使用，例如电视和显示器通过为每个像素添加红色，绿色和蓝色光的混合物来再现许多不同的颜色。这也是渲染器可以只使用三个值来描述每种颜色的原因。

三色泛化使我们能够在任何给定的刺激和其他三种颜色刺激的添加剂混合物之间进行颜色匹配。赫尔曼·格拉斯曼是第一个描述颜色匹配所遵循的代数规则的人。它们被称为Grassmann's laws of additive color matching (Grassmann, 1853)，如下所示：

- ***对称定律。** 如果颜色刺激A与颜色刺激B匹配，则B matches A.
- ***传递定律。** 如果A与B匹配，B与C匹配，则A与C匹配。
- ***比例法。** 如果A与B匹配，则 aA 与 aB 匹配，其中 a 是正比例因子。



- **Additivity law.** If A matches B , C matches D , and $A + C$ matches $B + D$, then it follows that $A + D$ matches $B + C$.

The additivity law forms the basis for color matching and colorimetry as a whole.

19.1.2 Cone Responses

Each cone type is sensitive to a range of wavelengths, spanning most of the full visible range. However, sensitivity to wavelengths is not evenly distributed, but contains a peak wavelength at which sensitivity is greatest. The location of this peak wavelength is different for each cone type. The three cone types are classified as S, M, and L cones, where the letters stand for short, medium, and long, indicating where in the visible spectrum the peak sensitivity is located.

The response of a given cone is then the magnitude of the electrical signal it outputs, as a function of the spectrum of wavelengths incident upon the cone. The cone response functions for each cone type as a function of wavelength λ are then given by $L(\lambda)$, $M(\lambda)$, and $S(\lambda)$. They are plotted in Figure 19.2.

The actual response to a stimulus with a given spectral composition $\Phi(\lambda)$ is then given for each cone type by

$$L = \int_{\lambda} \Phi(\lambda) L(\lambda) d\lambda,$$

$$M = \int_{\lambda} \Phi(\lambda) M(\lambda) d\lambda,$$

$$S = \int_{\lambda} \Phi(\lambda) S(\lambda) d\lambda.$$

These three integrated responses are known as tristimulus values.

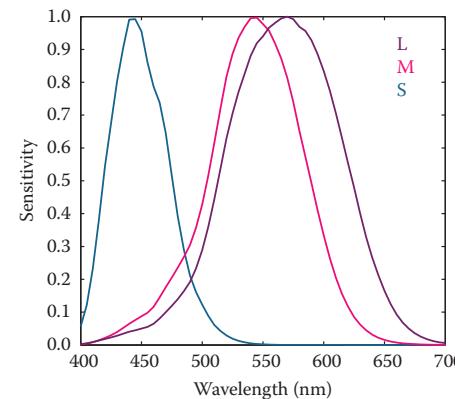


Figure 19.2. The cone response functions for L, M, and S cones.

*加法定律。如果A匹配B, C匹配D, 并且A+C匹配B+D, 那么它遵循A+D匹配B+C。

加性定律构成了整个配色和比色的基础。

19.1.2 锥体响应

每种锥体类型对一系列波长敏感，跨越整个可见光范围的大部分。然而，对波长的灵敏度不是均匀分布的，而是包含灵敏度最大的峰值波长。该峰值波长的位置对于每个锥型是不同的。三种锥体类型通常被定义为S, M和L锥体，其中字母代表短，中，长，表示峰值灵敏度在可见光谱中的位置。

给定锥体的响应是它输出的电信号的大小，作为入射到锥体上的波长光谱的函数。然后由 $L(\lambda)$ 、 $M(\lambda)$ 和 $S(\lambda)$ 给出作为波长 λ 函数的每种锥体类型的锥体响应函数。它们在图19.2中绘制。

对给定光谱组成 $\Phi(\lambda)$ 的刺激的实际响应，然后通过

$$L = \int_{\lambda} \Phi(\lambda) L(\lambda) d\lambda,$$

$$M = \int_{\lambda} \Phi(\lambda) M(\lambda) d\lambda,$$

$$S = \int_{\lambda} \Phi(\lambda) S(\lambda) d\lambda.$$

这三个综合响应被称为三刺激值。

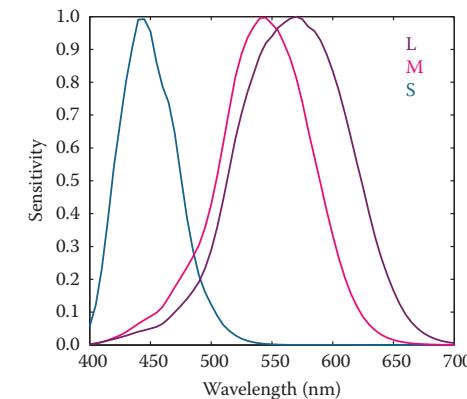


图19.2。 锥体响应函数为L、M和S锥体。

19.1.3 Color Matching Experiments

Given that tristimulus values are created by integrating the product of two functions over the visible range, it is immediately clear that the human visual system does not act as a simple wavelength detector. Rather, our photo-receptors act as approximately linear integrators. As a result, it is possible to find two different spectral compositions, say $\Phi_1(\lambda)$ and $\Phi_2(\lambda)$, that after integration yield the same response (L, M, S). This phenomenon is known as *metamerism*, an example of which is shown in Figure 19.3.

Metamerism is the key feature of human vision that allows the construction of color reproduction devices, including the color figures in this book and anything reproduced on printers, televisions, and monitors.

Color matching experiments also rely on the principle of metamerism. Suppose we have three differently colored light sources, each with a dial to alter its intensity. We call these three light sources primaries. We should now be able to adjust the intensity of each in such a way that when mixed together additively, the resulting spectrum integrates to a tristimulus value that matches the perceived color of a fourth unknown light source. When we carry out such an experiment, we have essentially matched our primaries to an unknown color. The positions of our three dials are then a representation of the color of the fourth light source.

In such an experiment, we have used Grassmann's laws to add the three spectra of our primaries. We have also used metamerism, because the combined spectrum of our three primaries is almost certainly different from the spectrum of the

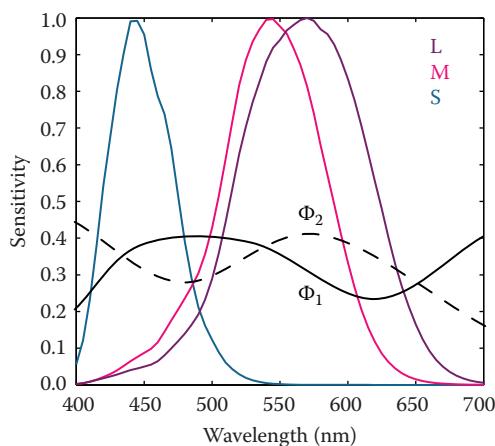


Figure 19.3. Two stimuli $\Phi_1(\lambda)$ and $\Phi_2(\lambda)$ leading to the same tristimulus values after integration.

19.1.3 配色实验

鉴于三刺激值是通过在可见光范围内积分两个频率的乘积而产生的，很明显人类视觉系统并不是一个简单的波长检测器。相反，我们的光受体充当近似线性积分器。因此，可以找到两种不同的光谱组成，例如 $\Phi_1(\lambda)$ 和 $\Phi_2(\lambda)$ ，在积分后产生相同的响应 (L, M, S)。这种现象被称为metamerism，其示例如图19.3所示。

Metamerism是人类视觉的关键特征，允许构建色彩再现设备，包括本书中的彩色数字以及打印机，电视和显示器上复制的任何内容。

配色实验也依赖于metamerism原理。Suppose我们有三种不同颜色的光源，每种光源都有一个表盘来改变其强度。我们称这三种光源为原色。我们现在应该能够以这样的方式调整每个的强度，即当相加地混合在一起时，产生的光谱集成到与第四个未知光源的感知颜色相匹配的三刺激值。当我们进行这样的实验时，我们基本上已经将我们的初选与未知的颜色相匹配。我们的三个刻度盘的位置是第四个光源的颜色的表示。

在这样的实验中，我们已经使用格拉斯曼定律来添加我们初选的三个spectra。我们也使用了metamerism，因为我们三个初选的组合光谱几乎可以肯定不同于

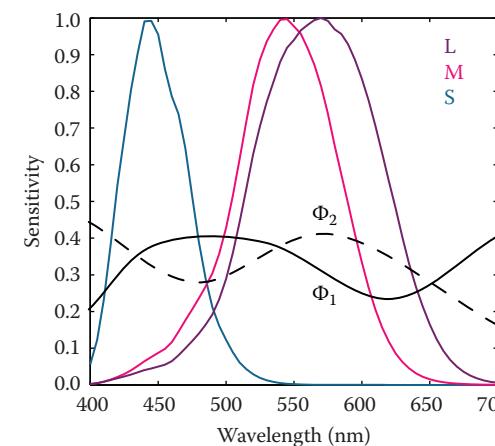


图19.3。 两个刺激 $\Phi_1(\lambda)$ 和 $\Phi_2(\lambda)$ 导致积分后相同的三刺激值。



fourth light source. However, the tristimulus values computed from these two spectra will be identical, having produced a color match.

Note that we do not actually have to know the cone response functions to carry out such an experiment. As long as we use the same observer under the same conditions, we are able to match colors and record the positions of our dials for each color. However, it is quite inconvenient to have to carry out such experiments every time we want to measure colors. For this reason, we do want to know the spectral cone response functions and average those for a set of different observers to eliminate interobserver variability.

19.1.4 Standard Observers

If we perform a color matching experiment for a large range of colors, carried out by a set of different observers, it is possible to generate an average color matching dataset. If we specifically use monochromatic light sources against which to match our primaries, we can repeat this experiment for all visible wavelengths. The resulting tristimulus values are then called *spectral tristimulus values*, and can be plotted against wavelength λ , shown in Figure 19.4.

By using a well-defined set of primary light sources, the spectral tristimulus values lead to three color matching functions. The Commission Internationale d'Eclairage (CIE) has defined three such primaries to be monochromatic light sources of 435.8, 546.1, and 700 nm, respectively. With these three monochro-

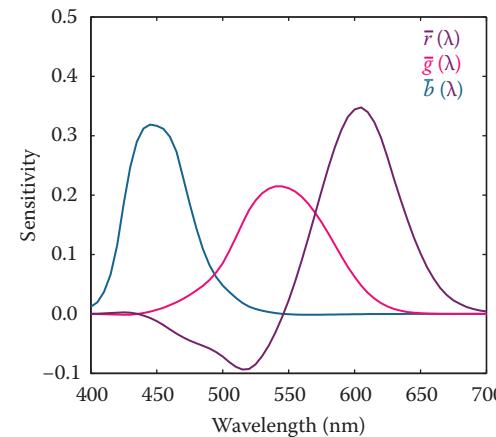


Figure 19.4. Spectral tristimulus values averaged over many observers. The primaries were monochromatic light sources with wavelengths of 435.8, 546.1, and 700 nm.



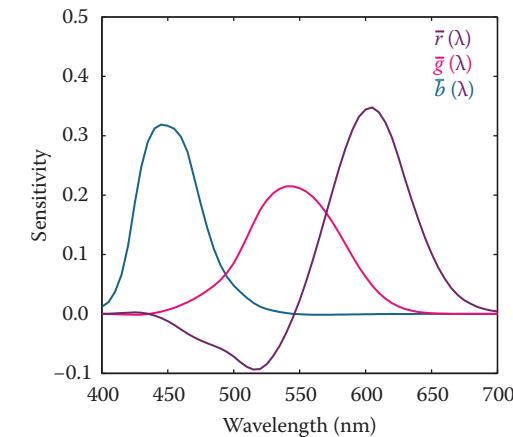
第四光源。然而，从这两个光谱计算的三刺激值将是相同的，产生了颜色匹配。

请注意，我们实际上不需要知道锥响应函数来进行这样的实验。只要我们在相同的条件下使用相同的观察者，我们就能够匹配颜色并记录每种颜色的表盘位置。然而，每次测量颜色时都必须进行这样的实验是相当不方便的。出于这个原因，我们确实想知道锥响应函数，并对一组不同观察者的锥响应函数进行平均，以消除服务器间的变异性。

19.1.4 标准观察员

如果我们对大范围的颜色进行颜色匹配实验，由一组不同的观察者进行，则可以生成平均颜色匹配ing数据集。如果我们专门使用单色光源来匹配我们的原始光源，我们可以对所有可见波长重复这个实验。由此产生的三刺激值被称为光谱三刺激值，可以根据波长 λ 绘制，如图19.4所示。

通过使用一组明确定义的主光源，光谱三刺激值导致三个颜色匹配函数。国际电平委员会（Cie）已经定义了三个这样的原始光源分别为435.8、546.1和700nm的单色光源。用这三种单色



光谱三刺激值在许多观察者上平均。波长为435.8、546.1和700nm的单色光源的原色。

matic light sources, all other visible wavelengths can be matched by adding different amounts of each. The amount of each required to match a given wavelength λ is encoded in color matching functions, given by $\bar{r}(\lambda)$, $\bar{g}(\lambda)$, and $\bar{b}(\lambda)$ and plotted in Figure 19.4. Tristimulus values associated with these color matching functions are termed R , G , and B .

Given that we are adding light, and light cannot be negative, you may have noticed an anomaly in Figure 19.4: to create a match for some wavelengths, it is necessary to subtract light. Although there is no such thing as negative light, we can use Grassmann's laws once more, and instead of subtracting light from the mixture of primaries, we can add the same amount of light to the color that is being matched.

The CIE $\bar{r}(\lambda)$, $\bar{g}(\lambda)$, and $\bar{b}(\lambda)$ color matching functions allow us to determine if a spectral distribution Φ_1 matches a second spectral distribution Φ_2 by simply comparing the resulting tristimulus values obtained by integrating with these color matching functions:

$$\begin{aligned}\int_{\lambda} \Phi_1(\lambda) \bar{r}(\lambda) &= \int_{\lambda} \Phi_2(\lambda) \bar{r}(\lambda), \\ \int_{\lambda} \Phi_1(\lambda) \bar{g}(\lambda) &= \int_{\lambda} \Phi_2(\lambda) \bar{g}(\lambda), \\ \int_{\lambda} \Phi_1(\lambda) \bar{b}(\lambda) &= \int_{\lambda} \Phi_2(\lambda) \bar{b}(\lambda).\end{aligned}$$

Of course, a color match is only guaranteed if all three tristimulus values match.

The importance of these color matching functions lies in the fact that we are now able to communicate and describe colors compactly by means of tristimulus values. For a given spectral function, the CIE color matching functions provide a precise way in which to calculate tristimulus values. As long as everybody uses the same color matching functions, it should always be possible to generate a match.

If the same color matching functions are not available, then it is possible to transform one set of tristimulus values into a different set of tristimulus values appropriate for a corresponding set of primaries. The CIE has defined one such a transform for two specific reasons. First, in the 1930s numerical integrations were difficult to perform, and even more so for functions that can be both positive and negative. Second, the CIE had already developed the photopic luminance response function, CIE $V(\lambda)$. It became desirable to have three integrating functions, of which $V(\lambda)$ is one and all three being positive over the visible range.

To create a set of positive color matching functions, it is necessary to define imaginary primaries. In other words, to reproduce any color in the visible spectrum, we need light sources that cannot be physically realized. The color match-

matic光源，所有其他可见波长可以通过添加不同的量的每一个来匹配。匹配给定波长所需的每个量以颜色匹配函数进行编码，由 $r(\lambda)$ ， $g(\lambda)$ 和 $b(\lambda)$ 给出，并在图19.4中绘制。与这些颜色匹配函数相关联的三刺激值被称为R、G和B。

鉴于我们正在添加光，并且光不能是负的，您可能已经注意到图19.4中的一个异常：要创建一些波长的匹配，有必要减去光。虽然没有负光这样的东西，但我们可以再次使用格拉斯曼定律，而不是从原色的混合物中减去光，我们可以向正在匹配的颜色添加相同数量的光。

CIE $r(\lambda)$ ， $g(\lambda)$ 和 $b(\lambda)$ 颜色匹配函数允许我们通过简单地比较通过与这些颜色匹配函数积分获得的结果三刺激值来确定光谱分布 Φ_1 是否与第二光谱：

$$\begin{aligned}\int_{\lambda} \Phi_1(\lambda) \bar{r}(\lambda) &= \int_{\lambda} \Phi_2(\lambda) \bar{r}(\lambda), \\ \int_{\lambda} \Phi_1(\lambda) \bar{g}(\lambda) &= \int_{\lambda} \Phi_2(\lambda) \bar{g}(\lambda), \\ \int_{\lambda} \Phi_1(\lambda) \bar{b}(\lambda) &= \int_{\lambda} \Phi_2(\lambda) \bar{b}(\lambda).\end{aligned}$$

当然，只有当所有三个三刺激值都匹配时，才保证颜色匹配。这些颜色匹配功能的重要性在于我们现在能够通过三刺激值来紧凑地交流和描述颜色。对于给定的光谱函数，CIE颜色匹配函数提供了计算三刺激值的精确方法。只要每个人都使用相同颜色匹配功能，就应该总是可以生成匹配。

如果相同的颜色匹配函数不可用，那么可以将一组三刺激值转换为适合于一组相应的原色的不同三刺激值。CIE出于两个具体原因定义了一个这样的转换。首先，在20世纪30年代，数值积分很难执行，对于既可能是正数又可能是负数的函数来说更是如此。其次，CIE已经开发了光度响应函数CIE $V(\lambda)$ 。理想的是有三个积分函数，其中 $V(\lambda)$ 是一个，三个在可见范围内都是正的。

要创建一组正色匹配函数，有必要定义假想的原色。换句话说，要在可见光谱中再现任何颜色，我们需要无法物理实现的光源。颜色搭配

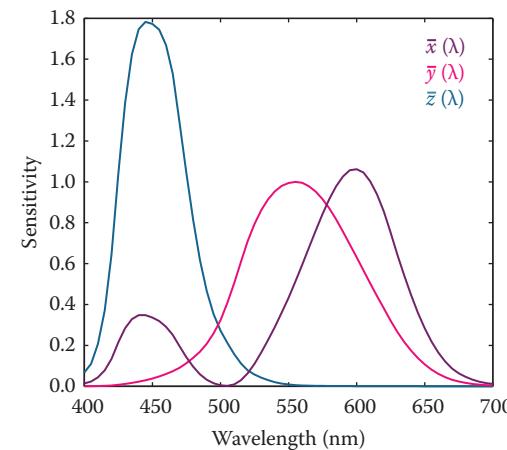


Figure 19.5. The CIE $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, and $\bar{z}(\lambda)$ color matching functions.

ing functions that were settled upon by the CIE are named $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, and $\bar{z}(\lambda)$ and are shown in Figure 19.5. Note that $\bar{y}(\lambda)$ is equal to the photopic luminance response function $V(\lambda)$ and that each of these functions is indeed positive. They are known as the CIE 1931 standard observer.

The corresponding tristimulus values are termed X , Y , and Z , to avoid confusion with R , G , and B tristimulus values that are normally associated with realizable primaries. The conversion from (R, G, B) tristimulus values to (X, Y, Z) tristimulus values is defined by a simple 3×3 transform:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{0.17697} \begin{bmatrix} 0.4900 & 0.3100 & 0.2000 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.0000 & 0.0100 & 0.9900 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

To calculate tristimulus values, we typically directly integrate the standard observer color matching functions with the spectrum of interest $\Phi(\lambda)$, rather than go through the CIE $\bar{r}(\lambda)$, $\bar{g}(\lambda)$, and $\bar{b}(\lambda)$ color matching functions first, followed by the above transformation. It allows us to calculate consistent color measurements and also determine when two colors match each other.

19.1.5 Chromaticity Coordinates

Every color can be represented by a set of three tristimulus values (X, Y, Z) . We could define an orthogonal coordinate system with X , Y , and Z axes and plot each color in the resulting 3D space. This is called a *color space*. The spatial extent of the volume in which colors lie is then called the *color gamut*.

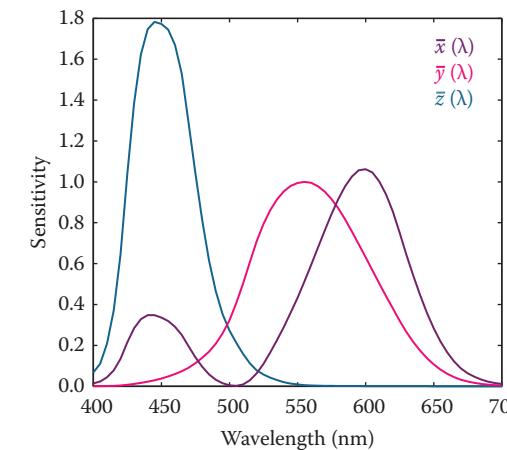


图19.5。CIE $x(\lambda)$ 、 $y(\lambda)$ 和 $z(\lambda)$ 颜色匹配函数。

由CIE确定的ing函数被命名为 $x(\lambda)$ 、 $y(\lambda)$ 和 $z(\lambda)$ ，如图19.5所示。注意， $y(\lambda)$ 等于光度响应函数 $V(\lambda)$ 并且这些函数中的每一个确实是正的。它们被称为CIE1931标准观察者。

相应的三刺激值被称为 X ， Y 和 Z ，以避免与 r ， G 和 B 三刺激值的con融合，这些三刺激值通常与真正的izableprimaries相关联。从 (R, G, B) 三刺激值到 (X, Y, Z) 三刺激值的转换由一个简单的 3×3 转换定义：

$$\begin{array}{rccccc} X & = & 0.4900 & 0.3100 & 0.2000 & R \\ Y & = & 0.17697 & 0.81240 & 0.01063 & G \\ Z & = & 0.0000 & 0.0100 & 0.9900 & B \end{array}.$$

为了计算三刺激值，我们通常直接将标准观察者颜色匹配函数与感兴趣的光谱 $\Phi(\lambda)$ 集成，而不是先经过CIE $r(\lambda)$ ， $g(\lambda)$ 和 $b(\lambda)$ 颜色匹配函数，然后进行上述变它允许我们计算一致的颜色测量，并确定两种颜色何时相互匹配。

19.1.5 Chromaticity Coordinates

每种颜色都可以由一组三个三刺激值 (X, Y, Z) 表示。我们可以用 X 、 Y 和 Z 轴定义一个正交坐标系，并在生成的3D空间中绘制每种颜色。这被称为颜色空间。然后，颜色所在的体积的空间范围称为色域。

Visualizing colors in a 3D color space is fairly difficult. Moreover, the Y -value of any color corresponds to its luminance, by virtue of the fact that $\bar{y}(\lambda)$ equals $V(\lambda)$. We could therefore project tristimulus values to a 2D space which approximates chromatic information, i.e., information which is independent of luminance. This projection is called a *chromaticity diagram* and is obtained by normalization while at the same time removing luminance information:

$$\begin{aligned}x &= \frac{X}{X + Y + Z}, \\y &= \frac{Y}{X + Y + Z}, \\z &= \frac{Z}{X + Y + Z}.\end{aligned}$$

Given that $x + y + z$ equals 1, the z -value is redundant, allowing us to plot the x and y chromaticities against each other in a chromaticity diagram. Although x and y by themselves are not sufficient to fully describe a color, we can use these two chromaticity coordinates and one of the three tristimulus values, traditionally Y , to recover the other two tristimulus values:

$$\begin{aligned}X &= \frac{x}{y} Y, \\Z &= \frac{1 - x - y}{y} Y.\end{aligned}$$

By plotting all monochromatic (spectral) colors in a chromaticity diagram, we obtain a horseshoe-shaped curve. The points on this curve are called *spectrum loci*. All other colors will generate points lying inside this curve. The spectrum locus for the 1931 standard observer is shown in Figure 19.6. The purple line

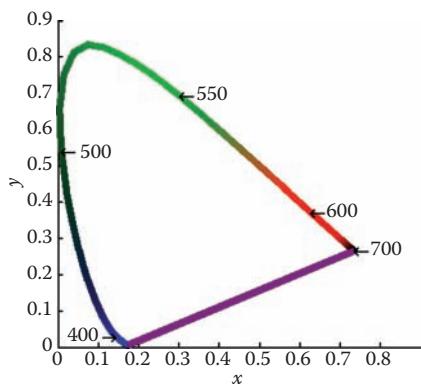


Figure 19.6. The spectrum locus for the CIE 1931 standard observer.

在3d色彩空间中可视化颜色相当困难。此外，任何颜色的Y值都对应于其亮度，因为 $y(\lambda)$ 等于 $V(\lambda)$ 。因此，我们可以将三刺激值投影到一个近似色度信息的2D空间，即与亮度无关的信息。这种投影称为色度图，是在去除亮度信息的同时通过归一化得到的：

$$\begin{aligned}x &= \frac{X}{X + Y + Z}, \\y &= \frac{Y}{X + Y + Z}, \\z &= \frac{Z}{X + Y + Z}.\end{aligned}$$

鉴于 $x+y+z$ 等于 1， z 值是多余的，允许我们在色度图中相互绘制 x 和 y 色度。虽然 x 和 y 本身不足以完全描述一种颜色，但我们可以使用这两个色度坐标和三个三刺激值中的一个，传统上是 Y ，来恢复另外两个三刺激值：

$$\begin{aligned}X &= \frac{x}{y} Y, \\Z &= \frac{1 - x - y}{y} Y.\end{aligned}$$

通过在色度图中绘制所有单色（光谱）颜色，我们获得马蹄形曲线。该曲线上的点称为谱位点。所有其他颜色将生成位于此曲线内的点。1931年标准观测器的光谱轨迹如图19.6所示。紫线

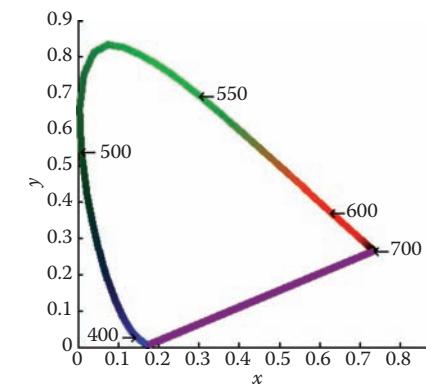


图19.6。Cie1931标准观测器的光谱轨迹。

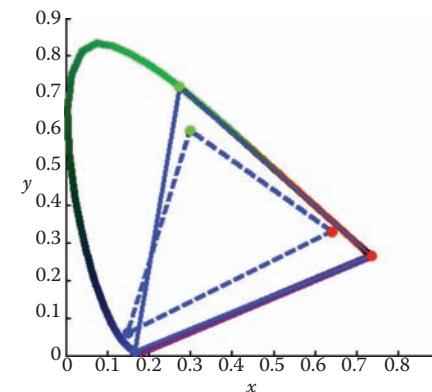


Figure 19.7. The chromaticity boundaries of the CIE RGB primaries at 435.8, 546.1, and 700 nm (solid) and a typical HDTV (dashed).

between either end of the horseshoe does not represent a monochromatic color, but rather a combination of short and long wavelength stimuli.

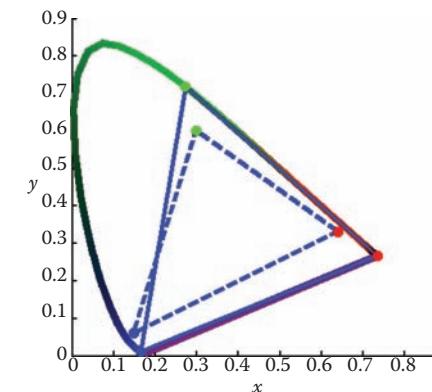
A (non-monochromatic) primary can be integrated over all visible wavelengths, leading to (X, Y, Z) tristimulus values, and subsequently to an (x, y) chromaticity coordinate, i.e., a point on a chromaticity diagram. Repeating this for two or more primaries yields a set of points on a chromaticity diagram that can be connected by straight lines. The volume spanned in this manner represents the range of colors that can be reproduced by the additive mixture of these primaries. Examples of three-primary systems are shown in Figure 19.7.

Chromaticity diagrams provide insight into additive color mixtures. However, they should be used with care. First, the interior of the horseshoe should not be colored, as any color reproduction system will have its own primaries and can only reproduce some parts of the chromaticity diagram. Second, as the CIE color matching functions do not represent human cone sensitivities, the distance between any two points on a chromaticity diagram is not a good indicator for how differently these colors will be perceived.

A more uniform chromaticity diagram was developed to at least in part address the second of these problems. The CIE $u'v'$ chromaticity diagram provides a perceptually more uniform spacing and is therefore generally preferred over (x, y) chromaticity diagrams. It is computed from (X, Y, Z) tristimulus values by applying a different normalization,

$$u' = \frac{4X}{X + 15Y + 3Z},$$

$$v' = \frac{9Y}{X + 15Y + 3Z}.$$



CIE RGB原色在435.8、546.1和700nm（实心）和典型HDTV（虚线）处的色度边界。

马蹄铁两端之间不代表单色，而是短波长和长波长刺激的组合。

一个（非单色）原色可以在所有可见波长上积分，从而得到 (X, Y, Z) 三刺激值，然后得到 (x, y) 色度坐标，即色度图上的一个点。对两个或多个初选重复此操作会在色度图上产生一组可以通过直线连接的点。以这种方式跨越的体积表示可以由这些原色的添加剂混合物再现的颜色的范围。三主系统的例子如图19.7所示。

色度图提供了添加剂颜色混合物的洞察力。但是，它们应该小心使用。首先，马蹄铁的内部不应该着色，因为任何颜色再现系统都会有自己的原色，只能再现色度图的某些部分。其次，由于CIE颜色匹配函数不代表人的锥体灵敏度，色度图上任意两点之间的距离并不是一个很好的指标，可以看出这些颜色的不同程度。

开发了更均匀的色度图，以至少部分地反映这些问题中的第二个。Cieuv'色度图提供了感知上更均匀的间距，因此通常比 (x, y) 色度图更受欢迎。它是通过应用不同的归一化从 (X, Y, Z) 三刺激值计算的

$$u' = \frac{4X}{X + 15Y + 3Z},$$

$$v' = \frac{9Y}{X + 15Y + 3Z}.$$

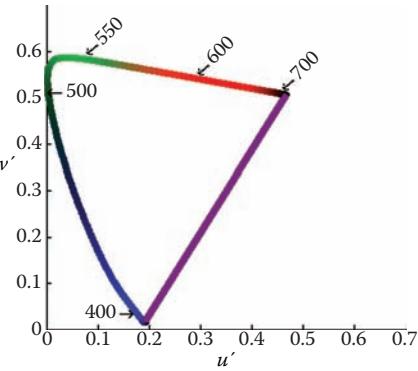


Figure 19.8. The CIE $u'v'$ chromaticity diagram.

and can alternatively be computed directly from (x, y) chromaticity coordinates:

$$u' = \frac{4x}{-2x + 12y + 3},$$

$$v' = \frac{9y}{-2x + 12y + 3}.$$

A CIE $u'v'$ chromaticity diagram is shown in Figure 19.8.

19.2 Color Spaces

As explained above, each color can be represented by three numbers, for instance defined by (X, Y, Z) tristimulus values. However, its primaries are imaginary, meaning that it is not possible to construct a device that has three light sources (all positive) that can reproduce all colors in the visible spectrum.

For the same reason, image encoding and computations on images may not be practical. There is, for instance, a large number of possible $X Y Z$ values that do not correspond to any physical color. This would lead to inefficient use of available bits for storage and to a higher requirement for bit-depth to preserve visual integrity after image processing. Although it may be possible to build a capture device that has primaries that are close to the CIE $X Y Z$ color matching functions, the cost of hardware and image processing make this an unattractive option. It is not possible to build a display that corresponds to CIE $X Y Z$. For these reasons, it is necessary to design other color spaces: physical realizability, efficient encoding, perceptual uniformity, and intuitive color specification.

The CIE $X Y Z$ color space is still actively used, mostly for the conversion between other color spaces. It can be seen as a device-independent color space.

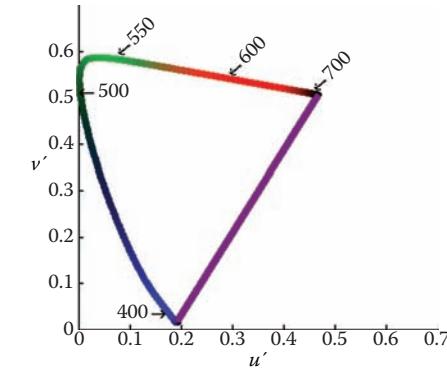


图19.8。的CIEu'v'色度图。

也可以直接从 $(x y)$ 色度坐标计算:

$$u' = \frac{4x}{-2x + 12y + 3},$$

$$v' = \frac{9y}{-2x + 12y + 3}.$$

Cieu'v'色度图如图19.8所示。

19.2 色彩空间

如上所述，每种颜色可以由三个数字表示，例如由 (X, Y, Z) 三刺激值定义。然而，它的原色是虚构的，这意味着不可能构建具有三个光源（均为正）的设备，这些光源可以再现可见光谱中的所有颜色。

出于同样的原因，对图像进行图像编码和计算可能不实用。例如，有大量可能的XYZ值不对应于任何物理颜色。这将导致可用位用于存储的效率低下，并对位深度的要求更高，以便在图像处理后保持视觉完整性。虽然有可能构建具有接近CIEXYZ颜色匹配功能的原色的捕获设备，但硬件和图像处理的成本使这成为一个没有吸引力的选择。无法构建与CIEXYZ相对应的显示器。由于这些原因，有必要设计其他颜色空间：物理可实现性，高效编码，感知均匀性和直观的颜色规范。

CIEXYZ颜色空间仍在积极使用，主要用于其他颜色空间之间的转换。它可以被看作是一个与设备无关的颜色空间。



Other color spaces can then be defined in terms of their relationship to CIE $X Y Z$, which is often specified by a specific transform. For instance, linear and additive trichromatic display devices can be transformed to and from CIE $X Y Z$ by means of a simple 3×3 matrix. Some nonlinear additional transform may also be specified, for instance to minimize perceptual errors when data is stored with a limited bit-depth, or to enable display directly on devices that have a nonlinear relationship between input signal and the amount of light emitted.

19.2.1 Constructing a Matrix Transform

For a display device with three primaries, say red, green, and blue, we can measure the spectral composition of the emitted light by sending the color vectors $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. These vectors represent the three cases namely where one of the primaries is full on, and the other two are off. From the measured spectral output, we can then compute the corresponding chromaticity coordinates (x_R, y_R) , (x_G, y_G) , and (x_B, y_B) .

The *white point* of a display is defined as the spectrum emitted when the color vector $(1, 1, 1)$ is sent to the display. Its corresponding chromaticity coordinate is (x_W, y_W) . The three primaries and the white point characterize the display and are each required to construct a transformation matrix between the display's color space and CIE $X Y Z$.

These four chromaticity coordinates can be extended to chromaticity triplets reconstructing the z -coordinate from $z = 1 - x - y$, leading to triplets (x_R, y_R, z_R) , (x_G, y_G, z_G) , (x_B, y_B, z_B) , and (x_W, y_W, z_W) . If we know the maximum luminance of the white point, we can compute its corresponding tristimulus value (X_W, Y_W, Z_W) and then solve the following set of equations for the luminance ratio scalars S_R , S_G , and S_B :

$$\begin{aligned} X_W &= x_R S_R + x_G S_G + x_B S_B, \\ Y_W &= y_R S_R + y_G S_G + y_B S_B, \\ Z_W &= z_R S_R + z_G S_G + z_B S_B. \end{aligned}$$

The conversion between RGB and XYZ is then given by

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} x_R S_R & x_G S_G & x_B S_B \\ y_R S_R & y_G S_G & y_B S_B \\ z_R S_R & z_G S_G & z_B S_B \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

The luminance of any given color can be computed by evaluating the middle row of a matrix constructed in this manner:

$$Y = y_R S_R R + y_G S_G G + y_B S_B B.$$



然后可以根据它们与CIEXYZ的关系来定义其他颜色空间，这通常由特定变换指定。例如，线性和相加三色显示设备可以通过简单的 3×3 矩阵转换到和从CIEXYZ。还可以指定一些非线性附加变换，例如，当数据以有限的位深度存储时，可以最大限度地减少感知误差，或者可以直接在输入信号和发出的光量之间具有非线性关系的设备上显示。

19.2.1 构造矩阵变换

对于具有三原色（例如红色、绿色和蓝色）的显示设备，我们可以通过发送颜色矢量来确定发射光的光谱组成 $(1, 0, 0)$ ， $(0, 1, 0)$ 和 $(0, 0, 1)$ 。这些向量表示三种情况，即其中一个初选为全开，另外两个为关。从测量的光谱输出，我们可以计算相应的色度坐标 (x_R, y_R) ， (x_G, y_G) 和 (x_B, y_B) 。

显示器的白点定义为当颜色矢量 $(1, 1, 1)$ 发送到显示器时发出的光谱。其对应的色度坐标为 (x_W, y_W) 。三个原色和白点表征显示器，并且每个都需要在显示器的颜色空间和CIEXYZ之间构建变换矩阵。

这四个色度坐标可以扩展到重建 $z=1-x-y$ 的 z 坐标的色度三元组，导致三元组 (x_R, y_R, z_R) ， (x_G, y_G, z_G) ， (x_B, y_B, z_B) 和 (x_W, y_W, z_W) 。如果我们知道白点的最大亮度，我们可以计算其对应的三刺激值 (X_w, Y_w, Z_w) ，然后求解以下亮度比标量 S_R ， S_G 和 S_B 的一组方程：

$$\begin{aligned} X_W &= x_R S_R + x_G S_G + x_B S_B, \\ Y_W &= y_R S_R + y_G S_G + y_B S_B, \\ Z_W &= z_R S_R + z_G S_G + z_B S_B. \end{aligned}$$

RGB和XYZ之间的转换然后由

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} x_R S_R & x_G S_G & x_B S_B \\ y_R S_R & y_G S_G & y_B S_B \\ z_R S_R & z_G S_G & z_B S_B \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

任何给定颜色的亮度都可以通过评估以这种方式构造的矩阵的中间行来计算：

$$Y = y_R S_R R + y_G S_G G + y_B S_B B.$$

	R	G	B	White
<i>x</i>	0.6400	0.3000	0.1500	0.3127
<i>y</i>	0.3300	0.6000	0.0600	0.3290

Table 19.1. The (x, y) chromaticity coordinates for the primaries and white point specified by ITU-R BT.709. The sRGB standard also uses these primaries and white point.

To convert between XYZ and RGB of a given device, the above matrix can simply be inverted.

If an image is represented in an RGB color space for which the primaries and white point are unknown, then the next best thing is to assume that the image was encoded in a standard RGB color space. A reasonable choice is then to assume that the image was specified according to ITU-R BT.709, which is the specification used for encoding and broadcasting of HDTV. Its primaries and white point are specified in Table 19.1. Note that the same primaries and white point are used to define the well-known sRGB color space. The transformation between this RGB color space and CIE XYZ is and vice versa given by

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix};$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.2405 & -1.5371 & -0.4985 \\ -0.9693 & 1.8706 & 0.0416 \\ 0.0556 & -0.2040 & 1.0572 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}.$$

By substituting the maximum *RGB* values of the device, we can compute the white point. For ITU-R BT.709, the maximum values are $(R_W, G_W, B_W) = (100, 100, 100)$, leading to a white point of $(X_W, Y_W, Z_W) = (95.05, 100.00, 108.90)$.

In addition to a linear transformation, the sRGB color space is characterized by a subsequent nonlinear transform. The nonlinear encoding is given by

$$R_{\text{sRGB}} = \begin{cases} 1.055 R^{1/2.4} - 0.055 & R > 0.0031308, \\ 12.92 R & R \leq 0.0031308; \end{cases}$$

$$G_{\text{sRGB}} = \begin{cases} 1.055 G^{1/2.4} - 0.055 & G > 0.0031308, \\ 12.92 G & G \leq 0.0031308; \end{cases}$$

$$B_{\text{sRGB}} = \begin{cases} 1.055 B^{1/2.4} - 0.055 & B > 0.0031308, \\ 12.92 B & B \leq 0.0031308. \end{cases}$$

This nonlinear encoding helps minimize perceptual errors due to quantization errors in digital applications.

	R	G	B	White
<i>x</i>	0.6400	0.3000	0.1500	0.3127
<i>y</i>	0.3300	0.6000	0.0600	0.3290

Table 19.1. 由指定的原色和白点的(x,y)色度坐标
ITU-RBT。709.SRGB标准也使用这些初选和白点。

要在给定设备的XYZ和RGB之间转换，可以简单地反转上述矩阵。

如果图像在原色和白点未知的RGB颜色空间中表示，那么下一个最好的事情是假设图像是在标准RGB颜色空间中编码的。一个合理的选择是假设图像是根据ITU-RBT指定的。709，这是用于HDTV编码和广播的具体规定。其原色和白点在表19.1中指定。注意，相同的原色和白点用于定义公知的sRGB色彩空间。此RGB颜色空间和CIEXYZ之间的转换是，反之亦然给出

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix};$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.2405 & -1.5371 & -0.4985 \\ -0.9693 & 1.8706 & 0.0416 \\ 0.0556 & -0.2040 & 1.0572 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}.$$

通过替换设备的最大RGB值，我们可以计算白点。为ITU-RBT。709，最大值分别为 $(R_W, G_W, B_W)=(100 100 100)$ 导致白点为 $(X_W Y_W Z_W)=(95.05 100.00 108.90)$ 。

除了线性变换之外，sRGB颜色空间的特征在于随后的非线性变换。非线性编码由下式给出

$$R_{\text{sRGB}} = \begin{cases} 1.055 R^{1/2.4} - 0.055 & R > 0.0031308, \\ 12.92 R & R \leq 0.0031308; \end{cases}$$

$$G_{\text{sRGB}} = \begin{cases} 1.055 G^{1/2.4} - 0.055 & G > 0.0031308, \\ 12.92 G & G \leq 0.0031308; \end{cases}$$

$$B_{\text{sRGB}} = \begin{cases} 1.055 B^{1/2.4} - 0.055 & B > 0.0031308, \\ 12.92 B & B \leq 0.0031308. \end{cases}$$

这种非线性编码有助于最小化由于数字应用中的量化误差引起的感知误差。



19.2.2 Device-Dependent RGB Spaces

As each device typically has its own set of primaries and white point, we call the associated RGB color spaces device-dependent. It should be noted that even if all these devices operate in an RGB space, they may have very different primaries and white points. If we therefore have an image specified in some RGB space, it may appear very different to us, depending upon which device we display it.

This is clearly an undesirable situation, resulting from a lack of color management. However, if the image is specified in a known RGB color space, it can first be converted to XYZ, which is device independent, and then subsequently it can be converted to the RGB space of the device on which it will be displayed.

There are several other RGB color spaces that are well defined. They each consist of a linear matrix transform followed by a nonlinear transform, akin to the aforementioned sRGB color space. The nonlinear transform can be parameterized as follows:

$$R_{\text{nonlinear}} = \begin{cases} (1+f) R^\gamma - f & t < R \leq 1, \\ sR & 0 \leq R \leq t; \end{cases}$$

$$G_{\text{nonlinear}} = \begin{cases} (1+f) G^\gamma - f & t < G \leq 1, \\ sG & 0 \leq G \leq t; \end{cases}$$

$$B_{\text{nonlinear}} = \begin{cases} (1+f) B^\gamma - f & t < B \leq 1, \\ sB & 0 \leq B \leq t. \end{cases}$$

The parameters s , f , t and γ , together with primaries and white point, specify a class of RGB color spaces that are used in various industries. Several common transformations are listed in Table 19.2.

19.2.3 LMS Cone Space

The aforementioned cone signals can be expressed in terms of the CIE XYZ color space. The matrix transform to compute *LMS* signals from *XYZ* and vice versa are given by

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.38971 & 0.68898 & -0.07868 \\ -0.22981 & 1.18340 & 0.04641 \\ 0.00000 & 0.00000 & 1.00000 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix};$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1.91019 & -1.11214 & 0.20195 \\ 0.37095 & 0.62905 & 0.00000 \\ 0.00000 & 0.00000 & 1.00000 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}.$$



19.2.2 Device-Dependent RGB Spaces

由于每个设备通常都有自己的一组原色和白点，我们将相关的RGB颜色空间称为设备依赖性。应该注意的是，即使所有这些设备都在RGB空间中操作，它们也可能具有非常不同的原色和白点。如果我们因此在某些RGB空间中指定了一个图像，它可能会与我们看起来非常不同，这取决于我们显示它的设备。

这显然是一个不希望的情况。由于缺乏颜色人的刺激。然而，如果图像在已知的RGB颜色空间中指定，则可以首先将其转换为XYZ，这是设备独立的，然后随后可以将其转换为其将在其上显示的设备的RGB空间。还有几个其他的RGB颜色空间是很好的定义。它们各自由线性矩阵变换和非线性变换组成，类似于上述sRGB颜色空间。非线性变换可以参数化如下：

$$R_{\text{nonlinear}} = \begin{cases} (1+f) R^\gamma - f & t < R \leq 1, \\ sR & 0 \leq R \leq t; \end{cases}$$

$$G_{\text{nonlinear}} = \begin{cases} (1+f) G^\gamma - f & t < G \leq 1, \\ sG & 0 \leq G \leq t; \end{cases}$$

$$B_{\text{nonlinear}} = \begin{cases} (1+f) B^\gamma - f & t < B \leq 1, \\ sB & 0 \leq B \leq t. \end{cases}$$

参数s、f、t和 γ ，以及原色和白点，指定了一类用于各种行业的RGB颜色空间。表19.2中列出了几种常见的变换。

19.2.3 Lms锥体空间

前述锥信号可以用CIEXYZ颜色空间来表示。从XYZ计算LMS信号的矩阵变换，反之亦然，由下式给出

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.38971 & 0.68898 & -0.07868 \\ -0.22981 & 1.18340 & 0.04641 \\ 0.00000 & 0.00000 & 1.00000 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix};$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1.91019 & -1.11214 & 0.20195 \\ 0.37095 & 0.62905 & 0.00000 \\ 0.00000 & 0.00000 & 1.00000 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}.$$



Color space	XYZ to RGB matrix	RGB to XYZ matrix	Nonlinear transform
sRGB	$\begin{bmatrix} 3.2405 & -1.5371 & -0.4985 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0572 \end{bmatrix}$	$\begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix}$	$\gamma = 1/2.4 \approx 0.42$ $f = 0.055$ $s = 12.92$ $t = 0.0031308$
Adobe RGB (1998)	$\begin{bmatrix} 2.0414 & -0.5649 & -0.3447 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0134 & -0.1184 & 1.0154 \end{bmatrix}$	$\begin{bmatrix} 0.5767 & 0.1856 & 0.1882 \\ 0.2974 & 0.6273 & 0.0753 \\ 0.0270 & 0.0707 & 0.9911 \end{bmatrix}$	$\gamma = \frac{1}{2\frac{51}{256}} \approx \frac{1}{2.2}$ $f = N.A.$ $s = N.A.$ $t = N.A.$
HDTV (HD-CIF)	$\begin{bmatrix} 3.2405 & -1.5371 & -0.4985 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0572 \end{bmatrix}$	$\begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.099$ $s = 4.5$ $t = 0.018$
NTSC (1953)/ ITU-R BT.601-4	$\begin{bmatrix} 1.9100 & -0.5325 & -0.2882 \\ -0.9847 & 1.9992 & -0.0283 \\ 0.0583 & -0.1184 & 0.8976 \end{bmatrix}$	$\begin{bmatrix} 0.6069 & 0.1735 & 0.2003 \\ 0.2989 & 0.5866 & 0.1145 \\ 0.0000 & 0.0661 & 1.1162 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.099$ $s = 4.5$ $t = 0.018$
PAL/SECAM	$\begin{bmatrix} 3.0629 & -1.3932 & -0.4758 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0679 & -0.2289 & 1.0694 \end{bmatrix}$	$\begin{bmatrix} 0.4306 & 0.3415 & 0.1783 \\ 0.2220 & 0.7066 & 0.0713 \\ 0.0202 & 0.1296 & 0.9391 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.099$ $s = 4.5$ $t = 0.018$
SMPTE-C	$\begin{bmatrix} 3.5054 & -1.7395 & -0.5440 \\ -1.0691 & 1.9778 & 0.0352 \\ 0.0563 & -0.1970 & 1.0502 \end{bmatrix}$	$\begin{bmatrix} 0.3936 & 0.3652 & 0.1916 \\ 0.2124 & 0.7010 & 0.0865 \\ 0.0187 & 0.1119 & 0.9582 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.099$ $s = 4.5$ $t = 0.018$
SMPTE-240M	$\begin{bmatrix} 2.042 & -0.565 & -0.345 \\ -0.894 & 1.815 & 0.032 \\ 0.064 & -0.129 & 0.912 \end{bmatrix}$	$\begin{bmatrix} 0.567 & 0.190 & 0.193 \\ 0.279 & 0.643 & 0.077 \\ 0.000 & 0.073 & 1.016 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.1115$ $s = 4.0$ $t = 0.0228$
Wide Gamut	$\begin{bmatrix} 1.4625 & -0.1845 & -0.2734 \\ -0.5228 & 1.4479 & 0.0681 \\ 0.0346 & -0.0958 & 1.2875 \end{bmatrix}$	$\begin{bmatrix} 0.7164 & 0.1010 & 0.1468 \\ 0.2587 & 0.7247 & 0.0166 \\ 0.0000 & 0.0512 & 0.7740 \end{bmatrix}$	$\gamma = N.A.$ $f = N.A.$ $s = N.A.$ $t = N.A.$

Table 19.2. Transformations for standard RGB color spaces (after (Pascale, 2003)).

This transform is known as the Hunt-Pointer-Estevez transform (Hunt, 2004) and is used in chromatic adaptation transforms as well as in color appearance modeling.

19.2.4 CIE 1976 $L^*a^*b^*$

Color opponent spaces are characterized by a channel representing an achromatic channel (luminance), as well as two channels encoding color opponency. These are frequently red-green and yellow-blue channels. These color opponent chan-

色彩空间	XYZ到RGB矩阵	RGB到XYZ矩阵	非线性变换
sRGB	$\begin{bmatrix} 3.2405 & -1.5371 & -0.4985 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0572 \end{bmatrix}$	$\begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix}$	$\gamma = 1/2.4 \approx 0.42$ $f = 0.055$ $s = 12.92$ $t = 0.0031308$
Adobe RGB (1998)	$\begin{bmatrix} 2.0414 & -0.5649 & -0.3447 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0134 & -0.1184 & 1.0154 \end{bmatrix}$	$\begin{bmatrix} 0.5767 & 0.1856 & 0.1882 \\ 0.2974 & 0.6273 & 0.0753 \\ 0.0270 & 0.0707 & 0.9911 \end{bmatrix}$	$\gamma = \frac{1}{2\frac{51}{256}} \approx \frac{1}{2.2}$ $f = N.A.$ $s = N.A.$ $t = N.A.$
HDTV (HD-CIF)	$\begin{bmatrix} 3.2405 & -1.5371 & -0.4985 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0572 \end{bmatrix}$	$\begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.099$ $s = 4.5$ $t = 0.018$
NTSC (1953)/ ITU-R BT.601-4	$\begin{bmatrix} 1.9100 & -0.5325 & -0.2882 \\ -0.9847 & 1.9992 & -0.0283 \\ 0.0583 & -0.1184 & 0.8976 \end{bmatrix}$	$\begin{bmatrix} 0.6069 & 0.1735 & 0.2003 \\ 0.2989 & 0.5866 & 0.1145 \\ 0.0000 & 0.0661 & 1.1162 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.099$ $s = 4.5$ $t = 0.018$
PAL/SECAM	$\begin{bmatrix} 3.0629 & -1.3932 & -0.4758 \\ -0.9693 & 1.8760 & 0.0416 \\ 0.0679 & -0.2289 & 1.0694 \end{bmatrix}$	$\begin{bmatrix} 0.4306 & 0.3415 & 0.1783 \\ 0.2220 & 0.7066 & 0.0713 \\ 0.0202 & 0.1296 & 0.9391 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.099$ $s = 4.5$ $t = 0.018$
SMPTE-C	$\begin{bmatrix} 3.5054 & -1.7395 & -0.5440 \\ -1.0691 & 1.9778 & 0.0352 \\ 0.0563 & -0.1970 & 1.0502 \end{bmatrix}$	$\begin{bmatrix} 0.3936 & 0.3652 & 0.1916 \\ 0.2124 & 0.7010 & 0.0865 \\ 0.0187 & 0.1119 & 0.9582 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.099$ $s = 4.5$ $t = 0.018$
SMPTE-240M	$\begin{bmatrix} 2.042 & -0.565 & -0.345 \\ -0.894 & 1.815 & 0.032 \\ 0.064 & -0.129 & 0.912 \end{bmatrix}$	$\begin{bmatrix} 0.567 & 0.190 & 0.193 \\ 0.279 & 0.643 & 0.077 \\ 0.000 & 0.073 & 1.016 \end{bmatrix}$	$\gamma = 0.45$ $f = 0.1115$ $s = 4.0$ $t = 0.0228$
广色域	$\begin{bmatrix} 1.4625 & -0.1845 & -0.2734 \\ -0.5228 & 1.4479 & 0.0681 \\ 0.0346 & -0.0958 & 1.2875 \end{bmatrix}$	$\begin{bmatrix} 0.7164 & 0.1010 & 0.1468 \\ 0.2587 & 0.7247 & 0.0166 \\ 0.0000 & 0.0512 & 0.7740 \end{bmatrix}$	$\gamma = N.A.$ $f = N.A.$ $s = N.A.$ $t = N.A.$

表19.2。标准RGB颜色空间的转换 (after (Pascale, 2003))。

这种变换被称为Hunt-Pointer-Estevez变换 (Hunt, 2004) , 用于色度适应变换以及颜色外观建模。

19.2.4 CIE 1976 $L^*a^*b^*$

颜色对手空间的特征在于表示消色差通道 (亮度) 的通道, 以及编码颜色对手的两个通道。这些通道通常是红-绿和黄-蓝通道。这些颜色的对手陈



nels thus encode two chromaticities along one axis, which can have both positive and negative values. For instance, a red-green channel encodes red for positive values and green for negative values. The value zero encodes a special case: neutral which is neither red or green. The yellow-blue channel works in much the same way.

As at least two colors are encoded on each of the two chromatic axes, it is not possible to encode a mixture of red and green. Neither is it possible to encode yellow and blue simultaneously. While this may seem a disadvantage, it is known that the human visual system computes similar attributes early in the visual pathway. As a result, humans are not able to perceive colors that are simultaneously red and green, or yellow and blue. We do not see anything resembling reddish-green, or yellowish-blue. We are, however, able to perceive mixtures of colors such as yellowish-red (orange) or greenish-blue, as these are encoded across the chromatic channels.

The most relevant color opponent system for computer graphics is the CIE 1976 $L^*a^*b^*$ color model. It is a perceptually more or less uniform color space, useful, among other things, for the computation of color differences. It is also known as CIELAB.

The input to CIELAB are the stimulus (X, Y, Z) tristimulus values as well as the tristimulus values of a diffuse white reflecting surface that is lit by a known illuminant, (X_n, Y_n, Z_n) . CIELAB therefore goes beyond being an ordinary color space, as it takes into account a patch of color in the context of a known illumination. It can thus be seen as a rudimentary color appearance space.

The three channels defined in CIELAB are L^* , a^* , and b^* . The L^* channel encodes the lightness of the color, i.e., the perceived reflectance of a patch with tristimulus value (X, Y, Z) . The a^* and b^* are chromatic opponent channels. The transform between XYZ and CIELAB is given by

$$\begin{bmatrix} L^* \\ a^* \\ b^* \end{bmatrix} = \begin{bmatrix} 0 & 116 & 0 & -16 \\ 500 & -500 & 0 & 0 \\ 0 & 200 & -200 & 0 \end{bmatrix} \begin{bmatrix} f(X/X_n) \\ f(Y/Y_n) \\ f(Z/Z_n) \\ 1 \end{bmatrix}.$$

The function f is defined as

$$f(r) = \begin{cases} \sqrt[3]{r} & \text{for } r > 0.008856, \\ 7.787r + \frac{16}{116} & \text{for } r \leq 0.008856. \end{cases}$$

As can be seen from this formulation, the chromatic channels do depend on the luminance Y . Although this is perceptually accurate, it means that we cannot plot the values of a^* and b^* in a chromaticity diagram. The lightness L^* is normalized



因此, nels沿着一个轴编码两个色度, 可以具有正值和负值。例如, 红绿通道将红色编码为正值, 将绿色编码为负值。值零编码一个特殊情况: neutral, 它既不是红色也不是绿色。黄蓝色通道的工作方式大致相同。

由于在两个色轴的每一个上编码至少两种颜色, 因此不可能编码红色和绿色的混合物。也不可能同时编码黄色和蓝色。虽然这可能看起来是一个缺点, 但已知人类视觉系统在视觉路径方式的早期计算类似的属性。因此, 人类无法感知同时为红色和绿色, 或黄色和蓝色的颜色。我们没有看到任何类似红绿色或黄蓝色的东西。然而, 我们能够感知到黄红色(橙色)或绿蓝色等颜色的混合物, 因为这些颜色是通过彩色通道编码的。

与计算机图形学最相关的颜色模型系统是CIE1976L a b 颜色模型。它是一个感知上或多或少均匀的颜色空间, 除其他外, 对于计算颜色差异很有用。它也被称为CIELAB。

Cielab的输入是刺激 (X, Y, Z) 三刺激值以及由已知的发光体 (X_n, Y_n, Z_n) 照亮的漫反射白色表面的三刺激值。因此, CIELAB不仅仅是一个普通的颜色空间, 因为它考虑到一个已知的照明环境中的一个补丁的颜色。因此, 它可以被看作是一个基本的颜色外观空间。

CIELAB中定义的三个通道是 L^* , a^* 和 b^* 。 L^* 通道编码颜色的亮度, 即具有三刺激值 (X, Y, Z) 的贴片的感知反射率。 a^* 和 b^* 是半音阶对手通道。XYZ和CIELAB之间的变换由下式给出

$$\begin{bmatrix} L^* \\ a^* \\ b^* \end{bmatrix} = \begin{bmatrix} 0 & 116 & 0 & -16 \\ 500 & -500 & 0 & 0 \\ 0 & 200 & -200 & 0 \end{bmatrix} \begin{bmatrix} f(X/X_n) \\ f(Y/Y_n) \\ f(Z/Z_n) \\ 1 \end{bmatrix}.$$

函数f定义为

$$f(r) = \begin{cases} \sqrt[3]{r} & \text{for } r > 0.008856, \\ 7.787r + \frac{16}{116} & \text{for } r \leq 0.008856. \end{cases}$$

从该公式可以看出, 色度通道确实取决于亮度Y。虽然这在感知上是准确的, 但这意味着我们无法在色度图中绘制 a^{∞} 和 b^{∞} 的值。明度 L^* 归一化

between 0 and 100 for black and white. Although the a^* and b^* channels are not explicitly constrained, they are typically in the range $[-128, 128]$.

As CIELAB is approximately perceptually linear, it is possible to take two colors, convert them to CIELAB, and then estimate the perceived color difference by computing the Euclidean distance between them. This leads to the following color difference formula:

$$\Delta E_{ab}^* = \left[(\Delta L^*)^2 + (\Delta a^*)^2 + (\Delta b^*)^2 \right]^{1/2}.$$

The letter E stands for difference in sensation (in German, Empfindung) (Judd, 1932).

Finally, the inverse transform between CIELAB and XYZ is given by

$$X = X_n \begin{cases} \left(\frac{L^*}{116} + \frac{a^*}{500} + \frac{16}{116} \right)^3 & \text{if } L^* > 7.9996, \\ \frac{1}{7.787} \left(\frac{L^*}{116} + \frac{a^*}{500} \right) & \text{if } L^* \leq 7.9996, \end{cases}$$

$$Y = Y_n \begin{cases} \left(\frac{L^*}{116} + \frac{16}{116} \right)^3 & \text{if } L^* > 7.9996, \\ \frac{1}{7.787} \frac{L^*}{116} & \text{if } L^* \leq 7.9996, \end{cases}$$

$$Z = Z_n \begin{cases} \left(\frac{L^*}{116} - \frac{b^*}{200} + \frac{16}{116} \right)^3 & \text{if } L^* > 7.9996, \\ \frac{1}{7.787} \left(\frac{L^*}{116} - \frac{b^*}{200} \right) & \text{if } L^* \leq 7.9996. \end{cases}$$

19.3 Chromatic Adaptation

The CIELAB color space just described takes as input both a tristimulus value of the stimulus and the tristimulus value of light reflected off a white diffuse patch. As such, it forms the beginnings of a system in which the viewing environment is taken into account.

The environment in which we observe objects and images has a large influence on how we perceive those objects. The range of viewing environments that we encounter in daily life is very large, from sunlight to starlight and from candlelight to fluorescent light. The lighting conditions not only constitute a very large range in the amount of light that is present, but also vary greatly in the color of the emitted light.

黑色和白色在0到100之间。虽然 a^* 和 b^* 通道没有明确约束，但它们通常在[128 128]范围内。

由于CIELAB近似为感知线性，因此可以取两种颜色，将它们转换为CIELAB，然后通过计算它们之间的欧几里德距离来估计感知的颜色差异。这导致以下色差公式：

$$\Delta E_{ab}^* = \left[(\Delta L^*)^2 + (\Delta a^*)^2 + (\Delta b^*)^2 \right]^{1/2}.$$

字母E代表感觉差异（德语，Empfindung）（Judd, 1932）。

最后，cielab和XYZ之间的逆变换由下式给出

$$X = X_n \begin{cases} \left(\frac{L^*}{116} + \frac{a^*}{500} + \frac{16}{116} \right)^3 & \text{if } L^* > 7.9996, \\ \frac{1}{7.787} \left(\frac{L^*}{116} + \frac{a^*}{500} \right) & \text{if } L^* \leq 7.9996, \end{cases}$$

$$Y = Y_n \begin{cases} \left(\frac{L^*}{116} + \frac{16}{116} \right)^3 & \text{if } L^* > 7.9996, \\ \frac{1}{7.787} \frac{L^*}{116} & \text{if } L^* \leq 7.9996, \end{cases}$$

$$Z = Z_n \begin{cases} \left(\frac{L^*}{116} - \frac{b^*}{200} + \frac{16}{116} \right)^3 & \text{if } L^* > 7.9996, \\ \frac{1}{7.787} \left(\frac{L^*}{116} - \frac{b^*}{200} \right) & \text{if } L^* \leq 7.9996. \end{cases}$$

19.3 半音适应

刚才描述的cielab颜色空间将刺激的三刺激值和白色漫反射贴片反射的光的三刺激值作为输入。因此，它形成了一个系统的开端，在这个系统中，观看环境被考虑在内。

我们观察物体和图像的环境对我们如何感知这些物体有很大的影响。我们在日常生活中遇到的观看环境范围非常大，从阳光到星光，从烛光到荧光灯。照明条件不仅在存在的光量中构成非常大的范围，而且在发射光的颜色中也有很大的变化。

The human visual system accommodates these changes in the environment through a process called adaptation. Three different types of adaptation can be distinguished, namely light adaptation, dark adaptation, and chromatic adaptation. Light adaptation refers to the changes that occur when we move from a very dark to a very light environment. When this happens, at first we are dazzled by the light, but soon we adapt to the new situation and begin to distinguish objects in our environment. Dark adaptation refers to the opposite—when we go from a light environment to a dark environment. At first, we see very little, but after a given amount of time, details will start to emerge. The time needed to adapt to the dark is generally much longer than for light adaptation.

Chromatic adaptation refers to our ability to adapt, and largely ignore, variations in the color of the illumination. Chromatic adaptation is, in essence, the biological equivalent of the white balancing operation that is available on most modern cameras. The human visual system effectively normalizes the viewing conditions to present a visual experience that is fairly consistent. Thus, we exhibit a certain amount of color constancy: object reflectances appear relatively constant despite variations in illumination.

Although we are able to largely ignore changes in viewing environment, we are not able to do so completely. For instance, colors appear much more colorful on a sunny day than they do on a cloudy day. Although the appearances have changed, we do not assume that object reflectances themselves have actually changed their physical properties. We thus understand that the lighting conditions have influenced the overall color appearance.

Nonetheless, color constancy does apply to chromatic content. Chromatic adaptation allows white objects to appear white for a large number of lighting conditions, as shown in Figure 19.9.

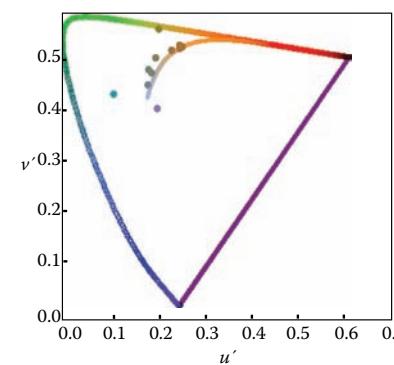


Figure 19.9. A series of light sources plotted in the CIE $u'v'$ chromaticity diagram. A white piece of paper illuminated by any of these light sources maintains a white color appearance.

人类视觉系统通过称为适应的过程来适应环境中的这些变化。可以区分三种不同类型的适应，即光适应、暗适应和半音适应。光适应是指当我们从一个非常黑暗的环境移动到一个非常光明的环境时发生的变化。当这种情况发生时，起初我们被光线所眩目，但很快我们就适应了新的情况，并开始区分我们环境中的物体。黑暗适应指的是相反的—当我们从光环境到黑暗环境时。起初，我们看到的很少，但经过一段时间后，细节将开始出现。适应黑暗所需的时间通常比光适应要长得多。

色度适应是指我们能够适应，并且在很大程度上忽略照明颜色的变化。从本质上讲，色度适应是大多数现代相机上可用的白平衡操作的生物等价物。人类视觉系统有效地使观看条件标准化，以呈现相当一致的视觉体验。因此，我们排除了一定量的颜色恒常性：尽管照明变化，物体反射看起来相对恒定。

虽然我们能够在很大程度上忽略查看环境的变化，但我们无法完全做到这一点。例如，在阳光明媚的日子里，颜色看起来比在阴天更冷或更多。虽然外观发生了变化，但我们并不认为物体反射本身实际上已经改变了它们的物理性质。因此，我们知道照明条件影响了整体颜色外观。

尽管如此，颜色恒定性确实适用于色度内容。色度适应允许白色物体在大量照明条件下呈现白色，如图19.9所示。

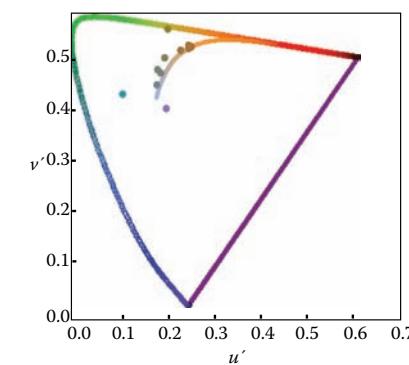


图19.9。在CIE $u'v'$ 色度图中绘制了一系列光源。由任何这些光源照明的白色纸片保持白色外观。

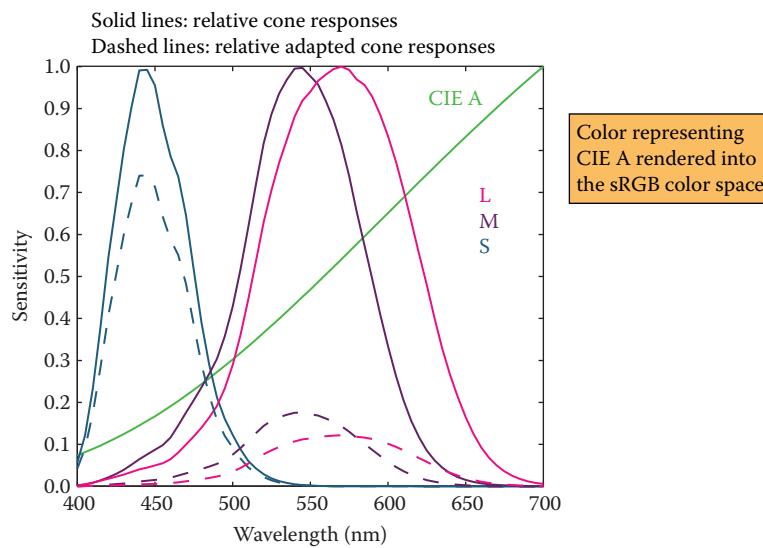


Figure 19.10. An example of von Kries-style independent photoreceptor gain control. The relative cone responses (solid line) and the relative adapted cone responses to CIE illuminant A (dashed) are shown. The separate patch of color represents CIE illuminant A rendered into the sRGB color space.

Computational models of chromatic adaptation tend to focus on the gain control mechanism in the cones. One of the simplest models assumes that each cone adapts independently to the energy that it absorbs. This means that different cone types adapt differently dependent on the spectrum of the light being absorbed. Such adaptation can then be modeled as an adaptive and independent rescaling of the cone signals:

$$L_a = \alpha L,$$

$$M_a = \beta M,$$

$$S_a = \gamma S,$$

where (L_a, M_a, S_a) are the chromatically adapted cone signals, and α , β , and γ are the independent gain controls which are determined by the viewing environment. This type of independent adaptation is also known as von-Kries adaptation. An example is shown in Figure 19.10.

The adapting illumination can be measured off a white surface in the scene. In the ideal case, this would be a Lambertian surface. In a digital image, the adapting illumination can also be approximated as the maximum tristimulus values of the scene. The light measured or computed in this manner is the adapting white, given by (L_w, M_w, S_w) . Von Kries adaptation is then simply a scaling by the reciprocal

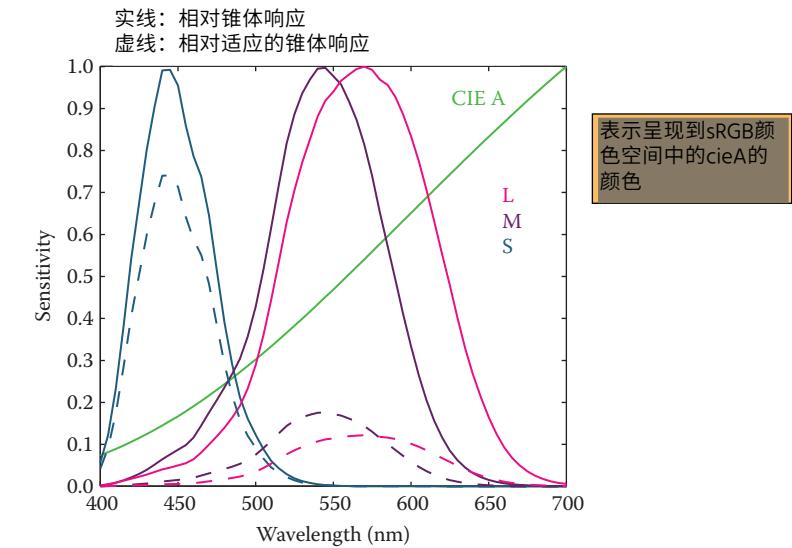


图19.10。冯克里斯式独立感光体增益控制的一个例子。示出了对cie光源A的相对锥响应（实线）和相对适配锥响应（虚线）。颜色的单独补丁表示呈现到sRGB颜色空间中的cie光源A。

半音适应的计算模型倾向于关注锥体中的增益控制机制。其中一个最简单的模型假设每个锥体独立地适应它吸收的能量。这意味着不同的锥体类型根据被吸收的光的光谱不同地适应。这样的适应可以被建模为锥体信号的自适应和独立的重新缩放：

$$L_a = \alpha L,$$

$$M_a = \beta M,$$

$$S_a = \gamma S,$$

其中 (L_a, M_a, S_a) 是色度适应的锥体信号， α 、 β 和 γ 是独立的增益控制，由观察环境决定。这种类型的独立适应也被称为von-Kries适应。一个例子如图19.10所示。

可以在场景中的白色表面上测量自适应照明。在理想的情况下，这将是朗伯表面。在数字图像中，自适应照明也可以近似为场景的最大三刺激值。以这种方式测量或计算的光是适应的白色，由 (L_w, M_w, S_w) 给出。冯*克里斯的适应只是一个倒数的缩放



of the adapting white, carried out in cone response space:

$$\begin{bmatrix} L_a \\ M_a \\ S_a \end{bmatrix} = \begin{bmatrix} \frac{1}{L_w} & 0 & 0 \\ 0 & \frac{1}{M_w} & 0 \\ 0 & 0 & \frac{1}{S_w} \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}.$$

In many cases, we are interested in what stimulus should be generated under one illumination to match a given color under a different illumination. For example, if we have a colored patch illuminated by daylight, we may ask ourselves what tristimulus values should be generated to create a matching color patch that will be illuminated by incandescent light.

We are thus interested in computing corresponding colors, which can be achieved by cascading two chromatic adaptation calculations. In essence, the previously mentioned von Kries transform divides out the adapting illuminant—in our example, the daylight illumination. If we subsequently multiply in the incandescent illuminant, we have computed a corresponding color. If the two illuminants are given by $(L_{w,1}, M_{w,1}, S_{w,1})$ and $(L_{w,2}, M_{w,2}, S_{w,2})$, the corresponding color (L_c, M_c, S_c) is given by

$$\begin{bmatrix} L_c \\ M_c \\ S_c \end{bmatrix} = \begin{bmatrix} L_{w,2} & 0 & 0 \\ 0 & M_{w,2} & 0 \\ 0 & 0 & S_{w,2} \end{bmatrix} \begin{bmatrix} \frac{1}{L_{w,1}} & 0 & 0 \\ 0 & \frac{1}{M_{w,1}} & 0 \\ 0 & 0 & \frac{1}{S_{w,1}} \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}.$$

There are several more complicated and, therefore, more accurate chromatic adaptation transform in existence (Reinhard et al., 2008). However, the simple von Kries model remains remarkably effective in modeling chromatic adaptation and can thus be used to achieve white balancing in digital images.

The importance of chromatic adaptation in the context of rendering, is that we have moved one step closer to taking into account the viewing environment of the observer, without having to correct for it by adjusting the scene and rerendering our imagery. Instead, we can model and render our scenes, and then, as an image postprocess, correct for the illumination of the viewing environment. To ensure that white balancing does not introduce artifacts, however, it is important to ensure that the image is rendered to a floating-point format. If rendered to traditional 8-bit image formats, the chromatic adaptation transform may amplify quantization errors.



适应的白色，在锥体响应空间中进行：

$$\begin{bmatrix} L_a \\ M_a \\ S_a \end{bmatrix} = \begin{bmatrix} \frac{1}{L_w} & 0 & 0 \\ 0 & \frac{1}{M_w} & 0 \\ 0 & 0 & \frac{1}{S_w} \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}.$$

在许多情况下，我们感兴趣的是在一种照明下应该产生什么刺激，以匹配不同照明下的给定颜色。例如，如果我们有一个被日光照明的彩色补丁，我们可能会问自己应该产生什么三刺激值来创建一个匹配的颜色补丁，该补丁将被白炽灯照亮。

因此，我们对计算相应的颜色感兴趣，这可以通过级联两个色度适应计算来实现。从本质上讲，前面提到的冯克里斯变换划分出适应的光源—在我们的例子中，日光照明。如果我们随后在印加下降光源中相乘，我们已经计算了相应的颜色。如果两个illuminant由 $(L_w, 1, M_w, 1, S_w, 1)$ 和 $(L_w, 2, M_w, 2, S_w, 2)$ 给出，则相应的颜色 (L_c, M_c, S_c) 由

$$\begin{bmatrix} L_c \\ M_c \\ S_c \end{bmatrix} = \begin{bmatrix} L_{w,2} & 0 & 0 \\ 0 & M_{w,2} & 0 \\ 0 & 0 & S_{w,2} \end{bmatrix} \begin{bmatrix} \frac{1}{L_{w,1}} & 0 & 0 \\ 0 & \frac{1}{M_{w,1}} & 0 \\ 0 & 0 & \frac{1}{S_{w,1}} \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}.$$

存在几个更复杂，因此更准确的色度适应变换 (Reinhard et al. 2008)。然而，简单的vonKries模型在建模色度适应方面仍然非常有效，因此可以用于实现数字图像中的白平衡。

在渲染的背景下，色彩适应的重要性在于，我们已经向考虑观察者的观看环境更近了一步，而不必通过调整场景和重新呈现我们的图像来纠正它。相反，我们可以建模和渲染我们的场景，然后，作为一个图像后处理，纠正观看环境的照明。但是，要确保白平衡不会引入伪影，必须确保将图像呈现为浮点格式。如果呈现为传统的8位图像格式，则色度适应变换可能放大量化误差。

19.4 Color Appearance

While colorimetry allows us to accurately specify and communicate color in a device-independent manner, and chromatic adaptation allows us to predict color matches across changes in illumination, these tools are still insufficient to describe what colors actually look like.

To predict the actual perception of an object, we need to know more information about the environment and take that information into account. The human visual system is constantly adapting to its environment, which means that the perception of color will be strongly influenced by such changes. Color appearance models take into account measurements of the stimulus itself, as well as the viewing environment. This means that the resulting description of color is independent of viewing condition.

The importance of color appearance modeling can be seen in the following example. Consider an image being displayed on an LCD screen. When making a print of the same image and viewing it in a different context, more often than not the image will look markedly different. Color appearance models can be used to predict the changes required to generate an accurate cross-media color reproduction (Fairchild, 2005).

Although color appearance modeling offers important tools for color reproduction, actual implementations tend to be relatively complicated and cumbersome in practical use. It can be anticipated that this situation may change over time. However, until then, we leave their description to more specialized textbooks (Fairchild, 2005).

Notes

Of all the books on color theory, Reinhard et al.'s work (Reinhard et al., 2008) is most directly geared toward engineering disciplines, including computer graphics, computer vision, and image processing. Other general introductions to color theory are given by Berns (Berns, 2000) and Stone (Stone, 2003). Wyszecki and Stiles have produced a comprehensive volume of data and formulae, forming an indispensable reference work (Wyszecki & Stiles, 2000). For color reproduction, we recommend Hunt's book (Hunt, 2004). Color appearance models are comprehensively described in Fairchild's book (Fairchild, 2005). For color issues related to video and HDTV Poynton's book is essential (Poynton, 2003).

虽然比色法使我们能够以与设备无关的方式准确指定和传达颜色，而色度适应使我们能够预测照明变化的颜色匹配，但这些工具仍然不足以描述实际

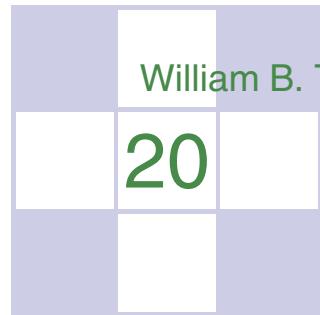
为了预测物体的实际感知，我们需要了解更多关于环境的信息，并将这些信息考虑在内。人类的视觉系统不断适应环境，这意味着色彩的每个感受都会受到这种变化的强烈影响。颜色外观模型考虑刺激本身的测量，以及视图ing环境。这意味着由此产生的颜色描述与观看条件无关。

颜色外观建模的重要性可以在下面的例子中看到。考虑在LCD屏幕上显示的图像。当打印相同的图像并在不同的上下文中查看时，图像通常看起来会明显不同。颜色外观模型可用于预测生成准确的跨媒体颜色再现所需的变化 (Fairchild, 2005)。

虽然色彩外观建模为色彩再现提供了重要的工具，但实际实现往往相对复杂，并且在实际使用中有一些。可以预期，这种情况可能随时间而改变。然而，在此之前，我们将它们的描述留给更专业的教科书 (Fairchild, 2005)。

Notes

在所有关于色彩理论的书籍中，Reinhard等人。的工作 (Reinhard et al., 2008) 最直接面向工程学科，包括计算机图形学，计算机视觉和图像处理。颜色理论的其他一般介绍由Berns (Berns, 2000) 和Stone (Stone, 2003) 给出。Wyszecki和Stiles产生了大量的数据和公式，形成了不可或缺的参考工作 (Wyszecki & Stiles, 2000)。对于色彩再现，我们推荐Hunt的书 (Hunt, 2004)。在Fairchild的书 (Fairchild, 2005) 中全面描述了颜色外观模型。对于与视频和HDTV有关的色彩问题，Poynton的书是必不可少的 (Poynton, 2003)。



Visual Perception

The ultimate purpose of computer graphics is to produce images for viewing by people. Thus, the success of a computer graphics system depends on how well it conveys relevant information to a human observer. The intrinsic complexity of the physical world and the limitations of display devices make it impossible to present a viewer with the identical patterns of light that would occur when looking at a natural environment. When the goal of a computer graphics system is physical realism, the best we can hope for is that the system be *perceptually effective*: displayed images should “look” as intended. For applications such as technical illustration, it is often desirable to visually highlight relevant information and perceptual effectiveness becomes an explicit requirement.

Artists and illustrators have developed empirically a broad range of tools and techniques for effectively conveying visual information. One approach to improving the perceptual effectiveness of computer graphics is to utilize these methods in our automated systems. A second approach builds directly on knowledge of the human vision system by using perceptual effectiveness as an optimization criterion in the design of computer graphics systems. These two approaches are not completely distinct. Indeed, one of the first systematic examinations of visual perception is found in the notebooks of Leonardo da Vinci.

The remainder of this chapter provides a partial overview of what is known about visual perception in people. The emphasis is on aspects of human vision that are most relevant to computer graphics. The human visual system is extremely complex in both its operation and its architecture. A chapter such as this



视觉感知

计算机图形学的最终目的是产生供人们观看的图像。因此，计算机图形系统的成功取决于它向人类观察者传达相关信息的能力。物理世界的内在复杂性和显示设备的局限性使得观看者无法向自然环境呈现相同的光模式。当计算机图形系统的目标是物理现实主义时，我们所能期望的最好的是系统在感知上是有效的：显示的图像应该“看起来”像预期的那样。对于诸如技术说明的应用，通常希望视觉上突出相关信息并且感知有效性成为明确的要求。

艺术家和插画家凭经验开发了广泛的工具和技术，以有效地传达视觉信息。提高计算机图形学感知有效性的一种方法是在我们的自动化系统中利用这些方法。第二种方法直接建立在人类视觉系统的知识基础上，在计算机图形系统的设计中使用感知有效性作为优化criterion。这两种方法并不完全不同。事实上，在列奥纳多·达·芬奇的笔记本中发现了视觉感知的第一个系统检查之一。

本章的其余部分提供了关于人类视觉感知的部分概述。重点是与计算机图形学最相关的人类视觉方面。人类视觉系统的操作和结构都非常复杂。这样的一章

can at best provide a summary of key points, and it is important to avoid overgeneralizing from what is presented here. More in-depth treatments of visual perception can be found in Wandell (1995) and Palmer (1999); Gregory (1997) and Yantis (2000) provide additional useful information. A good computer vision reference such as Forsyth and Ponce (2002) is also helpful. It is important to note that despite over 150 years of intensive research, our knowledge of many aspects of vision is still very limited and imperfect.

20.1 Vision Science

Light:

- travels far
- travels fast
- travels in straight lines
- interacts with stuff
- bounces off things
- is produced in nature
- has lots of energy

— Steven Shafer

Figure 20.1. The nature of light makes vision a powerful sense.

Vision is generally agreed to be the most powerful of the senses in humans. Vision produces more useful information about the world than does hearing, touch, smell, or taste. This is a direct consequence of the physics of light (Figure 20.1). Illumination is pervasive, especially during the day but also at night due to moonlight, starlight, and artificial sources. Surfaces reflect a substantial portion of incident illumination and do so in ways that are idiosyncratic to particular materials and that are dependent on the shape of the surface. The fact that light (mostly) travels in straight lines through the air allows vision to acquire information from distant locations.

The study of vision has a long and rich history. Much of what we know about the eye traces back to the work of philosophers and physicists in the 1600s. Starting in the mid-1800s, there was an explosion of work by perceptual psychologists exploring the phenomenology of vision and proposing models of how vision might work. The mid-1900s saw the start of modern neuroscience, which investigates both the fine-scale workings of individual neurons and the large-scale architectural organization of the brain and nervous system. A substantial portion of neuroscience research has focused on vision. More recently, computer science has contributed to the understanding of visual perception by providing tools for precisely describing hypothesized models of visual computations and by allowing empirical examination of computer vision programs. The term *vision science* was coined to refer to the multidisciplinary study of visual perception involving perceptual psychology, neuroscience, and computational analysis.

Vision science views the purpose of vision as producing information about objects, locations, and events in the world from imaged patterns of light reaching the viewer. Psychologists use the term *distal stimulus* to refer to the physical world under observation and *proximal stimulus* to refer to the retinal image.¹ Us-

¹In computer vision, the term *scene* is often used to refer to the external world, while the term *image* is used to refer to the projection of the scene onto a sensing plane.

充其量只能提供关键点的总结，重要的是要避免从这里介绍的内容过度概括。视觉感知的更深入治疗可以在Wandell (1995) 和Palmer (1999) 中找到;Gregory (1997) 和Yantis (2000) 提供了额外的有用信息。一个很好的计算机视觉参考，如Forsyth和Ponce (2002) 也很有帮助。值得注意的是，尽管经过150多年的深入研究，我们对视觉的许多方面的知识仍然非常有限和不完善。

20.1 视觉科学

视觉通常被认为是人类最强大的感官。视觉产生更多关于世界的有用信息，而不是听觉

触摸，嗅觉或味觉。这是光物理学的直接结果（Figure 20.1）。照明是普遍的，特别是在白天，但也在夜间由于月光，星光和人工来源。表面反射入射照明的很大一部分，并且以与particular材料特殊的方式进行反射，并且取决于表面的形状。光（大部分）在空中直线传播的事实允许视觉从远处获取信息。

Light:

- 远行•快速旅行•直线旅行•与事物相互作用•反弹事物•在自然界中产生•有很多能量

光的本质使视觉成为一种强大的感觉。

视觉的研究有着悠久而丰富的历史。我们所知道的关于眼睛的很多东西都可以追溯到1600年代哲学家和物理学家的工作。从19世纪中期开始，感知心理的chologists探索视觉现象学并提出视觉如何工作的模型。在20世纪中期，现代神经科学开始了研究个体神经元的精细规模运作以及大脑和神经系统的大规模建筑组织的研究。神经科学的研究的很大一部分集中在视觉上。最近，计算机科学提供了精确描述视觉计算假设模型的工具，并允许对计算机视觉程序进行实证检查，从而促进了对视觉感知的理解。视觉科学一词是指涉及感知心理学、神经科学和计算分析的视觉感知的多学科研究。

视觉科学将视觉的目的视为从到达观察者的成像光模式中产生有关世界上物体，位置和事件的信息。心理学家使用术语远端刺激来指代观察下的物理世界，近端刺激来指代视网膜图像。1美国

1在计算机视觉中，术语场景通常用于指代外部世界，而术语图像用于指代场景到感测平面上的投影。

ing this terminology, the function of vision is to generate a description of aspects of the distal stimulus given the proximal stimulus. Visual perception is said to be *veridical* when the description that is produced accurately reflects the real world. In practice, it makes little sense to think of these descriptions of objects, locations, and events in isolation. Rather, vision is better understood in the context of the motor and cognitive functions that it serves.

20.2 Visual Sensitivity

Vision systems create descriptions of the visual environment based on properties of the incident illumination. As a result, it is important to understand what properties of incident illumination the human vision system can actually detect. One critical observation about the human vision system is that it is primarily sensitive to *patterns* of light rather than being sensitive to the absolute magnitude of light energy. The eye does not operate as a photometer. Instead, it detects spatial, temporal, and spectral patterns in the light imaged on the retina and information about these patterns of light form the basis for all of visual perception.

There is a clear ecological utility to the vision system's sensitivity to variations in illumination over space and time. Being able to accurately sense changes in the environment is crucial to our survival.² A system which measures changes in light energy rather than the magnitude of the energy itself also makes engineering sense, since it makes it easier to detect patterns of light over large ranges in light intensity. It is a good thing for computer graphics that vision operates in this manner. Display devices are physically limited in their ability to project light with the power and dynamic range typical of natural scenes. Graphical displays would not be effective if they needed to produce the identical patterns of light as the corresponding physical world. Fortunately, all that is required is that displays be able to produce similar patterns of spatial and temporal change to the real world.

20.2.1 Brightness and Contrast

In bright light, the human visual system is capable of distinguishing gratings consisting of high-contrast parallel light and dark bars as fine as 50–60 cycles/degree. (In this case, a “cycle” consists of an adjacent pair of light and dark bars.)

²It is sometimes said that the primary goals of vision are to support eating, avoiding being eaten, reproduction, and avoidance of catastrophe while moving. Thinking about vision as a goal-directed activity is often useful, but needs to be done so at a more detailed level.

在这个术语中，视觉的功能是产生给定近端刺激的远端刺激方面的描述。当所产生的描述准确地反映了现实世界时，视觉感知被认为是真实的。在实践中，孤立地考虑对象，位置和事件的这些描述是没有意义的。相反，视觉在它所服务的运动和认知功能的背景下得到更好的理解。

20.2 视觉敏感度

视觉系统根据入射照明的属性创建视觉环境的描述。因此，了解人类视觉系统实际可以检测到的入射照明工具非常重要。关于人类视觉系统的一个关键观察是，它主要对光的模式敏感，而不是对光能的绝对量值敏感。眼睛不作为光度计操作。相反，它检测在视网膜上成像的光中的空间，时间和光谱模式，并且关于这些光模式的信息构成了所有视觉感知的基础。

视觉系统对空间和时间内照明变化的敏感性具有明显的生态效用。能够准确地感知环境的变化对我们的生存至关重要。测量光能量变化而不是能量本身大小的系统也具有工程意义，因为它可以更容易地检测光强度较大范围内的光模式。视觉以这种方式运行对于计算机图形学来说是一件好事。显示设备以自然场景典型的功率和动态范围投射光线的能力受到物理限制。如果图形显示需要产生与相应的物理世界相同的光模式，则不会有效。幸运的是，所需要的只是显示器能够产生与现实世界相似的空间和时间变化模式。

20.2.1 亮度和对比度

在明亮的光线下，人类视觉系统能够区分由高对比度平行光栅和细至50–60度的暗条组成的光栅。（在这种情况下，一个“循环”由相邻的一对明暗条组成。）

²有人说，视力的主要目标是支持进食，避免被吃掉，繁殖，以及在移动时避免灾难。将视觉视为目标导向的活动通常是有用的，但需要在更详细的层面上进行。

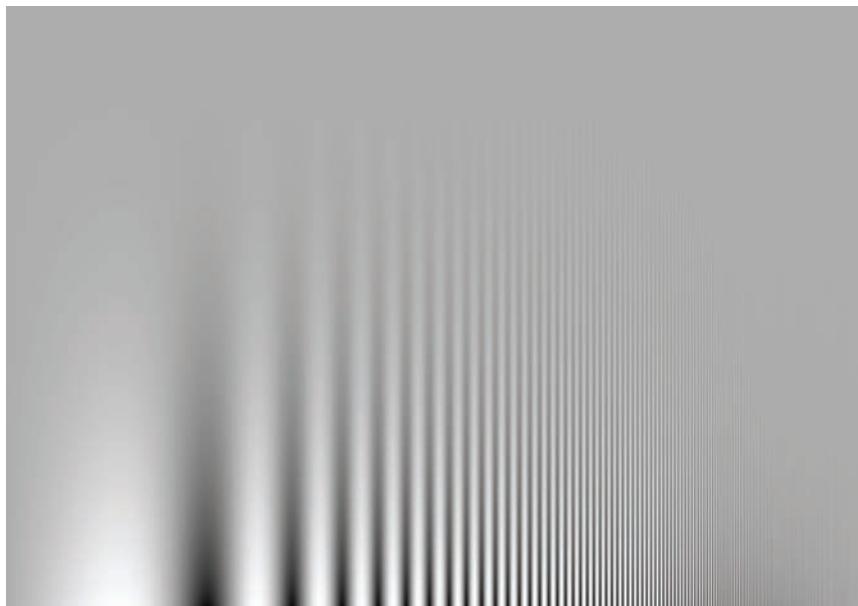


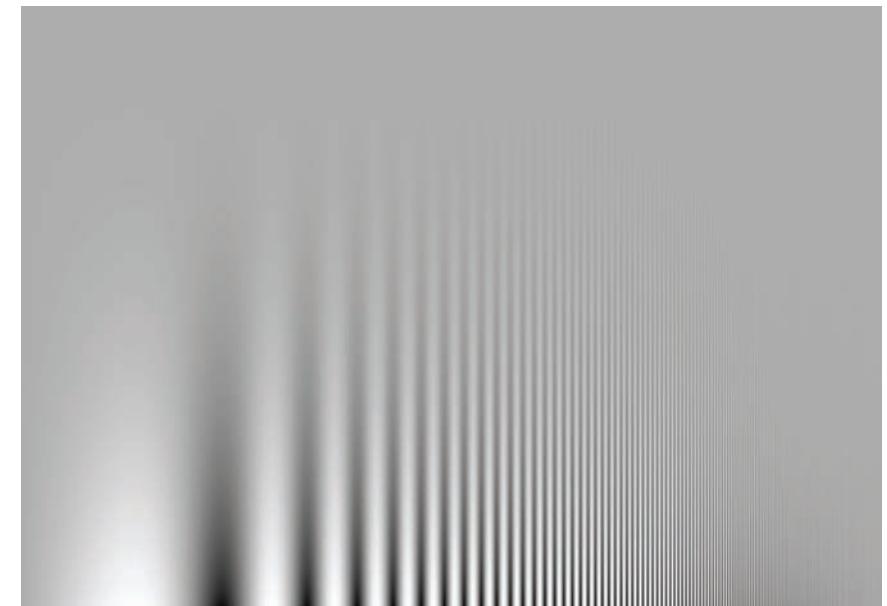
Figure 20.2. The contrast between stripes increases in a constant manner from top to bottom, yet the threshold of visibility varies with frequency.

For comparison, the best currently available LCD computer monitor, at a normal viewing distance, can display patterns as fine as about 20 cycles/degree. The minimum contrast difference at an edge detectable by the human visual system in bright light is about 1% of the average luminance across the edge. In most 8-bit displays, differences of a single gray level are often noticeable over at least a portion of the range of intensities due to the nature of the mapping from gray levels to actual display luminance.

Characterizing the ability of the visual system to detect fine scale patterns (*visual acuity*) and to detect changes in brightness is considerably more complicated than for cameras and similar image acquisition devices. As shown in Figure 20.2, there is an interaction between contrast and acuity in human vision. In the figure, the scale of the pattern decreases from left to right while the contrast increases from top to bottom. If you view the figure at a normal viewing distance, it will be clear that the lowest contrast at which a pattern is visible is a function of the spatial frequency of the pattern.

There is a linear relationship between the intensity of light L reaching the eye from a particular surface point in the world, the intensity of light I illuminating that surface point, and the reflectivity R of the surface at the point being observed:

$$L = \alpha I \cdot R, \quad (20.1)$$



条纹之间的对比度从上到下以恒定的方式增加，但可见度阈值随频率而变化。

为了比较，最好的目前可用的LCD计算机监视器，在正常的观看距离下，可以显示精细到大约200度的图案。人类视觉系统在强光下可检测到的边缘处的最小对比度差约为整个边缘的平均亮度的1%。在大多数8位显示器中，由于从灰度级到实际显示亮度的映射的性质，单个灰度级的差异通常在强度范围的至少一个范围内是明显的。

表征视觉系统检测精细尺度模式 (visual acuity) 和检测亮度变化的能力比相机和类似图像采集设备要复杂得多。如图20.2所示，人的视觉中存在对比度和敏锐度之间的相互作用。在图中，图案的比例从左到右减小，而对比度从上到下增加。如果以正常观看距离观看图，则将清楚的是图案可见的最低对比度是图案的空间频率的函数。

在世界上从特定表面点到达眼睛的光 I 的强度、照射该表面点的光 I 的强度、以及在被观察的点处的表面的反射率 R 之间存在线性关系：

$$L = \alpha I \cdot R, \quad (20.1)$$



Figure 20.3. *Lightness constancy.* Cast a shadow over one of the patterns with your hand and notice that the apparent brightness of the two center squares remains nearly the same.

where α is dependent on the relationship between the surface geometry, the pattern of incident illumination, and the viewing direction. While the eye is only able to directly measure L , human vision is much better at estimating R than L . To see this, view Figure 20.3 in bright direct light. Use your hand to shadow one of the patterns, leaving the other directly illuminated. While the light reflected off of the two patterns will be significantly different, the apparent brightness of the two center squares will seem nearly the same. The term *lightness* is often used to describe the apparent brightness of a surface, as distinct from its actual luminance. In many situations, lightness is invariant to large changes in illumination, a phenomenon referred to as *lightness constancy*.

The mechanisms by which the human visual system achieves lightness constancy are not well understood. As shown in Figure 20.2, the vision system is relatively insensitive to slowly varying patterns of light, which may serve to discount the effects of slowly varying illumination. Apparent brightness is affected by the brightness of surrounding regions (Figure 20.4). This can aid lightness constancy when regions are illuminated dissimilarly. While this *simultaneous contrast* effect is often described as a modification of the perceived lightness of



图20.3。轻盈恒常。用手在其中一个图案上投下阴影，并注意到两个中心方块的表观亮度几乎保持不变。

其中 α 取决于表面几何形状、入射照明的拍子和观察方向之间的关系。虽然眼睛只能直接测量 L ，但人类视觉在估计 R 方面比 L 要好得多。为了看到这一点，请在明亮的直射光下查看图20.3。用你的手阴影其中一个图案，留下另一个直接照亮。虽然两个图案反射的光将显着不同，但两个中心方块的表观亮度将看起来几乎相同。亮度一词通常用来描述表面的表观亮度，与其实际亮度不同。在许多情况下，亮度不变于照明的大变化，这种现象称为亮度恒定性。

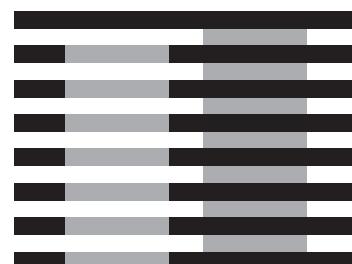
人类视觉系统实现亮度和稳定性的机制还没有得到很好的理解。如图20.2所示，视觉系统对缓慢变化的光模式相对不敏感，这可能有助于计算缓慢变化的照明的影响。表观亮度受周围区域亮度的影响（图20.4）。当区域被不同地照亮时，这可以帮助亮度恒定。虽然这种同时的对比效应通常被描述为对



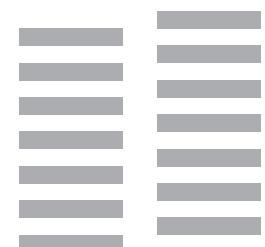
Figure 20.4. (a) Simultaneous contrast: the apparent brightness of the center bar is affected by the brightness of the surrounding area; (b) The same bar without a variable surround.



(a) 同时对比度：中心条的表观亮度受周围区域亮度的影响；(b) 没有可变环绕的相同条。



(a)



(b)

Figure 20.5. The Munker-White illusion shows the complexity of simultaneous contrast. In Figure 20.4, the central region looked lighter when the surrounding area was darker. In (a), the gray strips on the left look *lighter* than the gray strips on the right, even though they are nearly surrounded by regions of white; (b) shows the gray strips without the black lines.

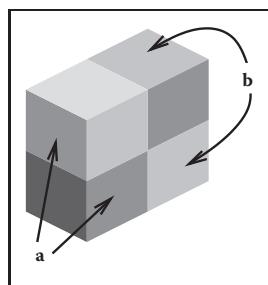


Figure 20.6. The perception of lightness is affected by the perception of 3D structure. The two surfaces marked (a) have the same brightness, as do the two surfaces marked (b) (after Adelson (1999)).

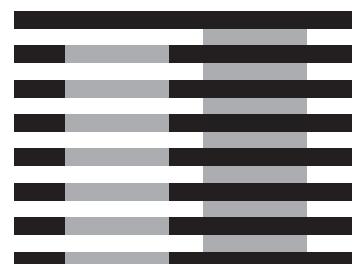


(a)

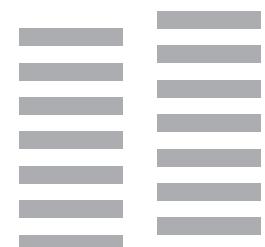


(b)

Figure 20.7. (a) Original gray scale image, (b) image *edges*, which are lines of high spatial variability in some direction.

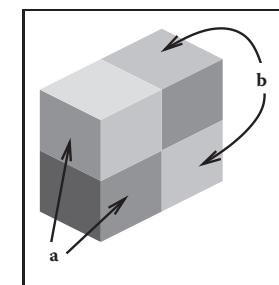


(a)



(b)

图20.5。 Munker-White幻觉显示了同时对比的复杂性。在图20.4中，当周围区域较暗时，中心区域看起来较亮。在 (a) 中，左侧的灰色条带看起来比右侧的灰色条带更轻，尽管它们几乎被白色区域包围；(b) 显示没有黑色线条的灰色条带。



亮度的感知受三维结构感知的影响。标记 (a) 的两个表面具有相同的亮度，标记 (b) 的两个表面也是如此（在 Adelson (1999) 之后）。



(a)



(b)

图20.7。 (a) 原始灰度图像，(b) 图像边缘，它们是在某些方向上具有高度空间可变性的线条。



Figure 20.8. The visual system sometimes sees “edges” even when there are no sharp discontinuities in brightness, as is the case at the right side of the central pattern in this image.

image properties. The vision system has very little ability, however, to detect spatial discontinuities in color when not accompanied by differences in one of these other properties.

Perception of edges seems to interact with perception of form. While edges give the visual system the information it needs to recognize shapes, slowly varying brightness can appear as a sharp edge if the resulting edge creates a more complete form (Figure 20.8). Figure 20.9 shows a *subjective contour*, an extreme form of this effect in which a closed contour is seen even though no such contour exists in the actual image. Finally, the vision system’s sensitivity to edges also appears to be part of the mechanism involved in lightness perception. Note that the region enclosed by the subjective contour in Figure 20.9 appears a bit brighter than the surrounding area of the page. Figure 20.10 shows a different interaction between edges and lightness. In this case, a particular brightness profile at the edge has a dramatic effect on the apparent brightness of the surfaces to either side of the edge.

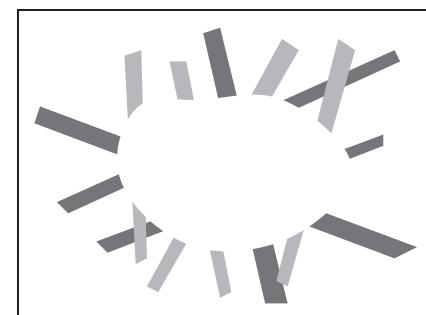


Figure 20.9. Sometimes, the visual system will “see” subjective contours without any associated change in brightness.



视觉系统有时会看到“边缘”，即使在亮度上没有尖锐的不连续点，就像在图像的中心图案的右侧一样。

图像属性。然而，当不伴随这些其他属性之一的差异时，视觉系统检测颜色的空间不连续性的能力很小。

对边缘的感知似乎与对形式的感知相互作用。虽然边缘为视觉系统提供了识别形状所需的信息，但如果生成的边缘创建了更完整的形状，则缓慢变化的亮度可能会显示为尖锐边缘（图20.8）。图20.9显示了一个主观轮廓，这是这种效果的一种极端形式，即使在实际图像中不存在这样的轮廓，也可以看到一个封闭的轮廓。最后，视觉系统对边缘的敏感性似乎也是参与亮度感知的机制的一部分。请注意，图20.9中的主观轮廓所包围的区域看起来比页面的周围区域亮一点。图20.10显示了边缘和亮度之间的不同相互作用。在这种情况下，边缘处的特定亮度分布对边缘两侧表面的表现亮度有显著影响。

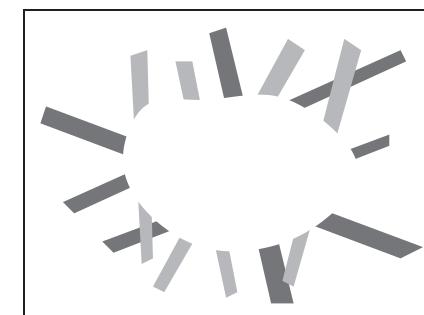


图20.9。有时，视觉系统将“看到”主观轮廓而没有任何相关的亮度变化。



Figure 20.10. Perceived lightness depends more on local contrast at edges than on brightness across surfaces. Try covering the vertical edge in the middle of the figure with a pencil. This figure is an instance of the Craik-O'Brien-Cornsweet illusion.

As indicated above, people can detect differences in the brightness between two adjacent regions if the difference is at least 1% of the average brightness. This is an example of *Weber's law*, which states that there is a constant ratio between the *just noticeable differences* (jnd) in a stimulus and the magnitude of the stimulus:

$$\frac{\Delta I}{I} = k_1, \quad (20.2)$$

where I is the magnitude of the stimulus, ΔI is the magnitude of the just noticeable difference, and k_1 is a constant particular to the stimulus. Weber's law was postulated in 1846 and still remains a useful characterization of many perceptual effects. Fechner's law, proposed in 1860, generalized Weber's law in a way that allowed for the description of the strength of any sensory experience, not just jnd's:

$$S = k_2 \log(I), \quad (20.3)$$

where S is the perceptual strength of the sensory experience, I is the physical magnitude of the corresponding stimulus, and k_2 is a scaling constant specific to the stimulus. Current practice is to model the association between perceived and actual strength of a stimulus using a power function (*Stevens's law*):

$$S = k_3 I^b, \quad (20.4)$$

where S and I are as before, k_3 is another scaling constant, and b is an exponent specific to the stimulus. For a large number of perceptual quantities involving vision, $b < 1$. The CIE $L^*a^*b^*$ color space, described elsewhere, uses a modified Stevens's law representation to characterize perceptual differences between brightness values. Note that in the first two characterizations of the perceptual strength of a stimulus and in Stevens's Law when $b < 1$, changes in the stimulus when it has a small average magnitude create larger perceptual effects than do the same physical change in the stimulus when it has a larger magnitude.



图20.10。感知到的亮度更多地取决于边缘的局部对比度，而不是表面的亮度。尝试用铅笔复盖图中间的垂直边缘。这个数字是Craik-O'Brien-Cornsweet幻觉的一个例子。

如上所述，如果两个相邻区域之间的亮度差异至少为平均亮度的1%，则人们可以检测到该亮度差异。这是韦伯定律的一个例子，该定律指出，刺激物中刚刚明显的差异 (jnd) 与刺激物的大小之间存在恒定的比率：

$$\frac{\Delta I}{I} = k_1, \quad (20.2)$$

其中 I 是刺激的量值， ΔI 是刚好能差的量值， k_1 是刺激的常数。韦伯定律是在1846年提出的，至今仍是许多感知效应的有用表征。Fechner定律于1860年提出，广义的韦伯定律允许描述任何感官经验的强度，而不仅仅是jnd的：

$$S = k_2 \log(I), \quad (20.3)$$

其中 S 是感觉体验的感知强度， I 是相应刺激的物理幅度， k_2 是特定于刺激的缩放常数。目前的做法是使用幂函数（史蒂文斯定律）对刺激的感知强度和实际强度之间的关联进行建模：

$$S = k_3 I^b, \quad (20.4)$$

其中 S 和 I 与之前一样， k_3 是另一个缩放常数， b 是特定于刺激的指数。对于大量涉及视觉的感知量， $b < 1$ 。其他地方描述的ciel a b 颜色空间使用修改的史蒂文斯定律表示来表征亮度值之间的感知差异。请注意，在刺激的感知强度的前两个表征和 $B < 1$ 时史蒂文斯定律中，当刺激具有较小的平均幅度时，刺激的变化会产生更大的感知效果，而当刺激具有较大的幅度时，刺激的物理变化会产生更大的感知效果。



The “laws” described above are not physical constraints on how perception operates. Rather, they are generalizations about how the perceptual system responds to particular physical stimuli. In the field of perceptual psychology, the quantitative study of the relationships between physical stimuli and their perceptual effects is called *psychophysics*. While psychophysical laws are empirically derived observations rather than mechanistic accounts, the fact that so many perceptual effects are well modeled by simple power functions is striking and may provide insights into the mechanisms involved.

20.2.2 Color

In 1666, Isaac Newton used prisms to show that apparently white sunlight could be decomposed into a *spectrum* of colors and that these colors could be recombined to produce light that appeared white. We now know that light energy is made up of a collection of photons, each with a particular wavelength. The *spectral distribution* of light is a measure of the average energy of the light at each wavelength. For natural illumination, the spectral distribution of light reflected off of surfaces varies significantly depending on the surface material. Characterizations of this spectral distribution can therefore provide visual information for the nature of surfaces in the environment.

Most people have a pervasive sense of color when they view the world. Color perception depends on the frequency distribution of light, with the visible spectrum for humans ranging from a wavelength of about 370 nm to a wavelength of about 730 nm (see Figure 20.11). The manner in which the visual systems derives a sense of color from this spectral distribution was first systematically examined in 1801 and remained extremely controversial for 150 years. The problem is that the visual system responds to patterns of spectral distribution very differently than patterns of luminance distribution.

Even accounting for phenomena such as lightness constancy, distinctly different spatial distributions almost always look distinctly different. More importantly given that the purpose of the visual system is to produce descriptions of the distal stimulus given the proximal stimulus, perceived patterns of lightness correspond at least approximately to patterns of brightness over surfaces in the environment.



Figure 20.11. The visible spectrum. Wavelengths are in nanometers.



上面描述的“法则”不是感知如何运作的物理约束。相反，它们是关于感知系统如何响应特定物理刺激的概括。在感知心理学领域，对物理刺激与其感知效应之间关系的定量研究被称为心理物理学。虽然心理物理定律是经验推导的观察而不是机械的帐户，但如此多的感知效应通过简单的幂函数很好地建模这一事实是惊人的，并且可能提供对所涉机

20.2.2 Color

1666年，艾萨克·牛顿 (IsaacNewton) 使用棱镜表明，显然白色的阳光可以分解成一系列颜色，并且这些颜色可以重新组合以产生看起来是白色的光。我们现在知道光能是由一组光子组成的，每个光子都有一个特定的波长。光的光谱分布是光在每个波长处的平均能量的量度。对于自然照明，表面反射的光的光谱分布因表面材料的不同而有很大差异。因此，这种光谱分布的特征izations可以为环境中表面的性质提供视觉信息。

大多数人在看世界时都有一种普遍的色彩感。颜色感知取决于光的频率分布，人类的可见光谱范围从约370nm的波长到约730nm的波长（见图20.11）。视觉系统从这种光谱分布中获得色彩感的方式在1801年首次被系统地检查，并且在150年里仍然存在极大的争议。问题是视觉系统对光谱分布模式的响应与亮度分布模式非常不同。

即使考虑到诸如亮度恒定性等现象，明显不同的空间分布几乎总是看起来明显不同。更重要的是，鉴于视觉系统的目的是在近端刺激的情况下产生对远端刺激的描述，感知到的亮度模式至少近似地对应于环境中表面上的亮度模式。



图20.11。的可见光谱。波长以纳米为单位。

“色觉研究的历史以其尖刻而着称。”—理查德格雷戈里 (1997)

The same is not true of color perception. Many quite different spectral distributions of light can produce a sense of any specific color. Correspondingly, the sense that a surface is a specific color provides little direct information about the spectral distribution of light coming from the surface. For example, a spectral distribution consisting of a combination of light at wavelengths of 700 nm and 540 nm, with appropriately chosen relative strengths, will look indistinguishable from light at the single wavelength of 580 nm. (Perceptually indistinguishable colors with different spectral compositions are referred to as *metamers*.) If we see the color “yellow,” we have no way of knowing if it was generated by one or the other of these distributions or an infinite family of other spectral distributions. For this reason, in the context of vision the term *color* refers to a purely perceptual quality, not a physical property.

There are two classes of photoreceptors in the human retina. *Cones* are involved in color perception, while *rods* are sensitive to light energy across the visible range and do not provide information about color. There are three types of cones, each with a different spectral sensitivity (Figure 20.12). *S-cones* respond to short wavelengths in the blue range of the visible spectrum. *M-cones* respond to wavelengths in the middle (greenish) region of the visible spectrum. *L-cones* respond to somewhat longer wavelengths covering the green and red portions of the visible spectrum.

While it is common to describe the three types of cones as *red*, *green*, and *blue*, this is neither correct terminology nor does it accurately reflect the cone sensitivities shown in Figure 20.12. The *L-cones* and *M-cones* are broadly tuned, meaning that they respond to a wide range of frequencies. There is also substantial overlap between the sensitivity curves of the three cone types. Taken together, these two properties mean that it is not possible to reconstruct an approximation

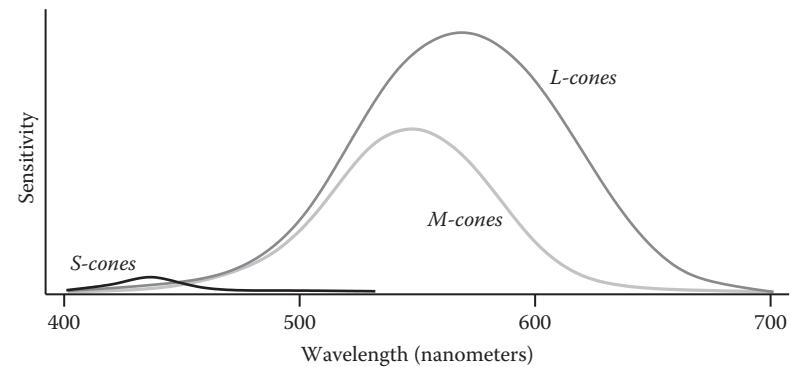


Figure 20.12. Spectral sensitivity of the *short*, *medium*, and *long* cones in the human retina.

色彩感知也是如此。许多完全不同的光谱扭曲的光可以产生任何特定颜色的感觉。相应地，表面是特定颜色的感觉提供了很少关于来自表面的光的光谱分布的直接信息。例如，由波长为700nm和540nm的光组合组成的光谱分布，具有适当选择的相对强度，看起来与单一波长为580nm的光没有区别。（具有不同光谱组成的感知上不可区分的颜色被称为metamers。）如果我们看到颜色“黄色”，我们无法知道它是由这些分布中的一个或另一个或其他光谱分布的无限族产生的。出于这个原因，在视觉的背景下，术语颜色指的是纯粹的感知质量，而不是物理属性。

人类视网膜中有两类光感受器。锥体在颜色感知方面处于volved状态，而棒对可见光范围内的光能敏感，并且不提供有关颜色的信息。锥体有三种类型，每种类型具有不同的光谱灵敏度（图20.12）。S-锥体响应可见光谱蓝色范围内的短波长。M-锥体响应可见光谱中间（绿色）区域的波长。L-锥体对覆盖可见光谱的绿色和红色部分的较长波长有反应。

虽然通常将三种类型的锥体描述为红色，绿色和蓝色，但这不是正确的术语，也不能准确反映图20.12所示的锥体灵敏度。L锥和M锥是广泛调谐的，这意味着它们响应广泛的频率范围。三种锥体类型的灵敏度曲线之间也存在实质性重叠。综合起来，这两个属性意味着不可能重建近似值

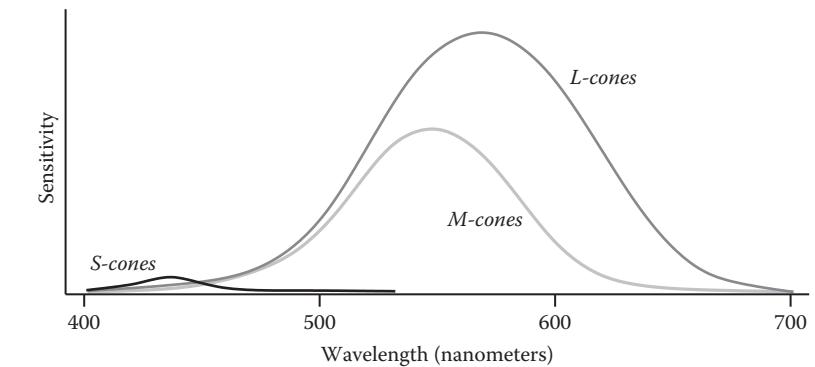


图20.12。人视网膜中短、中、长视锥细胞的光谱灵敏度。



to the original spectral distribution given the responses of the three cone types. This is in contrast to spatial sampling in the retina (and in digital cameras), where the receptors are narrowly tuned in their spatial sensitivity in order to be able to detect fine detail in local contrast.

The fact that there are only three types of color sensitive photoreceptors in the human retina greatly simplifies the task of displaying colors on computer monitors and in other graphical displays. Computer monitors display colors as a weighted combination of three fixed-color distributions. Most often, the three colors are a distinct red, a distinct green, and a distinct blue. As a result, in computer graphics, color is often represented by a *red-green-blue* (RGB) triple, representing the intensities of red, green, and blue primaries needed to display a particular color. Three *basis colors* are sufficient to display most perceptible colors, since appropriately weighted combinations of three appropriately chosen colors can produce metamers for these perceptible colors.

There are at least two significant problems with the RGB color representation. The first is that different monitors have different spectral distributions for their red, green, and blue primaries. As a result, perceptually correct color rendition involves remapping RGB values for each monitor. This is, of course, only possible if the original RGB values satisfy some well-defined standard, which is often not the case. (See Chapter 19 for more information on this issue.) The second problem is that RGB values do not define a particular color in a way that corresponds to subjective perception. When we see the color "yellow," we do not have the sense that it is made up of equal parts of red and green light. Rather, it looks like a single color, with additional properties involving brightness and the "amount" of color. Representing color as the output of the S-cones, M-cones, and L-cones is no help either, since we have no more phenomenological sense of color as characterized by these properties than we do as characterized by RGB display properties.

There are two different approaches to characterizing color in a way that more closely reflects human perception. The various CIE color spaces aim to be "perceptually uniform" so that the magnitude of the difference in the represented values of two colors is proportional to the perceived difference in color (Wyszecki & Stiles, 2000). This turns out to be a difficult goal to accomplish, and there have been several modifications to the CIE model over the years. Furthermore, while one of the dimensions of the CIE color spaces corresponds to perceived brightness, the other two dimensions that specify chromaticity have no intuitive meaning.

The second approach to characterizing color in a more natural manner starts with the observation that there are three distinct and independent properties that



给出了三种锥体类型的响应的原始光谱分布。这与视网膜（和数码相机）中的空间采样形成鲜明对比，其中受体在其空间灵敏度上进行了狭隘的调整，以便能够在局部对比度中检测到精细细节。

在人类视网膜中只有三种类型的颜色敏感光感受器的事实大大简化了在计算机显示器和其他图形显示器中显示颜色的任务。计算机显示器将颜色显示为三种固定颜色分布的加权组合。大多数情况下，这三种颜色是一种独特的红色，一种独特的绿色和一种独特的蓝色。因此，在计算机图形学中，颜色通常由红-绿-蓝（RGB）三重表示，表示显示特定颜色所需的红、绿和蓝原色的强度。三种基色足以显示大多数可感知的颜色，因为三种适当选择的颜色的适当加权组合可以产生用于这些可感知颜色的元素。

RGB颜色表示至少存在两个显着问题。首先，不同的显示器的红色、绿色和蓝色原色具有不同的光谱分布。因此，感知正确的色彩再现涉及为每个显示器重新映射RGB值。当然，这是唯一可能的，如果原始RGB值满足一些明确定义的标准，这往往不是这种情况。（有关此问题的更多信息，请参阅第19章。）第二个问题是RGB值没有以与主观感知相对应的方式定义特定颜色。当我们看到颜色"黄色"时，我们没有感觉它是由红色和绿色光的相等部分组成的。相反，它看起来像一个单一的颜色，具有涉及亮度和颜色的"数量"的附加属性。将颜色表示为S-锥体，M-锥体和L-锥体的输出也没有帮助，因为我们没有更多的现象学色彩感，因为这些属性的特征比RGB显示属性的特征。

有两种不同的方法来表征颜色，以更密切地反映人类感知的方式。各种CIE颜色空间旨在"感知上均匀"，以便两种颜色的表示值差异的大小与感知的颜色差异成比例（Wyszecki & Stiles, 2000）。事实证明，这是一个难以实现的目标，多年来对CIE模型进行了多次修改。此外，虽然CIE颜色空间的维度之一对应于感知亮度，但指定色度的其他两个维度没有直观意义。

以更自然的方式表征颜色的第二种方法始于观察到有三种不同和独立的属性

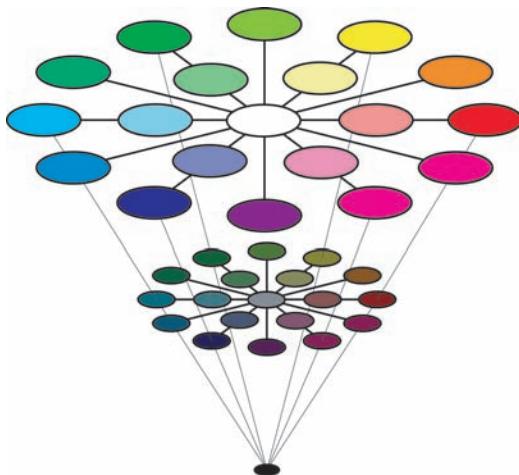
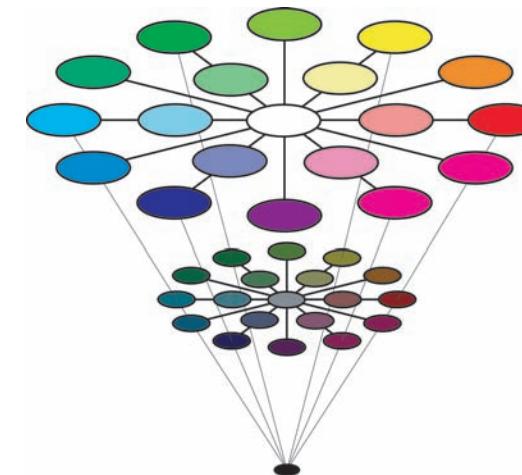


Figure 20.13. HSV color space. Hue varies around the circle, saturation varies with radius, and value varies with height.

dominate the subjective sense of color. *Lightness*, the apparent brightness of a surface, has already been discussed. *Saturation* refers to the purity or vividness of a color. Colors can range from totally unsaturated gray to partially saturated pastels to fully saturated “pure” colors. The third property, *hue*, corresponds most closely to the informal sense of the word “color” and is characterized in a manner similar to colors in the visible spectrum, ranging from dark violet to dark red. Figure 20.13 shows a plot of the hue-saturation-lightness (HSV) color space. Since the relationship between brightness and lightness is both complex and not well understood, HSV color spaces almost always use brightness instead of attempting to estimate lightness. Unlike wavelengths in the spectrum, however, hue is usually represented in a manner that reflects the fact that the extremes of the visible spectrum are actually similar in appearance (Figure 20.14). Simple transformations exist between RGB and HSV representations of a particular color value. As a result, while the HSV color space is motivated by perceptual considerations, it contains no more information than does an RGB representation.



Figure 20.14. Which color is closer to red: green or violet?



HSV色彩空间。色调在圆周围变化，饱和度随半径变化，值随高度变化。

支配色彩的主观感觉。亮度 表面的表观亮度 已经讨论过了.饱和度是指颜色的纯度或鲜艳度。颜色可以从完全不饱和的灰色到部分饱和的粉彩到完全饱和的“纯”颜色。第三个属性，色调，最接近于“颜色”这个词的非正式意义，其特征类似于可见光谱中的颜色，从深紫色到深红色。图20.13显示了色调-饱和度-亮度(HSV)颜色空间的图。由于亮度和亮度之间的关系既复杂又不被很好理解，HSV色彩空间几乎总是使用亮度而不是试图估计亮度。然而，与光谱中的波长不同，色调通常以反映可见光谱的极值在外观上实际上相似的事实的方式表示（图20.14）。特定颜色值的RGB和HSV表示之间存在简单的转换。因此，虽然HSV颜色空间是由感知考虑驱动的，但它包含的信息并不比RGB表示更多。



图20.14。哪种颜色更接近红色：绿色还是紫色？



The hue-saturation-lightness approach to describing color is based on the spectral distribution at a single point and so only approximates the perceptual response to spectral distributions of light distributed over space. Color perception is subject to similar constancy and simultaneous contrast effects as is lightness/brightness, neither of which are captured in the RGB representation and as a result are not captured in the HSV representation. For an example of color constancy, look at a piece of white paper indoors under incandescent light and outdoors under direct sunlight. The paper will look “white” in both cases, even though incandescent light has a distinctly yellow hue and so the light reflected off of the paper will also have a yellow hue, while sunlight has a much more uniform color spectrum.

Another aspect of color perception not captured by either the CIE color spaces or HSV encoding is the fact that we see a small number of distinct colors when looking at a continuous spectrum of visible light (Figure 20.11) or in a naturally occurring rainbow. For most people, the visible spectrum appears to be divided into four to six distinct colors: red, yellow, green, and blue, plus perhaps light blue and purple. Considering non-spectral colors as well, there are only 11 basic color terms commonly used in English: *red, green, blue, yellow, black, white, gray, orange, purple, brown, and pink*. The partitioning of the intrinsically continuous space of spectral distributions into a relatively small set of perceptual categories associated with well-defined linguistic terms seems to be a basic property of perception, not just a cultural artifact (Berlin & Kay, 1969). The exact nature of the process, however, is not well understood.

20.2.3 Dynamic Range

Natural illumination varies in intensity over 6 orders of magnitude (Figure 20.15). The human vision system is able to operate over this full range of brightness levels. However, at any one point in time, the visual system is only able to detect variations in light intensity over a much smaller range. As the average brightness to which the visual system is exposed changes over time, the range of discriminable brightnesses changes in a corresponding manner. This effect is most obvious if we move rapidly from a brightly lit outdoor area to a very dark room. At first, we are able to see little. After a while, however, details in the room start to become apparent. The *dark adaptation* that occurs involves a number of physiological changes in the eye. It takes several minutes for significant dark adaptation to occur and 40 minutes or so for complete dark adaptation. If we then move back into the bright light, not only is vision difficult but it can actually be painful. *Light adaptation* is required before it is again possible to see clearly. Light adaptation

<i>direct sunlight</i>	10^5
<i>indoor lighting</i>	10^2
<i>moonlight</i>	10^{-1}
<i>starlight</i>	10^{-3}

Figure 20.15. Approximate luminance level of a white surface under different types of illumination in candelas per meter squared (cd/m^2). (Wandell, 1995).



用于描述颜色的色相-饱和度-亮度方法基于单个点上的光谱分布，因此仅近似于对分布在空间上的光的光谱分布的感知响应。颜色感知与光的亮度一样受到类似的恒常性和同时的对比度效应，这两种效应都不会在RGB表示中捕获，结果也不会在HSV表示中捕获。对于颜色恒定性的一个例子，看看一张白纸在室内白炽灯下和室外阳光直射下。在这两种情况下，纸张看起来都是“白色”的，即使白炽灯具有明显的黄色色调，因此从纸张反射出来的光线也会具有黄色色调，而阳光具有更均匀的色谱。

Cie色彩空间或HSV编码没有捕捉到色彩感知的另一个方面是，当我们看到连续的可见光光谱（图20.11）或自然出现的彩虹时，我们看到少量不同的颜色。对于大多数人来说，可见光谱似乎分为四到六种不同的颜色：红色，黄色，绿色和蓝色，以及可能的浅蓝色和紫色。考虑到非光谱颜色，只有11个英语中常用的基本颜色术语：红色，绿色，蓝色，黄色，黑色，白色，灰色，橙色，紫色，棕色和粉红色。将光谱分布的内在连续空间划分为相对较小的一组与明确定义的语言术语相关的感知类别似乎是每个感知的基本属性，而不仅仅是文化工作（Berlin & Kay, 1969）。然而，这个过程的确切性质尚不清楚。

20.2.3 动态范围

自然照明的强度变化超过6个数量级（图20.15）。人类视觉系统能够在整个亮度范围内运行。然而，在任何一个时间点，视觉系统只能检测到小得多的范围内的光强度变化。随着视觉系统所暴露到的平均亮度随时间变化，可辨别亮度的范围以相应的方式变化。如果我们从一个明亮的室外区域迅速移动到一个非常黑暗的房间，这种效果是最明显的。在

首先，我们能够看到很少。然而，过了一段时间，房间里的细节开始变得明显。发生的黑暗适应涉及眼睛的许多生理变化。发生显着的黑暗适应需要几分钟，完全黑暗适应需要40分钟左右。如果我们回到明亮的光线中，不仅视力困难，而且实际上可能是痛苦的。在再次能够看清楚之前，需要进行光适应。光适应

阳光直射	10^5
室内照明	10^2
<i>moonlight</i>	10^{-1}
<i>starlight</i>	10^{-3}

图20.15。在不同类型的照明下，白色表面的近似亮度水平，每米平方烛台 (cd/m^2)。（Wandell, 1995）。

occurs much more quickly than dark adaptation, typically requiring less than a minute.

The two classes of photoreceptors in the human retina are sensitive to different ranges of brightness. The cones provide visual information over most of what we consider normal lighting conditions, ranging from bright sunlight to dim indoor lighting. The rods are only effective at very low light levels. *Photopic* vision involves bright light in which only the cones are effective. *Scotopic* vision involves dark light in which only the rods are effective. There is a range of intensities within which both cones and rods are sensitive to changes in light, which is referred to as *mesopic* conditions (see Chapter 21).

20.2.4 Field-of-View and Acuity

Each eye in the human visual system has a field-of-view of approximately 160° horizontal by 135° vertical. With binocular viewing, there is only partial overlap between the fields-of-view of the two eyes. This results in a wider overall field-of-view (approximately 200° horizontal by 135° vertical), with the region of overlap being approximately 120° horizontal by 135° vertical.

With normal or corrected-to-normal vision, we usually have the subjective experience of being able to see relatively fine detail wherever we look. This is an illusion, however. Only a small portion of the visual field of each eye is actually sensitive to fine detail. To see this, hold a piece of paper covered with normal-sized text at arm's length, as shown in Figure 20.16. Cover one eye with the

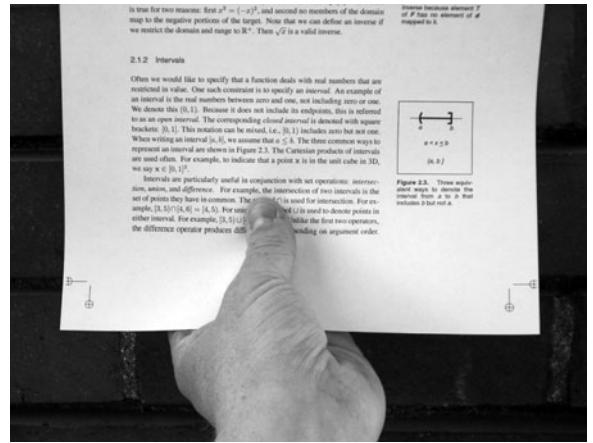


Figure 20.16. If you hold a page of text at arm's length and stare at your thumb, only the text near your thumb will be readable. Photo by Peter Shirley.

发生比黑暗适应快得多，通常需要不到一分钟。

人类视网膜中的两类光感受器对亮度的different范围敏感。锥体提供了我们认为的大多数正常照明条件的视觉信息，从明亮的阳光到昏暗的室内照明。这些棒只有在非常低的光照水平下才有效。光视觉涉及只有视锥细胞有效的强光。Scotopic视觉涉及暗光，其中只有棒是有效的。有一个范围内的视锥和视杆都对光的变化敏感，这被称为中视条件（见第21章）。

20.2.4 视野和敏锐度

人类视觉系统中的每只眼睛的视野约为160°水平由135°垂直。在双目观察中，两只眼睛的视场之间只有部分重叠。这导致更宽的整体视场（大约200°水平x135°垂直），重叠区域大约为120°水平x135°垂直。

对于正常或矫正到正常的视力，我们通常有主观体验，无论我们看到哪里，都能看到相对精细的细节。然而，这是一种幻觉。每只眼睛的视野中只有一小部分实际上对精细细节敏感。为了看到这一点，拿着一张复盖着正常大小文本的纸，在手臂的长度，如图20.16所示。用

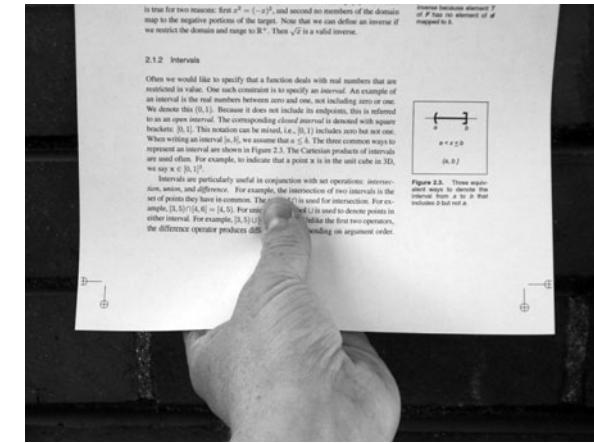


图20.16。如果你把一页文字放在一臂的长度上，盯着你的拇指，只有你的拇指附近的文字是可读的。彼得·雪莉的照片。



hand not holding the paper. While staring at your thumb and not moving your eye, note that the text immediately above your thumb is readable while the text to either side is not. High acuity vision is limited to a visual angle slightly larger than your thumb held at arm's length. We do not normally notice this because the eyes usually move frequently, allowing different regions of the visual field to be viewed at high resolution. The visual system then integrates this information over time to produce the subjective experience of the whole visual field being seen at high resolution.

There is not enough bandwidth in the human visual cortex to process the information that would result if there was a dense sampling of image intensity over the whole of the retina. The combination of variable density photoreceptor packing in the retina and a mechanism for rapid eye movements to point at areas of interest provides a way to simultaneously optimize acuity and field-of-view. Other animals have evolved different ways of balancing acuity and field-of-view that are not dependent on rapid eye movements. Some have only high acuity vision, but limited to a narrow field-of-view. Others have wide field-of-view vision, but limited ability to see detail.

The eye motions which focus areas of interest in the environment on the fovea are called *saccades*. Saccades occur very quickly. The time from a triggering stimulus to the completion of the eye movement is 150–200 ms. Most of this time is spent in the vision system planning the saccade. The actual motion takes 20 ms or so on average. The eyes are moving very quickly during a saccade, with the maximum rotational velocity often exceeding 500°/second. Between saccades, the eyes point toward an area of interest (*fixate*), taking 300 ms or so to acquire fine detail visual information. The mechanism by which multiple fixations are integrated to form an overall subjective sense of fine detail over a wide field of view is not well understood.

Figure 20.17 shows the variable packing density of cones and rods in the human retina. The cones, which are responsible for vision under normal lighting, are packed most closely at the *fovea* of the retina (Figure 20.17). When the eye is fixated at a particular point in the environment, the image of that point falls on the fovea. The higher packing density of cones at the fovea results in a higher sampling frequency of the imaged light (see Chapter 9) and hence greater detail in the sampled pattern. Foveal vision encompasses about 1.7°, which is the same visual angle as the width of your thumb held at arm's length.

While a version of Figure 20.17 appears in most introductory texts on human visual perception, it provides only a partial explanation for the neurophysiological limitations on visual acuity. The output of individual rods and cones is pooled in various ways by neural interconnects in the eye, before the information is shipped



手不拿着纸。当你盯着你的拇指而不移动你的眼睛时，请注意，你的拇指上方的文字是可读的，而两边的文字则不是。高敏锐度视力仅限于比拇指稍大的视角。我们通常不会注意到这一点，因为眼睛通常会频繁移动，从而允许以高分辨率观看视野的不同区域。然后，视觉系统随着时间的推移整合这些信息，以产生高分辨率看到的整个视野的主观体验。

人类视觉皮层中没有足够的带宽来处理如果整个视网膜上的图像强度密集采样所产生的信息。视网膜中可变密度光感受器填料与快速眼球运动指向感兴趣区域的机制相结合，提供了一种同时优化敏锐度和视野的方法。其他动物已经进化出不同的方式来平衡敏锐度和视野，而不依赖于快速的眼球运动。有些只有高敏锐度的视力，但仅限于狭窄的视野。其他人拥有广阔的视野视野，但看到细节的能力有限。

将环境中感兴趣的区域集中在中央凹上的眼睛运动称为眼跳。眼跳发生得非常快。从触发刺激到眼球运动完成的时间为150–200毫秒。大部分时间都花在视觉系统规划眼跳。实际运动平均需要20毫秒左右。眼跳时，眼睛的移动速度非常快，最大旋转速度通常超过500秒。在眼跳之间，眼睛指向一个感兴趣的区域（固定），需要300毫秒左右的时间来获得精细的视觉信息。在广泛的视场上，多重固定被整合以形成一个整体的主观细节感的机制还没有很好的理解。

图20.17显示了人视网膜中视锥细胞和视杆细胞的可变包装密度。视锥细胞在正常照明下负责视力，在视网膜中央凹处包装最紧密（图20.17）。当眼睛固定在环境中的特定点时，该点的图像落在中央凹上。视锥在中央凹处的更高填充密度导致成像光的更高采样频率（见第9章），因此采样模式的细节更多。中央凹视力包括约1.7%，这是相同的视觉角度，你的拇指的宽度举行在手臂的长度。

虽然图20.17的一个版本出现在大多数关于人类视觉感知的介绍性文本中，但它仅提供了对视觉敏锐度的神经生理学限制的部分解释。在信息发送之前，单个视杆和视锥细胞的输出通过眼睛中的神经互连以各种方式汇集

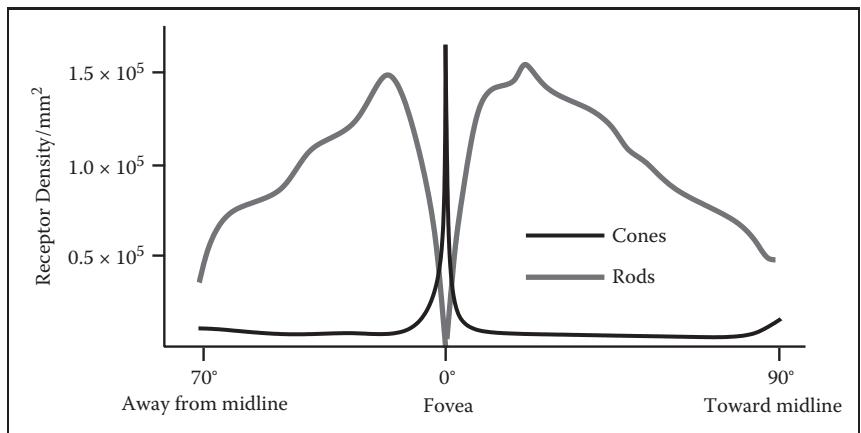


Figure 20.17. Density of rods and cone in the human retina (after Osterberg (1935)).

along the optic nerve to the visual cortex.³ This pooling filters the signal provided by the pattern of incident illumination in ways that have important impacts on the patterns of light that are detectable. In particular, the farther away from the fovea, the larger the area over which brightness is averaged. As a consequence, spatial acuity drops sharply away from the fovea. Most figures showing rod and cone packing density indicate the location of the retinal *blind spot*, where the nerve bundle carrying optical information from the eye to the brain passes through the retina, and there is no sensitivity to light. By and large, the only practical impact of the blind spot on real-world perception is its use as an illusion in introductory perception texts, since normal eye movements otherwise compensate for the temporary loss of information.

As shown in Figure 20.17, the packing density of rods drops to zero at the center of the fovea. Away from the fovea, the rod density first increases and then decreases. One result of this is that there is no foveal vision when illumination is very low. The lack of rods in the fovea can be demonstrated by observing a night sky on a moonless night, well away from any city lights. Some stars will be so dim that they will be visible if you look at a point in the sky slightly to the side of the star, but they will disappear if you look directly at them. This occurs because when you look directly at these features, the image of the features falls only on the cones in the retina, which are not sufficiently light sensitive to detect the feature. Looking slightly to the side causes the image to fall on the more light-sensitive cones. Scotopic vision is also limited in acuity, in part because

³All of the cells in the optic nerve and almost all cells in the visual cortex have an associated retinal receptive field. Patterns of light hitting the retina outside of a cell's receptive field have no effect on the firing rate of that cell.

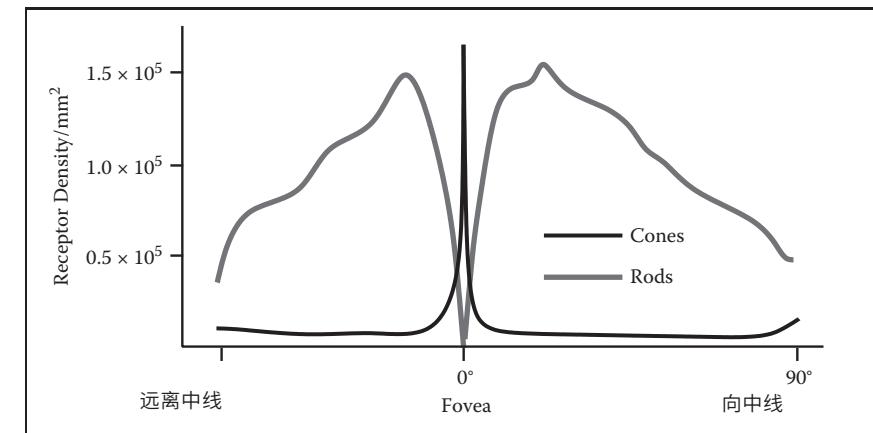


图20.17。人视网膜中棒和锥体的密度 (Osterberg (1935) 之后)。

沿着视神经到视觉皮层。3这种汇集以对可检测的光模式具有重要影响的方式过滤入射照明模式提供的信号。特别是，离中央凹越远，亮度平均的区域越大。结果，空间敏锐度从中央凹处急剧下降。显示棒和锥填料密度的大多数图指示视网膜盲点的位置，其中携带从眼睛到大脑的光学信息的神经束穿过视网膜，并且对光没有敏感性。总的来说，盲点对现实世界感知的唯一实际影响是它在视觉感知文本中作为幻觉的使用，因为正常的眼球运动否则会补偿信息的暂时损失。

如图20.17所示，杆的填料密度在中央凹中心处降至零。远离中央凹，杆密度首先增加然后减少。这样做的一个结果是，当照度非常低时，没有中央凹视力。中央凹中缺乏杆可以通过在没有月亮的夜晚观察夜空来证明，远离任何城市的灯光。有些星星会非常暗淡，如果你看天空中的一点稍微向星星的一侧看，它们就会可见，但如果你直视它们，它们就会消失。发生这种情况是因为当您直接查看这些特征时，特征的图像仅落在视网膜中的视锥细胞上，视锥细胞对检测特征没有足够的光敏感。稍微向侧面看会导致图像落在更光敏感的锥体上。Scotopic视力也受到敏锐度的限制，部分原因是

3视神经中的所有细胞和视觉皮层中的几乎所有细胞都具有相关的视网膜接受场。在细胞接受场外撞击视网膜的光线模式对该细胞的发射率没有影响。

of the lower density of rods over much of the retina and in part because greater pooling of signals from the rods occurs in the retina in order to increase the light sensitivity of the visual information passed back to the brain.

20.2.5 Motion

When reading about visual perception and looking at static figures on a printed page, it is easy to forget that motion is pervasive in our visual experience. The patterns of light that fall on the retina are constantly changing due to eye and body motion and the movement of objects in the world. This section covers our ability to detect visual motion. Section 20.3.4 describes how visual motion can be used to determine geometric information about the environment. Section 20.4.3 deals with the use of motion to guide our movement through the environment.

The detectability of motion in a particular pattern of light falling on the retina is a complex function of speed, direction, pattern size, and contrast. The issue is further complicated because simultaneous contrast effects occur for motion perception in a manner similar to that observed in brightness perception. In the extreme case of a single small pattern moving against a contrasting, homogeneous background, perceivable motion requires a rate of motion corresponding to $0.2^{\circ}\text{--}0.3^{\circ}/\text{second}$ of visual angle. Motion of the same pattern moving against a textured pattern is detectable at about a tenth this speed.

With this sensitivity to retinal motion, combined with the frequency and velocity of saccadic eye movements, it is surprising that the world usually appears stable and stationary when we view it. The vision system accomplishes this in three ways. Contrast sensitivity is reduced during saccades, reducing the visual effects generated by these rapid changes in eye position. Between saccades, a variety of sophisticated and complex mechanisms adjust eye position to compensate for head and body motion and the motion of objects of interest in the world. Finally, the visual system exploits information about the position of the eyes to assemble a mosaic of small patches of high-resolution imagery from multiple fixations into a single, stable whole.

The motion of straight lines and edges is ambiguous if no endpoints or corners are visible, a phenomenon referred to as the *aperture problem* (Figure 20.18). The aperture problem arises because the component of motion parallel to the line or edge does not produce any visual changes. The geometry of the real world is sufficiently complex that this rarely causes difficulties in practice, except for intentional illusions such as barber poles. The simplified geometry and texturing found in some computer graphics renderings, however, has the potential to introduce inaccuracies in perceived motion.

由于视网膜上大部分区域的视杆密度较低，部分原因是视网膜中出现了更多来自视杆的信号汇集，以增加传递回大脑的视觉信息的光敏感度。

20.2.5 Motion

当阅读视觉感知和查看打印页面上的静态数字时，很容易忘记运动在我们的视觉体验中无处不在。由于眼睛和身体的运动以及世界上物体的运动，落在视网膜上的光线模式不断变化。本节介绍我们检测视觉运动的能力。第20.3.4节描述了如何使用视觉运动来确定有关环境的几何信息。第20.4.3节涉及使用运动来引导我们在环境中的运动。

落在视网膜上的特定光线模式中的运动可检测性是速度，方向，图案大小和对比度的复杂函数。该问题进一步复杂化，因为每个感受的运动以类似于在亮度感知中观察到的方式同时发生对比度效应。在一个单一的小图案在一个对比的、同源的背景下移动的极端情况下，可感知的运动需要一个对应于 $0.2 : 0.3$ 秒的视觉角度的运动速率。在大约十分之一的速度下，可以检测到与纹理图案相反的相同图案的运动。

由于对视网膜运动的这种敏感性，再加上眼跳运动的频率和位置，令人惊讶的是，当我们观察它时，世界通常看起来是稳定和静止的。视觉系统通过三种方式实现这一点。眼跳过程中对比度灵敏度降低，降低了这些眼位快速变化所产生的视觉效果。在眼跳之间，各种复杂而复杂的机制调整眼睛的位置，以配合头部和身体的运动以及世界上感兴趣的物体的运动。最后，视觉系统利用有关眼睛位置的信息，将多个修复的高分辨率图像的小块拼接成一个单一的、稳定的整体。

直线和边的运动在没有端点或角点可见的情况下是模糊的，这种现象称为孔径问题（图20.18）。光圈问题的产生是因为平行于线或边缘的运动分量不会产生任何视觉变化。现实世界的几何形状足够复杂，这在实践中很少造成困难，除了有意的幻想，如理发杆。然而，在一些计算机图形渲染中发现的简化几何和文本可能会在感知运动中引入不准确性。

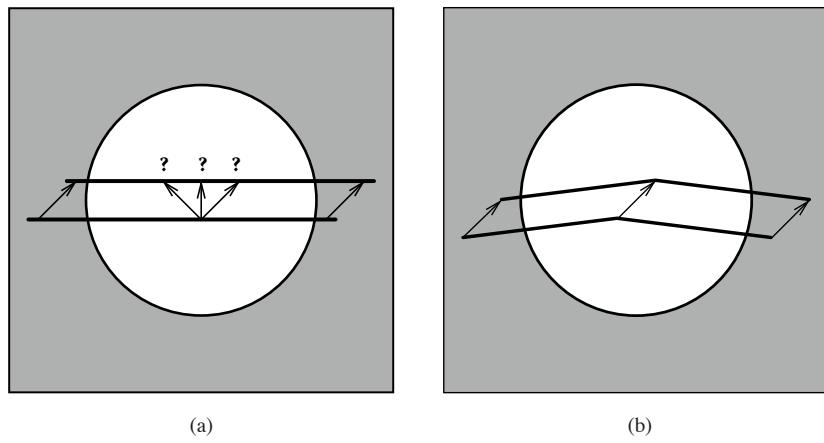
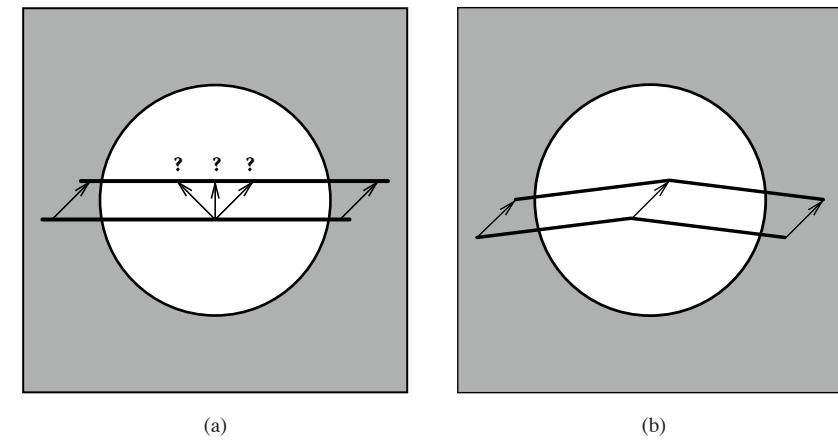


Figure 20.18. The aperture problem: (a) If a straight line or edge moves in such a way that its endpoints are hidden, the visual information is not sufficient to determine the actual motion of the line. (b) 2D motion of a line is unambiguous if there are any corners or other distinctive markings on the line.

Real-time computer graphics, film, and video would not be possible without an important perceptual phenomena: discontinuous motion, in which a series of static images are visible for discrete intervals in time and then move by discrete intervals in space, can be nearly indistinguishable from continuous motion. The effect is called *apparent motion* to highlight that the appearance of continuous motion is an illusion.

Figure 20.19 illustrates the difference between continuous motion, which is typical of the real world, and apparent motion, which is generated by almost all dynamic image display devices. The motion plotted in Figure 20.19 (b) consists of an average motion comparable to that shown in Figure 20.19 (a), modulated by a high space-time frequency that accounts for the alternation between a stationary pattern and one that moves discontinuously to a new location. Apparent perception of continuous motion occurs because the visual system is insensitive to the high-frequency component of the motion.

A compelling sense of apparent motion occurs when the rate at which individual images appear is above about 10 Hz, as long as the positional changes between successive images is not too great. This rate is not fast enough, however, to produce a satisfying sense of continuous motion for most image display devices. Almost all such devices introduce brightness variation as one image is switched to the next. In well-lit conditions, the human visual system is sensitive to this varying brightness for rates of variations up to about 80 Hz. In lower light, detectability is present up to about 40 Hz. When the rate of alternating brightness is sufficiently high, *flicker fusion* occurs and the variation is no longer visible.



光圈问题：(a) 如果直线或边缘以隐藏其端点的方式移动，则视觉信息不足以确定线的实际运动。(b)如果线上有任何角或其他明显的标记，则线的二维运动是明确的。

如果没有一个重要的感知现象，实时计算机图形、电影和视频就不可能实现：不连续的运动，在不连续的运动中，一系列静态图像在时间上以离散的间隔可见，然后在空间上以离散的间隔移动，几乎无法与连续的运动区分开来。这种效果被称为表观运动，以突出连续运动的出现是一种幻觉。

图20.19说明了现实世界中典型的连续运动和几乎所有动态图像显示设备产生的表观运动之间的区别。图20.19 (b) 中绘制的运动由与图20.19 (a) 中所示相当的平均运动组成，由高空间-时间频率调制，该频率占静止模式和不连续移动到新位置的模式之间的连续运动的明显感知是因为视觉系统对运动的高频分量不敏感。

只要连续图像之间的位置变化不是太大，当individual图像出现的速率超过约10Hz时，就会产生令人信服的表观运动感。这种速率还不够快，如何，对于大多数图像显示设备产生令人满意的连续运动感。几乎所有此类设备都会在一个图像切换到下一个图像时引入亮度变化。在光线充足的条件下，人类视觉系统对这种变化的亮度敏感，变化率高达约80Hz。在较低的光线下，可检测性最高可达约40Hz。当交替亮度的速率足够高时，发生闪烁融合并且不再可见变化。

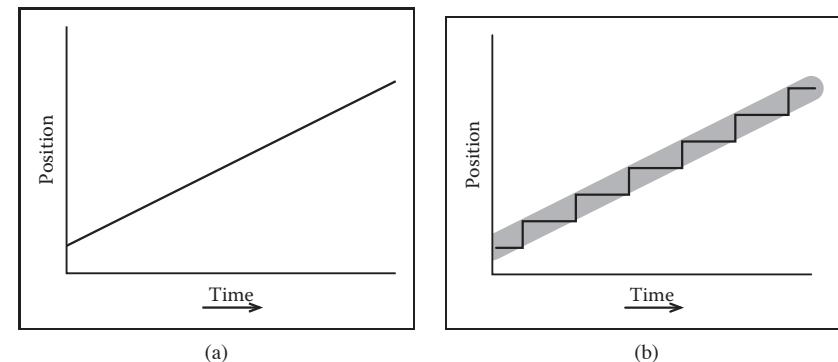


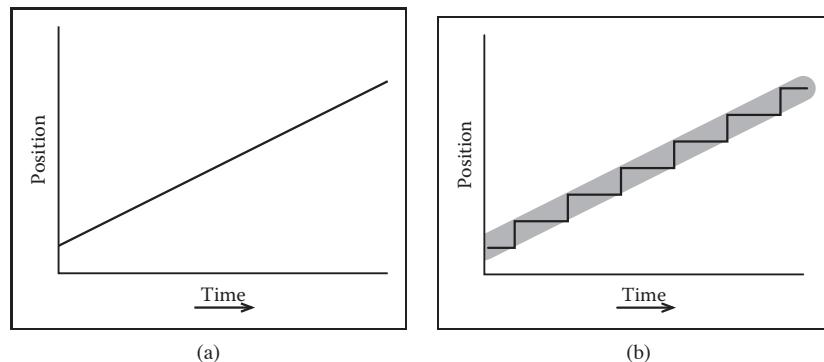
Figure 20.19. (a) Continuous motion. (b) Discontinuous motion with the same average velocity. Under some circumstances, the perception of these two motion patterns may be similar.

To produce a compelling sense of visual motion, an image display must therefore satisfy two separate constraints:

- images must be updated at a rate ≥ 10 Hz;
- any flicker introduced in the process of updating images must occur at a rate $\geq 60\text{--}80$ Hz.

One solution is to require that the image update rate be greater than or equal to 60–80 Hz. In many situations, however, this is simply not possible. For computer graphics displays, the frame computation time is often substantially greater than 12–15 msec. Transmission bandwidth and limitations of older monitor technologies limit normal broadcast television to 25–30 images per second. (Some HDTV formats operate at 60 images/sec.) Movies update images at 24 frames/second due to exposure time requirements and the mechanical difficulties of physically moving film any faster than that.

Different display technologies solve this problem in different ways. Computer displays refresh the displayed image at $\sim 70\text{--}80$ Hz, regardless of how often the contents of the image change. The term *frame rate* is ambiguous for such displays, since two values are required to characterize this display: *refresh rate*, which indicates the rate at which the image is redisplayed and *frame update rate*, which indicates the rate at which new images are generated for display. Standard non-HDTV broadcast television uses a refresh rate of 60 Hz (NTSC, used in North America and some other locations) or 50 Hz (PAL, used in most of the rest of the world). The frame update rate is half the refresh rate. Instead of displaying each new image twice, the display is *interlaced* by dividing alternating horizontal image lines into even and odd *fields* and alternating the display of these even and



(a)连续运动。(b)具有相同平均速度的不连续运动。在某些情况下，这两种运动模式的感知可能是相似的。

因此，为了产生令人信服的视觉运动感，图像显示必须满足两个单独的约束：

- *图像必须以 ≥ 10 赫兹的速度更新；
- *在更新图像过程中引入的任何闪烁必须以 $\geq 60\text{--}80$ Hz的速率发生。

一种解决方案是要求图像更新速率大于或等于60–80Hz。然而，在许多情况下，这根本不可能。对于计算机图形显示器，帧计算时间通常基本上大于12–15毫秒。传输带宽和旧显示器技术的限制将普通广播电视台限制在每秒25–30幅图像。（一些HDTV格式以60图像秒的速度运行）由于曝光时间要求和物理移动胶片的机械困难，电影以24帧秒的速度更新图像。

不同的显示技术以不同的方式解决了这个问题。无论图像内容更改的频率如何，计算机显示器都会以 70–80Hz刷新显示的图像。术语帧速率对于这种显示器是不明确的，因为需要两个值来表征这种显示器：刷新率，它指示图像被重新显示的速率和帧更新率，它指示为显示生成新图像的速率。标准的非HDTV广播电视台使用60Hz（NTSC，在北美和其他一些地方使用）或50Hz（PAL，在世界大部分地区使用）的刷新率。帧更新速率为刷新速率的一半。不是将每个新图像显示两次，而是通过将交替的水平图像线分成偶数和奇数场并交替显示这些偶数和奇数场来隔行扫描显示

odd fields. Flicker is avoided in movies by using a mechanical shutter to blink each frame of the film three times before moving to the next frame, producing a refresh rate of 72 Hz while maintaining the frame update rate of 24 Hz.

The use of apparent motion to simulate continuous motion occasionally produces undesirable artifacts. Best known of these is the *wagon wheel illusion* in which the spokes of a rotating wheel appear to revolve in the opposite direction from what would be expected given the translational motion of the wheel. The wagon wheel illusion is an example of temporal aliasing. Spokes, or other spatially periodic patterns on a rotating disk, produce a temporally periodic signal for viewing locations that are fixed with respect to the center of the wheel or disk. Fixed frame update rates have the effect of sampling this temporally periodic signal in time. If the temporal frequency of the sampled pattern is too high, undersampling results in an aliased, lower temporal frequency appearing when the image is displayed. Under some circumstances, this distortion of temporal frequency causes a spatial distortion in which the wheel appears to move backwards. Wagon wheel illusions are more likely to occur with movies than with video, since the temporal sampling rate is lower.

Problems can also occur when apparent motion imagery is converted from one medium to another. This is of particular concern when 24 Hz movies are transferred to video. Not only does a non-interlaced format need to be translated to an interlaced format, but there is no straightforward way to move from 24 frames per second to 50 or 60 fields per second. Some high-end display devices have the ability to partially compensate for the artifacts introduced when film is converted to video.

20.3 Spatial Vision

One of the critical operations performed by the visual system is the estimation of geometric properties of the visible environment, since these are central to determining information about objects, locations, and events. Vision has sometimes been described as *inverse optics*, to emphasize that one function of the visual system is to invert the image formation process in order to determine the geometry, materials, and lighting in the world that produced a particular pattern on light on the retina. The central problem for a vision system is that properties of the visible environment are confounded in the patterns of light imaged on the retina. Brightness is a function of both illumination and reflectance, and can depend on environmental properties across large regions of space due to the complexities of light transport. Image locations of a projected environmental location at best can

奇场。在电影中，通过使用机械快门在移动到下一帧之前将电影的每帧闪烁三次来避免闪烁，产生72Hz的刷新率，同时保持24Hz的帧更新率。

使用表观运动来模拟连续运动偶尔会产生不希望的伪影。其中最著名的是车轮假象，在这种假象中，旋转车轮的轮辐看起来与车轮的平移运动所预期的相反方向旋转。车轮假象是时间混叠的一个例子。轮辐或旋转盘上的其他spatially周期性图案产生时间周期性信号，用于查看相对于车轮或盘的中心固定的位置。固定的帧更新速率具有在时间上采样这种暂时性的odic信号的效果。如果采样图案的时间频率过高，则欠采样会导致图像显示时出现混叠的较低时间频率。在某些情况下，时间频率的这种失真导致空间失真，其中车轮似乎向后移动。由于时间采样率较低，电影比视频更容易出现车轮幻象。

当表观运动图像从一种介质转换到另一种介质时，也会出现问题。当24Hz电影传输到视频时，这一点尤其值得关注。非隔行格式不仅需要转换为隔行格式，而且没有直接的方法可以从每秒24帧移动到每秒50或60场。一些高端显示设备具有部分补偿当胶片转换为视频时引入的伪影的能力。

20.3 空间视觉

视觉系统执行的关键操作之一是对可见环境的几何属性的估计，因为这些是确定有关对象、位置和事件的信息的核心。视觉有时被描述为逆光学，以强调视觉系统的一个功能是反转图像形成过程，以确定在视网膜上产生特定图案的世界中的几何形状，材料和照明。视觉系统的核心问题是可见环境的属性在视网膜上成像的光模式中是混杂的。亮度是照明和反射率的函数，并且由于光传输的复杂性，可以依赖于大空间区域的环境属性。投影环境位置的图像位置充其量可以

be used to constrain the position of that location to a half-line. As a consequence, it is rarely possible to uniquely determine the nature of the world that produced a particular imaged pattern of light.

Determining *surface layout*—the location and orientation of visible surfaces in the environment—is thought to be a key step in human vision. Most discussions of how the vision system extracts information about surface layout from the patterns of light it receives divide the problem into a set of *visual cues*, with each cue describing a particular visual pattern which can be used to infer properties of surface layout along with the needed rules of inference. Since surface layout can rarely be determined accurately and unambiguously from vision alone, the process of inferring surface layout usually requires additional, nonvisual information. This can come from other senses or assumptions about what is likely to occur in the real world.

Visual cues are typically categorized into four categories. *Ocularmotor cues* involve information about the position and focus of the eyes. *Disparity cues* involve information extracted from viewing the same surface point with two eyes, beyond that available just from the positioning of the eyes. *Motion cues* provide information about the world that arises from either the movement of the observer or the movement of objects. *Pictorial cues* result from the process of projecting 3D surface shapes onto a 2D pattern of light that falls on the retina. This section deals with the visual cues relevant to the extraction of geometric information about individual points on surfaces. More general extraction of location and shape information is covered in Section 20.4.

20.3.1 Frames of Reference and Measurement Scales

Descriptions of the location and orientation of points on a visible surface must be done within the context of a particular frame of references that specifies the origin, orientation, and scaling of the coordinate system used in representing the geometric information. The human vision system uses multiple frames of reference, partially because of the different sorts of information available from different visual cues and partly because of the different purposes to which the information is put (Klatzky, 1998). *Egocentric* representations are defined with respect to the viewer's body. They can be subdivided into coordinate systems fixed to the eyes, head, or body. *Allocentric* representations, also called *exocentric* representations, are defined with respect to something external to the viewer. Allocentric frames of reference can be local to some configuration of objects in the environment or can be globally defined in terms of distinctive locations, gravity, or geographic properties.

用于将该位置的位置约束为半线。因此，很少能够唯一地确定产生特定成像光模式的世界的性质。

确定表面布局—环境中可见表面的位置和方向—被认为是人类视觉的关键步骤。大多数关于视觉系统如何从它接收的光模式中提取表面布局的信息的讨论将问题分为一组视觉线索，每个线索描述一个特定的视觉模式，可以用来推断表面布局的属性以及所需的推理规则。由于表面布局很少能从视觉上准确和明确地确定，因此推断表面布局的过程通常需要额外的非视觉信息。这可能来自于关于现实世界中可能发生的事情的其他感觉或假设。

视觉线索通常分为四类。*Ocularmotor*线索涉及关于眼睛的位置和焦点的信息。从用两只眼睛观察同一个表面点中提取的`volve`信息中的差异线索，超出了仅从眼睛的定位中获得的差异线索。运动线索提供了关于由观察者的运动或物体的运动产生的世界的信息。图像线索是将3d表面形状投射到落在视网膜上的2d光图案上的过程的结果。这部分处理与提取表面上的单个点的几何信息相关的视觉线索。位置和形状信息的更一般提取在第20.4节中。

20.3.1 参考框架和测量尺度

对可见表面上点的位置和方向的描述必须在指定用于表示几何信息的坐标系的原点、方向和缩放的特定参考框架的上下文中完成。人类视觉系统使用多个参照系，部分原因是不同视觉线索提供的信息种类不同，部分原因是信息的目的不同（Klatzky, 1998）。自我中心表示是相对于观看者的身体来定义的。它们可以细分为固定在眼睛、头部或身体上的坐标系。外心表示，也称为外心表示，是相对于查看器外部的东西定义的。中心参照系可以是环境中对象的某些配置的局部参照系，也可以根据独特的位置、重力或地理属性进行全局定义。

Cue	a	r	o	Requirements for Absolute Depth
Accommodation	x	x	x	very limited range
Binocular convergence	x	x	x	limited range
Binocular disparity	-	x	x	limited range
Linear perspective, height in picture, horizon ratio	x	x	x	requires viewpoint height
Familiar size	x	x	x	
Relative size	-	x	x	
Aerial perspective	?	x	x	adaptation to local conditions
Absolute motion parallax	?	x	x	requires viewpoint velocity
Relative motion parallax	-	-	x	
Texture gradients	-	x	-	
Shading	-	x	-	
Occlusion	-	-	x	

Table 20.1. Common visual cues for absolute (a), relative (r), and ordinal (o) depth.

The distance from the viewer to a particular visible location in the environment, expressed in an egocentric representation, is often referred to as *depth* in the perception literature. Surface orientation can be represented in either egocentric or allocentric coordinates. In egocentric representations of orientation, the term *slant* is used to refer to the angle between the line of sight to the point and the surface normal at the point, while the term *tilt* refers to the orientation of the projection of the surface normal onto a plane perpendicular to the line of sight.

Distance and orientation can be expressed in a variety of *measurement scales*. *Absolute* descriptions are specified using a standard that is not part of the sensed information itself. These can be culturally defined standards (e.g., meters), or standards relative to the viewer's body (e.g., eye height, the width of one's shoulders). *Relative* descriptions relate one perceived geometric property to another (e.g., point a is twice as far away as point b). *Ordinal* descriptions are a special case of relative measure in which the sign, but not the magnitude, of the relation is all that is represented. Table 20.1 provides a list of the most commonly considered visual cues, along with a characterization of the sorts of information they can potentially provide.

20.3.2 Ocularmotor Cues

Ocularmotor information about depth results directly from the muscular control of the eyes. There are two distinct types of ocularmotor information. *Accommo-*

Cue	a	r	o	绝对深度的要求
Accommodation	x	x	x	范围非常有限
双目收敛	x	x	x	有限范围
双眼视差	-	x	x	有限范围
线性透视, 高度图中, 视界比	x	x	x	需要视点高度
熟悉的尺寸	x	x	x	
相对尺寸	-	x	x	
空中透视	?	x	x	因地制宜
绝对运动视差	?	x	x	需要视点速度
相对运动视差	-	-	x	
纹理渐变	-	x	-	
Shading	-	x	-	
Occlusion	-	-	x	

表20.1。绝对 (a) , 相对 (r) 和序数 (o) 深度的常见视觉线索。

从观察者到环境中特定可见位置的距离，以自我中心表示，在感知文献中通常被称为深度。表面取向可以用egocen坐标或allocentric坐标表示。在定向的自我中心表示中，术语倾斜用于指到该点的视线与在该点处的表面法线之间的角度，而术语倾斜是指表面法线投影到垂直于视线的平面上的距离和方位可以用各种测量尺度来表示。

绝对描述使用不是感测信息本身的一部分的标准来指定。这些可以是文化上定义的标准（例如，米），或相对于观看者的身体的标准（例如，眼睛高度，一个人的眼睛的宽度）。相对描述将一个感知的几何属性与另一个感知的几何属性相关联（例如，点a的距离是点b的两倍）。序数描述是相对度量的一种特殊情况，其中关系的符号（而不是大小）是表示的全部。表20.1提供了最常见的视觉线索列表，以及它们可能提供的各种信息的表征。

20.3.2 Ocularmotor Cues

有关深度的Ocularmotor信息直接来自眼睛的肌肉控制。有两种不同类型的眼动信息。N.普通-

dation is the process by which the eye optically focuses at a particular distance. Convergence (often referred to as *vergence*) is the process by which the two eyes are pointed toward the same point in three-dimensional space. Both accommodation and convergence have the potential to provide absolute information about depth.

Physiologically, focusing in the human eye is accomplished by distorting the shape of the lens at the front of the eye. The vision system can infer depth from the amount of this distortion. Accommodation is a relatively weak cue to distance and is ineffective beyond about 2 m. Most people have increasing difficulty in focusing over a range of distances as they get beyond about 45 years old. For them, accommodation becomes even less effective.

Those not familiar with the specifics of visual perception sometimes confuse depth estimation from accommodation with depth information arising out of the blur associated with limited depth-of-field in the eye. The accommodation depth cue provides information about the distance to that portion of the visual field that it is in focus. It does not depend on the degree to which other portions of the visual field are out of focus, other than that blur is used by the visual system to adjust focus. Depth-of-field does seem to provide a degree of ordinal depth information (Figure 20.20), though this effect has received only limited investigation.

If two eyes fixate on the same point in space, trigonometry can be used to determine the distance from the viewer to the viewed location (Figure 20.21). For the simplest case, in which the point of interest is directly in front of the viewer,

$$z = \frac{ipd/2}{\tan \theta}, \quad (20.5)$$

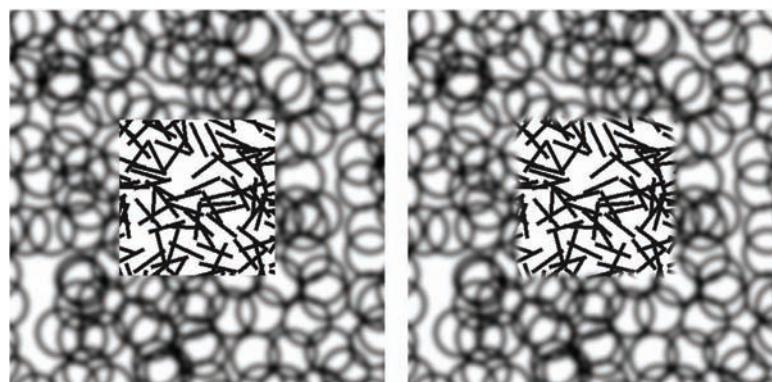


Figure 20.20. Does the central square appear in front of the pattern of circles or is it seen as appearing through a square hole in the pattern of circles? The only difference in the two images is the sharpness of the edge between the line and circle patterns (Marshall, Burbeck, Arely, Rolland, and Martin (1999), used by permission).

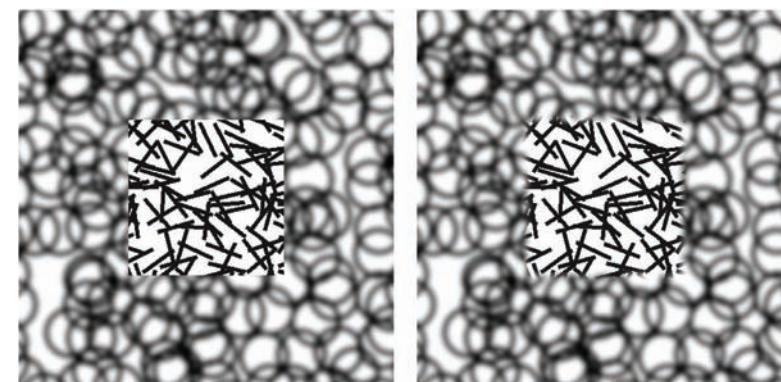
dation是眼睛在特定距离上光学聚焦的过程。收敛（通常称为收敛）是两只眼睛指向三维空间中的同一点的过程。Accommodation和convergence都有可能提供有关深度的绝对信息。

在生理上，人眼的聚焦是通过扭曲眼睛前部的晶状体形状来实现的。视觉系统可以从这种失真的量推断深度。住宿是一个相对较弱的距离提示，并且在超过2米时无效。大多数人在超过45岁的时候越来越难以集中注意力。对他们来说，住宿变得更加无效。

那些不熟悉视觉感知细节的人有时会将来自适应的深度估计与由于与眼睛中有限的景深相关的模糊而产生的深度信息混淆。调节深度提示提供了与它所聚焦的视场部分的距离有关的信息。它不依赖于视野的其他部分失焦的程度，除了模糊被视觉系统用来调整焦点。景深似乎确实提供了一定程度的序数深度信息（图20.20），尽管这种效应只得到了有限的调查。

如果两只眼睛固定在空间中的同一点上，则可以使用三角学来确定从观察者到被观察位置的距离（图20.21）。对于最简单的情况，其中兴趣点直接在观看者的前面

$$z = \frac{ipd/2}{\tan \theta}, \quad (20.5)$$



中央广场是否出现在圆形图案的前面，还是被视为通过圆形图案中的方形孔出现？两幅图像的唯一区别是线条和圆形图案之间边缘的清晰度（Marshall, Burbeck, Arely, Rolland and Martin (1999)，经许可使用）。

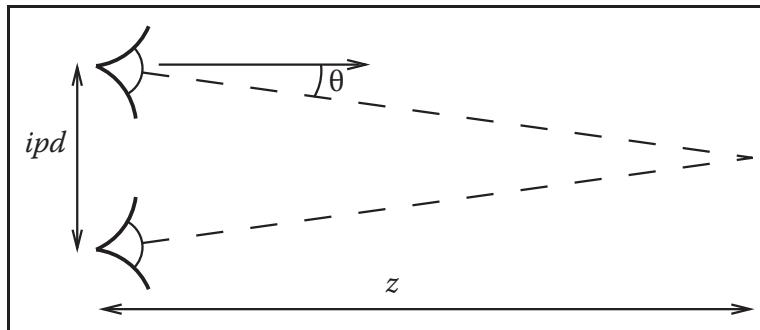


Figure 20.21. The *vergence* of the two eyes provides information about the distance to the point on which the eyes are fixated.

where z is the distance to a point in the world, ipd is the *interpupillary distance* indicating the distance between the eyes, and θ is the *vergence angle* indicating the orientation of the eyes relative to straight ahead. For small θ , which is the case for the geometric configuration of human eyes, $\tan \theta \approx \theta$ when θ is expressed in radians. Thus, differences in vergence angle specify differences in depth by the following relationship:

$$\Delta\theta \approx \frac{ipd}{2} \cdot \frac{1}{\Delta z}. \quad (20.6)$$

As $\theta \rightarrow 0$ in uniform steps, Δz gets increasingly larger. This means that stereo vision is less sensitive to changes in depth as the overall depth increases. Convergence in fact only provides information on absolute depth for distances out to a few meters. Beyond that, changes in distance produce changes in vergence angle that are too small to be useful.

There is an interaction between accommodation and convergence in the human visual system: accommodation is used to help determine the appropriate vergence angle, while vergence angle is used to help set the focus distance. Normally, this helps the visual system when there is uncertainty in setting either accommodation or vergence. However, stereographic computer displays break the relationship between focus and convergence that occurs in the real world, leading to a number of perceptual difficulties (Wann, Rushton, & Mon-Williams, 1995).

20.3.3 Binocular Disparity

The vergence angle of the eyes, when fixated on a common point in space, is only one of the ways that the visual system is able to determine depth from binocular stereo. A second mechanism involves a comparison of the retinal images in the

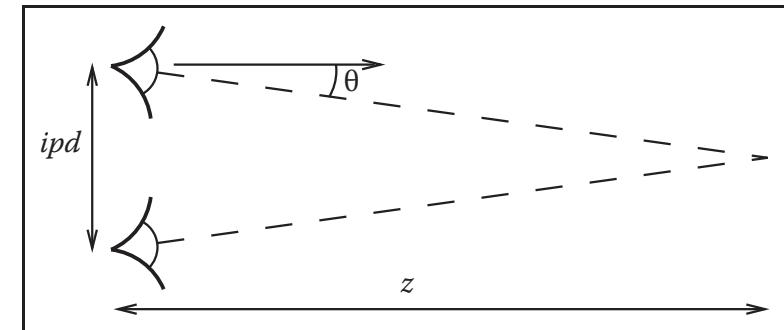


图20.21。两只眼睛的亮度提供了关于到眼睛所注视点的距离的信息。

其中 z 是到世界某一点的距离, ipd 是表示两眼之间距离的瞳孔间距离, θ 是表示两眼相对于直线前方的方位的俯角。对于小 θ , 也就是人眼的几何构型的情况, 当 θ 以弧度表示时, $\tan\theta\neq\theta$ 。因此, 收敛角的差异通过以下关系指定深度的差异:

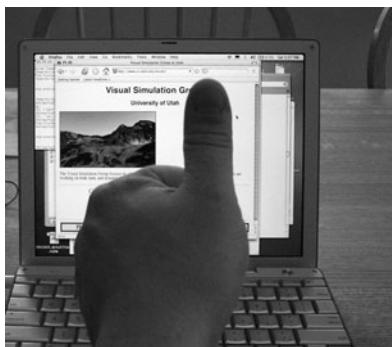
$$\Delta\theta \approx \frac{ipd}{2} \cdot \frac{1}{\Delta z}. \quad (20.6)$$

随着 $\theta\rightarrow 0$ 在均匀步长中, Δz 变得越来越大。这意味着立体视觉随着整体深度的增加而对深度的变化不太敏感。事实上, Convergence只提供了几米远的绝对深度信息。除此之外, 距离的变化会产生偏角的变化, 这些变化太小而不起作用。

在胡曼视觉系统中存在调节和收敛之间的相互作用: 调节用于帮助确定适当的偏斜角, 而偏斜角用于帮助设置焦点距离。也不是马利, 这有助于视觉系统时, 有不确定性设置accommodation或vergence。然而, 立体计算机显示器打破了现实世界中发生的焦点和收敛之间的关系, 导致了一些感知困难 (Wann, Rushton, & Mon-Williams, 1995)。

20.3.3 双眼视差

当眼睛聚焦在空间中的一个公共点上时, 眼睛的倾斜角只是视觉系统能够从双目立体中确定深度的方法之一。第二种机制涉及在视网膜图像的比较



(left eye image)



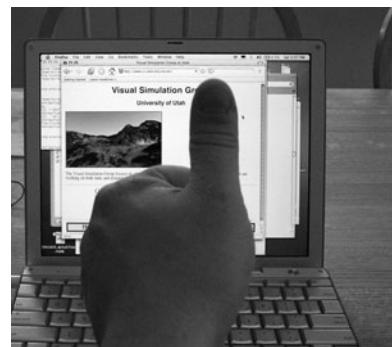
(right eye image)

Figure 20.22. Binocular disparity. The view from the left and right eyes shows an offset for surface points at depths different from the point of fixation. *Images courtesy Peter Shirley.*

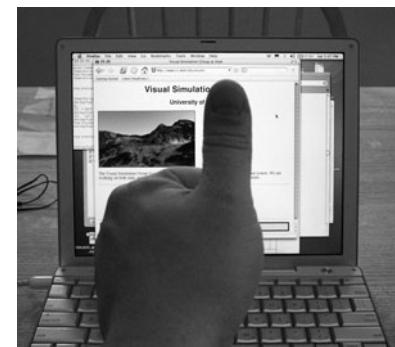
two eyes and does not require information about where the eyes are pointed. A simple example demonstrates the effect. Hold your arm straight out in front of you, with your thumb pointed up. Stare at your thumb and then close one eye. Now, simultaneously open the closed eye and close the open eye. Your thumb will appear to be more or less stationary, while the more distant surfaces seen behind your thumb will appear to move from side to side (Figure 20.22). The change in retinal position of points in the scene between the left and right eyes is called *disparity*.

The binocular disparity cue requires that the vision system be able to match the image of points in the world in one eye with the imaged locations of those points in the other eye, a process referred to as the *correspondence problem*. This is a relatively complicated process and is only partially understood. Once correspondences have been established, the relative positions on which particular points in the world project onto the left and right retinas indicate whether the points are closer than or farther away than the point of fixation. *Crossed disparity* occurs when the corresponding points are displaced outward relative to the fovea and indicates that the surface point is closer than the point of fixation. *Uncrossed disparity* occurs when the corresponding points are displaced inward relative to the fovea and indicates that the surface point is farther away than the point of fixation (Figure 20.23).⁴ Binocular disparity is a relative depth cue, but it can provide information about absolute depth when scaled by convergence. Equation (20.5) applies to binocular disparity as well as binocular convergence. As with

⁴Technically, crossed and uncrossed disparities indicate that the surface point generating the disparity is closer to or farther away from the *horopter*. The horopter is not a fixed distance away from the eyes but rather it is a curved surface passing through the point of fixation.



(left eye image)



(right eye image)

图20.22。双眼视差。左眼和右眼的视图显示了不同于注视点的深度处的表面点的偏移。图片由彼得·雪莉提供。

两只眼睛并且不需要关于眼睛指向哪里的信息。一个简单的例子演示了效果。把你的手臂伸直在你面前，拇指向上。盯着你的拇指，然后闭上一只眼睛。现在，同时打开关闭的眼睛和关闭打开的眼睛。你的拇指看起来或多或少是静止的，而你的拇指后面看到的更远的表面看起来会从一侧移动到另一侧（图20.22）。左右眼之间的场景中的点的视网膜位置的变化被称为视差。

双眼视差提示要求视觉系统能够将一只眼睛中的世界点的图像与另一只眼睛中的这些点的成像位置相匹配，这一过程称为对应问题。这是一个相对复杂的过程，只是部分理解。一旦确定了眼动反应，世界上特定点投射到左右视网膜上的相对位置表明这些点是比注视点更近还是更远。当相应的点相对于中央凹向外位移并且指示表面点比固定点更接近时，就会发生交叉视差。当相应点相对于中央凹向内位移时，会发生未交叉的视差，并表明表面点比固定点更远（图20.23）。4双眼视差是一种相对深度提示，但它可以在按收敛缩放时提供有关绝对深度的信息。等式（20.5）适用于双眼视差以及双眼收敛。

4从技术上讲，交叉和未交叉的差异表明产生差异的表面点更接近或更远离horopter。霍普特不是一个固定的距离远离眼睛，而是一个弯曲的表面通过固定点。

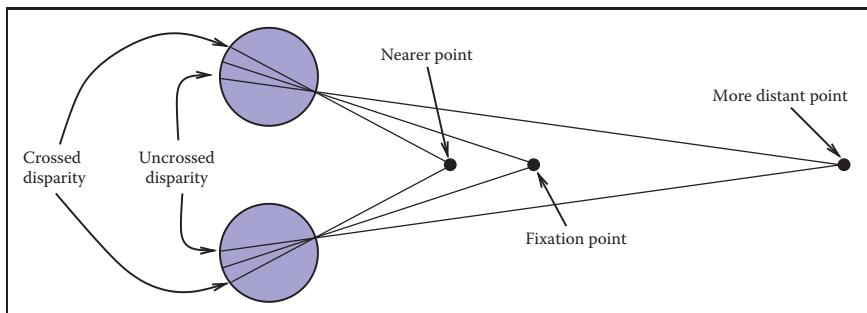


Figure 20.23. Near the line of sight, surface points nearer than the fixation point produce disparities in the opposite direction from those associated with surface points more distant than the fixation point.

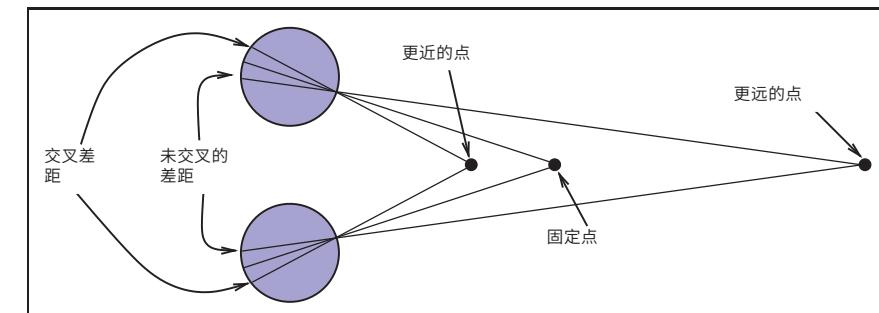
convergence, the sensitivity of binocular disparity to changes in depth decreases with depth.

20.3.4 Motion Cues

Relative motion between the eyes and visible surfaces will produce changes in the image of those surfaces on the retina. Three-dimensional relative motion between the eye and a surface point produces two-dimensional motion of the projection of the surface point on the retina. This retinal motion is given the name *optic flow*. Optic flow serves as the basis for several types of depth cues. In addition, optic flow can be used to determine information about how a person is moving in the world and whether or not a collision is imminent (Section 20.4.3).

If a person moves to the side while continuing to fixate on some surface point, then optic flow provides information about depth similar to stereo disparity. This is referred to as *motion parallax*. For other surface points that project to retinal locations near the fixation point, zero optic flow indicates a depth equivalent to the fixation point; flow in the opposite direction to head translation indicates nearer points, equivalent to crossed disparity; and flow in the same direction as head translation indicates farther points, equivalent to uncrossed disparity (Figure 20.24). Motion parallax is a powerful cue to relative depth. In principle, motion parallax can provide absolute depth information if the visual system has access to information about the velocity of head motion. In practice, motion parallax appears at best to be a weak cue for absolute depth.

In addition to egocentric depth information due to motion parallax, visual motion can also provide information about the three-dimensional shape of objects moving relative to the viewer. In the perception literature, this is known as the *kinetic depth effect*. In computer vision, it is referred to as *structure-from-*



在视线附近，比注视点更近的表面点在与比注视点更远的表面点相关的相反方向上产生差异。

收敛，双眼视差对深度变化的敏感性随深度而降低。

20.3.4 运动提示

眼睛和可见表面之间的相对运动将在视网膜上的那些表面的图像中产生变化。眼睛和表面点之间的三维相对运动产生表面点在视网膜上的投影的二维运动。这种视网膜运动被命名为视流。光学流作为几种类型的深度线索的基础。此外，opticflow可用于确定有关一个人在世界上如何移动以及碰撞是否迫在眉睫的信息（第20.4.3节）。

如果一个人在继续固定在某个表面点上的同时向侧面移动，则opticflow提供有关类似于立体视差的深度的信息。这被称为运动视差。对于投影到注视点附近的不同位置的其他表面点，零光流表示与注视点相当的深度；与头部平移相反方向的流动表示更近的点，相当于交叉的视差；而与头部平移相同方向的流动表示更远的点，相当于未交叉的视差（Figure 20.24）。运动视差是相对深度的有力提示。原则上，如果视觉系统能够访问关于头部运动速度的信息，运动视差可以提供绝对深度信息。在实践中，motionparallax充其量似乎是绝对深度的弱提示。

除了由于运动视差而产生的自我中心深度信息之外，视觉运动还可以提供关于相对于观看者移动的objects的三维形状的信息。在感知文献中，这被称为动力学深度效应。在计算机视觉中，它被称为结构-从

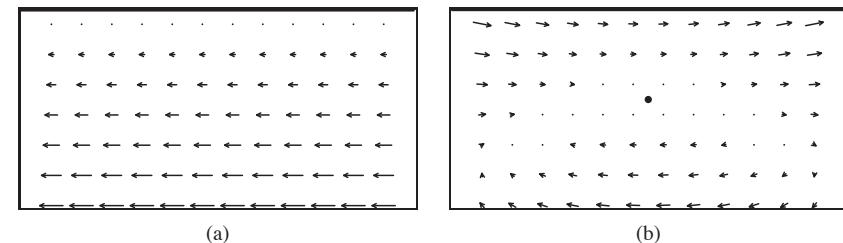


Figure 20.24. (a) Motion parallax generated by sideways movement to the right while looking at an extended ground plane. (b) The same motion, with eye tracking of the fixation point.

motion. The kinetic depth effect presumes that one component of object motion is *rotation in depth*, meaning that there is a component of rotation around an axis perpendicular to the line of sight.

Optic flow can also provide information about the shape and location of surface boundaries, as shown in Figure 20.25. Spatial discontinuities in optic flow almost always either correspond to depth discontinuities or result from independently moving objects. Simple comparisons of the magnitude of optic flow are insufficient to determine the sign of depth changes, except in the special case of a viewer moving through an otherwise static world. Even when independently moving objects are present, however, the sign of the change in depth across surface boundaries can often be determined by other means. Motion often changes the portion of the more distant surface visible at surface boundaries. The appearance (*accretion*) or disappearance (*deletion*) of surface texture occurs because the nearer, occluding surface progressively uncovers or covers portions of the more distant, *occluded* surface. Comparisons of the motion of surface texture to either side of a boundary can also be used to infer ordinal depth, even in the absence of accretion or deletion of the texture. Discontinuities in optic flow and accretion/deletion of surface texture are referred to as *dynamic occlusion* cues and are another powerful source of visual information about the spatial structure of the environment.

The speed that a viewer is traveling relative to points in the world cannot be determined from visual motion alone (see Section 20.4.3). Despite this limitation, it is possible to use visual information to determine the time it will take to reach a visible point in the world, even when speed cannot be determined. When velocity is constant, *time-to-contact* (often referred to as *time-to-collision*) is given by the retinal size of an entity toward which the observer is moving, divided by the rate at which that image size is increasing.⁵ In the biological vision literature, this is

⁵The terms *time-to-collision* and *time-to-contact* are misleading, since contact will only occur if the viewer's trajectory actually passes through or near the entity under view.

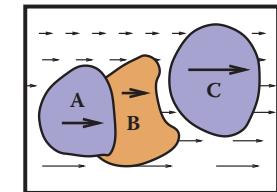
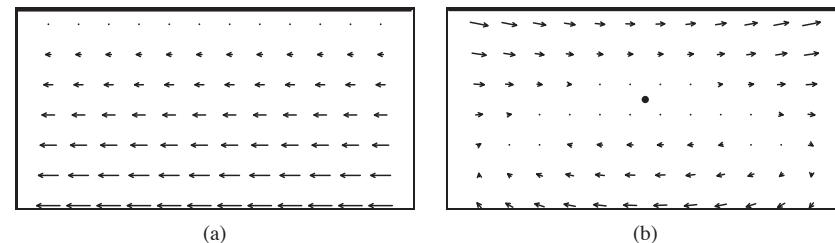


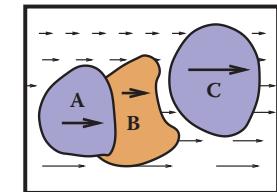
Figure 20.25. Discontinuities in optic flow signal surface boundaries. In many cases, the sign of the depth change (i.e., the ordinal depth) can be determined.



(a) 在观察扩展的接地平面时向右侧向运动产生的运动视差。(b) 相同的运动, 用眼动追踪注视点。

运动。动力学深度效应假定物体运动的一个分量是深度旋转, 这意味着存在围绕垂直于视线的轴旋转的分量。

光流还可以提供关于面部边界形状和位置的信息, 如图20.25所示。光流中的空间不连续性几乎总是与深度不连续性相对应, 或者是由于物体的深度不连续性引起的。光流大小的简单比较不足以确定深度变化的迹象, 除非观看者在静态世界中移动。然而, 即使当存在独立运动的物体时, 跨越面部边界的深度变化的标志也常常可以通过其他手段来确定。运动经常改变在表面边界处可见的较远表面的部分。表面纹理的出现ance (吸积) 或消失 (缺失) 是因为较近的闭塞表面逐渐揭开或复盖较远的闭塞表面的一部分。将表面纹理的运动与边界的任一侧进行比较也可以用来推断序数深度, 即使在纹理没有增加或删除的情况下也是如此。光流中的不连续性和表面纹理的缺失被称为动态遮挡线索, 是关于环境空间结构的另一个强大的视觉信息来源。



光流信号表面边界的不均匀性。
在许多情况下, 可以确定深度变化的标志 (即序数深度)。

观看者相对于世界上的点移动的速度不能仅仅从视觉运动来确定 (见第20.4.3节)。尽管存在这种限制, 但可以使用视觉信息来确定到达世界上可见点所需的时间, 即使在无法确定速度的情况下也是如此。当速度恒定时, 接触时间 (通常称为碰撞时间) 由观察者移动的实体的视网膜大小除以图像大小增加的速度给出。5在生物视觉文献中, 这是

5术语碰撞时间和接触时间具有误导性, 因为只有当观看者的轨迹实际穿过或靠近所看到的实体时, 才会发生接触。

often called the τ function (Lee & Reddish, 1981). If distance information to the structure in the world on which the time-to-collision estimate is based is available, then this can be used to determine speed.

20.3.5 Pictorial Cues

An image can contain much information about the spatial structure of the world from which it arose, even in the absence of binocular stereo or motion. As evidence for this, note that the world still appears three-dimensional even if we close one eye, hold our head stationary, and nothing moves in the environment. (As discussed in Section 20.5, the situation is more complicated in the case of photographs and other displayed images.) There are three classes of such *pictorial depth cues*. The best known of these involve *linear perspective*. There are also a number of *occlusion cues* that provide information about ordinal depth even in the absence of perspective. Finally, *illumination cues* involving shading, shadows and interreflections, and aerial perspective also provide visual information about spatial layout.

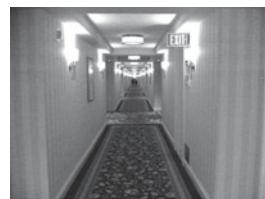


Figure 20.26. The classical linear perspective effects include object size scaled by distance, the convergence of parallel lines, the ground plane extending to a visible horizon, and the relationship between the distance to objects on the ground plane and the image location of those objects relative to the horizon (Figure 20.26). More formally, linear perspective cues are those visual cues which exploit the fact that under perspective projection, the image location onto which points in the world are projected is scaled by $\frac{1}{z}$, where z is the distance from the point of projection to the point in the environment. Direct consequences of this relationship are that points that are farther away are projected to points closer to the center of the image (convergence of parallel lines) and that the spacing between the image of points in the world decreases for more distant world points (object size in the image is scaled by distance).⁶ The fact that the image of an infinite flat surface in the world ends at a finite horizon is explained by examining the perspective projection equation as $z \rightarrow \infty$.

With the exception of size-related effects described in Section 20.4.2, most pictorial depth cues involving linear perspective depend on objects of interest being in contact with a ground plane. In effect, these cues estimate not the distance to the objects but, instead, the distance to the contact point on the ground plane. Assuming observer and object are both on top of a horizontal ground plane, then

⁶The actual mathematics for analyzing the specifics of biological vision are different, since eyes are not well approximated by the planar projection formulation used in computer graphics and most other imaging applications.

常被称为 τ 函数 (Lee & Reddish, 1981)。如果到碰撞时间估计所基于的世界中的结构的距离信息是可用的，那么这可用于确定速度。

20.3.5 图片线索

一幅图像可以包含很多关于它产生的世界空间结构的信息，即使在没有双目立体或运动的情况下也是如此。作为证据，请注意，即使我们闭上一只眼睛，保持头部静止，环境中没有任何东西移动，世界仍然呈现三维。（如第20.5节所述，在照片和其他显示图像的情况下，情况更复杂。）有三类这样的图像深度线索。其中最著名的涉及线性透视。还有



Classical线性透视效果包括按距离缩放的对象大小、平行线的会聚、延伸到可见地平线的地平面以及地平面上相对于地平线的位置。图片由SamPullara提供。

许多遮挡线索，即使在没有透视的情况下也能提供关于序数深度的信息。最后，涉及阴影、阴影和反射以及空中透视的照明线索也提供了关于空间布局的视觉信息。

术语线性透视通常用于指图像中按距离缩放的物体大小的图像属性，平行线的收敛，延伸到可见地平线的地平面，以及与地平面上物体的距离和这些物体相对于地平线的图像位置的关系（图20.26）。更正式地说，线性透视线索是那些利用透视投影下的事实的视觉线索，将世界上点投影到的图像位置按1缩放

z ，其中 z 是从投影点到周围点的距离。这种关系的直接后果是，距离较远的点被投影到距离图像中心较近的点（平行线的收敛），并且对于距离较远的世界点，世界上点的图像之间的间距减小（图像中的对象大小按距离缩放）。6世界中无限平坦表面的图像以有限地平线结束的事实通过检查透视投影方程为 $z \rightarrow \infty$ 来解释。

除了在第20.4.2节中描述的与尺寸相关的效果外，涉及线性透视的大多数图像深度线索取决于感兴趣的对象与地平面接触。实际上，这些线索不是估计到物体的距离，而是估计到地平面上接触点的距离。假设观察者和物体都在水平地平面之上，那么

6分析生物视觉细节的实际数学是不同的，因为在计算机图形学和大多数其他成像应用中使用的平面投影公式不能很好地近似眼睛。

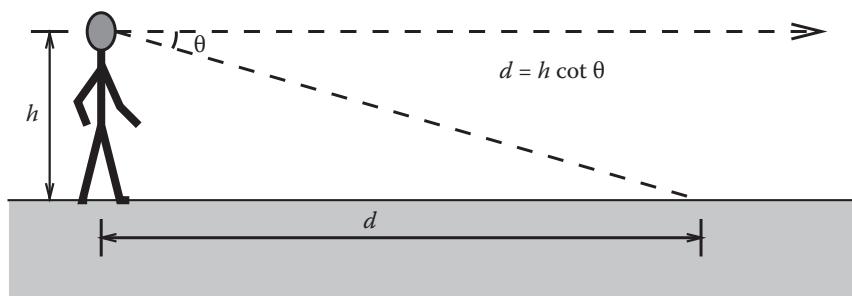


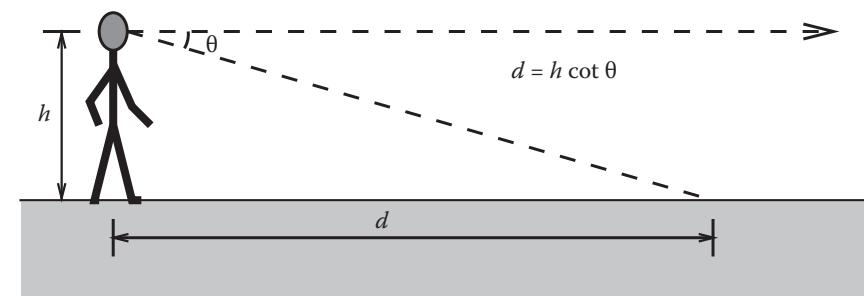
Figure 20.27. Absolute distance to locations on the ground plane can be determined based on declination angle from the horizon and eye height.

locations on the ground plane lower in the view will be close. Figure 20.27 illustrates this effect quantitatively. For a viewpoint h above the ground and an *angle of declination* θ between the horizon and a point of interest on the ground, the point in question is a distance $d = h \cot \theta$ from the point at which the observer is standing. The angle of declination provides relative depth information for arbitrary fixed viewpoints and can provide absolute depth when scaling by eye height (h) is possible.

While the human visual system almost certainly makes use of angle of declination as a depth cue, the exact mechanisms used to acquire the needed information are not clear. The angle θ could be obtained relative to either gravity or the visible horizon. There is some evidence that both are used in human vision. Eye height h could be based on posture, visually determined by looking at the ground at one's feet, or learned by experience and presumed to be constant. While a number of researchers have investigated this issue, if and how these values are determined is not yet known with certainty.

Shadows provide a variety of types of information about three-dimensional spatial layout. *Attached shadows* indicate that an object is in contact with another surface, often consisting of the ground plane. *Detached shadows* indicate that an object is close to some surface, but not in contact with that surface. Shadows can serve as an indirect depth cue by causing an object to appear at the depth of the location of the shadow on the ground plane (Yonas, Goldsmith, & Hallstrom, 1978). When utilizing this cue, the visual system seems to make the assumption that light is coming from directly above (Figure 20.28).

Vision provides information about surface orientation as well as distance. It is convenient to represent visually determined surface orientation in terms of *tilt*, defined as the orientation in the image of the projection of the surface normal, and *slant*, defined as the angle between the surface normal and the line of sight.



与地平面上的位置的绝对距离可以根据与地平线的偏角和眼睛高度来确定。

视图中较低的地平面上的位置将接近。图20.27定量地说明了这种效应。对于地面上方的视点 h 和地平线与地面上的兴趣点之间的偏角 θ ，所讨论的点是距观察者所站立的点的距离 $d=h\cot\theta$ 。偏角为任意固定视点提供相对深度信息，并且可以在按眼睛高度（ h ）缩放时提供绝对深度。

虽然人类视觉系统几乎可以肯定地使用赤纬角作为深度提示，但用于获取所需信息的确切机制尚不清楚。角 θ 可以相对于重力或可见地平线获得。有一些证据表明两者都用于人类视觉。眼睛高度 h 可以基于姿势，通过观察脚上的地面视觉确定，或者通过经验学习并假定为恒定的。虽然许多研究人员已经调查了这个问题，但这些值是否以及如何确定尚不清楚。

阴影提供关于三维空间布局的多种类型的信息。附加阴影表示物体与另一个表面接触，通常由接地层组成。分离阴影表示对象靠近某个表面，但不与该表面接触。阴影可以通过使对象出现在地平面上阴影位置的深度来作为间接深度提示 (Yonas, Goldsmith, & Hallstrom, 1978)。当利用这个提示时，视觉系统似乎假设光线来自正上方 (图20.28)。

视觉提供有关表面方向以及距离的信息。在倾斜方面方便地表示视觉上确定的表面取向，定义为表面法线的投影的图像中的取向，以及倾斜，定义为表面法线与视线之间的角度。

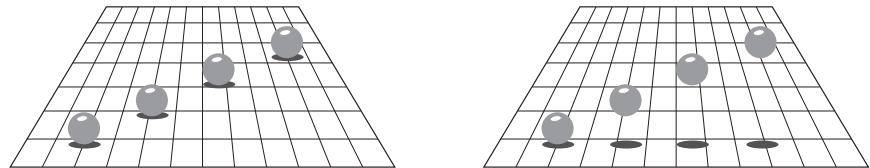


Figure 20.28. Shadows can indirectly function as a depth cue by associating the depth of an object with a location on the ground plane (after Kersten, Mamassian, and Knill (1997)).

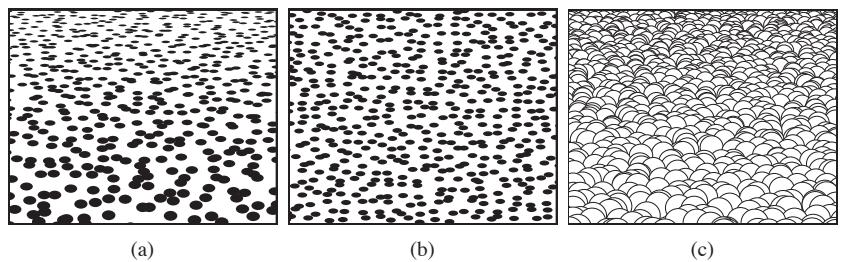


Figure 20.29. Texture cues for slant. (a) Near surface exhibiting compression and texture gradient; (b) distant surface exhibiting only compression; (c) variability in appearance of near surface with regular geometric variability.

A visible surface horizon can be used to find the orientation of an (effectively infinite) surface relative to the viewer. Determining tilt is straightforward, since the tilt of the surface is the orientation of the visible horizon. Slant can be recovered as well, since the lines of sight from the eye point to the horizon define a plane parallel to the surface. In many situations, either the surface horizon is not visible or the surface is small enough that its far edge does not correspond to an actual horizon. In such cases, visible texture can still be used to estimate orientation.

In the context of perception, the term *texture* refers to visual patterns consisting of sub-patterns replicated over a surface. The sub-patterns and their distribution can be fixed and regular, as for a checkerboard, or consistent in a more statistical sense, as in the view of a grassy field.⁷ When a textured surface is viewed from an oblique angle, the projected view of the texture is distorted relative to the actual markings on the surface. Two quite distinct types of distortions occur (Knill, 1998), both affected by the amount of slant. The position and size of texture elements are subject to the linear perspective effects described above. This produces a *texture gradient* (Gibson, 1950) due to both element size and spacing decreasing with distance (Figure 20.29(a)). Both the image of individual

⁷In computer graphics, the term *texture* has a different meaning, referring to any image that is applied to a surface as part of the rendering process.

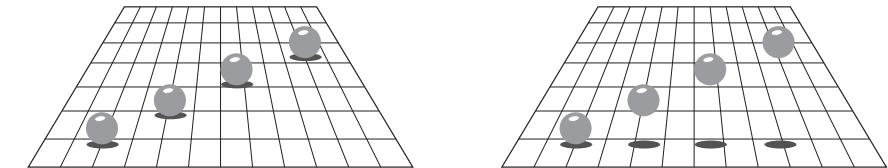


图20.28。阴影可以通过将对象的深度与地平面上的位置相关联（在Kersten Mamassian And Knill(1997)之后）来间接地充当深度提示。

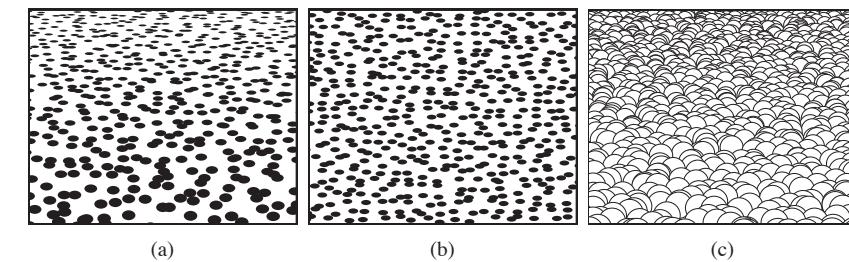
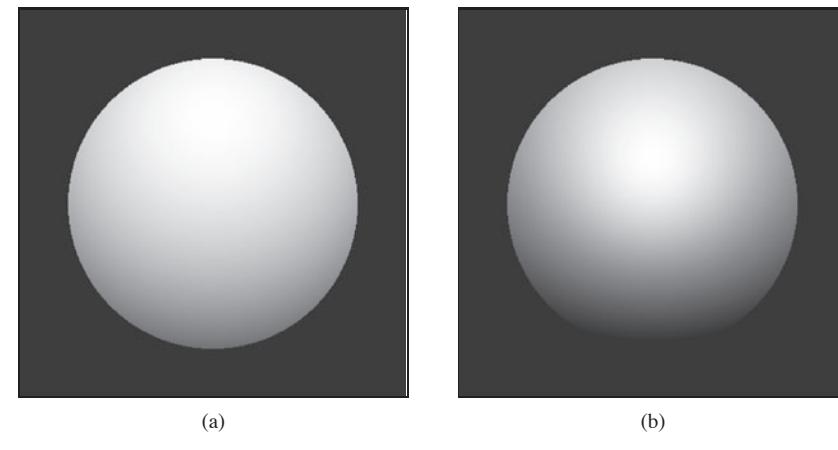


图20.29。倾斜的纹理线索。(a)表现出压缩和纹理梯度的近表面;(b)仅表现出压缩的远表面;(c)具有规则几何变异性的近表面外观的变异性。

可见表面视界可用于查找（实际上是无限的）表面相对于观看者的方向。确定倾斜是直接的，因为表面的倾斜是可见地平线的方向。倾斜也可以被重新覆盖，因为从眼睛点到地平线的视线定义了一个平行于表面的平面。在许多情况下，要么表面视界不可见，要么表面足够小，以至于其远边与实际视界不对应。在此类情况下，可见纹理仍可用于估计取向。

在感知的背景下，术语纹理是指视觉模式由复制在表面上的子模式组成。子模式和它们的距离可以是固定的和规则的，就像棋盘一样，或者在更统计的意义上是一致的，就像在草地上一样。当从斜角观察纹理表面时，纹理的投影视图会根据表面上的实际标记而扭曲。发生两种截然不同的扭曲类型 (Knill, 1998)，两者都受到倾斜量的影响。纹理元素的位置和大小受上述线性透视效果的影响。这会产生纹理梯度 (Gibson, 1950)，因为元素大小和间距都随着距离而减小（图20.29 (a)）。个人的形象

⁷在计算机图形学中，术语纹理具有不同的含义，指的是作为渲染过程的一部分应用于表面的任何图像。



(a)

(b)

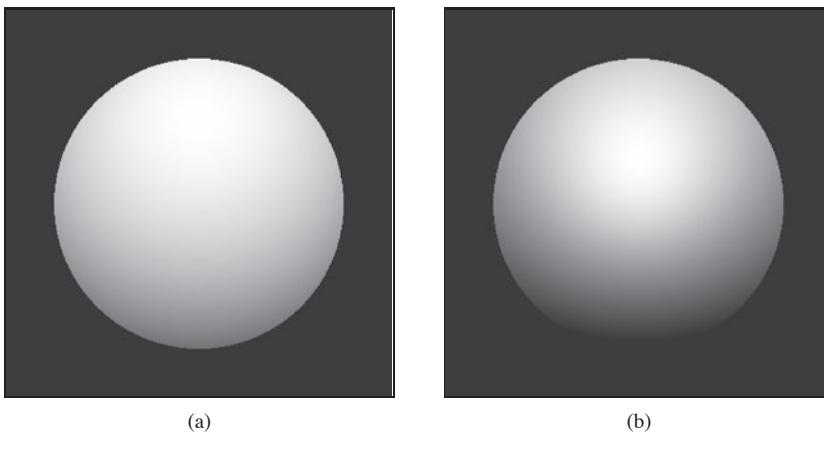
Figure 20.30. Shape-from-shading. The images in (a) and (b) appear to have different 3D shapes because of differences in the rate of change of brightness over their surfaces.

texture elements and the distribution of elements are *foreshortened* under oblique viewing (Figure 20.29(b)). This produces a compression in the direction of tilt. For example, an obliquely viewed circle appears as an ellipse, with the ratio of the minor to major axes equal to the cosine of the slant. Note that foreshortening itself is not a result of linear perspective, though in practice both linear perspective and foreshortening provide information about slant.⁸

For texture gradients to serve as a cue to surface slant, the average size and spacing of texture elements must be constant over the textured surface. If spatial variability in size and spacing in the image is not due entirely to the projection process, then attempts to invert the effects of projection will produce incorrect inferences about surface orientation. Likewise, the foreshortening cue fails if the shape of texture elements is not isotropic, since then asymmetric texture element image shapes would occur in situations not associated with oblique viewing. These are examples of the assumptions often required in order for spatial visual cues to be effective. Such assumptions are reasonable to the degree that they reflect commonly occurring properties of the world.

Shading also provides information about surface shape (Figure 20.30). The brightness of viewed points on a surface depends on the surface reflectance and the orientation of the surface with respect to directional light sources and the observation point. When the relative position of an object, viewing direction, and illumination direction remain fixed, changes in brightness over a constant

⁸A third form of visual distortion occurs when surfaces with distinct 3D surface relief are viewed obliquely (Leung & Malik, 1997), as shown in Figure 20.29(c). Nothing is currently known about if or how this effect might be used by the human vision system to determine slant.



(a)

(b)

形状-从阴影。 (A) 和 (b) 中的图像似乎具有不同的3D形状，因为其表面上的亮度变化率存在差异。

纹理元素和元素的分布在斜视下被缩短（图20.29 (b)）。这在倾斜方向上产生压缩。例如，斜向观察的圆显示为椭圆，短轴与长轴的比率等于倾斜的余弦。请注意，前缩本身不是线性透视的结果，尽管在实践中，线性透视和前缩都提供了有关倾斜的信息。⁸

要使纹理渐变作为表面倾斜的提示，纹理元素的平均大小和间距必须在纹理表面上保持不变。如果图像中大小和间距的spatial可变性不是完全由于投影过程，那么试图反转投影的影响将产生关于表面取向的错误推断。同样，如果纹理元素的形状不是各向同性的，则预缩提示也会失败，因为在与斜视无关的情况下，不对称的纹理元素图像形状会发生。这些是为了使spatial视觉线索有效而经常需要的假设的例子。这种假设在一定程度上是合理的，因为它们反映了世界上常见的性质。

阴影还提供有关表面形状的信息（图20.30）。表面上观察点的亮度取决于表面反射率以及表面相对于定向光源和观察点的取向。当物体的相对位置、观察方向和照明方向保持固定时，亮度的变化会超过一个常数

⁸第三种形式的视觉扭曲发生在具有明显3D表面浮雕的表面倾斜观察时（Leung & Malik, 1997），如图20.29 (c) 所示。目前还不知道人类视觉系统是否或如何使用这种效应来确定倾斜。

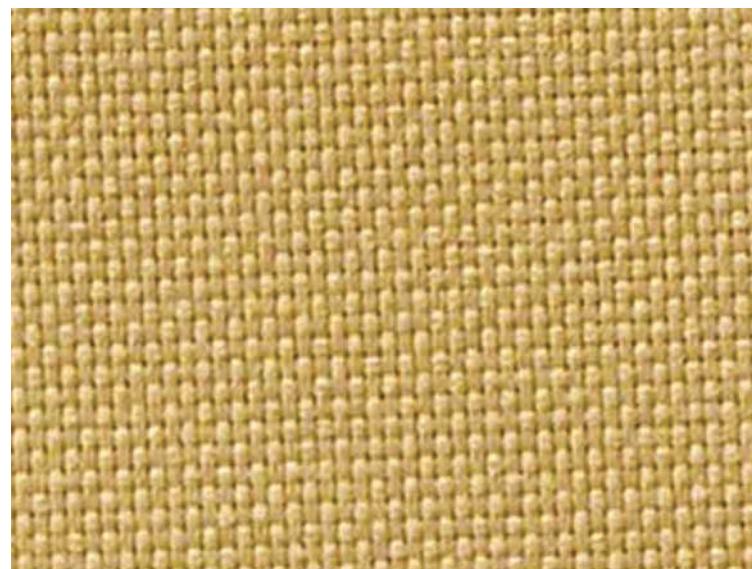


Figure 20.31. Shading can generate a strong perception of three-dimensional shape. In this figure, the effect is stronger if you view the image from several meters away using one eye. It becomes yet stronger if you place a piece of cardboard in front of the figure with a hole cut out slightly smaller than the picture (see Section 20.5). *Image courtesy Albert Yonas.*

reflectance surface are indications of changes in the orientation of the surface of the object. *Shape-from-shading* is the process of recovering surface shape from these variations in observed brightness. It is almost never possible to recover the actual orientation of surfaces from shading alone, though shading can often be combined with other cues to provide an effective indication of surface shape. For surfaces with fine-scale geometric variability, shading can provide a compelling three-dimensional appearance, even for an image rendered on a two-dimensional surface (Figure 20.31).

There are a number of pictorial cues that yield ordinal information about depth, without directly indicating actual distance. In line drawings, different types of junctions provide constraints on the 3D geometry that could have generated the drawing (Figure 20.32). Many of these effects occur in more natural images as well. Most perceptually effective of the junction cues are *T-junctions*, which are strong indicators that the surface opposite the stem of the T is occluding at least one more distant surface. T-junctions often generate a sense of *amodal completion*, in which one surface is seen to continue behind a nearer, occluding surface (Figure 20.33).

Atmospheric effects cause visual changes that can provide information about depth, particularly outdoors over long distances. Leonardo da Vinci was the first

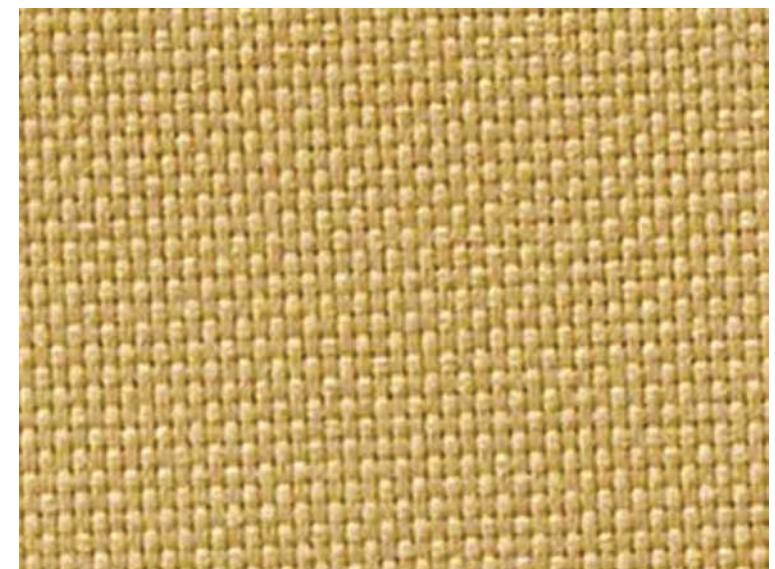


图20.31。阴影可以产生三维形状的强烈感知。在这个图中，如果你用一只眼睛从几米远的地方观看图像，效果会更强。如果你把一块纸板放在图的前面，上面有一个比图略小的孔（见第20.5节），它就会变得更强。图像礼貌阿尔伯特Yonas.

反射表面是物体表面取向变化的指示。从阴影中恢复形状是从观察到的亮度的这些变化中恢复表面形状的过程。仅仅从阴影中恢复表面的实际方向几乎是不可能的，尽管阴影通常可以与其他线索结合起来提供表面形状的有效指示。对于具有精细尺度几何可变性的表面，着色可以提供引人注目的三维外观，即使对于在二维表面上渲染的图像也是如此（图20.31）。

有许多图形线索产生关于深度的序数信息，而不直接指示实际距离。在线条图中，不同类型的交汇点对可能生成绘图的3d几何体提供了约束（图20.32）。其中许多效果也发生在更自然的图像中。大多数感知上有效的连接线索是T-连接，这是强烈的指标，表明与t茎相对的表面正在遮挡至少一个更远的表面。T型连接通常会产生一种amodalcompletion感，其中一个表面在更近的遮挡表面后面继续（图20.33）。

大气效应导致视觉变化，可以提供有关深度的信息，特别是长距离的户外。列奥纳多·达·芬奇是第一个

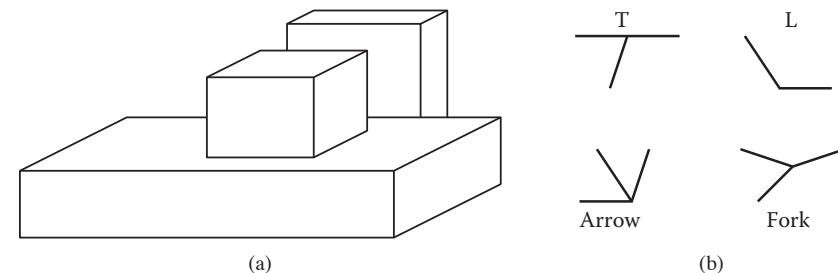


Figure 20.32. (a) Junctions provide information about occlusion and the convexity or concavity of corners. (b) Common junction types for planar surface objects.



Figure 20.33. T-junctions cause the left disk to appear to be continuing behind the rectangle, while the right disk appears in front of the rectangle, which is seen to continue behind the disk.

to describe *aerial perspective* (also called *atmospheric perspective*), in which scattering reduces the contrast of distant portions of the scene and causes them to appear more bluish than if they were nearer (da Vinci, 1970) (see Figure 20.34). Aerial perspective is predominately a relative depth cue, though there is some speculation that it may affect perception of absolute distance as well. While many people believe that more distant objects look blurrier due to atmospheric effects, atmospheric scattering actually causes little blur.

20.4 Objects, Locations, and Events

While there is fairly wide agreement among current vision scientists that the purpose of vision is to extract information about objects, locations, and events, there is little consensus on the key features of what information is extracted, how it is extracted, or how the information is used to perform tasks. Significant controversies exist about the nature of object recognition and the potential interactions between object recognition and other aspects of perception. Most of what we know about location involves low-level spatial vision, not issues associated with spatial relationships between complex objects or the visual processes required to

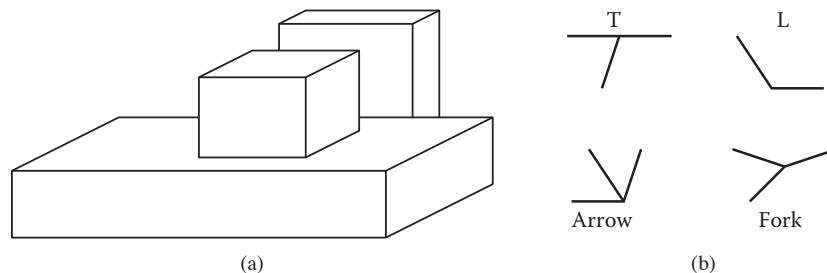


图20.32。 (a)交汇点提供有关遮挡和拐角凸度或凹度的信息。 (b)平面表面物体的常见结类型。



图20.33. T型连接导致左盘似乎继续在矩形后面，而右盘出现在矩形前面，这被认为是继续在磁盘后面。

描述空中透视（也称为大气透视），其中散射降低了场景远处部分的对比度，并使它们看起来比它们更近时更蓝（da Vinci, 1970）（见图20.34）。空中视角主要是一个相对深度的线索，尽管有一些猜测，它可能会影响绝对距离的感知。虽然许多人认为更远的物体由于大气效应而看起来更模糊，但大气散射实际上导致很少模糊。

20.4 对象、位置和事件

虽然目前的视觉科学家之间有相当广泛的共识，即视觉的目的是提取有关对象，位置和事件的信息，但对于提取哪些信息，如何提取信息或如何使用信息关于物体识别的性质以及物体识别与感知的其他方面之间的潜在相互作用存在重大争议。我们所知道的关于位置的大部分内容都涉及低层次的空间视觉，而不是与复杂物体之间的空间关系或所需的视觉过程相关的问题。



Figure 20.34. Aerial perspective, in which atmospheric effects reduce contrast and shift colors toward blue, provides a depth cue over long distances.

navigate in complex environments. We know a fair amount about how people perceive their speed and heading as they move through the world, but have only a limited understanding of actual event perception. Visual attention involves aspects of the perception of objects, locations, and events. While there is much data about the phenomenology of visual attention for relatively simple and well-controlled stimuli, we know much less about how visual attention serves high-level perceptual goals.

20.4.1 Object Recognition

Object recognition involves segregating an image into constituent parts corresponding to distinct physical entities and determining the identity of those entities. Figure 20.35 illustrates a few of the complexities associated with this process. We have little difficulty recognizing that the image on the left is some sort of vehicle, even though we have never before seen this particular view of a vehicle nor do most of us typically associate vehicles with this context. The image on the right is less easily recognizable until the page is turned upside down, indicating an orientational preference in human object recognition.

Object recognition is thought to involve two, fairly distinct steps. The first step organizes the visual field into *groupings* likely to correspond to objects and surfaces. These grouping processes are very powerful (see Figure 20.36), though there is little or no conscious awareness of the low-level image features that gener-



空中透视，其中大气效果降低对比度并将颜色转向蓝色，提供了长距离的深度提示。

在复杂环境中导航。我们对人们在世界各地移动时如何感知他们的速度和航向有相当多的了解，但对实际事件感知的理解有限。视觉注意涉及物体，位置和事件的感知方面。虽然关于相对简单和控制良好的刺激的视觉注意现象学的数据很多，但我们对视觉注意如何服务于高层次的感知目标知之甚少。

20.4.1 物体识别

对象识别涉及将图像分离成不同的物理实体的组成部分，并确定这些实体的身份。图20.35说明了与此过程相关的一些复杂性。我们很难认识到左边的图像是某种激烈的，尽管我们以前从未见过这种特定的车辆视图，也没有我们大多数人通常将车辆与这种背景联系起来。右侧的图像不太容易识别，直到页面被颠倒，表明人类对象识别中的定向偏好。

对象识别被认为涉及两个相当不同的步骤。第一步将视野组织成可能对应于对象和表面的分组。这些分组过程非常强大（见图20.36），尽管很少或根本没有意识到所产生的低级图像特征

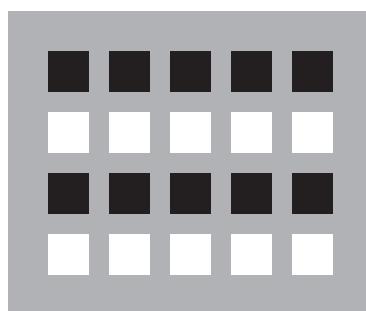


(a)

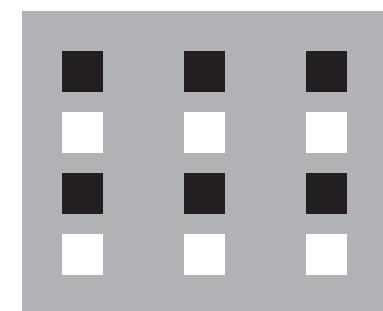


(b)

Figure 20.35. The complexities of object recognition. (a) We recognize a vehicle-like object even though we have likely never seen this particular view of a vehicle before. (b) The image is hard to recognize based on a quick view. It becomes much easier to recognize if the book is turned upside down.



(a)



(b)

Figure 20.36. Images are perceptually organized into groupings based on a complex set of similarity and organizational criteria. (a) Similarity in brightness results in four horizontal groupings. (b) Proximity resulting in three vertical groupings.

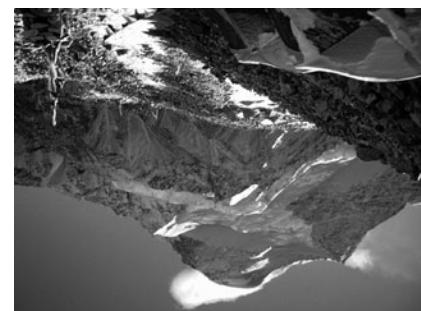
ate the grouping effect.⁹ Grouping is based on the complex interaction of proximity, similarities in the brightness, color, shape, and orientation of primitive structures in the image, common motion, and a variety of more complex relationships.

The second step in object recognition is to interpret groupings as identified objects. A computational analysis suggests that there are a number of distinctly different ways in which an object can be identified. The perceptual data is unclear as to which of these are actually used in human vision. Object recognition requires that the vision system have available to it descriptions of each class of object

⁹The most common form of visual camouflage involves adding visual textures that fool the perceptual grouping processes so that the view of the world cannot be organized in a way that separates out the object being camouflaged.

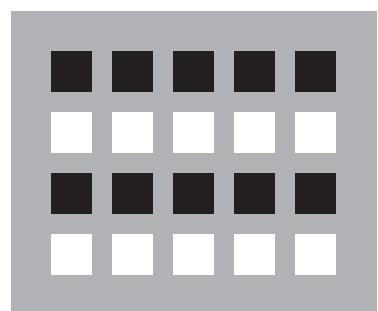


(a)



(b)

对象识别的复杂性。(a)即使我们以前可能从未见过这种特定的车辆视图，我们也能识别出类似车辆的物体。(b)图像很难根据快速视图识别。它变得更容易识别，如果这本书是颠倒的。



(a)



(b)

图像根据一组复杂的相似度和组织标准被感知地组织成组。(a)亮度的相似性导致四个水平分组。(b)接近导致三个垂直分组。

吃了分组效果。9分组是基于接近、图像中基元结构的亮度、颜色、形状和取向的相似性、共同运动以及各种更复杂的关系的复杂相互作用。对象识别的第二步是将分组解释为已识别的对象。一个计算分析表明，有许多明显不同的方法可以识别一个物体。感知数据尚不清楚其中哪些实际上用于人类视觉。对象识别要求视觉系统对每类对象都有可用的描述

⁹最常见的视觉伪装形式涉及添加欺骗感知分组过程的视觉纹理，以便世界视图不能以分离被伪装的对象的方式组织。

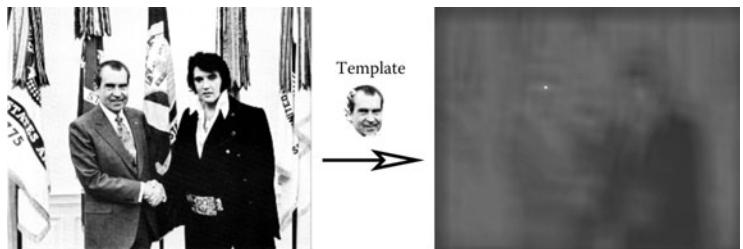


Figure 20.37. Template matching. The bright spot in the right image indicates the best match location to the template in the left image. *Image courtesy National Archives and Records Administration.*

sufficient to discriminate each class from all others. Theories of object recognition differ in the nature of the information describing each class and the mechanisms used to match these descriptions to actual views of the world.

Three general types of descriptions are possible. *Templates* represent object classes in terms of prototypical views of objects in each class. Figure 20.37 shows a simple example. *Structural descriptions* represent object classes in terms of distinctive features of each class likely to be easily detected in views of the object, along with information about the geometric relationships between the features. Structural descriptions can either be represented in 2D or 3D. For 2D models of objects types, there must be a separate description for each distinctly different potential view of the object. For 3D models, two distinct forms of matching strategies are possible. In one, the three-dimensional structure of the viewed object is determined prior to classification using whatever spatial cues are available, and then this 3D description of the view is matched to 3D prototypes of known objects. The other possibility is that some mechanism allows the determination of the orientation of the yet-to-be identified object under view. This orientation information is used to rotate and project potential 3D descriptions in a way that allows a 2D matching of the description and the viewed object. Finally, the last option for describing the properties of object classes involves *invariant features* which describe classes of objects in terms of more generic geometric properties, particularly those that are likely to be insensitive to different views of the object.

20.4.2 Size and Distance

In the absence of more definitive information about depth, objects which project onto a larger area of the retina are seen as closer compared with objects which project to a smaller retinal area, an effect called *relative size*. A more powerful cue involves *familiar size*, which can provide information for absolute distance

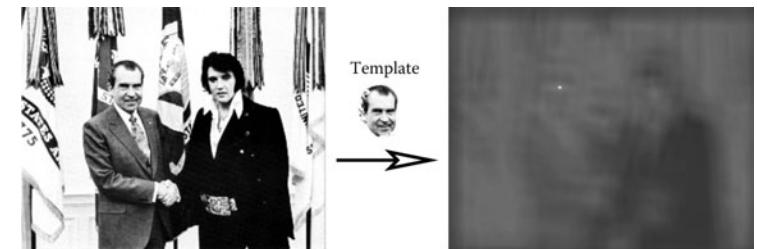


图20.37。模板匹配。右图像中的亮点指示与左图像中的模板的最佳匹配位置。图片由国家档案和记录管理局提供。

足以将每个班级与所有其他班级区分开来。物体识别理论在描述每个类的信息的性质以及用于将这些描述与世界的实际观点相匹配的机制方面有所不同。

三种一般类型的描述是可能的。模板根据每个类中对象的原型视图来表示对象类。图20.37显示了一个简单的例子。结构描述以对象视图中可能容易检测到的每个类的模糊特征以及有关特征之间几何关系的信息来表示对象类。结构描述可以用2D或3D来表示。对于对象类型的2D模型，对于每个明显不同的对象潜在视图，必须有一个单独的描述。对于3D模型，两种不同形式的匹配策略是可能的。在一个方面，在使用任何可用的空间线索进行分类之前确定所查看的object的三维结构，然后将视图的此3d描述匹配到已知对象的3d原型。另一种可能性是，一些机制允许确定在视图下的尚未被识别的对象的取向。该方向信息用于以允许描述和所查看对象的2D匹配的方式旋转和投影潜在的3d描述。最后，用于描述对象类的属性的最后一个选项涉及不变特征，这些特征用更通用的几何属性来描述对象类，特别是那些可能对对象的不同视图不敏感的

20.4.2 大小和距离

在缺乏有关深度的更明确信息的情况下，投射到视网膜较大区域的物体与投射到较小视网膜区域的物体相比被视为更接近，这种效应称为相对尺寸。一个更强大的提示涉及熟悉的大小，它可以提供绝对距离的信息



Figure 20.38. Left: perspective and familiar size cues are consistent. Right: perspective and familiar size cues are inconsistent. *Images courtesy Peter Shirley, Scott Kuhl, and J. Dylan Lacewell.*

to recognizable objects of known size. The strength of familiar size as a depth cue can be seen in illusions such as Figure 20.38, in which it is put in conflict with ground-plane, perspective-based depth cues. Familiar size is one part of the *size-distance* relationship, relating the physical size of an object, the optical size of the same object projected onto the retina, and the distance of the object from the eye (Figure 20.39).

When objects are sitting on top of a flat-ground plane, additional sources for depth information become available, particularly when the horizon is either visible or can be derived from other perspective information. The angle of declination to the contact point on the ground is a relative depth cue and provides absolute egocentric distance when scaled by eye height, as previously shown in

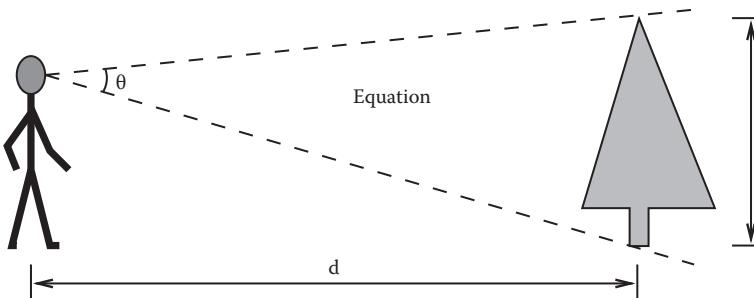


Figure 20.39. The *size-distance* relationship allows the distance to objects of known size to be determined based on the visual angle subtended by the object. Likewise, the size of an object at a known distance can be determined based on the visual angle subtended by the object.



左：透视和熟悉的尺寸线索是一致的。右：透视和熟悉的尺寸线索不一致。图片由PeterShirley, ScottKuhl和J.DylanLacewell提供。

到已知尺寸的可识别物体。熟悉的尺寸作为深度线索的强度可以在诸如图20.38之类的幻想中看到，其中它与地面平面，基于透视的深度线索相冲突。熟悉的尺寸是尺寸-距离关系的一部分，涉及物体的物理尺寸，投射到视网膜上的相同物体的光学尺寸以及物体距离眼睛的距离（图20.39）。

当物体位于平坦地面上时，深度信息的额外来源就会变得可用，特别是当地平线是可见的或可以从其他透视信息中获得时。Declination到地面接触点的角度是一个相对深度提示，并在按眼睛高度缩放时提供绝对的自我中心距离，如前所示

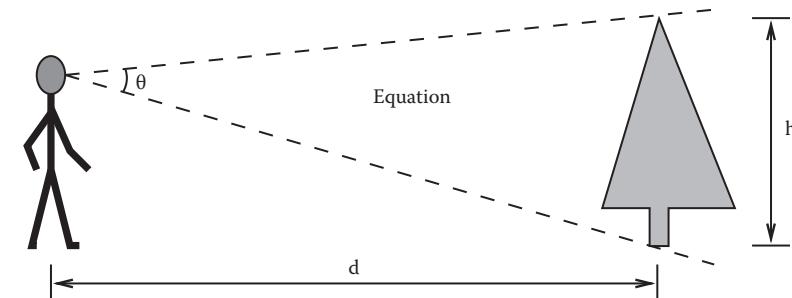


图20.39。尺寸-距离关系允许基于对象子化的视角来确定到已知尺寸的对象的距离。同样地，在已知距离处的物体的尺寸可以基于该物体子化的可视角度来确定。

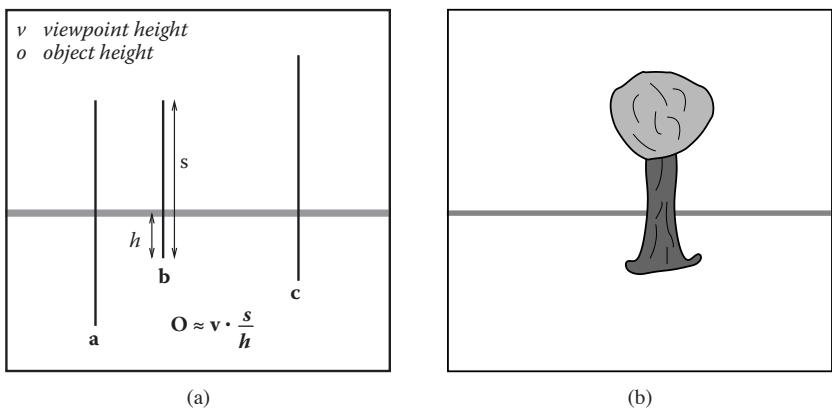


Figure 20.40. (a) The *horizon ratio* can be used to determine depth by comparing the visible portion of an object below the horizon to the total vertical visible extent of the object. (b) A real-world example.

Figure 20.27. The *horizon ratio*, in which the total visible height of an object is compared with the visible extent of that portion of the object appearing below the horizon, can be used to determine the actual size of objects, even when the distance to the objects is not known (Figure 20.40). Underlying the horizon ratio is the fact that for a flat-ground plane, the line of sight to the horizon intersects objects at a position that is exactly an eye height above the ground.

The human visual system is sufficiently able to determine the absolute size of most viewed objects; our perception of size is dominated by the the actual physi-

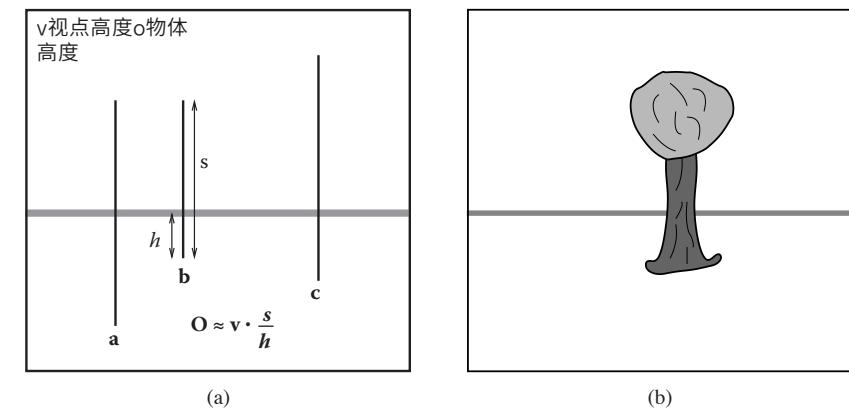


图20.40。(a) 视界比可用于通过将低于视界的物体的可见部分与物体的总垂直可见范围进行比较来确定深度。(b)一个现实世界的例子。

图20.27。视界比（视界比）可以用来确定物体的实际大小，即使不知道到物体的距离（图20.40）。视界比（视界比）是将物体的总可见高度与低于视界的那部分物体的可见程度进行比较的。视界比的基础是一个事实，即对于一个平面-地平面，视界线与物体相交的位置正好是一个眼睛的高度在地面上。

人类的视觉系统足以确定大多数被观察物体的绝对尺寸；我们对尺寸的感知由实际的植物支配—



Figure 20.41. (a) Size constancy makes hands positioned at different distances from the eye appear to be nearly the same size for real-world viewing, even though the retinal sizes are quite different. (b) The effect is less strong when one hand is partially occluded by the other, particularly when one eye is closed. *Images courtesy Peter Shirley and Pat Moulis.*



图20.41。(a) 大小恒定性使得位于与眼睛不同距离的手看起来几乎相同的大小，即使视网膜大小完全不同。(b) 当一只手被另一只手部分遮挡时，效果不那么强烈，特别是当一只眼睛闭合时。图片由彼得·雪莉和帕特·穆利斯提供。



cal size, and we have almost no conscious awareness of the corresponding retinal size of objects. This is similar to lightness constancy, discussed earlier, in that our perception is dominated by inferred properties of the world, not the low level features actually sensed by photoreceptors in the retina. Gregory (1997) describes a simple example of *size constancy*. Hold your two hands out in front of you, one at arm's length and the other at half that distance away from you (Figure 20.41(a)). Your two hands will look almost the same size, even though the retinal sizes differ by a factor of two. The effect is much less strong if the nearer hand partially occludes the more distant hand, particularly if you close one eye (Figure 20.41(b)). The visual system also exhibits *shape constancy*, where the perception of geometric structure is close to actual object geometry than might be expected given the distortions of the retinal image due to perspective (Figure 20.42).

20.4.3 Events

Most aspects of event perception are beyond the scope of this chapter, since they involve complex nonvisual cognitive processes. Three types of event perception are primarily visual, however, and are also of clear relevance to computer graphics. Vision is capable of providing information about how a person is moving in the world, the existence of independently moving objects in the world, and the potential for collisions either due to observer motion or due to objects moving toward the observer.

Vision can be used to determine rotation and the direction of translation relative to the environment. The simplest case involves movement toward a flat surface oriented perpendicularly to the line of sight. Presuming that there is sufficient surface texture to enable the recovery of optic flow, the flow field will form a symmetric pattern as shown in Figure 20.43(a). The location in the field of view of the *focus of expansion* of the flow field will have an associated line of sight corresponding to the direction of translation. While optic flow can be used to visually determine the direction of motion, it does not contain enough information to determine speed. To see this, consider the situation in which the world is made twice as large and the viewer moves twice as fast. The decrease in the magnitude of flow values due to the doubling of distances is exactly compensated for by the increase in the magnitude of flow values due to the doubling of velocity, resulting in an identical flow field.

Figure 20.43(b) shows the optic flow field resulting from the viewer (or more accurately, the viewer's eyes) rotating around the vertical axis. Unlike the situation with respect to translational motion, optic flow provides sufficient information to determine both the axis of rotation and the (angular) speed of rotation. The



Figure 20.42. Shape constancy—the table looks rectangular even though its shape in the image is an irregular four-sided polygon.

cal尺寸，并且我们对物体的相应视网膜尺寸几乎没有有意识的认识。这类似于前面讨论过的亮度恒定性，因为我们的感知是由世界的推断属性主导的，而不是视网膜中光感受器实际感知到的低级特征。Gregory(1997)描述了尺寸恒定性的一个简单示例。将你的两只手放在你面前，一只在手臂的长度，另一只在距离你一半的距离（图20.41 (a)）。你的两只手看起来几乎相同的大小，即使视网膜大小相差两倍。如果更近的手部分地使用，效果就不那么强烈了-

抓住更远的手，特别是如果你闭上一只眼睛（图20.41 (b)）。视觉系统还表现出形状恒定性，其中几何结构的感知接近实际物体几何形状，而不是由于透视引起的视网膜图像扭曲而预期的（图20.42）。



Figure 20.42. Shape表格看起来是矩形的，即使它在图像中的形状是一个不规则的四边多边形。

20.4.3 Events

事件感知的大多数方面都超出了本章的范围，因为它们涉及复杂的非视觉认知过程。然而，三种类型的事件感知主要是视觉的，并且与计算机图形集成电路也有明显的相关性。视觉能够提供关于一个人如何在世界上移动的信息，世界上独立移动的物体的存在，以及由于观察者运动或由于物体向观察者移动而发生碰撞的可能性。

视觉可用于确定旋转和平移方向相对于环境。最简单的情况是朝向垂直于视线的平坦表面移动。假设有足够的表面纹理使光流恢复，流场将形成对称模式，如图20.43 (a) 所示。流场的扩展焦点的视场中的位置将具有对应于平移方向的相关联的视线。虽然光流可以用来确定运动方向，但它不包含足够的信息来确定速度。为了看到这一点，请考虑这样的情况：世界是两倍大，观众的移动速度是两倍。由于距离增加一倍而导致的流量值幅度的减小正好通过由于速度增加一倍而导致的流量值幅度的增加来补偿，从而产生相同的流场。

图20.43 (b) 显示了观看者（或更准确地说，观看者的眼睛）绕垂直轴旋转产生的光学流场。与平移运动的情况不同，光流提供了足够的信息来确定旋转轴和（角）旋转速度。该

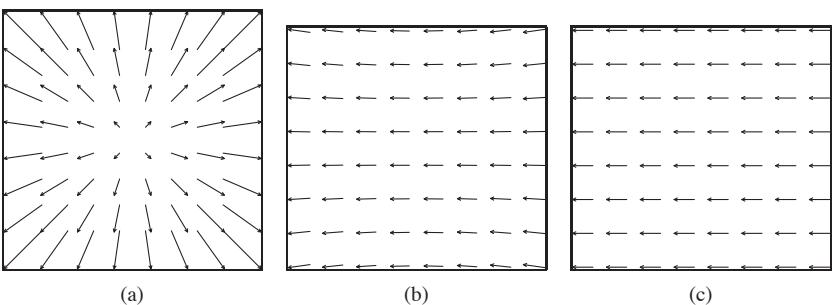
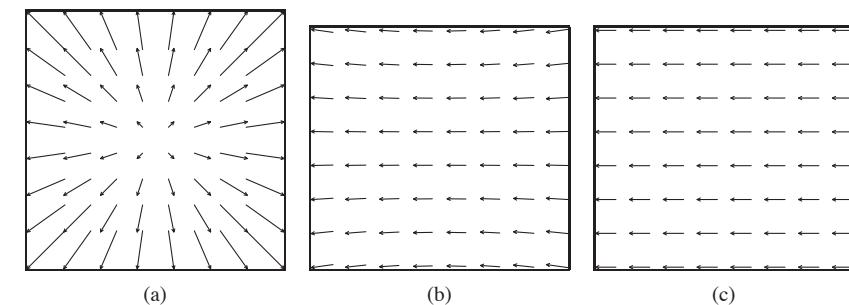


Figure 20.43. (a) Movement toward a flat, textured surface produces an expanding flow field, with the *focus of expansion* indicating the line of sight corresponding to the direction of motion. (b) The flow field resulting from rotation around the vertical axis while viewing a flat surface oriented perpendicularly to the line of sight. (c) The flow field resulting from translation parallel to a flat, textured surface.

practical problem in exploiting this is that the flow resulting from pure rotational motion around an axis perpendicular to the line of sight is quite similar to the flow resulting from pure translation in the direction that is perpendicular to both the line of sight and this rotational axis, making it difficult to visually discriminate between the two very different types of motion (Figure 20.43(c)). Figure 20.44 shows the optical flow patterns generated by movement through a more realistic environment.

If a viewer is completely stationary, visual detection of moving objects is easy, since such objects will be associated with the only nonzero optic flow in the field of view. The situation is considerably more complicated when the observer is moving, since the visual field will be dominated by nonzero flow, most or all of



(a) 朝向平坦的，纹理化的表面的运动产生膨胀的流场，膨胀的焦点指示与运动方向相对应的视线。(b) 观察垂直于视线的平面时，绕垂直轴旋转所产生的流场。(c) 平行于平坦的纹理表面平移产生的流场。

利用这一点的实际问题是，围绕垂直于视线的轴的纯旋转运动所产生的流动与垂直于视线和这一旋转轴的方向的纯平移所产生的流动非常相似，因此很难在视觉上区分这两种截然不同的运动（图20.43 (c)）。图20.44显示了通过更真实的环境运动产生的光流模式。

如果观看者是完全静止的，则移动物体的视觉检测是容易的，因为这样的物体将与视场中唯一的非零光流相关联。当观察者移动时，情况要复杂得多，因为视野将被非零流所支配，大部分或全部

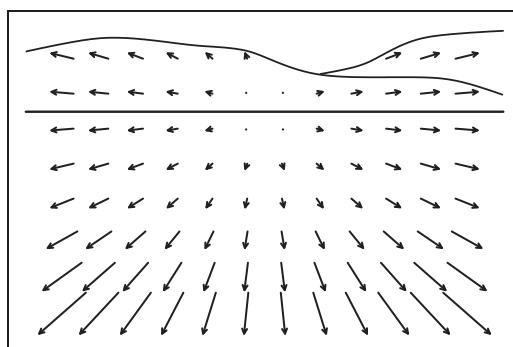
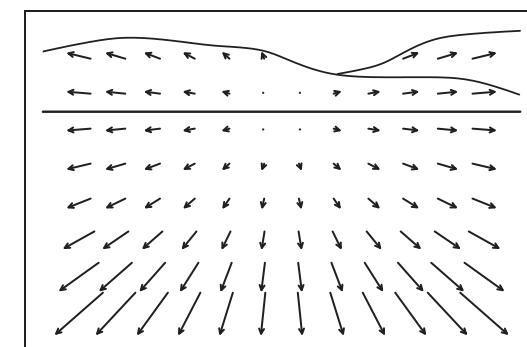


Figure 20.44. The optic flow generated by moving through an otherwise static environment provides information about both the motion relative to the environment and the distances to points in the environment. In this case, the direction of view is depressed from the horizon, but as indicated by the focus of expansion, the motion is parallel to the ground plane.



通过在静态环境中移动而产生的光流提供了关于相对于环境的运动和到环境中点的距离的信息。在这种情况下，视图方向从地平线上被压低，但如扩展焦点所示，运动平行于地平面。

which is due to relative motion between the observer and the static environment (Thompson & Pong, 1990). In such cases, the visual system must be sensitive to patterns in the optic flow field that are inconsistent with flow fields associated with observer movement relative to a static environment (Figure 20.45).

Section 20.3.4 described how vision can be used to determine time to contact with a point in the environment even when the speed of motion is not known. Assuming a viewer moving with a straight, constant-speed trajectory and no independently moving objects in the world, contact will be made with whatever surface is in the direction of the line of sight corresponding to the focus of expansion at a time indicated by the τ relationship. An independently moving object complicates the matter of determining if a collision will in fact occur. Sailors use a method for detecting potential collisions that may also be employed in the human visual system: for non-accelerating straight-line motion, collisions will occur with objects that are visually expanding but otherwise remain visually stationary in the egocentric frame of reference.

One form of more complex event perception merits discussion here, since it is so important in interactive computer graphics. People are particularly sensitive to motion corresponding to human movement. Locomotion can be recognized when the only features visible are lights on the walker's joints (Johansson, 1973). Such *moving light displays* are often even sufficient to recognize properties such as the sex of the walker and the weight of the load that the walker may be carrying. In computer graphics renderings, viewers will notice even small inaccuracies in animated characters, particularly if they are intended to mimic human motion.

The term *visual attention* covers a range of phenomenon from where we point our eyes to cognitive effects involving what we notice in a complex scene and how we interpret what we notice (Pashler, 1998). Figure 20.46 provides an example of how attentional processes affect vision, even for very simple images. In the left

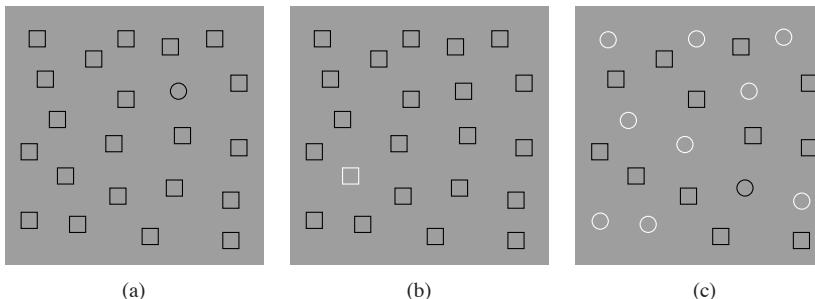


Figure 20.46. In (a) and (b), visual attention is quickly drawn to the item of different shape or color. In (c), sequential search appears to be necessary in order to find the one item that differs in both shape and color.

这是由于观察者和静态环境之间的相对运动 (Thompson & Pong, 1990)。在这种情况下，视觉系统必须对光学流场中的模式敏感，这些模式与观察者相对于静态环境的运动相关的流场不一致（图20.45）。

第20.3.4节描述了如何使用视觉来确定与环境中某个点接触的时间，即使不知道运动速度。假设一个观者以直线、恒定速度的轨迹移动，而没有在世界上可靠移动的物体中移动，那么任何东西都可以接触到

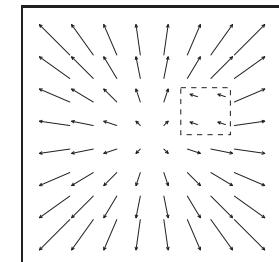
面是在 τ 关系所指示的时刻与所述sion的焦点相对应的视线方向上。一个独立运动的物体使确定碰撞是否会发生的问题变得复杂。水手们使用一种检测潜在碰撞的方法，这种方法也可以在人视觉系统中使用：对于非加速的直线运动，碰撞会在视觉上扩张但在以自我为中心的参照系中保持视觉静止的物体上。

更复杂的事件感知的一种形式值得在这里讨论，因为它在交互式计算机图形学中如此重要。人们对与人的运动相对应的运动特别敏感。当唯一可见的特征是步行者关节上的灯光时，可以识别运动 (Johansson, 1973)。这样的移动光显示器常常甚至足以识别诸如步行者的性别和步行者可能携带的负载的重量之类的属性。在计算机图形渲染中，观众甚至会注意到动画角色中的微小不准确之处，特别是如果它们旨在模仿人体运动。

视觉注意这个术语涵盖了一系列现象，从我们将眼睛指向认知效果，涉及我们在复杂场景中注意到的内容以及我们如何解释我们注意到的内容 (Pashler, 1998)。图20.46提供了一个注意力过程如何影响视力的例子，即使对于非常简单的图像也是如此。在左边



图20.46。在 (a) 和 (b) 中，视觉注意力被迅速吸引到不同形状或颜色的物品上。在 (c) 中，顺序搜索似乎是必要的，以便找到在形状和颜色上不同的一个项目。



从移动观察点对移动物体进行视觉检测需要识别光学流中的模式，这些模式不能通过静态环境与运动相关联。

two panels, the one pattern differing in shape or color from the rest immediately “pops out” and is easily noticed. In the panel on the right, the one pattern differing in both shape and color is harder to find. The reason for this is that the visual system can do a parallel search for items distinguished by individual properties, but requires more cognitive, sequential search when looking for items that are indicated by the simultaneous presence of two distinguishing features. Graphically based human-computer interfaces should be (but often are not!) designed with an understanding of how to take advantage of visual attention processes in people so as to communicate important information quickly and effectively.

20.5 Picture Perception

So far, this chapter has dealt with the visual perception that occurs when the world is directly imaged by the human eye. When we view the results of computer graphics, of course, we are looking at rendered images and not the real world. This has important perceptual implications. In principle, it should be possible to generate computer graphics that appear indistinguishable from the real world, at least for monocular viewing without either object or observer motion. Imagine looking out at the world through a glass window. Now, consider coloring each point on the window to exactly match the color of the world originally seen at that point.¹⁰ The light reaching the eye is unchanged by this operation, meaning that perception should be the same whether the painted glass is viewed or the real world is viewed through the window. The goal of computer graphics can be thought of as producing the colored window without actually having the equivalent real-world view available.

The problem for computer graphics and other visual arts is that we can't in practice match a view of the real world by coloring a flat surface. The brightness and dynamic range of light in the real world is impossible to re-create using any current display technology. Resolution of rendered images is also often less than the finest detail perceivable by human vision. Lightness and color constancy are much less apparent in pictures than in the real world, likely because the visual system attempts to compensate for variability in the brightness and color of the illumination based on the ambient illumination in the viewing environment, rather than the illumination associated with the rendered image. This is why the realistic appearance of color in photographs depends on film color balanced for the nature of the light source present when the photograph was taken and why

¹⁰This idea was first described by the painter Leon Battista Alberti in 1435 and is now known as *Alberti's Window*. It is closely related to the *camera obscura*.

两个面板，一个图案的形状或颜色不同的其余立即“弹出”，很容易被注意到。在右边的面板中，一个在形状和颜色上不同的图案很难找到。这样做的原因是视觉系统可以对由单个属性区分的项目进行并行搜索，但是在寻找由同时存在两个区分特征指示的项目时需要更多的认知，顺序搜索。基于图形的人机接口应该是（但往往不是！）设计时了解如何利用人的视觉注意力过程，以便迅速有效地传达重要信息。

20.5 图片感知

到目前为止，本章已经处理了当世界由人眼直接成像时发生的视觉感知。当然，当我们查看计算机图形的结果时，我们看的是渲染的图像，而不是真实的世界。这具有重要的感知意义。原则上，应该可以生成看起来与现实世界没有区别的计算机图形，至少对于没有物体或观察者运动的单眼观看。想象一下，透过玻璃窗眺望世界。现在，考虑着色窗口上的每个点，以完全匹配最初在该点看到的世界的颜色。通过此操作到达眼睛的光线不变，这意味着无论是观看彩绘玻璃还是通过窗户观看真实世界，感知都应该是相同的。计算机图形学的目标可以被认为是生产彩色窗口，而实际上没有equiva借给现实世界的视图可用。

计算机图形学和其他视觉艺术的问题是，我们在实践中无法通过着色平坦的表面来匹配真实世界的视图。现实世界中光的亮度和动态范围是不可能使用任何当前的显示技术重新创建的。渲染图像的分辨率通常不如人类视觉所能感知到的最精细细节。亮度和颜色恒定性在图片中比在现实世界中明显得多，这可能是因为visual系统试图根据观看环境中的环境照明来补偿照明亮度和颜色的变化，而不是与渲染图像相关的照明。这就是为什么照片中颜色的真实外观取决于胶片颜色的平衡，因为照片拍摄时存在的光源的性质以及原因

¹⁰这个想法首先由画家LeonBattistaAlberti于1435年描述，现在被称为Alberti的窗户。它与暗箱密切相关。

realistic color in video requires a white-balancing step. While much is known about how limitations in resolution, brightness, and dynamic range affect the detectability of simple patterns, almost nothing is known about how these display properties affect spatial vision or object identification.

We have a better understanding of other aspects of this problem, which psychologists refer to as the perception of *pictorial space* (S. Rogers, 1995). One difference between viewing images and viewing the real world is that accommodation, binocular stereo, motion parallax, and perhaps other depth cues may indicate that the surface under view is much different from the distances in the world that it is intended to represent. The depths that are seen in such a situation tend to be somewhere between the depths indicated by the pictorial cues in the image and the distance to the image itself. When looking at a photograph or computer display, this often results in a sense of scale smaller than intended. On the other hand, seeing a movie in a big-screen theater produces a more compelling sense of spaciousness than does seeing the same movie on television, even if the distance to the TV is such that the visual angles are the same, since the movie screen is farther away.

Computer graphics rendered using perspective projection has a viewpoint, specified as a position and direction in model space, and a view frustum, which specifies the horizontal and vertical field of view and several other aspects of the viewing transform. If the rendered image is not viewed from the correct location, the visual angles to the borders of the image will not match the frustum used in creating the image. All visual angles within the image will be distorted as well, causing a distortion in all of the pictorial depth and orientation cues based on linear perspective. This effect occurs frequently in practice, when a viewer is positioned either too close or too far away from a photograph or display surface. If the viewer is too close, the perspective cues for depth will be compressed, and the cues for surface slant will indicate that the surface is closer to perpendicular to the line of sight than is actually the case. The situation is reversed if the viewer is too far from the photograph or screen. The situation is even more complicated if the line of sight does not go through the center of the viewing area, as is commonly the case in a wide variety of viewing situations.

The human visual system is able to partially compensate for perspective distortions arising from viewing an image at the wrong location, which is why we are able to sit in different seats at a movie theater and experience a similar sense of the depicted space. When controlling viewing position is particularly important, *viewing tubes* can be used. These are appropriately sized tubes, mounted in a fixed position relative to the display, and through which the viewer sees the display. The viewing tube constrains the observation point to the (hopefully) cor-

视频中的逼真色彩需要一个白平衡步骤。虽然人们对分辨率、亮度和动态范围的限制如何影响简单图案的可构造性知之甚少，但对这些显示属性如何影响空间视觉或物体识别几乎一无所知。

我们对这个问题的其他方面有了更好的理解，psychologists将其称为图像空间的感知 (S.Rogers, 1995)。观看图像和观看真实世界之间的一个区别是，图像调制，双目立体，运动视差和也许其他深度线索可能表明所看到的表面与它旨在表示的世界中的距离在这种情况下看到的深度往往介于图像中图形线索指示的深度和到图像本身的距离之间。当看一张照片或computer显示器时，这通常会导致比预期的小规模感。另一方面，在大屏幕影院看电影会产生比在电视上看同一部电影更引人注目的空间感，即使与电视的距离相同，视角相同，因为电影屏幕更远。

使用透视投影渲染的计算机图形具有视点，指定为模型空间中的位置和方向，以及视锥，其指定水平和垂直视场以及观看变换的几个其他方面。如果没有从正确的位置查看渲染的图像，则与图像边界的可视角度将不匹配创建图像时使用的截头。图像中的所有视觉角度也将被扭曲，导致所有基于线性透视的图像深度和方向线索的扭曲。这种效应在实践中经常发生，当观看者被定位在离照片或显示表面太近或太远的地方时。如果观看者太近，则深度的透视线索将被压缩，而表面倾斜的线索将指示表面比实际情况更接近垂直于视线。如果观看者离照片或屏幕太远，情况就会相反。如果视线没有穿过观看区域的中心，情况就更加复杂，就像在各种各样的观看情况下通常的情况一样。

人类视觉系统能够部分补偿由于在错误的位置观看图像而产生的透视偏差，这就是为什么我们能够坐在电影院的不同座位上，体验所描绘空间的类似感。当控制观察位置特别重要时，可以使用观察管。这些是适当大小的管，安装在相对于显示器的固定位置，并通过该管观看者看到显示器。观察管将观察点约束到（希望）cor



558

20. Visual Perception

rect position. Viewing tubes are also quite effective at reducing the conflict in depth information between the pictorial cues in the image and the actual display surface. They eliminate both stereo and motion parallax, which, if present, would correspond to the display surface, not the rendered view. If they are small enough in diameter, they also reduce other cues to the location of the display surface by hiding the picture frame or edge of the display device. Exotic visually immersive display devices such as head-mounted displays (HMDs) go further in attempting to hide visual cues to the position of the display surface while adding binocular stereo and motion parallax consistent with the geometry of the world being rendered.



558

20. 视觉感知

rect位置。观察管在减少图像中的图案线索和实际显示表面之间的深度信息冲突方面也相当有效。它们消除了立体视差和运动视差，如果存在，则对应于显示表面，而不是渲染视图。如果它们的直径足够小，它们还通过隐藏显示设备的相框或边缘来减少对显示表面位置的其他提示。诸如头戴式显示器(Hmd)之类的奇异视觉沉浸式显示设备在尝试将视觉线索隐藏到显示表面的位置方面更进一步，同时添加与正在渲染的世界的几何相



Tone Reproduction

As discussed in Chapter 20, the human visual system adapts to a wide range of viewing conditions. Under normal viewing, we may discern a range of around 4 to 5 log units of illumination, i.e., the ratio between brightest and darkest areas where we can see detail may be as large as 100,000 : 1. Through adaptation processes, we may adapt to an even larger range of illumination. We call images that are matched to the capabilities of the human visual system *high dynamic range*.

Visual simulations routinely produce images with a high dynamic range (Ward Larson & Shakespeare, 1998). Recent developments in image-capturing techniques allow multiple exposures to be aligned and recombined into a single high dynamic range image (Debevec & Malik, 1997). Multiple exposure techniques are also available for video. In addition, we expect future hardware to be able to photograph or film high dynamic range scenes directly. In general, we may think of each pixel as a triplet of three floating point numbers.

As it is becoming easier to create high dynamic range imagery, the need to display such data is rapidly increasing. Unfortunately, most current display devices, monitors and printers, are only capable of displaying around 2 log units of dynamic range. We consider such devices to be of low dynamic range. Most images in existence today are represented with a byte-per-pixel-per-color channel, which is matched to current display devices, rather than to the scenes they represent.

Typically, low dynamic range images are not able to represent scenes without loss of information. A common example is an indoor room with an out-



音调再现

如第20章所讨论的，人类视觉系统适应广泛的观看条件。在正常观察下，我们可以分辨出大约4到5对数单位的照明范围，即我们可以看到细节的最亮和最暗区域之间的比例可能高达100 000 : 1。通过适应过程，我们可以适应更大范围的照明。我们将与人类视觉系统能力相匹配的图像称为高动态范围。

视觉模拟通常会产生具有高动态范围的图像 (WardLarson & Shakespeare, 1998)。图像捕获技术的最新发展允许将多个曝光对齐并重新组合成单个高动态范围图像 (Debevec & Malik, 1997)。多曝光技术也可用于视频。此外，我们期望未来的硬件能够直接拍摄或拍摄高动态范围的场景。一般来说，我们可能会将每个像素视为三个浮点数的三元组。

随着创建高动态范围图像变得越来越容易，显示此类数据的需求正在迅速增加。不幸的是，目前大多数显示器、显示器和打印机只能显示大约2个日志单位的动态范围。我们认为此类器件的动态范围较低。目前存在的大多数图像都是用每像素每颜色的字节表示的，它与当前的显示设备相匹配，而不是与它们所代表的场景相匹配。

通常，低动态范围图像不能够表示信息丢失的场景。一个常见的例子是一个带室外的室内房间



Figure 21.1. With conventional photography, some parts of the scene may be under- or overexposed. To visualize the snooker table, the view through the window is burned out in the left image. On the other hand, the snooker table will be too dark if the outdoor part of this scene is properly exposed. Compare with Figure 21.2, which shows a high dynamic range image prepared for display using a tone reproduction algorithm.

door area visible through the window. Humans are easily able to see details of both the indoor part and the outside part. A conventional photograph typically does not capture this full range of information—the photographer has to choose whether the indoor or the outdoor part of the scene is properly exposed (see Figure 21.1). These decisions may be avoided by using high dynamic range imaging and preparing these images for display using techniques described in this chapter (see Figure 21.2).

There are two strategies available to display high dynamic range images. First, we may develop display devices which can directly accommodate a high dynamic range (Seetzen, Whitehead, & Ward, 2003; Seetzen et al., 2004). Second, we may prepare high dynamic range images for display on low dynamic range display devices (Upstill, 1985). This is currently the more common approach and the topic of this chapter. Although we foresee that high dynamic range display devices will become widely used in the (near) future, the need to compress the dynamic range of an image may diminish, but will not disappear. In particular, printed media such as this book are, by their very nature, low dynamic range.

Compressing the range of values of an image for the purpose of display on a low dynamic range display device is called tonemapping or tone reproduction.



Figure 21.2. A high dynamic range image tonemapped for display using a recent tone reproduction operator (Reinhard & Devlin, 2005). In this image, both the indoor part and the view through the window are properly exposed.



使用传统摄影，场景的某些部分可能会曝光不足。为了可视化斯诺克桌，通过窗口的视图在左图中被烧掉。另一方面，如果这个场景的室外部分适当暴露，斯诺克桌就会太暗。与图21.2比较，其示出了使用色调再现算法准备用于显示的高动态范围图像。

通过窗户可见的门区域。人类很容易看到室内部分和室外部分的细节。传统的照片通常不能捕捉到这种全方位的信息—摄影师必须选择场景的室内或室外部分是否正确曝光（见图21.1）。这些决定可以通过使用高动态范围成像并使用本章中描述的技术准备这些图像以供显示来避免（见图21.2）。

有两种策略可用于显示高动态范围图像。第一个一个高动态范围的图像色调显示使用最近的色调再生产操作员(Reinhard&Devlin 2005)。在该图像中，室内部分和透过窗户的视图两者都被适当地曝光。我们可能会开发出可以直接适应高动态范围的显示设备 (Seetzen, Whitehead, & Ward, 2003; Seetzen et al. 2004)。Second，我们可能会准备高动态范围图像，以便在低动态范围显示设备上显示 (Upstill, 1985)。这是目前比较常见的方法，也是本章的主题。尽管我们预见到高动态范围显示设备将在（近）将来得到广泛应用，但压缩图像动态范围的需要可能会减少，但不会消失。特别是，像这本书这样的印刷媒体，就其本质而言，动态范围很低。

为了在低动态范围显示设备上显示而压缩图像的值范围被称为色调映射或色调再现。



Figure 21.3. Linear scaling of high dynamic range images to fit a given display device may cause significant detail to be lost (left and middle). The left image is linearly scaled. In the middle image high values are clamped. For comparison, the right image is tonemapped, allowing details in both bright and dark regions to be visible.

A simple compression function would be to normalize an image (see Figure 21.3 (left)). This constitutes a linear scaling which tends to be sufficient only if the dynamic range of the image is only marginally higher than the dynamic range of the display device. For images with a higher dynamic range, small intensity differences will be quantized to the same display value such that visible details are lost. In Figure 21.3 (middle) all pixel values larger than a user-specified maximum are set to this maximum (i.e., they are clamped). This makes the normalization less dependent on noisy outliers, but here we lose information in the bright areas of the image. For comparison, Figure 21.3 (right) is a tonemapped version showing detail in both the dark and the bright regions.

In general, linear scaling will not be appropriate for tone reproduction. The key issue in tone reproduction is then to compress an image while at the same time preserving one or more attributes of the image. Different tone reproduction algorithms focus on different attributes such as contrast, visible detail, brightness, or appearance.

Ideally, displaying a tonemapped image on a low dynamic range display device would create the same visual response in the observer as the original scene. Given the limitations of display devices, this will not be achievable, although we could aim for approximating this goal as closely as possible.

As an example, we created the high dynamic range image shown in Figure 21.4. This image was then tonemapped and displayed on a display device. The display device itself was then placed in the scene such that it displays its own background (Figure 21.5). In the ideal case, the display should appear transpar-



高动态范围图像的线性缩放以适合给定的显示设备可能会导致显着细节丢失（左侧和中间）。左图像进行线性缩放。在中间图像中，高值被夹紧。为了进行比较，正确的图像是色调，允许在明亮和黑暗区域的细节是可见的。

一个简单的压缩函数是对图像进行归一化（见图21.3（左））。这构成了仅当图像的动态范围仅略微高于显示装置的动态范围时趋于足够的线性缩放。对于具有较高动态范围的图像，小的强度差异将被量化为相同的显示值，从而丢失可见的细节。在图21.3（中间）中，所有大于用户指定最大值的像素值都被设置为该最大值（即，它们被钳位）。这使得归一化对噪声异常值的依赖性较小，但在这里我们会丢失图像明亮区域的信息。为了比较，图21.3（右）是一个色调的版本，显示了黑暗和明亮区域的细节。



Figure 21.4. Image used for demonstrating the goal of tone reproduction in Figure 21.5.

一般而言，线性缩放将不适用于音调再现。色调再现的关键问题是压缩图像，同时保留图像的一个或多个属性。不同的色调再现算法侧重于不同的属性，例如对比度、可见细节、亮度或外观。



图21.4。图21.5中用于演示色调再现目标的图像。

理想情况下，在低动态范围显示器device上显示色调映射图像将在观察者中创建与原始场景相同的视觉响应。鉴于显示设备的局限性，这是不可能实现的，尽管我们可以尽可能接近这个目标。

作为一个例子，我们创建了图21.4所示的高动态范围图像。然后对此图像进行调色并显示在显示设备上。然后将显示设备本身放置在场景中，使其显示自己的背景（图21.5）。在理想的情况下，显示器应该出现。



Figure 21.5. After tonemapping the image in Figure 21.4 and displaying it on a monitor, the monitor is placed in the scene approximately at the location where the image was taken. Dependent on the quality of the tone reproduction operator, the result should appear as if the monitor is transparent.

ent. Dependent on the quality of the tone reproduction operator, as well as the nature of the scene being depicted, this goal may be more or less achievable.

21.1 Classification

Although it would be possible to classify tone reproduction operators by which attribute they aim to preserve, or for which task they were developed, we classify algorithms according to their general technique. This will enable us to show the differences and similarities between a significant number of different operators, and so, hopefully, contribute to the meaningful selection of specific operators for given tone reproduction tasks.

The main classification scheme we follow hinges upon the realization that tone reproduction operators are based on insights gained from various disciplines. In particular, several operators are based on knowledge of human visual perception.

The human visual system detects light using photoreceptors located in the retina. Light is converted to an electrical signal which is partially processed in the retina and then transmitted to the brain. Except for the first few layers of cells in the retina, the signal derived from detected light is transmitted using impulse trains. The information-carrying quantity is the frequency with which these electrical pulses occur.

The range of light that the human visual system can detect is much larger than the range of frequencies employed by the human brain to transmit information. Thus, the human visual system effortlessly solves the tone reproduction problem—a large range of luminances is transformed into a small range of frequencies of impulse trains. Emulating relevant aspects of the human visual system is therefore a worthwhile approach to tone reproduction; this approach is explained in more detail in Section 21.7.



在对图21.4中的图像进行调色并将其显示在监视器上之后，监视器被放置在场景中，大约在拍摄图像的位置。取决于色调再现操作员的质量，结果应该看起来好像监视器是透明的。

耳鼻喉科依赖于音调再现算子的质量，以及被描绘的场景的性质，这个目标可能或多或少是可实现的。

21.1 Classification

虽然可以根据它们的目标保留哪个属性或它们被开发的任务来对音调再现操作符进行分类，但我们根据它们的一般技术对算法进行分类。这将使我们能够显示显着数量的不同运营商之间的差异和相似之处，因此，希望有助于为给定的音调再现任务有意义地选择特定运营商。

我们遵循的主要分类方案取决于音调再现运算符基于从各个学科获得的认识。特别是，几个操作员基于人类视觉感知的知识。人类视觉系统使用位于视网膜中的光感受器检测光。光被转换为电信号，该电信号在视网膜中被部分处理，然后传输到大脑。除了视网膜中的前几层细胞外，来自检测到的光的信号使用im脉冲列传输。信息承载量是这些电脉冲发生的频率。

人类视觉系统可以检测到的光的范围比人类大脑用来传输信息的频率范围大得多。因此，人类视觉系统毫不费力地解决了音调再现问题—大范围的亮度被转换成脉冲列车的小范围频率。因此，模拟人类视觉系统的相关方面是一种有价值的色调再现方法；这种方法在第21.7节中有更详细的解释。

A second class of operators is grounded in physics. Light interacts with surfaces and volumes before being absorbed by the photoreceptors. In computer graphics, light interaction is generally modeled by the rendering equation. For purely diffuse surfaces, this equation may be simplified to the product between light incident upon a surface (illuminance), and this surface's ability to reflect light (reflectance) (Oppenheim, Schafer, & Stockham, 1968).

Since reflectance is a passive property of surfaces, for diffuse surfaces it is, by definition, low dynamic range—typically between 0.005 and 1 (Stockham, 1972). The reflectance of a surface cannot be larger than 1, since then it would reflect more light than was incident upon the surface. Illuminance, on the other hand, can produce arbitrarily large values and is limited only by the intensity and proximity of the light sources.

The dynamic range of an image is thus predominantly governed by the illuminance component. In the face of diffuse scenes, a viable approach to tone reproduction may therefore be to separate reflectance from illuminance, compress the illuminance component, and then recombine the image.

However, the assumption that all surfaces in a scene are diffuse is generally incorrect. Many high dynamic range images depict highlights and/or directly visible light sources (Figure 21.3). The luminance reflected by a specular surface may be almost as high as the light source it reflects.

Various tone reproduction operators currently used split the image into a high dynamic range base layer and a low dynamic range detail layer. These layers would represent illuminance and reflectance if the depicted scene were entirely diffuse. For scenes containing directly visible light sources or specular highlights, separation into base and detail layers still allows the design of effective tone reproduction operators, although no direct meaning can be attached to the separate layers. Such operators are discussed in Section 21.5.

21.2 Dynamic Range

Conventional images are stored with one byte per pixel for each of the red, green and blue components. The dynamic range afforded by such an encoding depends on the ratio between smallest and largest representable value, as well as the step size between successive values. Thus, for low dynamic range images, there are only 256 different values per color channel.

High dynamic range images encode a significantly larger set of possible values; the maximum representable value may be much larger and the step size between successive values may be much smaller. The file size of high dynamic

第二类运算符以物理为基础。光在被光感受器吸收之前与表面和体积相互作用。在计算机图形学中，光交互通常由渲染方程建模。对于纯漫反射表面，该方程可以简化为入射到表面的光（照度）与该表面反射光的能力（反射率）之间的乘积（Oppenheim, Schafer, & Stockham, 1968）。

由于反射率是表面的被动属性，因此对于漫反射表面，根据定义，它的动态范围很低—通常在0之间。0.005和1（Stockham, 1972）。一个表面的反射率不能大于1，因为这样它会反射比入射到表面更多的光。另一方面，照度可以产生任意大的值，并且仅受光源强度和接近度的限制。

因此，图像的动态范围主要由illuminance分量控制。面对漫反射场景，色调重新制作的可行方法可能因此是将反射率与照度分开，压缩照度分量，然后重新组合图像。

但是，假设场景中的所有表面都是漫反射的，通常是不正确的。许多高动态范围图像描绘了高光和或直接可见光源（图21.3）。镜面反射的亮度可能几乎与其反射的光源一样高。

目前使用的各种色调再现算子将图像分割成高动态范围基础层和低动态范围细节层。如果所描绘的场景是完全漫反射的，这些层将表示照度和反射率。对于包含直接可见光源或镜面高光的场景，分离为基础层和细节层仍然允许设计有效的色调重新制作操作员，尽管没有直接意义可以附加到单独的层。此类运算符在第21.5节中讨论。

21.2 动态范围

传统的图像存储在每个像素一个字节的红色，绿色和蓝色分量。这种编码所提供的动态范围取决于最小值和最大值之间的比率，以及连续值之间的步长。因此，对于低动态范围图像，每个颜色通道只有256个不同的值。

高动态范围图像编码显着更大的一组可能的values;最大可表示值可能大得多，步长可能是tween连续值可能小得多。高动态的文件大小

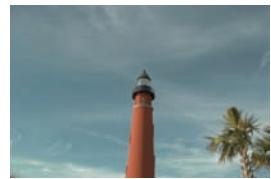


Figure 21.6. Dynamic range of $2.65 \log_2$ units.



Figure 21.7. Dynamic range of $3.96 \log_2$ units.



Figure 21.8. Dynamic range of $4.22 \log_2$ units.



Figure 21.9. Dynamic range of $5.01 \log_2$ units.

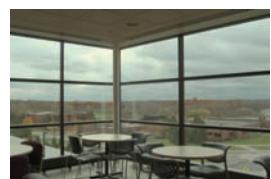


Figure 21.10. Dynamic range of $6.56 \log_2$ units.

range images is therefore generally larger as well, although at least one standard (the OpenEXR high dynamic range file format (Kainz, Bogart, & Hess, 2003)) includes a very capable compression scheme.

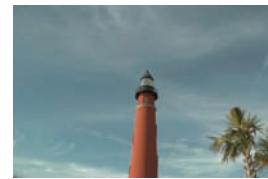
A different approach to limit file sizes is to apply a tone reproduction operator to the high dynamic data. The result may then be encoded in JPEG format. In addition, the input image may be divided pixel-wise by the tonemapped image. The result of this division can then be subsampled and stored as a small amount of data in the header of the same JPEG image (G. Ward & Simmons, 2004). The file size of such sub-band encoded images is of the same order as conventional JPEG encoded images. Display programs can display the JPEG image directly or may reconstruct the high dynamic range image by multiplying the tonemapped image with the data stored in the header.

In general, the combination of smallest step size and ratio of the smallest and largest representable values determines the dynamic range that an image encoding scheme affords. For computer-generated imagery, an image is typically stored as a triplet of floating point values before it is written to file or displayed on screen, although more efficient encoding schemes are possible (Reinhard, Ward, Debevec, & Pattanaik, 2005). Since most display devices are still fitted with eight-bit D/A converters, we may think of tone reproduction as the mapping of floating point numbers to bytes such that the result is displayable on a low dynamic range display device.

The dynamic range of individual images is generally smaller, and is determined by the smallest and largest luminances found in the scene. A simplistic approach to measure the dynamic range of an image may therefore compute the ratio between the largest and smallest pixel value of an image. Sensitivity to outliers may be reduced by ignoring a small percentage of the darkest and brightest pixels.

Alternatively, the same ratio may be expressed as a difference in the logarithmic domain. This measure is less sensitive to outliers. The images shown in the margin on this page are examples of images with different dynamic ranges. Note that the night scene in this case does not have a smaller dynamic range than the day scene. While all the values in the night scene are smaller, the ratio between largest and smallest values is not.

However, the recording device or rendering algorithm may introduce noise which will lower the useful dynamic range. Thus, a measurement of the dynamic range of an image should factor in noise. A better measure of dynamic range would therefore be a signal-to-noise ratio, expressed in decibels, as used in signal processing.



2.65 \log_2 单位的动态范围。



3.96 \log_2 单位的动态范围。



动态范围为 $4.22 \log_2$ 单位。



5.01 \log_2 单位的动态范围。



6.56 \log_2 单位的动态范围。

因此，范围图像通常也更大，尽管至少有一个标准（OpenEXR高动态范围文件格式 (Kainz, Bogart, & Hess, 2003)）包括一个非常有能力的压缩方案。

限制文件大小的另一种方法是对高动态数据应用音调再现运算符。然后可以将结果编码为JPEG格式。此外，输入图像可以像素地被色调映射的图像分割。

然后可以对该划分的结果进行二次采样，并将其作为少量数据存储在同一JPEG图像的标头中 (G.Ward & Simmons, 2004)。此类子带编码图像的文件大小与常规JPEG编码图像的量级相同。显示程序可以直接显示JPEG图像，或者可以通过将色调映射图像与存储在标题中的数据相乘来重建高动态范围图像。

一般来说，最小步长和最小和最大可表示值的比率的组合决定了图像编码方案所提供的动态范围。对于计算机生成的图像，图像通常在写入文件或显示在屏幕上之前存储为浮点值的三元组，尽管可以使用更有效的编码方案 (Reinhard, Ward, Debevec, & Pattanaik, 2005)。由于大多数显示设备仍然配备了8位Da转换器，我们可以将色调再现视为浮点数到字节的映射，以便结果可以在低动态范围显示设备上显示。

单个图像的动态范围通常较小，并且由场景中发现的最小和最大的发光来阻止。因此，测量图像动态范围的简单方法可以计算图像的最大像素值和最小像素值之间的比率。通过忽略一小部分最暗和最亮的像素，可能会降低对外层的敏感度。

或者，相同的比率可以表示为对数域的差异。该度量对异常值不太敏感。本页面边距中显示的图像是具有不同动态范围的图像示例。注意，这种情况下夜景没有比白天场景更小的动态范围。虽然夜景中的所有值都较小，但最大值和最小值之间的比率却不是。

然而，记录设备或渲染算法可能引入噪声，这会降低有用的动态范围。因此，图像动态范围的测量应该考虑噪声。因此，更好的动态范围测量方法是信号处理中使用的以分贝表示的信噪比。



Figure 21.11. Per-channel gamma correction may desaturate the image. The left image was desaturated with a value of $s = 0.5$. The right image was not desaturated ($s = 1$).

21.3 Color

Tone reproduction operators normally compress luminance values, rather than work directly on the red, green, and blue components of a color image. After these luminance values have been compressed into display values $L_d(x, y)$, a color image may be reconstructed by keeping the ratios between color channels the same as they were before compression (using $s = 1$) (Schlick, 1994b):

$$I_{r,d}(x, y) = \left(\frac{I_r(x, y)}{L_v(x, y)} \right)^s L_d(x, y),$$

$$I_{g,d}(x, y) = \left(\frac{I_g(x, y)}{L_v(x, y)} \right)^s L_d(x, y),$$

$$I_{b,d}(x, y) = \left(\frac{I_b(x, y)}{L_v(x, y)} \right)^s L_d(x, y).$$

The results frequently appear over-saturated, because human color perception is nonlinear with respect to overall luminance level. This means that if we view an image of a bright outdoor scene on a monitor in a dim environment, our eyes are adapted to the dim environment rather than the outdoor lighting. By keeping color ratios constant, we do not take this effect into account.

Alternatively, the saturation constant s may be chosen smaller than one. Such per-channel gamma correction may desaturate the results to an appropriate level, as shown in Figure 21.11 (Fattal, Lischinski, & Werman, 2002). A more comprehensive solution is to incorporate ideas from the field of color appearance modeling into tone reproduction operators (Pattanaik, Ferwerda, Fairchild, & Greenberg, 1998; Fairchild & Johnson, 2004; Reinhard & Devlin, 2005).



每通道伽马校正可能使图像去饱和。左图像以 $s=0.5$ 的值去饱和。右图像没有去饱和 ($s=1$)。

21.3 Color

色调再现运算符通常压缩亮度值，而不是直接处理彩色图像的红色、绿色和蓝色分量。在这些亮度值被压缩成显示值 $L_d(x, y)$ 之后，可以通过保持颜色通道之间的比率与压缩前相同（使用 $s=1$ ）来重建彩色图像（Schlick, 1994b）：

$$I_{r,d}(x, y) = \left(\frac{I_r(x, y)}{L_v(x, y)} \right)^s L_d(x, y),$$

$$I_{g,d}(x, y) = \left(\frac{I_g(x, y)}{L_v(x, y)} \right)^s L_d(x, y),$$

$$I_{b,d}(x, y) = \left(\frac{I_b(x, y)}{L_v(x, y)} \right)^s L_d(x, y).$$

结果经常出现过度饱和，因为人类的色彩感知相对于整体亮度水平是非线性的。这意味着，如果我们在昏暗的环境中在监视器上观看明亮的室外场景的图像，我们的眼睛适应昏暗的环境而不是室外照明。通过保持颜色比率不变，我们没有考虑到这种影响。

或者，饱和常数 s 可以选择得小于一。这种每通道伽马校正可能会使结果去饱和到适当的水平，如图21.11所示（Fattal, Lischinski, & Werman, 2002）。更具综合性的解决方案是将来自色彩外观modeling领域的想法纳入色调再现算子（Pattanaik, Ferwerda, Fairchild, & Greenberg, 1998; Fairchild & Johnson, 2004; Reinhard & Devlin, 2005）。

Finally, if an example image with a representative color scheme is already available, this color scheme may be applied to a new image. Such a mapping of colors between images may be used for subtle color correction, such as saturation adjustment or for more creative color mappings. The mapping proceeds by converting both source and target images to a decorrelated color space. In such a color space, the pixel values in each color channel may be treated independently without introducing too many artifacts (Reinhard, Ashikhmin, Gooch, & Shirley, 2001).

Mapping colors from one image to another in a decorrelated color space is then straightforward: compute the mean and standard deviation of all pixels in the source and target images for the three color channels separately. Then,

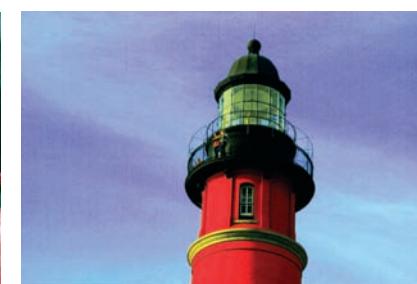
shift and scale the target image so that in each color channel the mean and standard deviation of the target image is the same as the source image. The resulting image is then obtained by converting from the decorrelated color space to RGB and clamping negative pixels to zero. The dynamic range of the image may have changed as a result of applying this algorithm. It is therefore recommended to apply this algorithm on high dynamic range images and apply a conventional tone reproduction algorithm afterward. A suitable decorrelated color space is the opponent space from Section 19.2.4.

Figure 21.12. Image used for demonstrating the color transfer technique. Results are shown in Figures 21.13 and 21.31.

The result of applying such a color transform to the image in Figure 21.12 is shown in Figure 21.13.



Figure 21.13. The image on the left is used to adjust the colors of the image shown in Figure 21.12. The result is shown on the right.



最后，如果具有代表性配色方案的示例图像已经可用，则此配色方案可应用于新图像。图像之间的颜色的这种映射可用于细微的颜色校正，例如饱和度调整或用于更有创意的颜色映射。映射通过将源图像和目标图像转换为去相关的颜色空间来进行。在这样的颜色空间中，每个颜色通道中的像素值可以被独立处理而不会引入太多伪影（Reinhard, Ashikhmin, Gooch, & Shirley, 2001）。

在去相关色彩空间中将颜色从一个图像映射到另一个图像非常简单：分别计算三个颜色通道的源图像和目标图像中所有像素的均值和标准差。然后

用于演示颜色转移技术的图像。结果见图21.13和图21.31。对目标图像进行移位和缩放，使得在每个颜色通道中目标图像的均值和标准差与源图像相同。然后通过从去相关色彩空间转换为RGB并将负像素钳位为零来获得所得图像。图像的动态范围可能由于应用该算法而改变。因此建议在高动态范围图像上应用该算法，之后应用常规色调再现算法。一个合适的去相关色彩空间是来自第19.2.4节的对手空间。

对图21.12中的图像应用这样的颜色变换的结果示于图21.13。

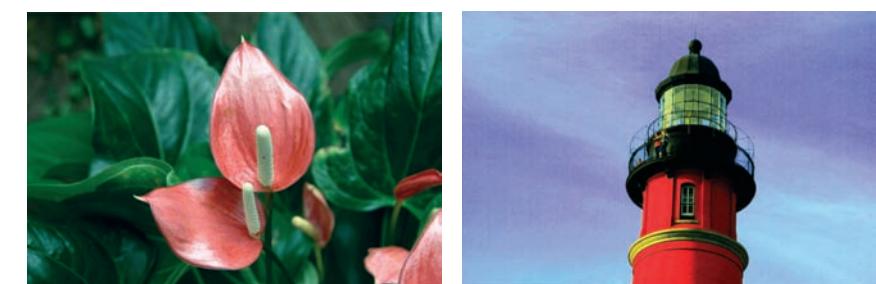


图21.13。左边的图像用于调整图21.12所示图像的颜色。结果显示在右侧。



21.4 Image Formation

For now, we assume that an image is formed as the result of light being diffusely reflected off of surfaces. Later in this chapter, we relax this constraint to scenes directly depicting light sources and highlights. The luminance L_v of each pixel is then approximated by the following product:

$$L_v(x, y) = r(x, y) E_v(x, y).$$

Here, r denotes the reflectance of a surface, and E_v denotes the illuminance. The subscript v indicates that we are using photometrically weighted quantities. Alternatively, we may write this expression in the logarithmic domain (Oppenheim et al., 1968):

$$\begin{aligned} D(x, y) &= \log(L_v(x, y)) \\ &= \log(r(x, y) E_v(x, y)) \\ &= \log(r(x, y)) + \log(E_v(x, y)). \end{aligned}$$

Photographic transparencies record images by varying the density of the material. In traditional photography, this variation has a logarithmic relation with luminance. Thus, in analogy with common practice in photography, we will use the term *density representation* (D) for log luminance. When represented in the log domain, reflectance and illuminance become additive. This facilitates separation of these two components, despite the fact that isolating either reflectance or illuminance is an under-constrained problem. In practice, separation is possible only to a certain degree and depends on the composition of the image. Nonetheless, tone reproduction could be based on disentangling these two components of image formation, as shown in the following two sections.

21.5 Frequency-Based Operators

For typical diffuse scenes, the reflectance component tends to exhibit high spatial frequencies due to textured surfaces as well as the presence of surface edges. On the other hand, illuminance tends to be a slowly varying function over space.

Since reflectance is low dynamic range and illuminance is high dynamic range, we may try to separate the two components. The frequency-dependence of both reflectance and illuminance provides a solution. We may, for instance, compute the Fourier transform of an image and attenuate only the low frequencies. This compresses the illuminance component while leaving the reflectance component



21.4 图像形成

目前，我们假设一个图像是由于光从表面扩散反射而形成的。在本章的后面，我们将此约束放宽到直接描绘光源和高光的场景。每个像素的亮度 L_v 然后由以下乘积近似：

$$L_v(x, y) = r(x, y) E_v(x, y).$$

这里， r 表示表面的反射率， E_v 表示照度。下标 v 表示我们正在使用光度加权量。或者，我们可以在对数域中写出这个表达式 (Oppenheim et al., 1968):

$$\begin{aligned} D(x, y) &= \log(L_v(x, y)) \\ &= \log(r(x, y) E_v(x, y)) \\ &= \log(r(x, y)) + \log(E_v(x, y)). \end{aligned}$$

照相透明胶片通过改变材料的密度来记录图像。在传统摄影中，这种变化与亮度呈对数关系。因此，与摄影中常见的做法类似，我们将使用术语密度表示 (D) 来表示对数亮度。当在对数域中表示时，反射率和照度变得相加。这有利于分离这两个组件，尽管隔离反射率或照度是一个受限不足的问题。在实践中，分离仅在一定程度上是可能的，并且取决于图像的组成。尽管如此，色调再现可以基于图像形成的这两个组成部分的解缠，如下面的两个部分所示。

21.5 Frequency-Based Operators

对于典型的漫反射场景，由于纹理表面以及表面边缘的存在，反射率分量倾向于表现出高空间频率。另一方面，照度往往是在空间上缓慢变化的函数。

由于反射率是低动态范围，而照度是高动态范围，我们可以尝试将这两个分量分开。反射率和照度两者的频率依赖性提供了解决方案。例如，我们可以计算图像的傅立叶变换并仅衰减低频。这压缩了照度分量，同时留下了反射率分量



Figure 21.14. Bilateral filtering removes small details but preserves sharp gradients (left). The associated detail layer is shown on the right.

largely unaffected—the very first digital tone reproduction operator known to us takes this approach (Oppenheim et al., 1968).

More recently, other operators have also followed this line of reasoning. In particular, bilateral and trilateral filters were used to separate an image into base and detail layers (Durand & Dorsey, 2002; Choudhury & Tumblin, 2003). Both filters are edge-preserving smoothing operators which may be used in a variety of different ways. Applying an edge-preserving smoothing operator to a density image results in a blurred image in which sharp edges remain present (Figure 21.14 (left)). We may view such an image as a base layer. If we then pixel-wise divide the high dynamic range image by the base layer, we obtain a detail layer which contains all the high-frequency detail (Figure 21.14 (right)).

For diffuse scenes, base and detail layers are similar to representations of illuminance and reflectance. For images depicting highlights and light sources,



Figure 21.15. An image tonemapped using bilateral filtering. The base and detail layers shown in Figure 21.14 are recombined after compressing the base layer.

Edge-preserving smoothing operators may also be used to compute a local adaptation level for each pixel, which

may be used in a spatially varying or local tone reproduction operator. We describe this use of bilateral and trilateral filters in Section 21.7.



双边滤波可去除小细节，但保留锐度梯度（左）。相关的细节图层显示在右侧。

很大程度上不受影响—我们所知的第一个数字音调再现运营商采用这种方法(Oppenheimer et al. 1968)。

最近，其他运营商也遵循了这一推理。特别是，使用双边和三边滤波器将图像分离为基础层和细节层 (Durand & Dorsey, 2002; Choudhury & Tumblin, 2003)。这两个滤波器都是边缘保持平滑运算符，可以以各种不同的方式使用。将边缘保持平滑算子应用于密度图像会导致模糊图像，其中锐边仍然存在（图21.14（左））。我们可以将这样的图像视为基础层。如果我们将高动态范围图像按像素划分为基础层，我们得到一个包含所有高频细节的细节层（图21.14（右））。

对于漫反射场景，基础层和细节层类似于照度和反射率的表示。用于描绘高光和光源的图像



这种平行不成立。但是，无论图像的内容如何，都可以将图像分离为基础层和细节层。

通过在重新组合成压缩密度图像之前压缩基础层，可能会产生低密度范围密度图像（图21.15）。指数化后，得到可显示图像。

边缘保持平滑运算符也可用于计算每个像素的局部适应水平，其可用于空间变化或局部色调再现运算符。我们在第21.7节中描述了双边和三边滤波器的这种使用。

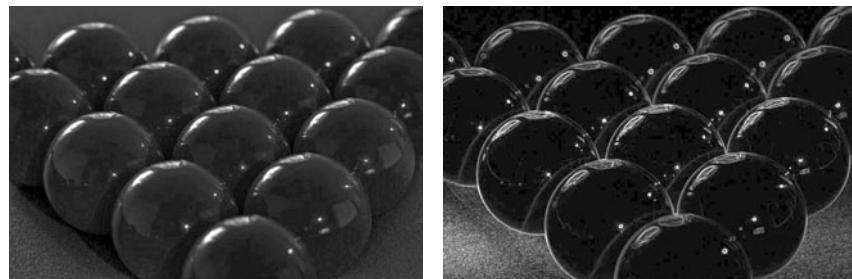


Figure 21.16. The image on the left (tonemapped using gradient-domain compression) shows a scene with highlights. These highlights show up as large gradients on the right, where the magnitude of the gradients is mapped to a grayscale (black is a gradient of 0, white is the maximum gradient in the image).

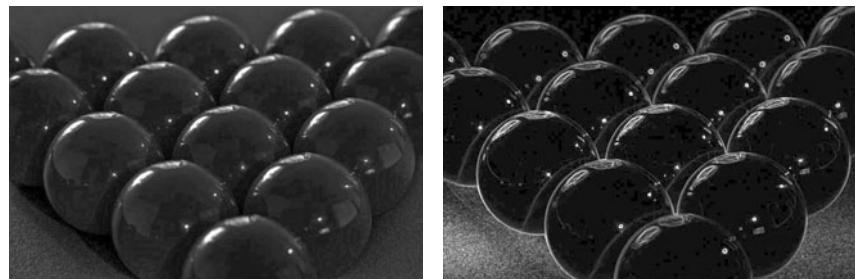
21.6 Gradient-Domain Operators

The arguments made for the frequency-based operators in the preceding section also hold for the gradient field. Assuming that no light sources are directly visible, the reflectance component will be a constant function with sharp spikes in the gradient field. Similarly, the illuminance component will cause small gradients everywhere.

Humans are generally able to separate illuminance from reflectance in typical scenes. The perception of surface reflectance after discounting the illuminant is called *lightness*. To assess the lightness of an image depicting only diffuse surfaces, B. K. P. Horn was the first to separate reflectance and illuminance using a gradient field (Horn, 1974). He used simple thresholding to remove all small gradients and then integrated the image, which involves solving a Poisson equation using the Full Multigrid Method (Press, Teukolsky, Vetterling, & Flannery, 1992).

The result is similar to an edge-preserving smoothing filter. This is according to expectation since Oppenheim's frequency-based operator works under the same assumptions of scene reflectivity and image formation. In particular, Horn's work was directly aimed at "mini-worlds of Mondrians," which are simplified versions of diffuse scenes which resemble the abstract paintings by the famous Dutch painter Piet Mondrian.

Horn's work cannot be employed directly as a tone reproduction operator, since most high dynamic range images depict light sources. However, a relatively small variation will turn this work into a suitable tone reproduction operator. If light sources or specular surfaces are depicted in the image, then large gradients will be associated with the edges of light sources and highlights. These cause the image to have a high dynamic range. An example is shown in Figure 21.16, where the highlights on the snooker balls cause sharp gradients.



左边的图像（使用梯度域压缩进行色调映射）显示了一个具有高光的场景。这些高光在右侧显示为大梯度，其中梯度的幅度映射到灰度（黑色是0的梯度，白色是图像中的最大梯度）。

21.6 Gradient-Domain Operators

上一节中为基于频率的运算符所做的参数也适用于梯度场。假设没有光源是直接可见的，反射率分量将是梯度场中具有尖锐尖峰的恒定函数。同样，照度分量会在各处引起小梯度。

在典型场景中，人类通常能够将照度与反射率分开。将光源贴现后的表面反射率的感知称为亮度。为了评估仅描绘漫反射表面的图像的亮度，B.K.P. Horn是第一个使用梯度场分离反射率和照度的人（Horn, 1974）。他使用简单的阈值去除所有小的gradients，然后对图像进行积分，这涉及使用全多网格方法求解泊松方程（Press, Teukolsky, Vetterling, & Flannery, 1992）。结果类似于边缘保留平滑滤波器。这符合预期，因为奥本海姆基于频率的算子在场景反射率和图像形成的相同假设下工作。特别是，霍恩的作品直接针对"蒙德里安的迷你世界"，这是漫射场景的简化版本，类似于荷兰着名画家彼得·蒙德里安的抽象绘画。

Horn的作品不能直接用作色调再现算子，因为大多数高动态范围图像都描绘了光源。然而，一个相对小的变化将把这个工作变成一个合适的音调再现操作器。如果在图像中描绘了光源或镜面，那么大的梯度将与光源和高光的边缘相关联。这些导致图像具有高动态范围。一个例子如图21.16所示，斯诺克球上的高光导致急剧的梯度。



Figure 21.17. An image tonemapped using gradient-domain compression.

21.7 Spatial Operators

In the following sections, we discuss tone reproduction operators which apply compression directly on pixels without transformation to other domains. Often global and local operators are distinguished. Tone reproduction operators in the former class change each pixel's luminance values according to a compressive function which is the same for each pixel. The term global stems from the fact that many such functions need to be anchored to some values determined by analyzing the full image. In practice, most operators use the geometric average \bar{L}_v to steer the compression:

$$\bar{L}_v = \exp \left(\frac{1}{N} \sum_{x,y} \log(\delta + L_v(x,y)) \right). \quad (21.1)$$

In Equation (21.1), a small constant δ is introduced to prevent the average to become zero in the presence of black pixels. The geometric average is normally mapped to a predefined display value. The effect of mapping the geometric average to different display values is shown in Figure 21.18. Alternatively, sometimes the minimum or maximum image luminance is used. The main challenge faced in the design of a global operator lies in the choice of the compressive function.

On the other hand, local operators compress each pixel according to a specific compression function which is modulated by information derived from a selection of neighboring pixels, rather than the full image. The rationale is that a bright pixel in a bright neighborhood may be perceived differently than a bright pixel in a dim neighborhood. Design challenges in the development of a local operator



使用梯度域压缩对图像进行色调映射。

21.7 空间运算符

在以下部分中，我们将讨论色调再现运算符，这些运算符直接在像素上应用压缩，而无需转换到其他域。通常全局和本地运营商是区分的。前一类中的色调再现运算符根据压缩函数改变每个像素的亮度值，该压缩函数对每个像素都是相同的。术语全局源于这样一个事实，即许多这样的函数需要锚定到通过分析全图像确定的一些值。在实践中，大多数操作员使用几何平均值 \bar{L}_v 来引导压缩：

$$\bar{L}_v = \exp \left(\frac{1}{N} \sum_{x,y} \log(\delta + L_v(x,y)) \right). \quad (21.1)$$

在等式(21.1)中，引入了一个小的常数 δ ，以防止平均值在存在黑色像素时变为零。几何平均值通常映射到预定义的显示值。将几何平均值映射到不同显示值的效果如图21.18所示。或者，有时使用最小或最大图像亮度。全局运营商设计面临的主要挑战在于压缩函数的选择。另一方面，局部运算符根据特定的压缩函数压缩每个像素，该压缩函数由从选择的邻近像素而不是完整图像中导出的信息调制。其理由是，明亮邻域中的明亮像素可能与暗淡邻域中的明亮像素不同地被感知。在开发本地运营商时面临的设计挑战



Figure 21.18. Spatial tonemapping operator applied after mapping the geometric average to different display values (left: 0.12, right: 0.38).

involves choosing the compressive function, the size of the local neighborhood for each pixel, and the manner in which local pixel values are used. In general, local operators achieve better compression than global operators (Figure 21.19), albeit at a higher computational cost.

Both global and local operators are often inspired by the human visual system. Most operators employ one of two distinct compressive functions, which is orthogonal to the distinction between local and global operators. Display values $L_d(x, y)$ are most commonly derived from image luminances $L_v(x, y)$ by the



图21.18。将几何平均值映射到不同显示值（左：0.12，右：0.38）后应用的空间色调映射算子。

涉及选择压缩函数、每个像素的局部邻域的大小以及使用局部像素值的方式。一般来说，本地运算符比全局运算符实现更好的压缩（图21.19），尽管计算成本更高。

全球和本地运营商通常都受到人类视觉系统的启发。大多数运算符使用两个不同的压缩函数之一，这与本地运算符和全局运算符之间的区别正交。显示值 $L_d(x, y)$ 最常见的是从图像亮度 $lv(x, y)$ 由



Figure 21.19. A global tone reproduction operator (left) and a local tone reproduction operator (right) (Reinhard, Stark, Shirley, & Ferwerda, 2002) of each image. The local operator shows more detail; for example, the metal badge on the right shows better contrast and the highlights are crisper.



每个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）。本地运算符显示更多细节；例如，右侧的金属徽章显示更好的对比度，高光更清晰。

following two functional forms:

$$L_d(x, y) = \frac{L_v(x, y)}{f(x, y)}, \quad (21.2)$$

$$L_d(x, y) = \frac{L_v(x, y)}{L_v(x, y) + f^n(x, y)}. \quad (21.3)$$

In these equations, $f(x, y)$ may either be a constant or a function which varies per pixel. In the former case, we have a global operator, whereas a spatially varying function $f(x, y)$ results in a local operator. The exponent n is usually a constant which is fixed for a particular operator.

Equation (21.2) divides each pixel's luminance by a value derived from either the full image or a local neighborhood. Equation (21.3) has an S-shaped curve on a log-linear plot and is called a sigmoid for that reason. This functional form fits data obtained from measuring the electrical response of photoreceptors to flashes of light in various species. In the following sections, we discuss both functional forms.

21.8 Division

Each pixel may be divided by a constant to bring the high dynamic range image within a displayable range. Such a division essentially constitutes linear scaling, as shown in Figure 21.3. While Figure 21.3 shows ad-hoc linear scaling, this approach may be refined by employing psychophysical data to derive the scaling constant $f(x, y) = k$ in Equation (21.2) (G. J. Ward, 1994; Ferwerda, Pattanaik, Shirley, & Greenberg, 1996).

Alternatively, several approaches exist that compute a spatially varying divisor. In each of these cases, $f(x, y)$ is a blurred version of the image, i.e., $f(x, y) = L_v^{\text{blur}}(x, y)$. The blur is achieved by convolving the image with a Gaussian filter (Chiu et al., 1993; Rahman, Jobson, & Woodell, 1996). In addition, the computation of $f(x, y)$ by blurring the image may be combined with a shift in white point for the purpose of color appearance modeling (Fairchild & Johnson, 2002; G. M. Johnson & Fairchild, 2003; Fairchild & Johnson, 2004).

The size and the weight of the Gaussian filter has a profound impact on the resulting displayable image. The Gaussian filter has the effect of selecting a weighted local average. Tone reproduction is then a matter of dividing each pixel by its associated weighted local average. If the size of the filter kernel is chosen too small, then haloing artifacts will occur (Figure 21.20 (left)). Haloing is a common problem with local operators and is particularly evident when tone mapping relies on division.

以下两种功能形式:

$$L_d(x, y) = \frac{L_v(x, y)}{f(x, y)}, \quad (21.2)$$

$$L_d(x, y) = \frac{L_v(x, y)}{L_v(x, y) + f^n(x, y)}. \quad (21.3)$$

在这些方程中, $f(x, y)$ 可以是常数, 也可以是每像素变化的函数。在前一种情况下, 我们有一个全局运算符, 而空间变化的函数 $f(x, y)$ 会产生一个局部运算符。指数 n 通常是一个常数, 对于特定的运算符是固定的。

公式(21.2)将每个像素的亮度除以从全图像或局部邻域导出的值。方程 (21.3) 在对数线性图上有一条S形曲线, 因此称为sigmoid。这种功能形式适合从测量光感受器对各种物种的闪光的电响应获得的数据。在以下部分中, 我们将讨论这两种功能形式。

21.8 Division

每个像素可以被一个常数分割, 以使高动态范围图像在一个可显示的范围内。这样的划分实质上构成线性缩放, 如图21.3所示。虽然图21.3显示了a d-hoc线性缩放, 但这种方法可以通过使用心理物理数据来推导公式 (21.2) 中的缩放常数 $f(x, y) = k$ 来改进 (G.J.Ward, 1994; Ferwerda, Pattanaik, Shirley, & Greenberg, 1996)。

或者, 有几种方法可以计算空间变化的di遮阳板。在每种情况下, $f(x, y)$ 是图像的模糊版本, 即 $f(x, y) = L_v^{\text{blur}}$

(x, y) 。模糊是通过用高斯滤波器对图像进行卷积来实现的 (Chiu et al., 1993; Rahman, Jobson, & Woodell, 1996)。此外, 通过模糊图像对 $f(x, y)$ 的计算可能与白点的偏移相结合, 以进行颜色外观建模 (Fairchild & Johnson 2002; G.M.Johnson & Fairchild 2003; Fairchild & Johnson 2004)。

高斯滤波器的大小和权重对得到的可显示图像有深远的影响。高斯滤波器具有选择加权局部平均值的效果。色调再现然后是将每个像素除以其相关的加权局部平均值的问题。如果筛选器内核的大小选择得太小, 则会出现晕影伪像 (图21.20 (左))。Haloing是本地运营商的common问题, 当色调映射依赖于除法时尤其明显。



Figure 21.20. Images tonemapped by dividing by Gaussian-blurred versions. The size of the filter kernel is 64 pixels for the left image and 512 pixels for the right image. For division-based algorithms, halo artifacts are minimized by choosing large filter kernels.

In general, haloing artifacts may be minimized in this approach by making the filter kernel large (Figure 21.20 (right)). Reasonable results may be obtained by choosing a filter size of at least one quarter of the image. Sometimes even larger filter kernels are desirable to minimize artifacts. Note, that in the limit, the filter size becomes as large as the image itself. In that case, the local operator becomes global, and the extra compression normally afforded by a local approach is lost.

The functional form whereby each pixel is divided by a Gaussian-blurred pixel at the same spatial position thus requires an undesirable tradeoff between amount of compression and severity of artifacts.

21.9 Sigmoids

Equation (21.3) follows a different functional form from simple division, and, therefore, affords a different tradeoff between amount of compression, presence of artifacts, and speed of computation.

Sigmoids have several desirable properties. For very small luminance values, the mapping is approximately linear, so that contrast is preserved in dark areas of the image. The function has an asymptote at one, which means that the output mapping is always bounded between 0 and 1.

In Equation (21.3), the function $f(x, y)$ may be computed as a global constant or as a spatially varying function. Following common practice in electrophysiology, we call $f(x, y)$ the *semi-saturation* constant. Its value determines which values in the input image are optimally visible after tonemapping. In particular, if we assume that the exponent n equals 1, then luminance values equal to the semi-saturation constant will be mapped to 0.5. The effect of choosing different semi-saturation constants is shown in Figure 21.21.



通过除以高斯模糊版本来绘制图像。滤波器内核的大小为左图像为64像素，右图像为512像素。对于基于除法的算法，通过选择大的滤波器内核来最小化光晕伪影。

通常，在这种方法中，可以通过使过滤器内核变大来最小化halo伪影（图21.20（右））。通过选择图像的至少四分之一的滤波器尺寸可以获得合理的结果。有时甚至更大的过滤器内核是可取的，以尽量减少伪影。请注意，在限制中，过滤器大小变得与图像本身一样大。在这种情况下，本地运算符将变为全局，并且通常由本地方法提供的额外压缩将丢失。因此，每个像素在相同空间位置被高斯模糊像素分割的功能形式需要在压缩量和伪像严重程度之间进行不希望的权衡。

21.9 Sigmoids

方程(21.3)遵循与简单除法不同的函数形式，因此在压缩量、伪像存在和计算速度之间提供了不同的折衷。

乙状结肠有几个理想的性质。对于非常小的亮度值，映射是近似线性的，以便在图像的暗区域中保留对比度。该函数在一处具有渐近线，这意味着输出映射始终在0和1之间有界。

在公式 (21.3) 中，函数 $f(x, y)$ 可以作为全局constant或空间变化函数计算。根据电生理学的常见实践，我们将 $f(x, y)$ 称为半饱和常数。它的值确定输入图像中的哪些值在色调映射后最佳可见。在 particular 中，如果我们假设指数 n 等于 1，那么等于半饱和常数的亮度值将被映射为 0.5。选择不同半饱和常数的效果如图21.21所示。

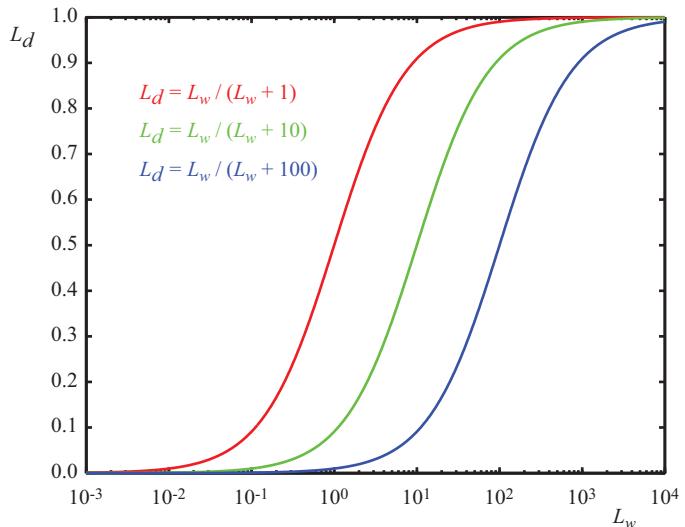


Figure 21.21. The choice of semi-saturation constant determines how input values are mapped to display values.

The function $f(x, y)$ may be computed in several different ways (Reinhard et al., 2005). In its simplest form, $f(x, y)$ is set to \bar{L}_v/k , so that the geometric average is mapped to user parameter k (Figure 21.22) (Reinhard et al., 2002). In this case, a good initial value for k is 0.18, although for particularly bright or dark scenes this value may be raised or lowered. Its value may be estimated from the image itself (Reinhard, 2003). The exponent n in Equation (21.3) may be set to 1.

In this approach, the semi-saturation constant is a function of the geometric average, and the operator is therefore global. A variation of this global operator computes the semi-saturation constant by linearly interpolating between the

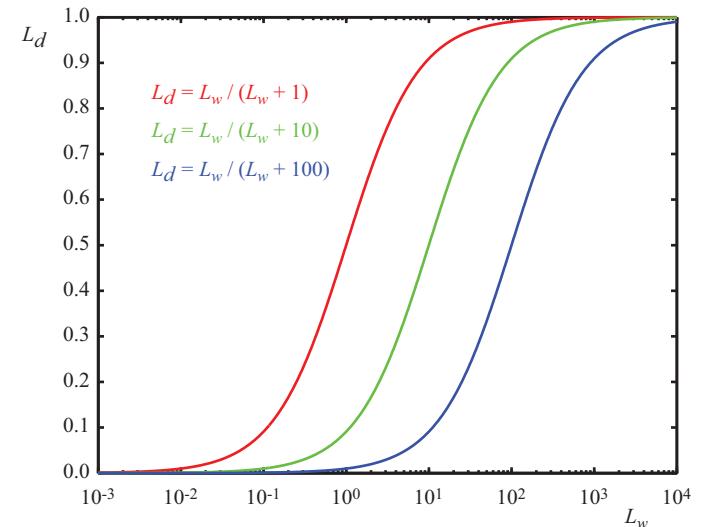


Figure 21.21. 半饱和常数的选择决定了如何将输入值映射到显示值。

函数 $f(x, y)$ 可以用几种不同的方式计算(Reinhard et al. 2005)。在其最简单的形式中, $f(x, y)$ 被设置为 \bar{L}_v/k , 使得几何平均值映射到用户参数 k (图21.22) (Reinhard et al. 2002)。在这种情况下, k 的良好初始值为0.18, 尽管对于特别明亮或黑暗的场景, 该值可以升高或降低。它的价值可以从图像本身估计 (Reinhard, 2003)。等式(21.3)中的指数 n 可以被设置为1。

在这种方法中, 半饱和常数是几何平均值的函数, 因此算子是全局的。此全局算子的一个变体通过在



Figure 21.22. A linearly scaled image (left) and an image tonemapped using sigmoidal compression (right).



图21.22。一个线性缩放的图像(左)和一个使用sigmoidal压缩的图像(右)。



Figure 21.23. Linear interpolation varies contrast in the tonemapped image. The parameter a is set to 0.0 in the left image, and to 1.0 in the right image.

geometric average and each pixel's luminance:

$$f(x, y) = a L_v(x, y) + (1 - a) \bar{L}_v.$$

The interpolation is governed by user parameter a which has the effect of varying the amount of contrast in the displayable image (Figure 21.23) (Reinhard & Devlin, 2005). More contrast means less visible detail in the light and dark areas and vice versa. This interpolation may be viewed as a halfway house between a fully global and a fully local operator by interpolating between the two extremes without resorting to expensive blurring operations.

Although operators typically compress luminance values, this particular operator may be extended to include a simple form of chromatic adaptation. It thus presents an opportunity to adjust the level of saturation normally associated with tonemapping, as discussed at the beginning of this chapter.

Rather than compress the luminance channel only, sigmoidal compression is applied to each of the three color channels:

$$\begin{aligned} I_{r,d}(x, y) &= \frac{I_r(x, y)}{I_r(x, y) + f^n(x, y)}, \\ I_{g,d}(x, y) &= \frac{I_g(x, y)}{I_g(x, y) + f^n(x, y)}, \\ I_{b,d}(x, y) &= \frac{I_b(x, y)}{I_b(x, y) + f^n(x, y)}. \end{aligned}$$

The computation of $f(x, y)$ is also modified to bilinearly interpolate between the geometric average luminance and pixel luminance and between each independent color channel and the pixel's luminance value. We therefore compute the geometric average luminance value \bar{L}_v , as well as the geometric average of the red, green, and blue channels (\bar{I}_r , \bar{I}_g , and \bar{I}_b). From these values, we compute $f(x, y)$ for each pixel and for each color channel independently. We show the equation



图21.23。线性插值会改变色调映射图像中的对比度。参数a在左图像中设置为0.0，在右图像中设置为1.0。

几何平均值和每个像素的亮度:

$$f(x, y) = a L_v(x, y) + (1 - a) \bar{L}_v.$$

插值由用户参数a控制，该参数会改变可显示图像中的对比度（图21.23）（Reinhard & Devlin, 2005）。更多的对比度意味着明暗区域的可见细节更少，反之亦然。通过在两个极值之间进行插值而不采用昂贵的模糊操作，这种插值可以被视为完全全局运算符和完全本地运算符之间的中间位置。

虽然运算符通常压缩亮度值，但这种特定的运算器可以扩展为包括简单形式的色度适应。因此，它提供了一个调整通常与色调映射相关的饱和度水平的机会，如本章开头所讨论的那样。

而不是只压缩亮度通道，sigmoidal压缩应用于三个颜色通道中的每一个：

$$\begin{aligned} I_{r,d}(x, y) &= \frac{I_r(x, y)}{I_r(x, y) + f^n(x, y)}, \\ I_{g,d}(x, y) &= \frac{I_g(x, y)}{I_g(x, y) + f^n(x, y)}, \\ I_{b,d}(x, y) &= \frac{I_b(x, y)}{I_b(x, y) + f^n(x, y)}. \end{aligned}$$

$F(x, y)$ 的计算也被修改为在几何平均亮度和像素亮度之间以及在每个独立的颜色通道和像素的亮度值之间进行双线性插值。因此，我们计算几何平均亮度值 L_v ，以及红色，绿色和蓝色通道的几何平均值(I_r , I_g 和 I_b)。根据这些值，我们分别计算每个像素和每个颜色通道的 $f(x, y)$ 。我们展示方程

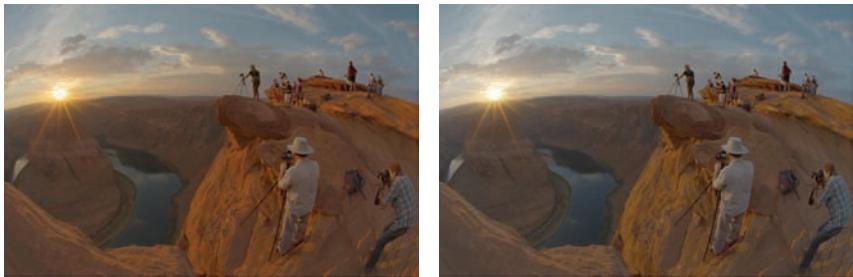


Figure 21.24. Linear interpolation for color correction. The parameter c is set to 0.0 in the left image, and to 1.0 in the right image.

for the red channel ($f_r(x, y)$):

$$\begin{aligned} G_r(x, y) &= c I_r(x, y) + (1 - c) L_v(x, y), \\ \bar{G}_r(x, y) &= c \bar{I}_r + (1 - c) \bar{L}_v, \\ f_r(x, y) &= a G_r(x, y) + (1 - a) \bar{G}_r(x, y). \end{aligned}$$

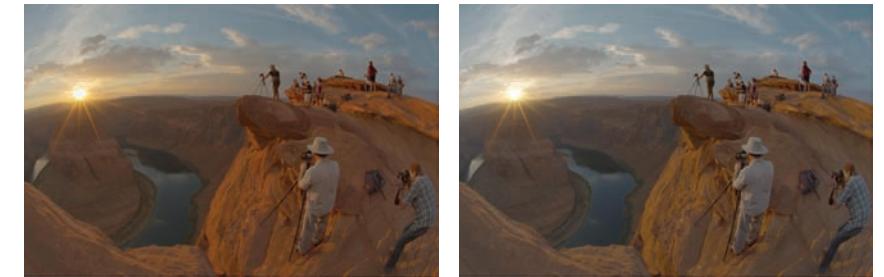
The interpolation parameter a steers the amount of contrast as before, and the new interpolation parameter c allows a simple form of color correction (Figure 21.24).

So far we have not discussed the value of the exponent n in Equation (21.3). Studies in electrophysiology report values between $n = 0.2$ and $n = 0.9$ (Hood, Finkelstein, & Buckingham, 1979). While the exponent may be user-specified, for a wide variety of images we may estimate a reasonable value from the geometric average luminance \bar{L}_v and the minimum and maximum luminance in the image (L_{\min} and L_{\max}) with the following empirical equation:

$$n = 0.3 + 0.7 \left(\frac{L_{\max} - \bar{L}_v}{L_{\max} - L_{\min}} \right)^{1.4}.$$

The several variants of sigmoidal compression shown so far are all global in nature. This has the advantage that they are fast to compute, and they are very suitable for medium to high dynamic range images. For very high dynamic range images, it may be necessary to resort to a local operator, since this may give some extra compression. A straightforward method to extend sigmoidal compression replaces the global semi-saturation constant by a spatially varying function, which may be computed in several different ways.

In other words, the function $f(x, y)$ is so far assumed to be constant, but may also be computed as a spatially localized average. Perhaps the simplest way to accomplish this is to once more use a Gaussian-blurred image. Each pixel in



进行颜色校正的线性插值。参数c在左图像中设置为0.0，在右图像中设置为1.0。

对于红色通道 (fr (x, y)) :

$$\begin{aligned} G_r(x, y) &= c I_r(x, y) + (1 - c) L_v(x, y), \\ \bar{G}_r(x, y) &= c \bar{I}_r + (1 - c) \bar{L}_v, \\ f_r(x, y) &= a G_r(x, y) + (1 - a) \bar{G}_r(x, y). \end{aligned}$$

插值参数a像以前一样引导对比度量，新的插值参数c允许简单形式的颜色校正（图21.24）。到目前为止，我们还没有讨论方程（21.3）中指数n的值。

电生理学研究报告n=0之间的值。2和n=0.9 (Hood, Finkelstein, & Buckingham, 1979)。虽然指数可以是用户指定的，但对于各种各样的图像，我们可以用下面的经验方程从几何平均亮度Lv和图像中的最小和最大亮度 (Lmin和Lmax) 估计一个合理的值：

$$n = 0.3 + 0.7 \left(\frac{L_{\max} - \bar{L}_v}{L_{\max} - L_{\min}} \right)^{1.4}.$$

到目前为止显示的sigmoidal压缩的几种变体在nature中都是全球性的。这具有这样的优点，即它们计算速度快，并且非常适合中到高动态范围图像。对于非常高的动态范围图像，可能需要求助于本地运营商，因为这可能会给出一些额外的压缩。扩展sigmoidal压缩的简单方法通过空间变化函数替换全局半饱和常数，该函数可以以几种不同的方式计算。

换句话说，函数f(x,y)迄今被假定为常数，但也可被计算为空间局部平均值。也许最简单的方法是再次使用高斯模糊图像。每个像素在

a blurred image represents a locally averaged value which may be viewed as a suitable choice for the semi-saturation constant¹.

As with division-based operators discussed in the previous section, we have to consider haloing artifacts. However, when an image is divided by a Gaussian-blurred version of itself, the size of the Gaussian filter kernel needs to be large in order to minimize halos. If sigmoids are used with a spatially variant semi-saturation constant, the Gaussian filter kernel needs to be made small in order to minimize artifacts. This is a significant improvement, since small amounts of Gaussian blur may be efficiently computed directly in the spatial domain. In other words, there is no need to resort to expensive Fourier transforms. In practice, filter kernels of only a few pixels width are sufficient to suppress significant artifacts while at the same time producing more local contrast in the tonemapped images.

One potential issue with Gaussian blur is that the filter blurs across sharp contrast edges in the same way that it blurs small details. In practice, if there

is a large contrast gradient in the neighborhood of the pixel under consideration, this causes the Gaussian-blurred pixel to be significantly different from the pixel itself. This is the direct cause for halos. By using a very large filter kernel in a division-based approach, such large contrasts are averaged out.

In sigmoidal compression schemes, a small Gaussian filter minimizes the chances of overlapping with a sharp contrast gradient. In that case, halos still occur, but their size is such that they

usually go unnoticed and instead are perceived as enhancing contrast.

Another way to blur an image, while minimizing the negative effects of nearby large contrast steps, is to avoid blurring over such edges. A simple, but computationally expensive way, is to compute a stack of Gaussian-blurred images with different kernel sizes. For each pixel, we may choose the largest Gaussian that does not overlap with a significant gradient.

In a relatively uniform neighborhood, the value of a Gaussian-blurred pixel should be the same regardless of the filter kernel size. Thus, the difference between a pixel filtered with two different Gaussians should be approximately zero. This difference will only change significantly if the wider filter kernel overlaps

¹Although $f(x, y)$ is now no longer a constant, we continue to refer to it as the semi-saturation constant.



Figure 21.25. Example image used to demonstrate the scale selection mechanism shown in Figure 21.26.

模糊图像表示局部平均值，其可被视为半饱和常数1的合适选择。

与上一节中讨论的基于除法的运算符一样，我们必须考虑halo伪像。然而，当图像被自身的高斯模糊版本分割时，高斯滤波器内核的大小需要很大，以便最小化光晕。如果sigmoids与空间变异半饱和常数一起使用，则需要使高斯滤波器内核变小以最小化伪影。这是一个显着的改进，因为少量高斯模糊可以直接在空间域中有效地计算。换句话说，没有必要诉诸昂贵的傅立叶变换。在实践中，只有几个像素宽度的滤波器内核就足以抑制显着的伪像，与此同时在色调映射的图像中产生更多的局部对比度。高斯模糊的一个潜在问题是，滤镜在鲜明对比边缘上的模糊与模糊小细节的方式相同。在实践中，如果有



图21.25。用于演示图21.26所示比例选择机制的示例图像。

是一个大的对比度梯度在附近的像素在考虑之下，这导致高斯模糊像素与像素本身显着不同。这是光晕的直接原因。

通过在基于除法的方法中使用非常大的filter内核，可以平均出如此大的对比度。在sigmoidal压缩方案中，一个小的高斯滤波器最大限度地减少了与鲜明对比度梯度重叠的机会。

在这种情况下，光晕仍然发生，但它们的大小是这样的，它们通常被忽视，而是被认为是增强对比度。

模糊图像的另一种方法是避免在这些边缘上模糊，同时尽量减少附近大对比度步骤的负面影响。一个简单但成本较高的方法是计算具有不同内核大小的高斯模糊图像的堆栈。对于每个像素，我们可以选择不与显着梯度重叠的最大高斯。

在相对均匀的邻域中，高斯模糊像素的值应该是相同的，而不管滤波器核大小如何。因此，用两个不同高斯滤波的像素的差值应该近似为零。只有当更宽的过滤器内核重叠时，这种差异才会发生显着变化

¹虽然 $f(x, y)$ 现在不再是常数，但我们继续将其称为半饱和常数。

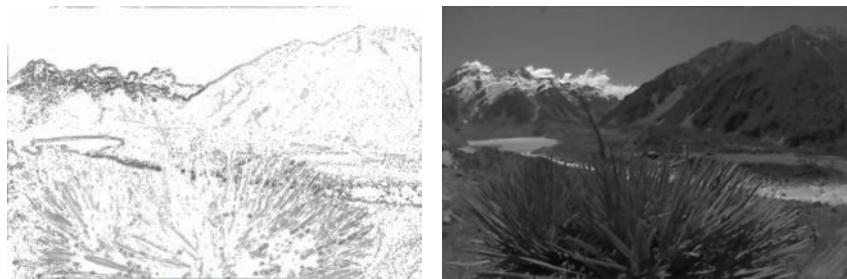


Figure 21.26. Scale selection mechanism: the left image shows the scale selected for each pixel of the image shown in Figure 21.25; the darker the pixel, the smaller the scale. A total of eight different scales were used to compute this image. The right image shows the local average computed for each pixel on the basis of the neighborhood selection mechanism.

with a neighborhood containing a sharp contrast step, whereas the smaller filter kernel does not.

It is possible, therefore, to find the largest neighborhood around a pixel that does not contain sharp edges by examining differences of Gaussians at different kernel sizes. For the image shown in Figure 21.25, the scale selected for each pixel is shown in Figure 21.26 (left). Such a scale selection mechanism is employed by the photographic tone reproduction operator (Reinhard et al., 2002) as well as in Ashikhmin's operator (Ashikhmin, 2002).

Once the appropriate neighborhood for each pixel is known, the Gaussian-blurred average L_{blur} for this neighborhood (shown on the right of Figure 21.26) may be used to steer the semi-saturation constant, such as for instance employed by the photographic tone reproduction operator:

$$L_d = \frac{L_w}{1 + L_{\text{blur}}}.$$

An alternative, and arguably better, approach is to employ edge-preserving smoothing operators, which are designed specifically for removing small details while keeping sharp contrasts in tact. Several such filters, such as the bilateral filter (Figure 21.27), trilateral filter, Susan filter, the LCIS algorithm and the mean shift algorithm are suitable, although some of them are expensive to compute (Durand & Dorsey, 2002; Choudhury & Tumblin, 2003; Pattanaik & Yee, 2002; Tumblin & Turk, 1999; Comaniciu & Meer, 2002).

21.10 Other Approaches

Although the previous sections together discuss most tone reproduction operators to date, there are one or two operators that do not directly fit into the above cate-

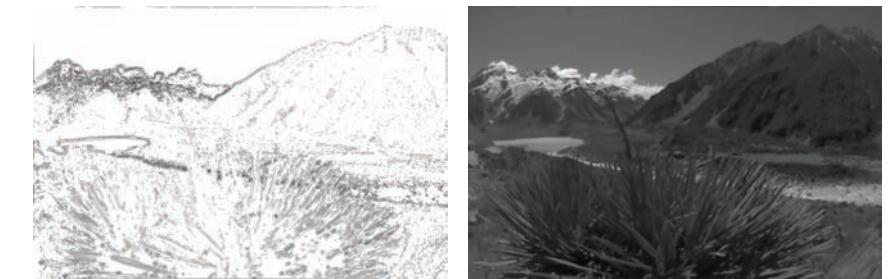


图21.26。比例选择机制：左图显示图21.25所示图像的每个像素所选择的比例；像素越暗，比例越小。总共使用了八个不同的尺度来计算这个图像。右图显示了在邻域选择机制的基础上为每个像素计算的局部平均值。

具有包含鲜明对比步骤的邻域，而较小的滤波器内核没有。

因此，通过检查不同内核大小的高斯差异，可以找到不包含锐边的像素周围的最大邻域。对于图21.25所示的图像，为每个像素选择的比例如图21.26所示（左）。这样的尺度选择机制由摄影色调再现操作员采用（Reinhard et al., 2002）以及在Ashikhmin的运营商（Ashikhmin, 2002）。

一旦已知每个像素的适当邻域，则可使用该邻域的高斯模糊平均|模糊（如图21.26的右侧所示）来引导半饱和常数，诸如例如由摄影色调再现算子所采用：

$$L_d = \frac{L_w}{1 + L_{\text{blur}}}.$$

另一种（可以说是更好的）方法是使用保留边缘的平滑运算符，该运算符专为去除小细节而设计，同时保持机智的鲜明对比。几个这样的滤波器，如双边滤波器（图21.27），三边滤波器，Susan滤波器，LCIS算法和meanshift算法是合适的，尽管其中一些计算昂贵（Durand & Dorsey, 2002; Choudhury & Tumblin, 2003; Pattanaik & Yee, 2002; Tumblin & Turk, 1999; Comaniciu & Meer, 2002）。

21.10 其他方法

虽然前面的部分一起讨论了迄今为止大多数音调再现运算符，但有一个或两个运算符不适合上述cate-



Figure 21.27. Sigmoidal compression (left) and sigmoidal compression using bilateral filtering to compute the semi-saturation constant (right). Note the improved contrast in the sky in the right image.

gories. The simplest of these are variations of logarithmic compression, and the other is a histogram-based approach.

Dynamic range reduction may be accomplished by taking the logarithm, provided that this number is greater than 1. Any positive number may then be non-linearly scaled between 0 and 1 using the following equation:

$$L_d(x, y) = \frac{\log_b(1 + L_v(x, y))}{\log_b(1 + L_{\max})}.$$

While the base b of the logarithm above is not specified, any choice of base will do. This freedom to choose the base of the logarithm may be used to vary the base with input luminance, and thus achieve an operator that is better matched to the image being compressed (Drago, Myszkowski, Annen, & Chiba, 2003). This method uses Perlin and Hoffert's bias function which takes user parameter p (Perlin & Hoffert, 1989):

$$\text{bias}_p(x) = x^{\log_{10}(p)/\log_{10}(1/2)}.$$



Figure 21.28. Logarithmic compression using base 10 logarithms (left) and logarithmic compression with varying base (right).



Sigmoidal压缩（左）和sigmoidal压缩使用双边滤波计算半饱和常数（右）。请注意右侧图像中天空对比度的改善。

戈里斯。其中最简单的是对数压缩的变化，另一种是基于直方图的方法。

动态范围缩小可以通过取对数来实现，假设这个数字大于1。然后，任何正数都可以使用以下等式在0和1之间非线性缩放：

$$L_d(x, y) = \frac{1}{\log_b(1 + L_{\max})} \log_b(1 + l_{\max}).$$

虽然上面的对数的底 b 没有指定，但任何选择的底都可以。这种选择对数底的自由可用于随输入亮度改变底，从而实现与正在压缩的图像更好匹配的算子 (Drago, Myszkowski, Annen, & Chiba, 2003)。该方法使用Perlin和Hoffert的偏置函数，该函数采用用户参数 p (Perlin & Hoffert, 1989)：

$$\text{bias}_p(x) = x^{\log_{10}(p)/\log_{10}(1/2)}.$$



对数压缩使用底10对数(左)和对数压缩与变化的底(右)。

Making the base b dependent on luminance and smoothly interpolating bases between 2 and 10, the logarithmic mapping above may be refined:

$$L_d(x, y) = \frac{\log_{10}(1 + L_v(x, y))}{\log_{10}(1 + L_{\max})} \cdot \frac{1}{\log_{10}\left(2 + 8 \left(\left(\frac{L_v(x, y)}{L_{\max}}\right)^{\log_{10}(p)/\log_{10}(1/2)}\right)\right)}.$$

For user parameter p , an initial value of around 0.85 tends to yield plausible results (Figure 21.28 (right)).

Alternatively, tone reproduction may be based on histogram equalization. Traditional histogram equalization aims to give each luminance value equal probability of occurrence in the output image. Greg Ward refines this method in a manner that preserves contrast (Ward Larson, Rushmeier, & Piatko, 1997).

First, a histogram is computed from the luminances in the high dynamic range image. From this histogram, a cumulative histogram is computed such that each bin contains the number of pixels that have a luminance value less than or equal to the luminance value that the bin represents. The cumulative histogram is a monotonically increasing function. Plotting the values in each bin against the luminance values represented by each bin therefore yields a function which may be viewed as a luminance mapping function. Scaling this function, such that the vertical axis spans the range of the display device, yields a tone reproduction operator. This technique is called histogram equalization.

Ward further refined this method by ensuring that the gradient of this function never exceeds 1. This means, that if the difference between neighboring values in the cumulative histogram is too large, this difference is clamped to 1. This avoids the problem that small changes in luminance in the input may yield large differences in the output image. In other words, by limiting the gradient of the cumulative histogram to 1, contrast is never exaggerated. The resulting algorithm is called histogram adjustment (see Figure 21.29).

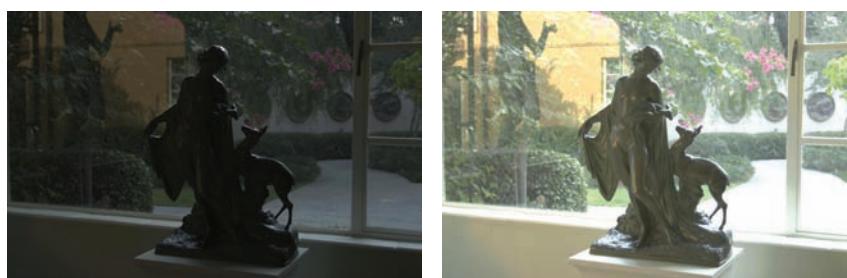


Figure 21.29. A linearly scaled image (left) and a histogram adjusted image (right). *Image created with the kind permission of the Albin Polasek museum, Winter Park, Florida.*

使底 b 依赖于亮度并平滑地内插2到10之间的底，可以细化上面的对数映射：

$$L_d(x, y) = \frac{\log_{10}(1 + L_v(x, y))}{\log_{10}(\text{最大}1+L)} \cdot \frac{1}{\log_{10}\left(2 + 8 \left(\left(\frac{L_v(x, y)}{L_{\max}}\right)^{\log_{10}(p)/\log_{10}(1/2)}\right)\right)}.$$

对于用户参数 p ，初始值约为0。85倾向于产生似是而非的结果（图21.28（右））。

或者，色调再现可以基于直方图均衡化。直方图均衡化的目的是使每个亮度值在输出图像中出现的概率相等。Greg Ward以保留对比度的方式改进了这种方法（Ward Larson, Rushmeier, & Piatko, 1997）。

首先，根据高动态范围图像中的亮度计算直方图。根据此直方图，计算累积直方图，使得每个bin包含亮度值小于或等于bin表示的亮度值的像素数。累积直方图是单调递增函数。因此，根据每个bin表示的亮度值绘制每个bin中的值会产生一个函数，该函数可以被视为亮度映射函数。缩放该功能，使得纵轴跨越显示设备的范围，产生色调再现算子。这种技术被称为直方图均衡化。

Ward通过确保此函数的梯度永远不会超过1来进一步完善此方法。这意味着，如果累积直方图中相邻值之间的差异过大，则将此差异钳位为1。这避免了输入中亮度的微小变化可能在输出图像中产生大差异的问题。换句话说，通过将累积直方图的梯度限制为1，对比度永远不会被夸大。由此产生的算法称为直方图调整（见图21.29）。

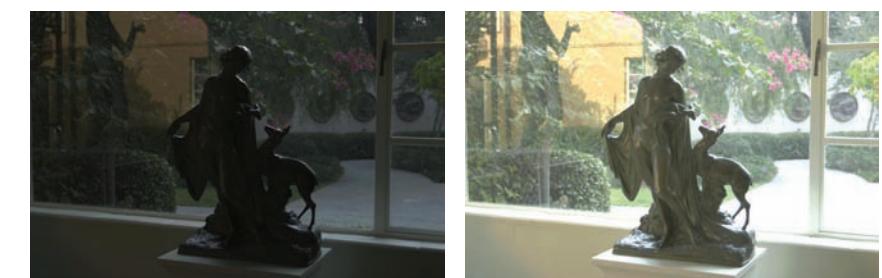


图21.29。线性缩放的图像（左）和直方图调整的图像（右）。图像创建与阿尔宾Polasek博物馆，冬季公园，佛罗里达州的亲切许可。



21.11 Night Tonemapping

The tone reproduction operators discussed so far nearly all assume that the image represents a scene under *photopic* viewing conditions, i.e., as seen at normal light levels. For *scotopic* scenes, i.e., very dark scenes, the human visual system exhibits distinctly different behavior. In particular, perceived contrast is lower, visual acuity (i.e., the smallest detail that we can distinguish) is lower, and everything has a slightly blue appearance.

To allow such images to be viewed correctly on monitors placed in photopic lighting conditions, we may preprocess the image such that it appears as if we were adapted to a very dark viewing environment. Such preprocessing frequently takes the form of a reduction in brightness and contrast, desaturation of the image, blue shift, and a reduction in visual acuity (Thompson, Shirley, & Ferwerda, 2002).

A typical approach starts by converting the image from RGB to XYZ. Then, scotopic luminance V may be computed for each pixel:

$$V = Y \left[1.33 \left(1 + \frac{Y+Z}{X} \right) - 1.68 \right].$$

This single channel image may then be scaled and multiplied by an empirically chosen bluish gray. An example is shown in Figure 21.30. If some pixels are in the photopic range, then the night image may be created by linearly blending the bluish-gray image with the input image. The fraction to use for each pixel depends on V .



Figure 21.30. Simulated night scene using the image shown in Figure 21.12.

像素在光度范围内，那么夜间图像可能是林早将蓝灰色图像与输入图像混合而成的。用于每个像素的分數取决于 V 。

Loss of visual acuity may be modeled by low-pass filtering the night image, although this would give an incorrect sense of blurriness. A better approach is to apply a bilateral filter to retain sharp edges while blurring smaller details (Tomasi & Manduchi, 1998).

Finally, the color transfer technique outlined in Section 21.3 may also be used to transform a day-lit image into a night scene. The effectiveness of this approach depends on the availability of a suitable night image from which to transfer colors. As an example, the image in Figure 21.12 is transformed into a night image in Figure 21.31.



21.11 Night Tonemapping

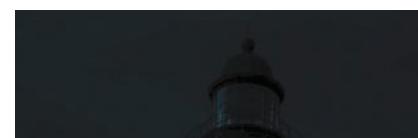
迄今为止所讨论的色调再现算子几乎都假设im年龄表示在照片观看条件下的场景，即，如在正常光水平下看到的那样。对于scotopic场景，即非常黑暗的场景，人类视觉系统表现出明显不同的行为。特别是，感知对比度较低，视觉敏锐度（即，我们可以区分的最小细节）较低，并且每件事物都具有略带蓝色的外观。

为了在光照条件下的显示器上正确地观看这些图像，我们可以对图像进行预处理，使其看起来像是适应了非常黑暗的观看环境。这样的预处理经常采取亮度和对比度降低，im年龄的去饱和，蓝移和视敏度降低的形式（Thompson, Shirley, & Ferwerda, 2002）。

典型的方法是将图像从RGB转换为XYZ。然后，可以计算每个像素的亮度 v :

$$V = Y \left[1.33 \left(1 + \frac{Y+Z}{X} \right) - 1.68 \right].$$

这个单通道图像然后可以缩放和乘以经验选择的蓝灰色。一个例子如图21.30所示。如果有的话



像素在光度范围内，那么夜间图像可能是林早将蓝灰色图像与输入图像混合而成的。用于每个像素的分數取决于 V 。

视敏度的损失可能通过夜间im年龄的低通滤波来modeled，尽管这会给人一种模糊感。一个更好的approach是应用双边滤波器来重新获得尖锐的边缘，同时模糊较小的细节 (Tomasi & Manduchi, 1998)。最后，第21.3节中概述的颜色转移技术也可用于将白天照明的图像转换为夜景。这种方法的有效性取决于合适的夜间图像的可用性，从中传输颜色。作为一个例子，图21.12中的图像在

Figure 21.31.



Figure 21.31. The image on the left is used to transform the image of Figure 21.12 into a night scene, shown here on the right.

21.12 Discussion

Since global illumination algorithms naturally produce high dynamic range images, direct display of the resulting images is not possible. Rather than resort to linear scaling or clamping, a tone reproduction operator should be used. Any tone reproduction operator is better than using no tone reproduction. Dependent on the requirements of the application, one of several operators may be suitable.

For instance, real-time rendering applications should probably resort to a simple sigmoidal compression, since these are fast enough to also run in real time. In addition, their visual quality is often good enough. The histogram adjustment technique (Ward Larson et al., 1997) may also be fast enough for real-time operation.

For scenes containing a very high dynamic range, better compression may be achieved with a local operator. However, the computational cost is frequently substantially higher, leaving these operators suitable only for noninteractive applications. Among the fastest of the local operators is the bilateral filter due to the optimizations afforded by this technique (Durand & Dorsey, 2002).

This filter is interesting as a tone reproduction operator by itself, or it may be used to compute a local adaptation level for use in a sigmoidal compression function. In either case, the filter respects sharp contrast changes and smoothes over smaller contrasts. This is an important feature that helps minimize halo artifacts, which are a common problem with local operators.

An alternative approach to minimize halo artifacts is the scale selection mechanism used in the photographic tone reproduction operator (Reinhard et al., 2002), although this technique is slower to compute.

In summary, while a large number of tone reproduction operators is currently available, only a small number of fundamentally different approaches exist. Fourier-domain and gradient-domain operators are both rooted in knowledge of



左侧的图像用于将图21.12的图像转换为夜景，此处显示在右侧。

21.12 Discussion

由于全局照明算法自然产生高动态范围ima，因此无法直接显示所得图像。而不是诉诸线性缩放或夹紧，应使用色调再现算子。任何音调再现运算符都比不使用音调再现更好。取决于应用的要求，几个操作员中的一个可能是合适的。

例如，实时渲染应用程序可能应该诉诸simplesigmoidal压缩，因为这些速度足够快，也可以实时运行。此外，它们的视觉质量往往足够好。的直方图调整技术（Ward Larson et al., 1997）对于实时操作来说也可能足够快。

对于包含非常高动态范围的场景，可以使用本地操作员实现更好的压缩。然而，计算成本通常要高得多，这使得这些运算符只适用于非交互式ap计算。其中最快的本地运营商是双边滤波器由于这种技术提供的优化（Durand & Dorsey, 2002）。

这个滤波器本身作为一个音调再现运算符是有趣的，或者它可以用来自计算一个局部适应水平，以用于sigmoidal压缩函数。无论哪种情况，滤镜都尊重鲜明的对比度变化，并在较小的对比度下平滑。这是一个重要的功能，有助于最大限度地减少光晕伪影，这是本地运营商的常见问题。

最小化光晕伪影的替代方法是在照相色调再现算子中使用的尺度选择anis m(Reinhard et al., 2002)，尽管这种技术计算速度较慢。

总之，虽然大量的音调再现运算符是可用的，但仅存在少量根本不同的方法。傅立叶域和梯度域运算符都根植于



image formation. Spatial-domain operators are either spatially variant (local) or global in nature. These operators are usually based on insights gained from studying the human visual system (and the visual system of many other species).



图像形成。空间域运算符在空间上是可变的（局部的）或全局的。这些操作符通常基于从研究人类视觉系统（以及许多其他物种的视觉系统）中获得的见解。



Implicit Modeling

Implicit modeling (also known as implicit surfaces) in computer graphics covers many different methods for defining models. These include *skeletal implicit modeling*, *offset surfaces*, *level sets*, *variational surfaces*, and *algebraic surfaces*. In this chapter, we briefly touch on these methods and describe how to build skeletal implicit models in more detail. Curves can be defined by implicit equations of the form

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

If we consider a closed curve, such as a circle, with radius r , then the implicit equation can be written as

$$f(x, y) = x^2 + y^2 - r^2 = 0. \quad (22.1)$$

The value of $f(x, y)$ can be positive (outside the circle), negative (inside the circle), or zero for points precisely on the circle. The equivalent in three dimensions is a closed surface around a set of points that occupy a given volume or region of space. The volume forms a scalar field, i.e., we can compute a value for every point and as can be seen for the circle, the negative values are bounded by the implicit curve or surface. The surface can be visualized as a contour in the field, connecting points with a particular value such as zero (see Equation (22.1)). To compute such a surface implies searching through space to find the points that satisfy the implicit equation; this method is unlikely to lead to an efficient algorithm for circle drawing (and even less likely in three dimensions). This was perhaps the reason that algorithmic methods for modeling with parametric curves



隐式建模

计算机图形学中的隐式建模（也称为隐式曲面）涵盖了定义模型的许多不同方法。这些包括骨架隐式modeling，偏移曲面，水平集，变分曲面和代数曲面。在本章中，我们将简要介绍这些方法，并更详细地描述如何构建骨架隐式模型。曲线可以通过形式的隐式方程来定义

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

如果我们考虑半径为r的闭合曲线，例如圆，那么隐式方程可以写为

$$f(x, y) = x^2 + y^2 - r^2 = 0. \quad (22.1)$$

对于圆上精确的点， $f(x, y)$ 的值可以是正（圆外）、负（圆内）或零。三维等价物是围绕一组占据给定体积或空间区域的点的封闭表面。体积形成一个标量场，即我们可以计算每个点的值，并且可以看到圆，负值由隐式曲线或曲面限定。表面可以在现场可视化为轮廓，将点与特定值（如零）连接起来（见方程（22.1））。计算这样一个表面意味着通过空间搜索来找到满足隐式方程的点；这种方法不太可能导致圆绘制的有效算法（甚至更不可能在三维中）。这也许是用参数曲线建模的算法方法的原因

and surfaces were investigated before implicit methods; however, there are some good reasons to develop algorithms to visualize implicit surfaces. In this chapter we explore the implications of deriving the data from a modeling process rather than from a scanner.

Despite the computational overhead of finding the implicit surface, designing with implicit modeling techniques offers some advantages over other modeling methods. Many geometric operations are simplified using implicit methods including:

- the definition of blends;
- the standard set operations (union, intersection, difference, etc.) of constructive solid geometry (CSG);
- functional composition with other implicit functions (e.g., R-functions, Barthe blends, Ricci blends, and warping);
- inside/outside tests, (e.g., for collision detection).



Figure 22.1. Blinn's Blobby Man 1980. *Image courtesy Jim Blinn.*

Visualizing the surfaces can be done either by direct ray tracing using an algorithm as described in (Kalra & Barr, 1989; Mitchell, 1990; Hart & Baker, 1996; deGroot & Wyvill, 2005) or by first converting to polygons (Wyvill, McPheeers, & Wyvill, 1986).

One of the first methods was proposed by Ricci as far back as 1973 (Ricci, 1973), who also introduced CSG in the same paper. Jim Blinn's algorithm for finding contours in electron density fields, known as *Blobby molecules* (J. Blinn, 1982), Nishimura's *Metaballs* (Nishimura et al., 1985) and Wyvills' *Soft Objects* (Wyvill et al., 1986) were all early examples of implicit modeling methods. Jim Blinn's *Blobby Man* (see Figure 22.1) was the first rendering of a non-algebraic implicit model.

22.1 Implicit Functions, Skeletal Primitives, and Summation Blending

In the context of modeling an *implicit* function is defined as a function f applied to a point $\mathbf{p} \in \mathbb{E}^3$ yielding a scalar value $\in \mathbb{R}$.

The implicit function $f_i(x, y, z)$ may be split into a distance function $d_i(x, y, z)$ and a *fall-off filter function*¹ $g_i(r)$, where r stands for the distance from the skeleton and the subscript refers to the i th skeletal element.

¹These functions have been given many names by researchers in the past, e.g., *filter*, *potential*, *radial-basis*, *kernel*, but we use *fall-off filter* as a simple term to describe their appearance.

在隐式方法之前研究了表面;然而,有一些很好的理由开发算法来可视化隐式表面。在本章中,我们将探讨从建模过程而不是扫描仪获取数据的含义。

尽管查找隐式曲面的计算开销很大,但使用隐式建模技术进行设计比其他建模方法具有一些优势。在cluding中使用隐式方法简化了许多几何操作:

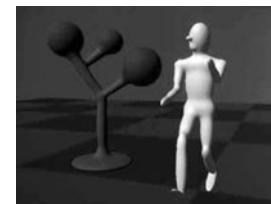


Figure 22.1. Blinn's BlobbyMan1980. *Image courtesy Jim Blinn.*

可视化表面可以通过使用algorithm的直接光线追踪来完成 (Kalra & Barr, 1989; Mitchell, 1990; Hart & Baker, 1996; deGroot & Wyvill, 2005) 或通过首先转换为多边形 (Wyvill, McPheeers, & Wyvill, 1986)。

早在1973年, Ricci就提出了第一种方法之一 (Ricci, 1973), 他也在同一篇论文中介绍了CSG。JimBlinn的算法在电子密度场中寻找轮廓,称为Blobby分子 (J.Blinn, 1982), Nishimura的Metaballs (Nishimura et al., 1985) 和Wyvills的SoftObjects (Wyvill et al., 1986) 都是隐式建模方法的早期例子。JimBlinn的BlobbyMan (见图22.1) 是第一个非代数隐式模型的渲染。

22.1 隐式函数、骨架基元和求和混合

在建模的上下文中, 隐式函数被定义为应用于点 $\mathbf{p} \in \mathbb{E}^3$ 产生标量值 $\in \mathbb{R}$ 的函数 f 。

隐式函数 $f_i(x y z)$ 可以被拆分为距离函数 $d_i(x y z)$ 和衰减滤波器函数 $g_i(r)$, 其中 r 表示与骨架的距离, 下标表示第*i*个骨架元素。

¹这些函数过去被研究人员赋予了许多名称, 例如filter, potential, radial-basis, kernel, 但是我们使用fall-offfilter作为一个简单的术语来描述它们的外观。

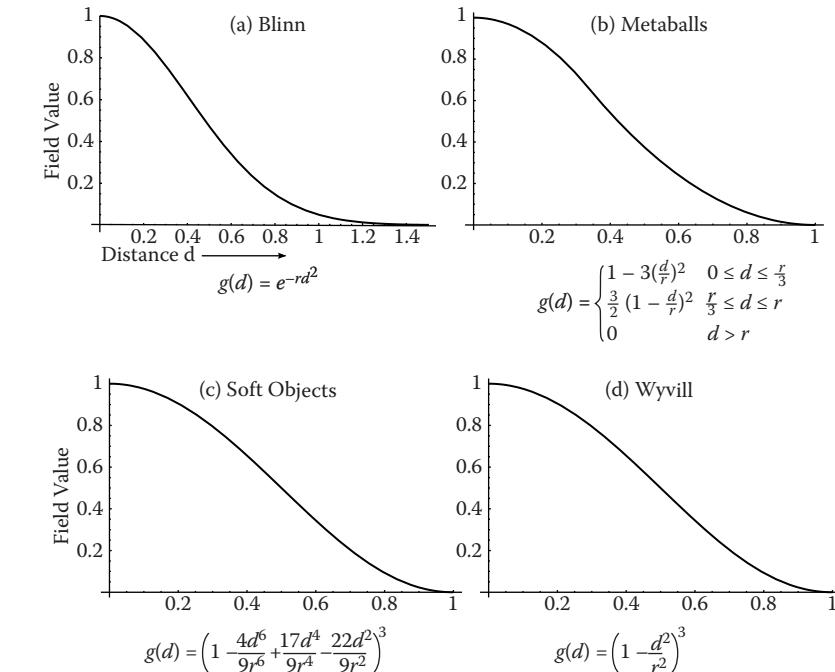


Figure 22.2. Fall-off filter functions ($0 \leq r \leq 1$). (a) Blinn's Gaussian or "Blobby" function; (b) Nishimura's "Metaball" function; (c) Wyvill et al.'s "soft objects" function; (d) the Wyvill function.

We will use the following notation:

$$f_i(x, y, z) = g_i \circ d_i(x, y, z) \quad (22.2)$$

A simple example is a point primitive, and we take the analogy of a star radiating heat into space. The field value (temperature in this example) may be measured at any point p and can be found by taking the distance from p to the center of the star and supplying the value to a fall-off filter function similar to one of those given in Figure 22.2. In these sample functions, the field is given a value of 1 at the center of the star; the value falls off with distance. The surface of a model may be derived from the implicit function $f(x, y, z)$ as the points of space whose values are equal to some desired *iso-value* (*iso*); in the star example, a spherical shell for values of $\text{iso} \in (0, 1)$.

In general, filter functions (g_i) are chosen so that the field values are maximized on the skeleton and fall off to zero at some chosen distance from the

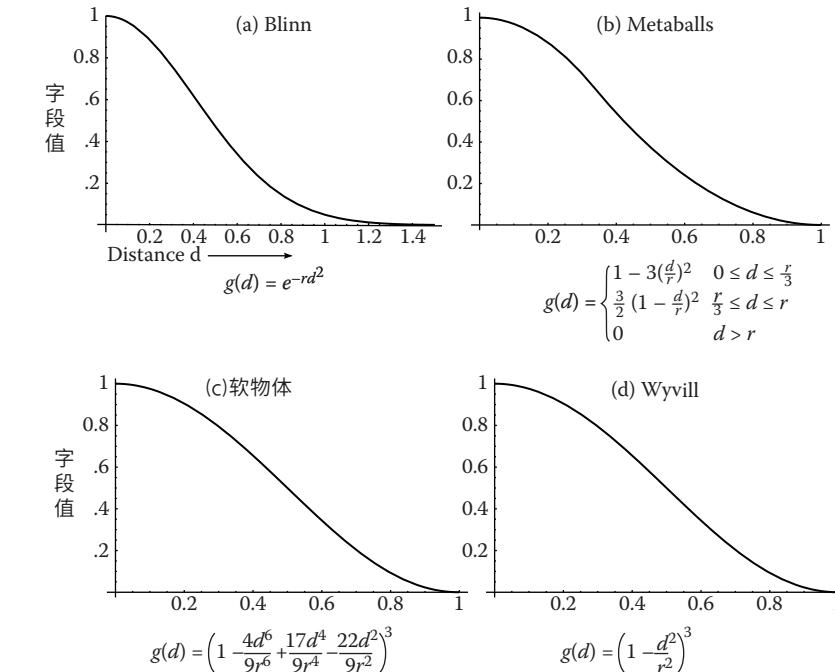


Figure 22.2. 衰減滤波器功能($0 \leq r \leq 1$)。 (a) Blinn的高斯或"Blobby"函数; (b) 西村的"Metaball"功能; (c) Wyvill等人。的"软对象"函数; (d)所述Wyvill函数。

我们将使用以下符号:

$$f_i(x, y, z) = g_i \circ d_i(x, y, z) \quad (22.2)$$

一个简单的例子是一个点基元，我们把一颗恒星的热量传递到太空的类比。场值（本例中的温度）可以在任何点 p 测量，并且可以通过取 p 到恒星中心的距离并将该值提供给类似于图22.2中给出的衰減滤波器函数之一来找到。在这些示例函数中，在恒星的中心给出了一个值1；该值随距离而下降。模型的表面可以从隐式函数 $f(x, y, z)$ 中导出，作为其值等于某些期望的*iso*值(*iso*)的空间点；在星形示例中，为*iso* ∈ (0, 1)值的球形壳。

通常，选择滤波器函数 (g_i)，以便在骨架上最大程度地模拟字段值，并与

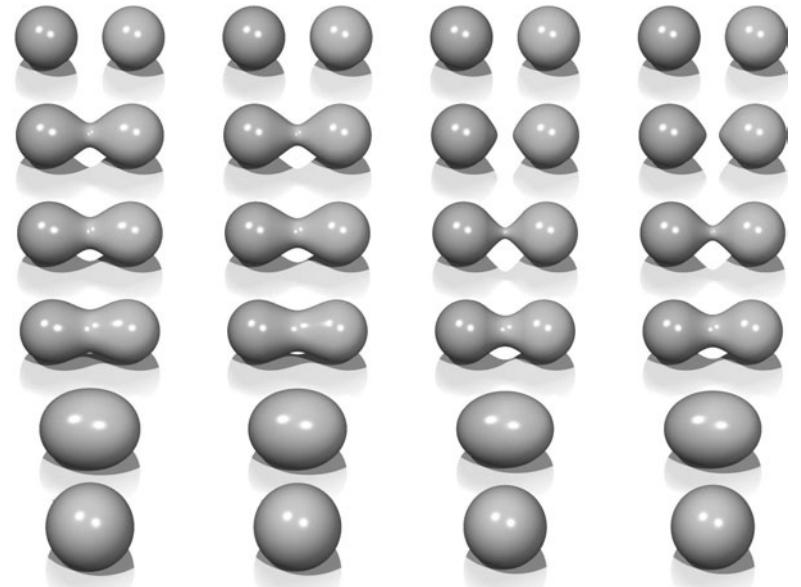


Figure 22.3. Each column shows two point primitives approaching each other. From left to right: the fall-off filter functions used are Blobby, Metaball, soft objects, and Wyvill. *Image courtesy Erwin DeGroot.*

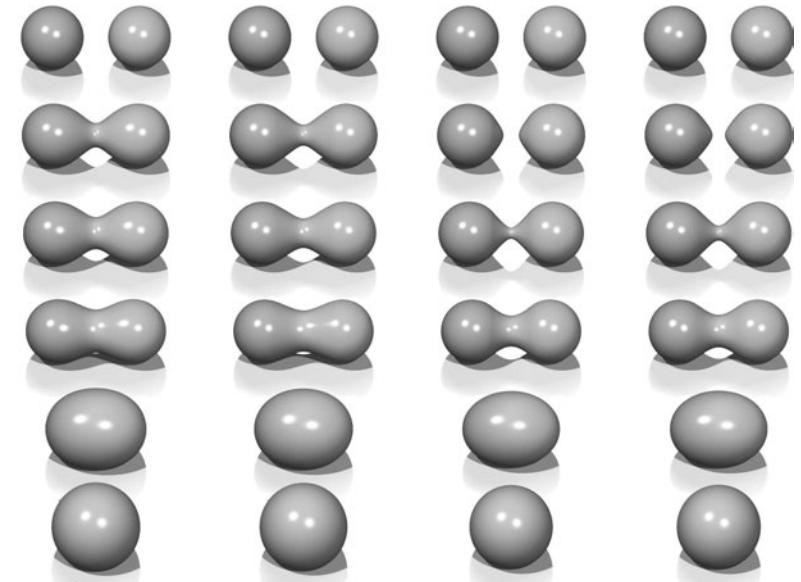
skeleton. In the simple case where the resulting surfaces are blended together, the global field $f(x, y, z)$ of an object, the implicit function, may be defined as

$$f(x, y, z) = \sum_{i=1}^{i=n} f_i(x, y, z), \quad (22.3)$$

where n skeletal elements contribute to the resulting field value. An example is shown in Figure 22.3 in which the field at any point (x, y, z) is calculated as in Equation (22.3).

In this case, two point primitives are placed in close proximity. As the two points are brought together, the surfaces bulge and then blend together. The term *filter* function is used because the function causes the primitives to be blurred together somewhat akin to a filter function for images. The summation blend is the most compact and efficient blending operation that can be applied to implicit surfaces (see Equation (22.3)).

One advantage of using filter functions with finite support is that primitives that are far from p will have zero contribution and thus need not be considered (Wyvill et al., 1986).



每列显示两个相互接近的点基元。从左到右：使用的衰减过滤器函数是Blobby，Metaball，softobjects和Wyvill。图片由ErwinDeGroot提供。

骨架。在生成的曲面混合在一起的简单情况下，对象的全局字段 $f(x, y, z)$ ，即隐式函数，可以定义为

$$f(x, y, z) = \sum_{i=1}^{i=n} f_i(x, y, z), \quad (22.3)$$

其中 n 个骨架元素有助于生成的字段值。一个例子如图22.3所示，其中任意点 (x, y, z) 的场计算如方程 (22.3) 所示。

在这种情况下，两个点基元被放置在接近。当两点结合在一起时，表面会膨胀，然后融合在一起。使用术语过滤器函数是因为该函数导致图元一起模糊，有点类似于图像的过滤器函数。求和混合是可应用于隐式曲面的最紧凑、最有效的混合操作（参见公式(22.3)）。

使用具有有限支持的滤波器函数的一个优点是远离 p 的基元将具有零贡献，因此不需要被限制(Wyvill et al. 1986).



22.1.1 C^1 Continuity and the Gradient

The most basic form of continuity is C^0 continuity, which ensures that there are no “jumps” in a function. Higher-order continuity is defined in terms of derivatives of functions (see Chapter 15).

In the case of a 3D scalar field f , the first derivative is a vector function known as the *gradient*, written ∇f and defined as

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

If ∇f is defined at all points, and the three one-dimensional partial derivatives are each C^0 , then f is C^1 . Informally, C^1 surface continuity means that the *surface normal* varies smoothly over the surface. The surface normal is the unit vector perpendicular to the surface. If no unique surface normal can be defined on the edge of a cube, for example, then the surface is not C^1 . For points on an implicit surface, the surface normal can be computed by normalizing the gradient vector ∇f . In the example of the circle, points inside have a negative value and those on the outside have a positive one. For many types of implicit surfaces, the sense of inside and outside is inverted, and since the normal vector must always point outward, it can be opposite to the gradient direction.

Skeletal implicit primitives are created by applying a fall-off filter function to an unsigned distance field as in Equation (22.2). Although the distance field is never C^1 at the skeleton, these discontinuities can be removed by using a suitable fall-off function (Akleman & Chen, 1999). If an operator, g , combines implicit functions, f_1 and f_2 , where all points are C^1 , then $g(f_1, f_2)$ is not necessarily C^1 . For example, it is possible to make a sharp CSG junction using the min and max operators. The combination is *not* C^1 continuous because the min and max operators don't have that property (see Section 22.5).

The analysis of operators is complicated by the fact that it is sometimes desirable to create a C^1 discontinuity. This case occurs whenever a crease in the surface is desired. For example, a cube is not C^1 because tangent discontinuities occur at each edge. To create creases using C^1 primitives, the operator must introduce C^1 discontinuities, and hence cannot be C^1 itself.

22.1.2 Distance Fields, R-Functions, and F-Rep

The *distance field* is defined with respect to some geometric object T :

$$F(T, \mathbf{p}) = \min_{\mathbf{q} \in T} |\mathbf{q} - \mathbf{p}|.$$

Visually, $F(T, \mathbf{p})$ is the shortest distance from \mathbf{p} to T . Hence, when \mathbf{p} lies on T ,



22.1.1 C^1 连续性和梯度

连续性的最基本形式是 C^0 连续性，它确保函数中没有“跳跃”。高阶连续性是用函数的导数来定义的（见第15章）。

在3D标量场f的情况下，一阶导数是称为梯度的向量函数，写为 ∇f 并定义为

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

如果在所有点上定义 ∇f ，并且三个一维偏导数各自为 C^0 ，则 f 为 C^1 。非正式地， C^1 表面连续性意味着表面法线在表面上平滑地变化。表面法线是垂直于表面的单位矢量。例如，如果在立方体的边缘上不能定义唯一的表面法线，则该表面不是 C^1 。对于隐式表面上的点，可以通过归一化梯度向量 ∇f 来计算表面法线。在圆的例子中，内部的点具有负值，而外部的点具有正值。对于许多类型的隐式表面，内外感是倒置的，并且由于法向量必须始终指向外，因此可以与梯度方向相反。

骨架隐式基元是通过将衰减滤波器函数应用于无符号距离场来创建的，如公式(22.2)所示。虽然距离场在骨架处从不是 C^1 ，但可以通过使用合适的脱落函数来去除这些不连续性 (Akleman & Chen, 1999)。如果运算符 g 组合隐式函数 f_1 和 f_2 ，其中所有点都是 C^1 ，则 $g(f_1, f_2)$ 不一定是 C^1 。例如，可以使用min和max运算符制作尖锐的CSG结。组合不是 C^1 连续的，因为min和max运算符没有该属性（请参阅第22.5节）。

算子的分析很复杂，因为有时创建 C^1 不连续性是可取的。这种情况发生在需要表面折痕的时候。例如，立方体不是 C^1 ，因为在每个边上发生切线不连续。要使用 C^1 基元创建折痕，运算符必须引入 C^1 不连续性，因此不能是 C^1 本身。

22.1.2 距离场、R函数和F代表

距离场是相对于某些几何对象 T 定义的：

$$F(T, \mathbf{p}) = \min_{\mathbf{q} \in T} |\mathbf{q} - \mathbf{p}|.$$

从视觉上看， $F(T, \mathbf{p})$ 是从 \mathbf{p} 到 T 的最短距离。因此，当 \mathbf{p} 位于 T

$F(T, p) = 0$ and the surface created by the implicit function is the object T . Outside of T , a nonzero distance is returned. The function T can be any geometric entity embedded in 3D—a point, curve, surface, or solid. Procedural modeling with distance fields started with Ricci (Ricci, 1973); *R-functions* (Rvachev, 1963) were first applied to shape modeling more than 20 years later (see (Shapiro, 1994) and (A. Pasko, Adzhiev, Sourin, & Savchenko, 1995)).

An R-function or Rvachev function is a function whose sign can change if and only if the sign of one of its arguments changes; that is, its sign is determined solely by its arguments. R-functions provide a robust theoretical framework for boolean composition of real functions, permitting the construction of C^n CSG operators (Shapiro, 1988). These CSG operators can be used to create blending operators simply by adding a fixed offset to the result (A. Pasko et al., 1995). Although these blending functions are no longer technically R-functions, they have most of the desirable properties and can be mixed freely with R-functions to create complex hierarchical models (Shapiro, 1988). These R-function-based blending and CSG operators are referred to as *R-operators* (see Section 22.4). The Hyperfun system (Adzhiev et al., 1999) is based on *F-reps* (function representation), another name for an implicit surface. The system uses a procedural C-like language to describe many types of implicit surfaces.

22.1.3 Level Sets

It is useful to represent an implicit field discretely via a regular grid (Barthe, Mora, Dodgson, & Sabin, 2002) or an adaptive grid (Frisken, Perry, Rockwood, & Jones, 2000). This is exactly what the polygonization algorithm does in the case of *level sets*; moreover, the grid can be used for various other purposes besides building polygons. Discrete representations of f are commonly obtained by sampling a continuous function at regular intervals. For example, the sampled function may be defined by other volume model representations (V. V. Savchenko, Pasko, Sourin, & Kunii, 1998). The data may also be a physical object sampled using three-dimensional imaging techniques. Discrete volume data has most often been used in conjunction with the *level sets* method (Osher & Sethian, 1988), which defines a means for dynamically modifying the data structure using curvature-dependent speed functions. Interactive modeling environments based on *level sets* have been defined (Museth, Breen, Whitaker, & Barr, 2002), although level sets are only one method employing a discrete representation of the implicit field. Methods for interactively defining discrete representations using standard implicit surfaces techniques have also been explored (Baerentzen & Christensen, 2002).

A key advantage to employing a discrete data structure is its ability to act as a unifying approach for all of the various volume models defined by potential

$F(T, p)=0$ 并且隐式函数创建的表面是对象 T 。 T 之外，返回非零距离。函数 T 可以是嵌入在 3D 中的任何几何实体——点、曲线、曲面或实体。具有距离场的程序建模始于 Ricci (Ricci, 1973); R-functions (Rvachev, 1963) 在 20 多年后首次应用于形状建模 (参见 (Shapiro, 1994) 和 (A.Pasko, Adzhiev, Sourin, & Savchenko, 1995))。

R 函数或 Rvachev 函数是一个函数，当且仅当它的一个参数的符号改变时，它的符号才会改变；也就是说，它的符号完全由它的参数决定。R-functions 为实函数的布尔组合提供了一个强大的理论框架，允许构建 CnCSG 运算符 (Shapiro, 1988)。这些 CSG 运算符可用于创建混合运算符，只需向结果添加固定偏移即可 (A.Pasko et al. 1995)。虽然这些混合函数在技术上不再是 R 函数，但它们具有大多数理想的属性，可以与 R 函数自由混合以创建复杂的分层模型 (Shapiro, 1988)。这些基于 R 函数的混合和 CSG 运算符称为 R 运算符 (参见第 22.4 节)。Hyperfun 系统 (Adzhiev 等, 1999) 基于 f-reps (function representation)，隐式表面的另一个名称。该系统使用类似过程的 C 语言来描述许多类型的隐式表面。

22.1.3 水平集

通过常规网格 (Barthe, Mora, Dodgson, & Sabin, 2002) 或自适应网格 (Frisken, Perry, Rockwood, & Jones, 2000) 离散地表示隐式字段是有用的。这正是多边形化算法在关卡集的情况下所做的；此外，除了构建多边形之外，网格还可以用于各种其他目的。F 的离散表示通常通过以规则间隔采样连续函数来获得。例如，采样函数可以由其它体积模型表示来定义 (V.V.Savchenko Pasko Sourin & Kunii 1998)。所述数据也可以是使用三维成像技术采样的物理对象。离散体积数据最常与水平集方法 (Osher & Sethian, 1988) 一起使用，该方法定义了一种使用 curvature-dependent speed 函数动态修改数据结构的方法。已经定义了基于级别集的交互式建模环境 (Museth, Breen, Whitaker, & Barr, 2002)，尽管级别集只是使用隐式字段的离散表示的一种方法。还探索了使用标准隐式曲面技术交互定义离散表示的方法 (Baerentzen & Christensen, 2002)。采用离散数据结构的一个关键优势是它能够作为由潜在定义的所有各种体积模型的统一方法。



fields (discrete or not) (V. V. Savchenko et al., 1998). The conversion of any continuous function to a discrete representation introduces the problem of how to reconstruct a continuous function, needed for the combined purposes of additional modeling operations and visualization of the resulting potential field. A well-known solution to this problem is to apply a filter g using the convolution operator (see Chapter 9). The choice of a filter is guided by the desired properties of the reconstruction, and many filters have been explored (Marschner & Lobb, 1994). The salient point is that there is typically a tradeoff between the efficiency of the chosen filter and the smoothness of the resulting reconstruction; see also Section 22.9.

To be interactive, a discrete system must restrict the size of the grid relative to the available computing power. This, in turn, limits the ability of the modeler to include high-frequency details. Additionally, the smoothing triquadratic filter makes it impossible to include sharp edges, should they be desired. A partial solution to this problem is the use of adaptive grids, although with any discrete representation there will be limitations. A discrete grid is used in (Schmidt, Wyvill, & Galin, 2005) to act as a cache representing a *BlobTree* node. The grid in this work is used for fast prototyping and uses trilinear interpolation for position and the slower, more accurate triquadratic interpolation to calculate gradient values, because the eye is more discerning in observing gradient errors than position errors.

22.1.4 Variational Implicit Surfaces

It is often required to convert sampled data to an implicit representation. Variational implicit surfaces interpolate or approximate a set of points using a weighted sum of globally supported basis functions (V. Savchenko, Pasko, Okunev, & Kunii, 1995; Turk & O'Brien, 1999; Carr et al., 2001; Turk & O'Brien, 2002). These radially symmetric basis functions are applied at each sample point. The continuity of such a surface depends on the choice of basis function. The C^2 thin-plate spline is most commonly used (Turk & O'Brien, 2002; Carr et al., 2001). Like Blinn's exponential function (see Figure 22.2), this function is unbounded as is the resulting variational implicit surface.

If the field is C^2 , creases cannot be defined;² however, anisotropic basis functions can be used to produce fields which change more rapidly and may appear to have creases (Dinh, Slabaugh, & Turk, 2001). At the appropriate scale, the surface is still smooth. The smooth field implies that self-intersections do not

²Except see Section 15.2.



场（离散与否）（V.V.Savchenko等人。1998）。将任何连续函数转换为离散表示引入了如何重建连续函数的问题，这对于additional建模操作和所得势场的可视化的组合目的是必需的。这个问题的一个众所周知的解决方案是使用卷积算子应用滤波器 g （参见第9章）。滤波器的选择由重建的期望性质指导，并且已经探索了许多滤波器（Marschner & Lobb, 1994）。最突出的一点是，在所选滤波器的效率和所产生的重建的平滑度之间通常有一个折衷；另请参见第22.9节。

为了实现交互式，离散系统必须相对于可用计算能力限制网格的大小。这反过来限制了modeler包含高频细节的能力。此外，平滑triquadratic滤波器使得它不可能包括尖锐的边缘，如果他们需要。这个问题的一个partial解决方案是使用自适应网格，尽管任何离散表示都会有局限性。在（Schmidt, Wyvill, & Galin, 2005）中使用离散网格作为表示BlobTree节点的缓存。这项工作中的网格用于快速原型设计，并使用位置的三线性插值和更慢，更精确的三次方插值来计算梯度值，因为眼睛在观察梯度误差方面比位置误差

22.1.4 变分隐曲面

通常需要将采样数据转换为隐式表示。Variational隐式曲面使用全局支持的基函数的加权和插值或近似一组点（V.Savchenko, Pasko, Okunev, & Kunii, 1995; Turk & O'Brien, 1999;Carretal., 2001;Turk & O'Brien, 2002）。这些径向对称基函数在每个采样点处应用。这样的表面的连续性取决于基函数的选择。C2薄板样条最常用（Turk & O'Brien, 2002;Carr等人。2001）。与Blinn的指数函数（见图22.2）一样，该函数是无界的，结果变分隐式表面也是如此。

如果字段是全局C2，则无法定义折痕²但是，各向异性基函数可用于产生变化更快且可能出现折痕的字段（Dinh, Slabaugh, & Turk, 2001）。在适当的尺度下，表面仍然是光滑的。平滑场意味着自相交不

²除了见第15.2节。

occur, and hence volumes are always well-defined. The thin-plate spline guarantees that global curvature is minimized (Duchon, 1977). Variational interpolation has many properties which are desirable for 3D modeling; however, controlling the resulting surfaces can be difficult.

Variational implicit surfaces can also be based on compactly supported radial basis functions (CS-RBFs) to reduce the computational cost of variational interpolation techniques (Morse, Yoo, Rheingans, Chen, & Subramanian, 2001). Each CS-RBF only influences a local region, so computing $f(\mathbf{p})$ requires only evaluation of basis functions within some small neighborhood of \mathbf{p} . As with the globally supported counterpart, the resulting field is C^k , creases are not supported, and self-intersections cannot occur.³ The local support of each basis function results in a bounded global field. This also guarantees that additional iso-contours will be present, as noted by various researchers (Ohtake, Belyaev, & Pasko, 2003; Reuter, 2003).

22.1.5 Convolution Surfaces

Convolution surfaces, introduced by Bloomenthal and Shoemake (Bloomenthal & Shoemake, 1991) are produced by convolving a geometric skeleton S with a *kernel* function h . Hence, the value at any position in space is defined by an integral over the skeleton:

$$f(\mathbf{p}) = \int_S g(\mathbf{r}) h(\mathbf{p} - \mathbf{r}) d\mathbf{r}.$$

Any finitely supported function can be used as h ; see (Sherstyuk, 1999) for a detailed analysis of different kernels.

Like skeletal primitives, convolution surfaces have bounded fields. Blinn's "Blobby molecules" is the simplest form of a convolution surface (J. Blinn, 1982); in this case, the skeleton consists of points only. This idea was extended by Bloomenthal to include line, arc, triangle, and polygon skeletons (Bloomenthal & Shoemake, 1991). These represent 1D and 2D primitives; 3D primitives were later described by Bloomenthal (Bloomenthal, 1995).

Combination of convolution surfaces is defined by composition of the underlying geometric skeletons and has the advantage of eliminating the bulges that tend to occur when composing multiple skeletal primitives with additive blending. The surface resulting from convolution of the combined skeleton does not have bulges, as in Figure 22.4, and the field is continuous even if the combined skele-

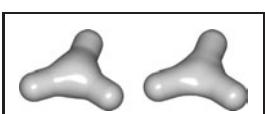


Figure 22.4. Two blended cylinders. Left: summation blend; right: convolution surface with barely discernible bulge (Bloomenthal, 1997). Image courtesy Erwin DeGroot.

³Note, $k > 0$ depending on the RBF (see Section 15.2).

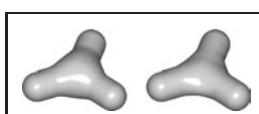
发生，因此卷总是定义良好。薄板样条保证了全局曲率最小化 (Duchon, 1977)。变分插值具有许多用于3D建模的属性；但是，控制生成的曲面可能很困难。

变分隐式曲面也可以基于紧凑支持的径向基函数 (CS-RBFs) 来降低变分间极化技术的计算成本 (Morse, Yoo, Rheingans, Chen, & Subramanian, 2001)。每个CS-RBF只影响一个局部区域，因此计算 $f(\mathbf{p})$ 只需要在 \mathbf{p} 的一些小邻域内评估基函数。与全局支持的对应项一样，结果字段为 C_k ，不支持折痕，并且不会发生自交。3个基函数的局部支持导致有界全局字段。这也保证了额外的iso轮廓将存在，正如各种研究人员所指出的那样 (Ohtake, Belyaev, & Pasko, 2003; Reuter, 2003)。

22.1.5 卷积表面

卷积表面，由Bloomenthal和Shoemake引入 (Bloomenthal & Shoemake, 1991) 是通过卷积具有核函数 h 的几何骨架 S 而产生的。因此，空间中任何位置的值都由骨架上的积分定义：

$$f(\mathbf{p}) = \int_S g(\mathbf{r}) h(\mathbf{p} - \mathbf{r}) d\mathbf{r}.$$



两个共混圆筒。左图：总和混合；右图：卷积表面几乎没有不可测的凸起(布卢门塔尔 1997)。图片由ErwinDeGroot提供。

卷积曲面的组合由基础几何骨架的组合来定义，并且具有消除在使用加性混合组合多个骨架基元时倾向于发生的凸起的优点。由组合骨架卷积产生的表面没有凸起，如图22.4所示，并且即使组合骨架也是连续的—

³注意， $k>0$ 取决于RBF（见第15.2节）。

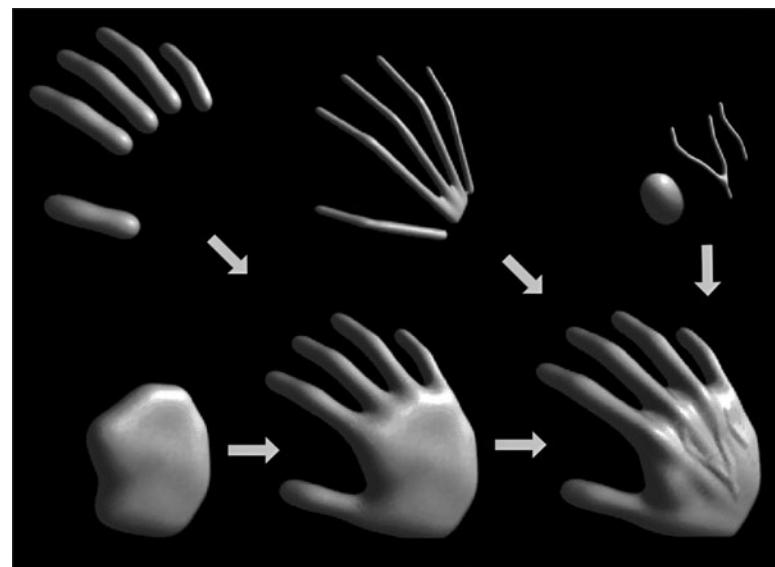


Figure 22.5. Skeletal elements convolved to build a hand model. *Image courtesy Jules Bloomenthal.*

ton is nonconvex. Convolution surfaces are offset a fixed distance from convex portions of a skeleton, but produce a fillet along concave portions of a skeleton.

An example of skeletal elements convolved to build a complex model is shown in Figure 22.5. The hand model contains fourteen primitives.

22.1.6 Defining Skeletal Primitives

As we will see in the following sections rendering the implicit models requires finding the field value and gradient for a large number of points. We need the distance to supply to Equation (22.2) and the gradient is useful for root finding as well as lighting calculations. Supplying the distance to the fall-off filter functions of Figure 22.2 is a matter of calculating the nearest distance to the skeletal primitive, simple for point primitives but a little trickier for more complex geometrical shapes. A line segment primitive (*AB*) can be defined as a cylinder around a line with hemispherical end caps (see Figure 22.6). Point P_0 lies on the surface where $f(P_0) = \text{iso}$ and $f(P_1) = 0$ since it lies outside of the influence of the line primitive. The distance from some P_i to the line is found by simply projecting onto the line *AB* and calculating the perpendicular distance, e.g., $|CP_0|$; this can be found

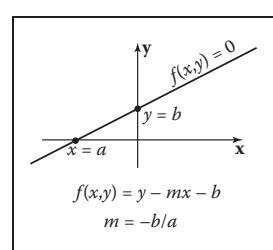


Figure 22.6. Line primitive *ab* and example points p_0 , p_1 , p_2 showing distance calculation.

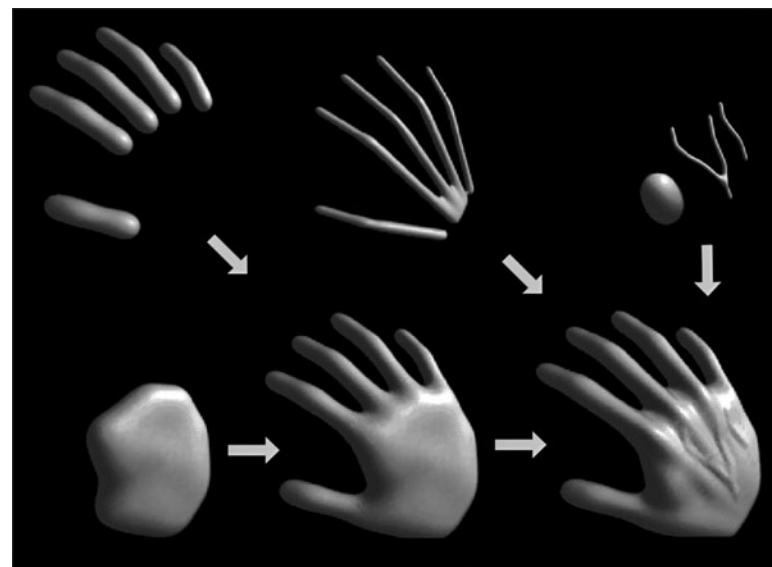


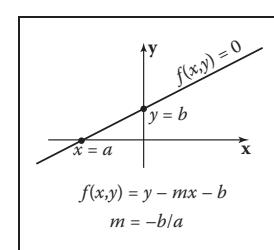
图22.5。 骨骼元素卷积以构建手部模型。图片由JulesBloomenthal提供。

ton是nonconvex。卷积表面与骨架的凸部偏移固定距离，但沿骨架的凹部产生圆角。图22.5显示了一个骨架单元卷积以构建复杂模型的例子。手模型包含十四个图元。

22.1.6 定义骨架基元

正如我们将在下面的章节中看到的，渲染隐式模型需要找到大量点的字段值和梯度。我们需要提供方程 (22.2) 的距离，梯度对于根查找和光照计算非常有用。向图22.2的衰减滤波器函数提供距离是计算到骨架基元的最近距离的问题，对于点基元来说很简单，但对于更复杂的几何形状来说有点棘手。线段基元 (*AB*) 可以定义为一条带半球形端盖的线周围的圆柱体（见图22.6）。点 P_0 位于其中的表面上

$f(P_0)=\text{iso}$ 和 $f(P_1)=0$ ，因为它位于线基元的影响之外。从一些 P_i 到线的距离是通过简单地投影到线 AB 并计算垂直距离来找到的，例如， CP_0 ；这可以找到



线primitive ab 和示出距离 calculation 的示例点 p0、p1、p2。



Figure 22.7. Cylinder primitive blended with a sphere. *Image courtesy Erwin DeGroot.*

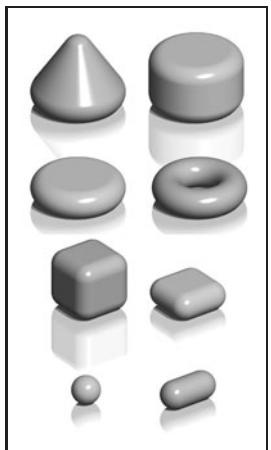


Figure 22.8. Implicit models from various skeletal primitives. *Image courtesy Erwin DeGroot.*

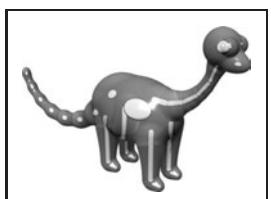


Figure 22.9. A ray-traced dinosaur model showing the underlying skeletal primitives. *Image courtesy Erwin DeGroot.*

from AC , since A , P_0 , and B , are all known:

$$\vec{AC} = \vec{AB} \frac{\vec{AP}_0 \cdot \vec{AB}}{\|\vec{AB}\|^2}.$$

In Figure 22.6, the field value of $P_2 > 0$, since P_2 is in the hemispherical endcap, which can be checked separately. Variations of this idea can define primitives with endcaps of different radii producing interesting cone shapes. An example is shown in Figure 22.7.

A great variety of geometrical skeletons have been described, and, in principle, it is simply a matter of defining the distance to the skeleton from some point p and also the gradient at p . For example, an offset surface of a triangle can be defined from the vertices of the triangle and a radius r . A simple way to implement this is to use line segment primitives to describe bounding cylinders connecting the vertices (radius r). The distance from a point q within the triangle that does not fall within the bounding fields of one of the line segment primitives is returned as the perpendicular distance to the plane of the triangle. Other examples include an implicit disk, defined by a circle and a thickness parameter, a torus also defined by a circle and the radius of the cross section (or inner and outer circle radii), a circular cone from a disk and a height, a cube with rounded corners, etc. (see Figure 22.8).

22.2 Rendering

Modeling methods, such as parametric surfaces, lend themselves to visualization, since it is easy to iterate over points on the surface that can be found directly from the defining equations; for example $(x, y) = (\cos \theta, \sin \theta)$, $\theta \in [0, 2\pi]$ produces a circle.

There are two techniques that are commonly used to render implicit surfaces: ray tracing and surface tiling. In practice, a designer wants to visualize an implicit surface model quickly, sacrificing quality for speed for interaction purposes. Prototyping algorithms have been concerned with producing a polygon mesh that can be rendered in real time on modern workstations. Finding the polygonal mesh which best approximates the desired surface is referred to as *polygonization* or *surface tiling*. For animation or for a final visualization, where quality is preferred over speed, ray tracing implicit surfaces directly without first polygonizing produces excellent results.

As previously mentioned, finding an implicit surface requires searching through space to find the points that satisfy, $f(p) = 0$. There are two main approaches to executing such a search: space partitioning—partitioning space into

从 AC , 由于 A , P_0 和 B , 都是已知的:

$$\vec{AC} = \vec{AB} \frac{\vec{AP}_0 \cdot \vec{AB}}{\|\vec{AB}\|^2}.$$

在图22.6中, $p_2>0$ 的字段值, 因为 P_2 在半球形端盖中, 可以单独检查。这种想法的变化可以定义具有不同半径的端盖的基元, 从而产生有趣的锥形形状。一个例子如图22.7所示。

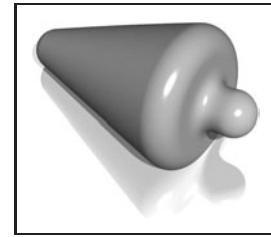
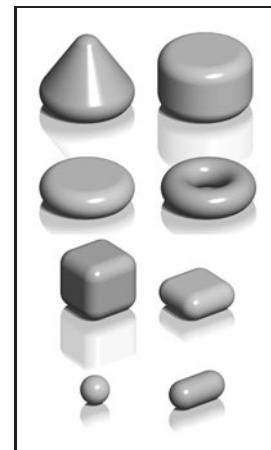
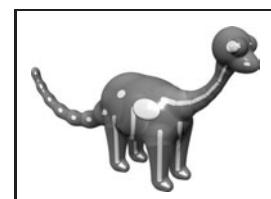


图22.7。圆柱体与球体混合。图片由ErwinDeGroot提供。



隐式模型来自各种骨骼primitives。图片由ErwinDeGroot提供。



一个光线追踪的恐龙模型, 显示了底层的骨骼基元。图片由ErwinDeGroot提供。

已经描述了各种各样的几何骨架, 原则上, 它只是定义从某个点 p 到骨架的距离以及 p 处的梯度的问题。例如, 可以从三角形的顶点和半径 r 定义三角形的偏移表面。实现这一点的一个简单方法是使用线段基元来描述连接顶点 (半径 r) 的边界圆柱体。三角形内不落在其中一个线段基元的边界域内的点 q 的距离作为到三角形平面的垂直距离返回。其他示例包括隐式盘, 由圆和厚度参数定义, 也由圆和横截面半径 (或内圆和外圆半径) 定义的环面, 来自盘和高度的圆形锥体, 具有圆角的立方体等。(见图22.8)。

22.2 Rendering

建模方法, 如参数化曲面, 适合可视化, 因为它很容易迭代面上可以直接从定义方程中找到的点; 例如 $(x, y) = (\cos \theta, \sin \theta)$, $\theta \in [0, 2\pi]$ 产生一个圆。

有两种常用于渲染隐式曲面的技术: 光线追踪和曲面平铺。在实践中, 设计人员希望快速可视化隐式曲面模型, 为交互目的而牺牲质量和速度。Prototyping 算法一直关注于生成可以在现代工作站上实时渲染的多边形网格。找到最接近所需曲面的多边形网格称为多边形化或曲面平铺。对于动画或最终可视化, 其中质量是通过速度预先考虑的, 光线跟踪隐式表面直接不首先多边形产生极好的结果。

如前所述, 找到隐式曲面需要通过空间搜索以找到满足, $f(p) = 0$ 的点。有两种主要的方法来执行这样的搜索: 空间分区-分区空间到

manageable units such as cubes, and non-space partitioning, e.g., marching triangles (Hartmann, 1998; Akkouche & Galin, 2001) and the shrinkwrap algorithm (van Overveld & Wyvill, 2004).

In this chapter, we describe the original space partitioning algorithm and leave it to the reader to explore the more advanced methods. This algorithm together with postprocessing for mesh refinement (see Chapter 12) and caching provide a method for interactive viewing of implicit models on modern workstations.

22.3 Space Partitioning

22.3.1 Exhaustive Search

The basic cubic space partitioning algorithm for tiling implicit surfaces was first published in (Wyvill et al., 1986) and a similar algorithm oriented toward volume visualization, called marching cubes in (Lorensen & Cline, 1987). Since then there have been many refinements and extensions.

A first approach to finding the implicit surface might be to subdivide space uniformly into a regular lattice of cubic cells and calculate a value for every vertex. Each cell is replaced with a set of polygons that best approximates the part of the surface contained within that cell. The problem with this method is that many of the cells will be completely outside or completely inside the volume; thus, many cells that contain no part of the surface are processed. For large grids of data this can be very time consuming and memory intensive.

To avoid storing the whole grid, a hash table is used to store only the cubes that contain a piece of the surface, based on the data structures used in (Wyvill et al., 1986). Working software was published in *Graphics Gems IV* (Bloomenthal, 1990). The algorithm is based on *numerical continuation*; it starts with a seed cube that intersects part of the surface and builds neighboring cubes as necessary to follow the surface.

The algorithm has two parts. In the first part, cubic cells are found that contain the surface and in the second part, each cube is replaced by triangles. The first part of the algorithm is driven by a queue of cubes, each of which contains part of the surface; the second part of the algorithm is table-driven.

22.3.2 Algorithm Description

A fast overview of the algorithm is as follows:

- divide space into cubic voxels;
- search for surface, starting from a skeletal element;

可管理的单元，如立方体和非空间分区，例如marchingtriangles (Hartmann, 1998; Akkouche & Galin, 2001) 和shrinkwrap算法 (vanOverveld & Wyvill, 2004)。

在本章中，我们描述了原始的空间分区算法，并留给读者去探索更高级的方法。该算法与用于网格细化的后处理（参见第12章）和缓存一起提供了一种在现代工作站上交互查看隐式模型的方法。

22.3 空间分区

22.3.1 详尽搜寻

用于平铺隐式曲面的基本立方空间分区算法首次发表于(Wyvilletal, 1986) 和面向体积可视化的类似算法，称为marchingcubesin (Lorensen & Cline, 1987)。从那时起，有许多改进和扩展。

找到隐式表面的第一种方法可能是将空间均匀地细分为立方单元的规则晶格，并计算每个vertex的值。每个单元格都替换为一组多边形，这些多边形最接近该单元格中包含的曲面部分。这种方法的问题是，许多细胞将完全在体积外部或完全在体积内部；因此，许多不包含表面部分的细胞被处理。对于大型数据网格，这可能非常耗时且占用大量内存。

为了避免存储整个网格，哈希表用于仅存储包含一块表面的立方体，基于中使用的数据结构(Wyvilletal 1986)。工作软件发表在GraphicsGemsIV (Bloomenthal, 1990) 中。该算法基于数值延续；它从与部分曲面相交的种子立方体开始，并根据需要构建邻近立方体以跟随曲面。

该算法有两个部分。在第一部分中，发现包含表面的立方细胞，在第二部分中，每个立方体被三角形替换。算法的第一部分由立方体队列驱动，每个立方体都包含部分表面；算法的第二部分是表驱动的。

22.3.2 算法描述

算法的快速概述如下：

- 将空间划分为立方体素
- *从骨架元素开始搜索表面；

- add voxel to queue, mark it visited;
- search neighbors;
- when done, replace voxel with polygons.

First, space is subdivided into a cubic lattice, and the next task is to find a seed cube containing part of the surface. A cube vertex v_i inside the surface will have a field value $v_i \geq iso$ and a vertex outside the surface will have a field value $v_i < iso$; thus, an edge with one of each type of vertex will intersect the surface. We call this an *intersecting* edge. The field value at the nearest cube vertex to the first primitive can be evaluated by summing the contributions of the primitives as per Equation (22.3), although other operators can also be used as will be seen later. We will assume that $f(v_0) > iso$, which indicates that v_0 lies within the solid. The value of iso is chosen by the user; an example is $iso = 0.5$ when using the *soft* fall-off function, which has some symmetry properties that lead to nice blending (see Figure 22.3). The vertices along one axis are evaluated in turn until a value $v_i < iso$ is found. The cube containing the *intersecting* edge is the seed cube.

The neighbors of the seed cube are examined, and those that contain at least one *intersecting* edge are added to the queue ready for processing. To process a cube, we examine each face. If any of the bounding edges have oppositely signed vertices, the surface will pass through that face and the face neighbor must be processed. When this process has been completed for all the faces, the second phase of the algorithm is applied to the cube. If the surface is closed, eventually a cube will be revisited and no more unmarked neighbors found, and the search algorithm will terminate. Processing a cube involves marking it as processed and processing its unmarked neighbors. Those that contain *intersecting* edges are processed until the entire surface has been covered (see Figure 22.10).

Each cube is indexed by an *identifying vertex* which we define to be the lower-left far corner (i.e., the vertex with the lowest (x, y, z) -coordinate values (see Figure 22.11)). For each vertex that is inside the surface, the corresponding bit will be set to form the address in an 8-bit table (see Figure 22.11 and Section 22.3.3).

The identifying vertex is addressed by integers i, j, k , computed from the (x, y, z) -coordinate location of the cube such that $x = side * i$, etc., where $side$ is the size of the cube. The identifying vertex of each cube may appear in as many as eight other cubes, and it would be inefficient to store these vertices more than once. Thus, the vertices are stored uniquely in a chained hash table. Since most of the space does not contain any part of the surface, only those cubes that are visited will be stored. The implicit function value is found for each vertex as it is stored in the hash table.

- 将体素添加到队列中，标记它。;
 - search neighbors;
- *完成后，将体素替换为多边形。

首先，空间被细分为立方晶格，接下来的任务是找到包含部分表面的种子立方体。曲面内的立方体顶点 v_i 将具有字段值 $v_i \geq iso$ ，曲面外的顶点将具有字段值 $v_i < iso$ ；因此，具有每种类型顶点之一的边将与曲面相交。我们称之为相交边。可以通过按照等式(22.3)对图元的贡献求和来评估到第一个图元的最近立方体顶点处的场值，尽管也可以如稍后将看到的那样使用其他运算符。我们将假设 $f(v_0) > iso$ ，这表明 v_0 位于固体内。 Iso 的值由用户选择；一个例子是 $Iso=0.5$ 。使用软衰减函数时，该函数具有一些对称性，可以很好地混合（见图22.3）。沿着一个轴的顶点依次计算，直到找到值 $v_i < iso$ 。包含相交边的立方体是种子立方体。

检查种子立方体的邻居，并且将包含至少一个相交边的那些添加到准备处理的队列中。要处理一个立方体，我们检查每个面。如果任何边界边具有相反签名的顶点，则曲面将穿过该面，并且必须处理面邻居。对于所有面完成此过程后，算法的第二阶段将应用于立方体。如果表面被关闭，最终将重新访问一个立方体，不再找到未标记的邻居，搜索算法将终止。处理多维数据集涉及将其标记为已处理并处理其未标记的邻居。那些包含相交边缘的被处理，直到整个表面被覆盖（见图22.10）。

每个立方体由我们定义为左下角的识别顶点（即具有最低 (x, y, z) 坐标值的顶点（见图22.11））索引。对于表面内的每个顶点，相应的位将被设置为形成8位表中的地址（见图22.11和22.3.3节）。识别顶点由整数 i, j, k 寻址，从立方体的 (x, y, z) 坐标位置计算，使得 $x=side*i$ 等。其中 $side$ 是立方体的大小。每个立方体的识别顶点可能出现在多达八个其他立方体中，并且不止一次存储这些顶点将是低效的。因此，顶点被唯一地存储在链式哈希表中。由于大部分空间不包含表面的任何部分，因此只会存储访问过的那些立方体。隐式函数值是为每个顶点找到的，因为它存储在哈希表中。

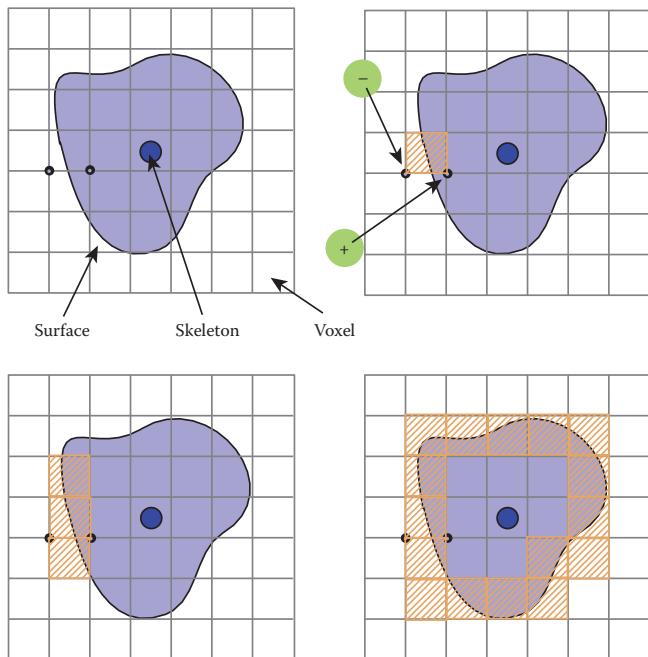


Figure 22.10. A section through the cubic lattice. The + sign indicates a vertex inside the surface ($f(v_i) \geq iso$) and - is outside $f(v_i) < iso$.

Nothing is known about the topology of the surface so a search must be started from every primitive to avoid any disconnected parts of the surface being missed. A scalar can be used to scale the influence of a primitive. If the scalar can be less than zero, then it is possible to search along an axis without finding an intersecting edge. In this case, a more sophisticated search must be done to find a seed cube (Galin & Akkouche, 1999).

Data Structures

The hash table entry holds five values:

- the i, j, k lattice indices of the identifying vertex (see Figure 22.11);
- f , the implicit function value of the identifying vertex;
- Boolean to indicate whether this cube has been visited.

The hash function computes an address in the hash table by selecting a few bits out of each of i, j, k and combining them arithmetically. For example, the five least significant bits produce a 15-bit address for a table, which must have a length

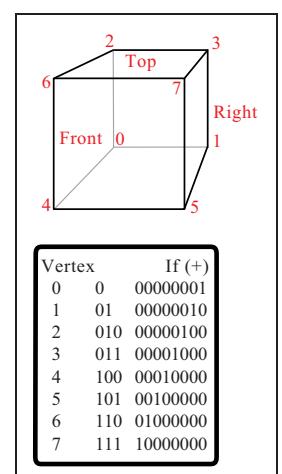


Figure 22.11. Vertex numbering.

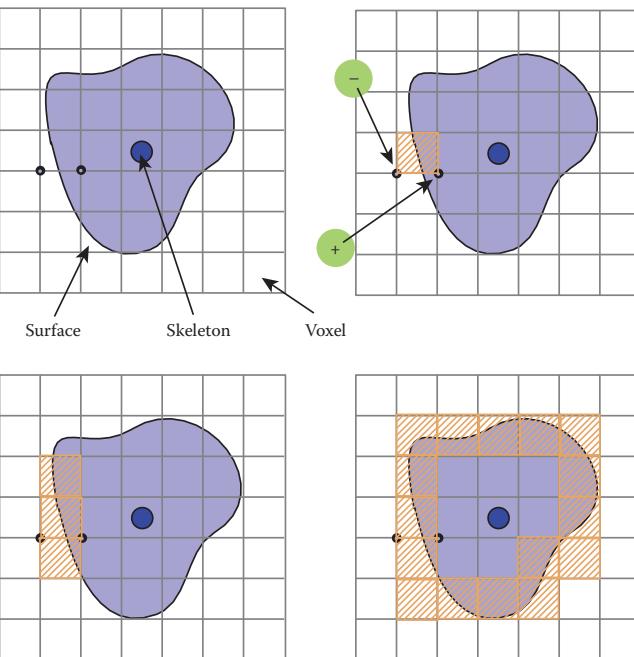


图22.10。 过立方晶格的一段。+号表示曲面内部的顶点 ($f(v_i) \geq iso$)，并且在 $f(v_i) < iso$ 之外。

对于曲面的拓扑一无所知，因此必须从每个图元开始搜索，以避免遗漏曲面的任何断开部分。标量可用于缩放图元的影响。如果标量可以小于零，那么可以沿着一个轴进行搜索而没有找到相交的边。在这种情况下，必须进行更复杂的搜索以找到种子立方体 (Galin & Akkouche, 1999)。

数据结构

哈希表条目包含五个值：

- *识别顶点的 i, j, k 晶格指数（见图22.11）；
- * f ，识别顶点的隐式函数值；
- *布尔值以指示此多维数据集是否已被访问。

哈希函数通过从 i, j, k 中的每一个中选择几个位并将它们进行算术组合来计算哈希表中的地址。例如，五个最低有效位为表生成一个15位地址，该地址必须具有长度

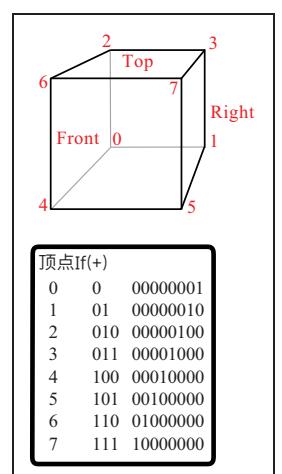


Figure 22.11. 顶点numbering.

of 2^{15} . Such a hash function can be neatly implemented in the C-preprocessor as follows:

```
#define NBITS      5
#define BMASK      037
#define HASH(a,b,c) (((a&BMASK)<<NBITS|b&BMASK)
                  <<NBITS|c&BMASK)
#define HSIZE      1<<NBITS*3
```

The queue (FIFO list) is used as temporary storage to identify the neighbors for processing. The algorithm begins with a seed cube that is marked as visited and placed on the queue. The first cube on the queue is dequeued and all its unvisited neighbors are added to the queue. Each cube is processed and passed to the second phase of the algorithm if it contains part of the surface. The queue is then processed until empty.

22.3.3 Polygonization Algorithm

The second phase of the algorithm treats each cube independently. The cell is replaced by a set of triangles that best matches the shape of the part of the surface that passes through the cell. The algorithm must decide how to polygonize the cell given the implicit function values at each vertex. These values will be positive or negative (i.e., less than or greater than the iso-value), giving 256 combinations of positive or negative vertices for the eight vertices of the cube. A table of 256 entries provides the right vertices to use in each triangle (Figure 22.12). For example, entry 4(00000100) points to a second table that records the vertices that bound the *intersecting* edges. In this example, vertex number 2 is inside the surface ($f(V_2) \geq \text{iso}$) and, therefore, we wish to draw a triangle that connects the points on the surface that intersect with edges bounded by (V_2, V_0) , (V_2, V_3) , and (V_2, V_6) as shown in Figure 22.13.

Finding Cube-Surface Intersections

Figure 22.13 shows a cube where vertex V_2 is inside the surface and all other vertices are outside. Intersections with the surface occur on three edges as shown. The surface intersects edge $V_2 - V_6$ at the point A . The fastest, but inaccurate, way to calculate A is to use linear interpolation:

$$\frac{f(A) - f(V_2)}{f(V_6) - f(V_2)} = \frac{|A - V_2|}{\text{side}}.$$

2美元。这样的散列函数可以在C-预处理器中整齐地实现，如下所示：

```
#define NBITS      5
#define BMASK      037
#define HASH(a,b,c) (((a&BMASK)<<NBITS|b&BMASK)
                  <<NBITS|c&BMASK)
#define HSIZE      1<<NBITS*3
```

队列（FIFO列表）用作临时存储，以识别用于处理的邻居。该算法从标记为已访问并放置在队列中的种子多维数据集开始。队列上的第一个多维数据集已出列，其所有未访问的邻居都将添加到队列中。如果每个立方体包含部分曲面，则会对其进行处理并传递到算法的第二阶段。然后处理队列，直到为空。

22.3.3 Polygonization Algorithm

算法的第二阶段独立地对待每个立方体。单元格由一组三角形替换，这些三角形与穿过单元格的表面部分的形状最匹配。给定每个顶点的隐式函数值，算法必须决定如何多边形化单元格。这些值将是正的或负的（即，小于或大于iso-值），给出立方体的八个顶点的正或负顶点的256个组合。一个包含256个条目的表格提供了在每个三角形中使用的正确顶点（图22.12）。例如，条目4(00000100)指向记录绑定相交边的顶点的第二个表。在本例中，顶点编号2位于曲面内 ($f(V_2) \geq \text{iso}$)，因此，我们希望绘制一个三角形，将曲面上与以 (V_2, V_0) , (V_2, V_3) 和 (V_2, V_6) 为界的边相交的点连接起来，如图22.13所示。

寻找立方体-表面交叉点

图22.13显示了一个立方体，其中顶点 V_2 在曲面内部，所有其他顶点都在外面。如图所示，在三条边上发生与表面的相交。面在点A处与边缘 $V_2 - V_6$ 相交。计算A的最快但不准确的方法是使用线性插值：

$$\frac{f(A) - f(V_2)}{f(V_6) - f(V_2)} = \frac{|A - V_2|}{\text{side}}.$$



Table 1		Table 2	
00000000	all unset	0	1 # polys
00000001	V0 set	1	3 # edges
00000010	V1 set	2	V2-V0 edges to intersect
00000011	V0 & V1 set	3	V2-V3
00000100	V2 set		V2-V6
•		•	
•		•	
•		•	
11111101	all set except V1 unset		
11111110	V1..V7 set		
11111111	V0..V7 set		

Figure 22.12. Table 2 contains the edges intersected by the surface. Table 1 points to the appropriate entry in Table 2.

If the cube side is 1 and the iso-value sought for $f(A)$ is 0.5, then

$$A = V_3 + \frac{0.5 - f(V_2)}{f(V_6) - f(V_2)}.$$

This works well for a static image, but in animation error differences between frames will be very noticeable. A root-finding method such as *regula falsi* should be employed. This becomes more computationally costly as the gradient is needed to evaluate the point of intersection. The gradient is also needed at surface points for rendering. For many types of primitives it is simpler to find a numerical approximation using sample points around p , as in

$$\nabla f(\mathbf{p}) = \left(\frac{f(\mathbf{p} + \Delta x) - f(\mathbf{p})}{\Delta x}, \frac{f(\mathbf{p} + \Delta y) - f(\mathbf{p})}{\Delta y}, \frac{f(\mathbf{p} + \Delta z) - f(\mathbf{p})}{\Delta z} \right).$$

A reasonable value for Δ has been found empirically to be $0.01 * \text{side}$ where side is the length of a cube edge.

For manufacturing a mesh, as opposed to a set of independent triangles, a second hash table can maintain a list of all the *intersecting edges*. Since each cube edge is shared by up to four neighbors, the edge hash table prevents repetition of the surface-cube edge intersection calculation. The hash address can be derived from the same hash function as for vertices (applied to the edge endpoints).

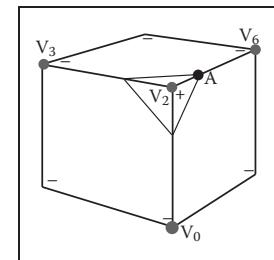


Figure 22.13. Finding the intersection of the surface with a cube edge.

Table 1		Table 2	
00000000	全部未设置	0	1 # polys
00000001	V0 set	1	3 # edges
00000010	V1 set	2	V2-V0 要相交的边
00000011	V0 & V1 set	3	V2-V3
00000100	V2 set		V2-V6
•		•	
•		•	
•		•	
11111101	除V1未设置外，所有设置		
11111110	V1..V7 set		
11111111	V0..V7 set		

图22.12。表2包含表面相交的边。表1指向表2中的适当条目。

如果立方体边为1, $f(A)$ 的iso值为0.5，则

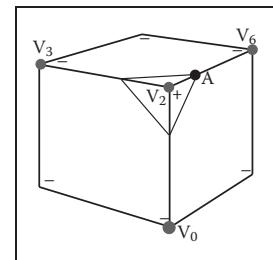
$$A = V_3 + \frac{0.5 - f(V_2)}{f(V_6) - f(V_2)}.$$

这对于静态图像效果很好，但在动画中，帧之间的错误差异将非常明显。应采用*regula falsi*等寻根方法。由于需要梯度来评估相交点，这会变得更加计算化。在表面点处也需要渐变以进行渲染。对于许多类型的基元，使用 p 周围的样本点找到数值近似更简单，如

$$\nabla f(\mathbf{p}) = \left(\frac{f(\mathbf{p} + \Delta x) - f(\mathbf{p})}{\Delta x}, \frac{f(\mathbf{p} + \Delta y) - f(\mathbf{p})}{\Delta y}, \frac{f(\mathbf{p} + \Delta z) - f(\mathbf{p})}{\Delta z} \right).$$

根据经验， Δ 的合理值为0。边是立方体边的长度。

对于制造网格，与一组独立三角形相反，第二哈希表可以保持所有相交边的列表。由于每个立方体边由最多四个邻居共享，因此边哈希表可防止重复曲面-立方体边相交计算。哈希地址可以从与顶点相同的哈希函数（应用于边缘端点）派生。



找到具有立方体边的曲面的交点。

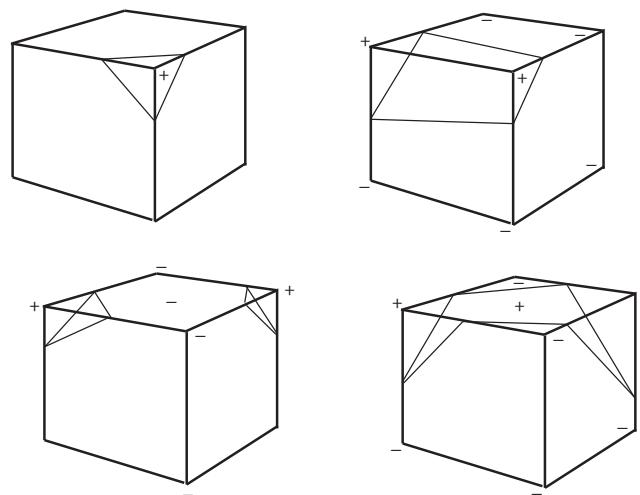


Figure 22.14. Examples of vertices inside (+) and outside (-) the surface. Note the extra sample gives a clue to avoid ambiguous cases.

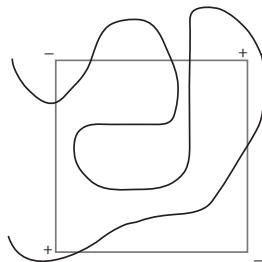


Figure 22.15. Cube too large to capture small variation in implicit function.

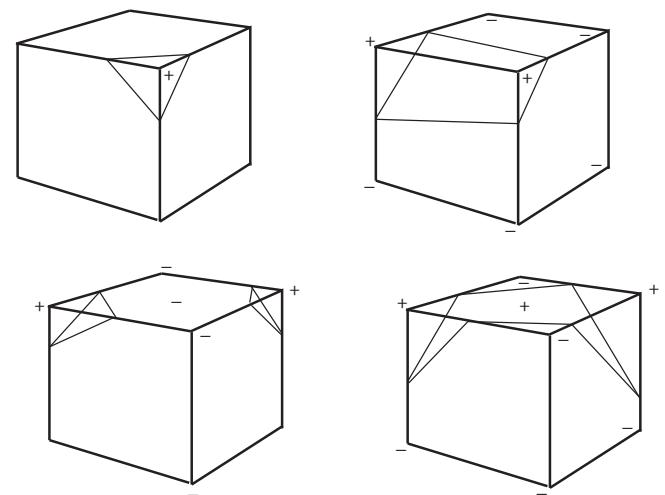
22.3.4 Sampling Problems

Ambiguities occur when opposite corners of a face (or the cube) have the same sign and the other pair of vertices on the face have the opposite sign (see Figure 22.14). A sample taken in the center of the face will give a clue as to whether the cube represents the meeting of two surfaces or a saddle. It should be made clear that a spatial grid stores a sample of the implicit function at every vertex. If the function happens to vary considerably within a cell, the polygonal representation will not show such variations (see Figure 22.15). The surface cannot be resolved by sampling alone unless something is known about the curvature of the surface. A good discussion of this topic appears in (Kalra & Barr, 1989).

This ambiguity problem (not the undersampling problem) is avoided by subdividing the cubic cell into tetrahedra. The tetrahedra can then be polygonized unambiguously. Since there are four vertices in each tetrahedron, a table of 16 entries will provide the correct triangle information. The disadvantage is that approximately twice the number of polygons will be generated.

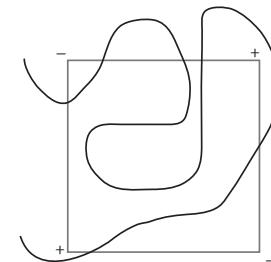
Subdividing a Cube

Without requiring additional cell vertices, a cube may be decomposed into five or six tetrahedra as shown in Figure 22.16. These decompositions introduce diagonals on the cube faces, and to maintain a consistent diagonal direction between



表面内部 (+) 和外部 (-) 顶点的示例。注意额外的样本提供了一个线索，以避免模棱两可的情况。

22.3.4 抽样问题



立方体太大无法在隐式函数中捕获小变异。

当面（或立方体）的相对角具有相同的符号，而面上的另一对顶点具有相反的符号时，就会出现歧义（见图22.14）。在脸部中心拍摄的样本将提供关于立方体是代表两个表面还是马鞍的会议的线索。应该明确的是，空间网格在每个顶点存储隐式函数的样本。如果函数恰好在一个单元格内变化很大，多边形表示将不会显示这种变化（见图22.15）。除非知道表面的曲率，否则无法单独通过采样来解析表面。关于这个主题的一个很好的讨论出现在 (Kalra & Barr, 1989)。

通过将立方单元细分为四面体来避免这种模糊问题（而不是欠采样问题）。然后可以明确地对四面体进行多边形化。由于每个四面体中有四个顶点，因此16个条目的表格将提供正确的三角形信息。缺点是将生成大约两倍数量的多边形。

细分立方体

不需要额外的单元顶点，一个立方体可以分解成五个或六个四面体，如图22.16所示。这些分解在立方体面上引入对角线，并在两者之间保持一致的对角线方向

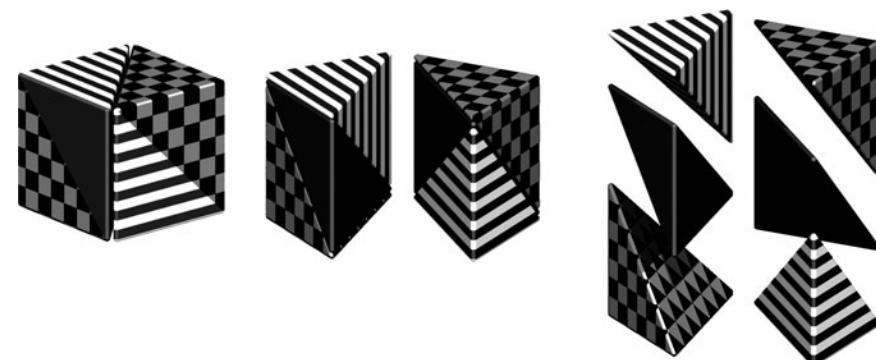


Figure 22.16. Decomposing a cube into six tetrahedra. *Image courtesy Erwin DeGroot.*

neighbors, the six decomposition is preferable. The introduction of diagonal edges produces a higher-resolution surface than replacing each cube directly with triangles. The decomposition into tetrahedra and the replacement of the tetrahedra with triangles are fast, table-driven algorithms, which produce topologically consistent meshes.

22.3.5 Cell Polygonization

Two obvious problems emerge from the use of uniform space subdivision. The size of triangles output by this algorithm do not adapt to the curvature of the surface and a further sample is required to solve the ambiguities, in which cubic cells are replaced by polygons. A space subdivision algorithm based on an octree was developed by Bloomenthal (Bloomenthal, 1988), which does adapt to the curvature of the surface. Cells are subdivided into eight octants and cracks are avoided by using a restricted octree scheme, i.e., neighboring cells cannot differ by more than one level of subdivision. This indeed reduces the number of polygons generated, but full advantage of large cells can only be taken if the flat regions of the surface happen to fall entirely within the appropriate octants. The algorithm proves in practice to be considerably slower than the uniform voxel algorithm and is more complicated to implement.

22.4 More on Blending

Section 22.1 showed that blending can be made to occur when field values are summed. Ricci, in his landmark paper (Ricci, 1973), describes super-elliptic

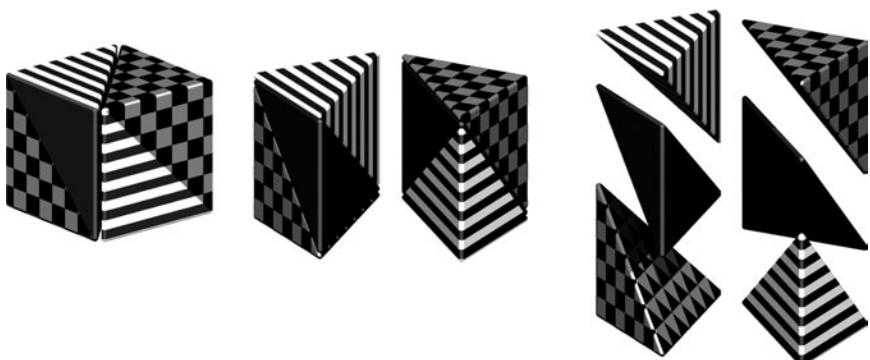


图22.16。把一个立方体分解成六个四面体。图片由ErwinDeGroot提供。

邻，优选六次分解。对角边的引入比直接用三角形替换每个立方体产生更高分辨率的表面。分解为四面体以及用三角形替换四面体是快速的、由表格驱动的算法，可生成拓扑一致的网格。

22.3.5 Cell Polygonization

使用均匀空间细分产生了两个明显的问题。该算法输出的三角形的大小不适合曲面的曲率，需要进一步的样本来解决模糊性，其中立方单元被多边形取代。由Bloomenthal (Bloomenthal, 1988) 开发了一种基于八叉树的空间细分算法，它确实适应了曲面的曲率。细胞被细分为八个八叉，并通过使用限制八叉树方案避免裂缝，即相邻细胞不能相差超过一个级别的细分。这确实减少了生成的多边形的数量，但只有当表面的平坦区域恰好完全落在适当的八边形内时，才能充分利用大单元格。该算法在实践中证明比均匀体素算法慢得多，并且实现起来更复杂。

22.4 更多关于混合

第22.1节表明 当字段值求和时 可以进行混合.利玛窦在其具有里程碑意义的论文 (Ricci, 1973) 中描述了超椭圆

blending. Given two functions F_A and F_B , previously we simply found the implicit value as $F_{\text{total}} = F_A + F_B$. We can denote this more general blending operator as $A \diamond B$. The Ricci blend is defined as:

$$f_{A \diamond B} = (f_A^n + f_B^n)^{\frac{1}{n}}. \quad (22.4)$$

It is interesting to point out the following properties:

$$\begin{aligned}\lim_{n \rightarrow +\infty} (f_A^n + f_B^n)^{\frac{1}{n}} &= \max(f_A, f_B), \\ \lim_{n \rightarrow -\infty} (f_A^n + f_B^n)^{\frac{1}{n}} &= \min(f_A, f_B).\end{aligned}$$

Moreover, this generalized blending is associative, i.e., $f_{(A \diamond B) \diamond C} = f_{A \diamond (B \diamond C)}$. The standard blending operator $+$ proves to be a special case of the super-elliptic blend with $n = 1$. When n varies from 1 to infinity, it creates a set of blends interpolating between blending $A + B$ and union $A \cup B$ (see Figure 22.17). Figure 22.27 shows the nodes to be binary or unary; in fact the binary nodes can easily be extended using the above formulation to n-ary nodes.

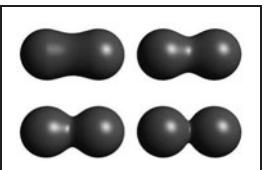


Figure 22.17. By varying n , the Ricci blend may be made to change smoothly from blend to union. *Image courtesy Erwin DeGroot.*

The power of Ricci's operators is that they are *closed* under the operations on the space of all possible implicit volumes, meaning that an application of an operator simply produces another scalar field defining another implicit volume. This new field can be composed with other fields, again using Ricci's operators. Equation (22.4) will always produce the exact union of two implicit volumes, regardless of how complex they are. Compared with the difficulties involved in applying boolean CSG operations to B-rep surfaces, solid modeling with implicit volumes is incredibly simple.

Following Pasko's functional representation (A. Pasko et al., 1995), another generalized blending function may be defined:

$$f_{A \diamond B} = \left(f_A + f_B + \alpha \sqrt{f_A^2 + f_B^2} \right) (f_A^2 + f_B^2)^{\frac{n}{2}}.$$

When $\alpha \in [-1, 1]$ varies from -1 to 1 , it creates a set of blends interpolating the union and the intersection operators. However, this operator is no longer associative which is incompatible with the definition of n-ary operators.

22.5 Constructive Solid Geometry

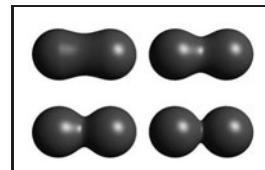
Implicit models are frequently termed *implicit surfaces*; however, they are inherently volume models and useful for *solid modeling* operations. Ricci introduced a

混合。给定两个函数 F_A 和 F_B , 以前我们简单地找到隐式值为 $F_{\text{total}} = F_A + F_B$ 。我们可以将这种更通用的混合运算符表示为 $A \in B$ 。利玛窦共混物定义为：

$$f_{A \diamond B} = (f_A^n + f_B^n)^{\frac{1}{n}}. \quad (22.4)$$

有趣的是指出以下属性：

$$\begin{aligned}\lim_{n \rightarrow +\infty} (f_A^n + f_B^n)^{\frac{1}{n}} &= \max(f_A, f_B), \\ \lim_{n \rightarrow -\infty} (f_A^n + f_B^n)^{\frac{1}{n}} &= \min(f_A, f_B).\end{aligned}$$



通过变化 n , 利玛窦混合可以顺利地从混合到联合。
图片由ErwinDeGroot提供。

此外, 这种广义混合是关联的, 即 $f(A \in B) \in C = fA \in (B \in C)$ 。
标准混合算子+证明是 $n=1$ 的超椭圆混合的特例。当 n 从 1 变化到无穷大时, 它会创建一组混合, 在混合 $A+B$ 和联合 $A \in B$ 之间进行插值 (见图 22.17)。图 22.27 显示了二进制或一元的节点;事实上, 使用上述公式可以很容易地将二进制节点扩展到 n -ary 节点。

Ricci 运算符的强大之处在于它们在所有可能的隐式卷的空间上的操作下被关闭, 这意味着一个运算符的应用程序只是产生另一个定义另一个隐式卷的标量字段。这个新字段可以与其他字段组成, 再次使用 Ricci 的运算符。方程 (22.4) 将始终产生两个隐含体积的精确联合, 无论它们有多复杂。与将布尔CSG 操作应用于 B-rep 表面所涉及的困难相比, 具有隐式体积的实体建模非常简单。

遵循 Pasko 的功能表示 (A. Pasko et al., 1995), 可以定义另一个广义混合函数:

$$f_{A \diamond B} = \left(f_A + f_B + \alpha \sqrt{f_A^2 + f_B^2} \right) (f_A^2 + f_B^2)^{\frac{n}{2}}.$$

当 $\alpha \in [-1, 1]$ 从 -1 到 1 变化时, 它会创建一组混合插值联合和交集运算符。但是, 此运算符不再是关联的, 这与 n -ary 运算符的定义不兼容。

22.5 构造实体几何

隐式模型通常被称为隐式曲面; 然而, 它们本质上是体积模型, 对实体建模操作很有用。利玛窦介绍了一个

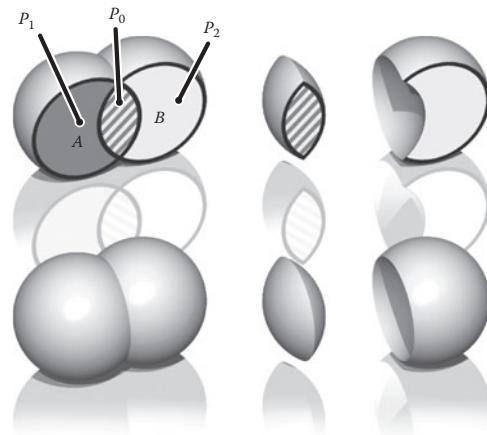


Figure 22.18. Ricci operators for CSG. Image courtesy Erwin DeGroot.

constructive geometry for defining complex shapes from operations such as union, intersection, difference, and blend upon primitives (Ricci, 1973). The surface was considered as the boundary between the half spaces $f(\mathbf{p}) < 1$, defining the inside, and $f(\mathbf{p}) > 1$ defining the outside. This initial approach to solid modeling evolved into *constructive solid geometry* or CSG (Ricci, 1973; Requicha, 1980). CSG is typically evaluated bottom-up according to a binary tree, with low-degree polynomial primitives as the leaf nodes and internal nodes representing Boolean set operations. These methods are readily adapted for use in implicit modeling, and in the case of skeletal implicit surfaces, the Boolean set operations union \cup_{\max} , intersection \cap_{\min} and difference $\setminus_{\min\max}$ are defined as follows (Wyvill, Galin, & Guy, 1999):

$$\begin{aligned}\cup_{\max} \quad f &= \max_{i=0}^{k-1} (f_i), \\ \cap_{\min} \quad f &= \min_{i=0}^{k-1} (f_i), \\ \setminus_{\min\max} \quad f &= \min \left(f_0, 2 * \text{iso} - \max_{j=1}^{k-1} (f_j) \right).\end{aligned}\quad (22.5)$$

The Ricci operators are illustrated in Figure 22.18 for point primitives A and B . For union (bottom left) the field at all points inside the union will be the greater of $f_A()$ and $f_B()$. For intersection (center), points in the region marked as P_1 will have value $\min(f_A(P_1), f_B(P_1)) = 0$, since the contribution of B will be zero outside of its range of influence. Similarly, for the region marked as P_2 , (influence of A is zero, i.e., the minimum) leaving only the intersection region with positive values. Difference works similarly using the iso-value in the three

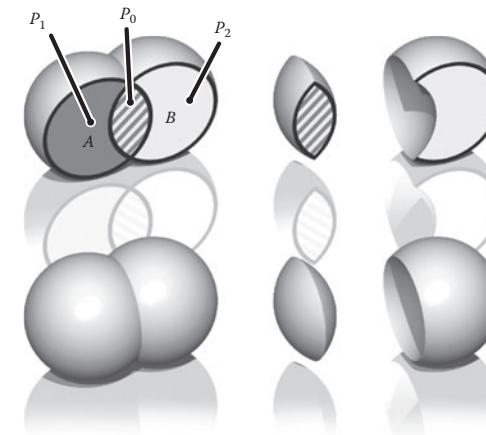


图22.18。CSG的利玛窦运营商。图片由ErwinDeGroot提供。

构造几何，用于从基元上的联合、交集、差异和混合等操作中定义复杂形状 (Ricci, 1973)。表面被认为是定义内部和定义外部的 $f(\mathbf{p}) > 1$ 之间的边界。这种最初的实体建模方法演变为建设性实体几何或CSG (Ricci, 1973; Requicha, 1980)。CSG通常根据二叉树进行自下而上的评估，以低度多项式基元作为叶节点和表示布尔集合操作的内部节点。这些方法很容易适用于隐式建模，在骨架隐式曲面的情况下，布尔集合运算 \cup_{\max} , \cap_{\min} 和 $\setminus_{\min\max}$ 定义如下 (Wyvill, Galin, & Guy, 1999)：

$$\begin{aligned}\cup_{\max} \quad f &= \max_{i=0}^{k-1} (f_i), \\ \cap_{\min} \quad f &= \min_{i=0}^{k-1} (f_i), \\ \setminus_{\min\max} \quad f &= \min \left(f_0, 2 * \text{iso} - \max_{j=1}^{k-1} (f_j) \right).\end{aligned}\quad (22.5)$$

图22.18为点基元A和B说明了Ricci运算符。对于union（左下），union内所有点的字段将是 $f_A()$ 和 $f_B()$ 中较大的一个。对于交点（中心），标记为 P_1 的区域中的点将具有值 $\min(f_A(P_1), f_B(P_1)) = 0$ ，因为 B 的贡献将在其影响范围之外为零。类似地，对于标记为 P_2 的区域，(A 的影响为零，即最小) 仅留下具有正值的交叉点区域。差异的工作原理类似使用三个iso值

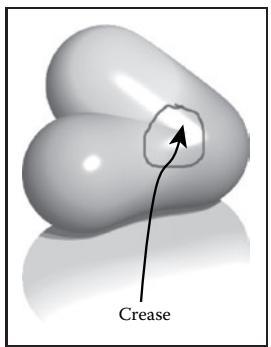


Figure 22.19. Two point primitives on the left are connected by the Ricci union. A third primitive is blended to the result, creating an unwanted crease in the field. *Image courtesy Erwin DeGroot.*

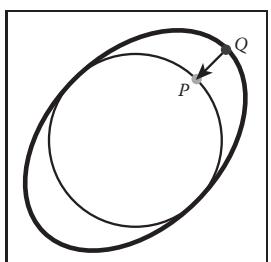
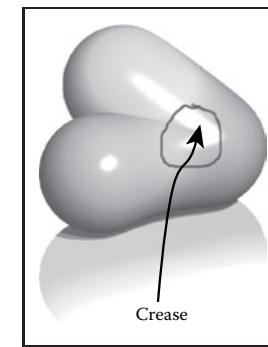


Figure 22.20. Point Q returns the field value for point P .

The ability to distort the shape of a surface by warping the space in its neighborhood is a useful modeling tool. A warp is a continuous function $w(x, y, z)$ that maps \mathbb{R}^3 onto \mathbb{R}^3 . Sederberg provides a good analogy for warping when describing free form deformations (Sederberg & Parry, 1986). He suggests that the warped space can be likened to a clear, flexible, plastic parallelepiped in which the objects to be warped are embedded. A warped element may be defined by simply applying some warp function $w(\mathbf{p})$ to the implicit equation:

$$f_i(x, y, z) = g_i \circ d_i \circ w_i(x, y, z). \quad (22.6)$$

A warped element may be fully characterized by the distance to its skeleton $d_i(x, y, z)$, its fall-off filter function $g_i(r)$, and eventually its warp function $w_i(x, y, z)$. To render or perform operations on an implicit surface, the implicit



标记区域(P_i)如下:

$$\begin{aligned} f(P_0) &= \min(f_B(P_0), 2 \div \text{iso} - f_A(P_0)) \\ &= \min([1, 2 \div \text{iso}], [2 \div \text{iso} - 1, \text{iso}]) \\ &= [2 \div \text{iso} - 1, \text{iso}] < \text{iso} \\ f(P_1) &= \min(f_B(P_1), 2 \div \text{iso} - f_A(P_1)) \\ &= \min([0, \text{iso}], [2 \div \text{iso} - 1, \text{iso}]) < \text{iso} \\ f(P_2) &= \min(f_B(P_2), 2 \div \text{iso} - f_A(P_2)) \\ &= \min([\text{iso}, 1], [\text{iso}, 2 \div \text{iso}]) \geq \text{iso} \end{aligned}$$

CSG运算符创建折痕，即C1不连续点。例如，min()运算符（方程 (22.5)）在 $f_1(p) = f_2(p)$ 的所有点上创建C1不连续性。当应用于两个球体时，由这个联合算子产生的不连续性导致表面上的折痕，如图22.18所示，这是所需的结果。不连续性不幸地延伸到表面之外的场中，这在该图像中是不可见的。如果混合然后应用于联合的结果，则场中的C1-不连续平面会产生阴影不连续性（图22.19）。

左边的两个点基元由利玛窦联合连接.第三个基元被混合到结果中，在字段中创建一个不需要的折痕。我的年龄礼貌ErwinDeGroot.

该问题可以在一定程度上避免 (G.Pasko, Pasko, Ikeda, & Kunii, 2002)，并且已经开发了CSG运营商，除了 $f_1(p) = f_2(p) = \text{iso}$ (Barthe, Dodgson, Sabin, Wyvill, & Gaildrat, 2003) 之外的所有点都是C1。

22.6 Warping

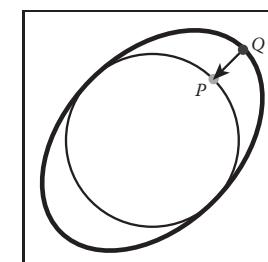


图22.20. 点Q重新为点P转场值。

22.6 Warping

通过扭曲表面附近的空间来扭曲表面形状的能力是一个有用的建模工具。经线是将 \mathbb{R}^3 映射到 \mathbb{R}^3 的连续函数 $w(x, y, z)$ 。在描述自由形式变形时，Sederberg为翘曲提供了一个很好的类比 (Sederberg & Parry, 1986)。他建议，扭曲的空间可以比作一个清晰，灵活，塑料平行六面体，其中要扭曲的物体被嵌入其中。扭曲的元素可以通过简单地将一些扭曲函数 $w(\mathbf{p})$ 应用于隐式方程来定义：

$$f_i(x, y, z) = g_i \circ d_i \circ w_i(x, y, z). \quad (22.6)$$

翘曲元件可以通过到其骨架的距离 $d_i(x, y, z)$ ，其脱落滤波器函数 $g_i(r)$ 以及最终其翘曲函数 $w_i(x, y, z)$ 来充分表征。要在隐式表面上呈现或执行操作，隐式



value of many points $f(P)$ must be found. First, P is transformed by the warp function to some new point Q , and $f(Q)$ is returned in place of $f(P)$. In Figure 22.20, instead of returning the implicit value of some point $f(Q)$, the value for $f(P)$ is returned. In this case, the iso-value is returned and the implicit surface (curve in 2D) passes through Q instead of P . Thus, the circle is warped into an ellipse.

Barr introduced the notion of global and local deformations using the operations of *twist*, *taper*, and *bend* applied to parametric surfaces (Barr, 1984). The deformations can be nested to produce models such as the one shown in Figure 22.27. Conceptually, these are easy to apply to an implicit surface, as indicated in Equation (22.6).

Note that the normal cannot be calculated in a similar manner to warping a point. This problem is similar to the problem outlined in Section 13.2 on instancing. In this case, the normal can most easily be approximated using Equation (22.3.3) although the use of the Jacobian, as suggested in (Barr, 1984), yields precise results. The Barr warps are described in the following sections.

22.6.1 Twist

In this example, the twist is around the z -axis by θ (see Figure 22.21) for three blended implicit cylinders with a twist warp applied to them.

The twist around z is expressed as

$$w(x, y, z) = \begin{cases} x * \cos(\theta(z)) - y * \sin(\theta(z)) \\ x * \sin(\theta(z)) + y * \cos(\theta(z)) \\ z \end{cases}.$$

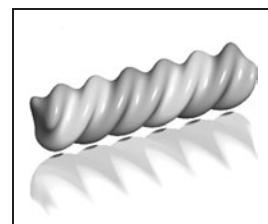


Figure 22.21. Three blended implicit cylinders twisted together. Image courtesy Erwin DeGroot.

22.6.2 Taper

Taper is applied along one major axis. A linear taper has proved to be the most useful although quadratic and cubic tapers are easily implemented. For example, a linear taper along the y -axis involves changing both x - and z -coordinates. (See Figure 22.22.) A linear scale is applied to y between y_{\max} and y_{\min} :

$$s(y) = \frac{y_{\max} - y}{y_{\max} - y_{\min}} \quad w(x, y, z) = \begin{cases} s(y)x \\ y \\ s(y)z \end{cases}$$

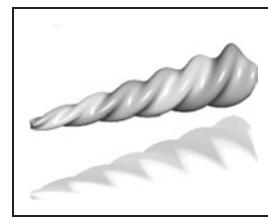


Figure 22.22. Three blended implicit cylinders, twisted then tapered. Image courtesy Erwin DeGroot.



必须找到许多点 $f(P)$ 的值。首先， P 由 $warp$ 函数变换到一些新的点 Q ，并且 $f(Q)$ 替代 $f(P)$ 返回。在图 22.20 中，不是返回某个点 $f(Q)$ 的隐式值，而是返回用于 $f(P)$ 的值。在这种情况下，返回 iso 值，隐式曲面（2D 中的曲线）通过 Q 而不是 P 。因此，圆被翘曲成椭圆。

Barr 引入了全局和局部变形的概念，使用扭曲、锥度和弯曲的变形应用于参数曲面（Barr, 1984）。变形可以嵌套以产生诸如图 22.27 所示的模型。从概念上讲，这些很容易应用于隐式表面，如方程 (22.6) 所示。

请注意，法线的计算方法与变形点的计算方法类似。此问题类似于第 13.2 节中概述的问题。在这种情况下，法线可以很容易地使用 Equation (22.3.3) 来近似，尽管如 (Barr, 1984) 中所建议的使用雅可比，可以产生精确的结果。Barr 翘曲将在以下各节中描述。

22.6.1 Twist

在这个例子中，对于三个混合的隐式圆柱体，扭曲是围绕 z 轴的 θ (见图 22.21)，并且在它们上施加了扭曲经纱。围绕 z 的扭曲表示为

$$w(x, y, z) = \begin{cases} x * \cos(\theta(z)) - y * \sin(\theta(z)) \\ x * \sin(\theta(z)) + y * \cos(\theta(z)) \\ z \end{cases}.$$

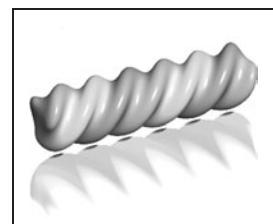


Figure 22.21. Three blended implicit cylinders在一起。图片由 Erwin DeGroot 提供。

22.6.2 Taper

锥度沿一个长轴施加。线性锥度被证明是最有用的，尽管二次和立方锥度很容易实现。例如，沿 y 轴的线性锥度涉及改变 x 和 z 坐标。(见图 22.22。) 在 y_{\max} 和 y_{\min} 之间对 y 应用线性刻度：

$$s(y) = \frac{y_{\max} - y}{y_{\max} - y_{\min}} \quad w(x, y, z) = \begin{cases} s(y)x \\ y \\ s(y)z \end{cases}$$

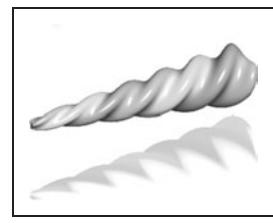


Figure 22.22. Three隐式圆柱体，扭曲然后锥形。图片由 Erwin DeGroot 提供。

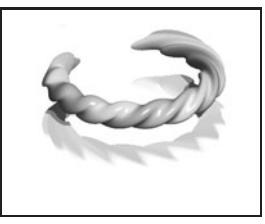


Figure 22.23. Three blended implicit cylinders, twisted together, tapered and bent. *Image courtesy Erwin DeGroot.*

22.6.3 Bend

Bend is also applied along one major axis. (See Figure 22.23.) For the bend example below, the bending rate is k measured in radians per unit length, the axis of the bend is $(x_0, 1/k)$, and the angle θ is defined as $(x - x_0) * k$. The bend around z is

$$w(x, y, z) = \begin{cases} -\sin(\theta) * (y - 1/k) + x_0 \\ \cos(\theta) * (y - 1/k) + 1/k \\ z \end{cases}$$

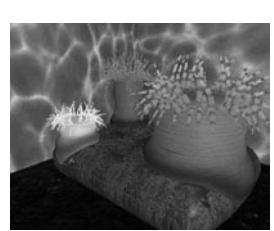


Figure 22.24. Sea anemone deforms to implicit rock. *Image courtesy Mai Nur and X. Liang.*

22.7 Precise Contact Modeling

Precise contact modeling (PCM) is a method of deforming implicit surface primitives in contact situations while maintaining a precise contact surface with C^1 continuity (Gascuel, 1993). PCM is important in that it is a simple and automatic way of showing how a model can react to its environment. This cannot be so easily done with non-implicit methods (see Figure 22.24).

PCM is implemented by the inclusion of a deforming function $s(p)$ that modifies the field value returned for each point. For each pair of objects, collision is first detected using a bounding-box test. Once it is established that a collision is likely, PCM is applied. A local, geometric deformation term s_i is computed and added to the implicit function f_i . The volume of the colliding objects is divided into an interpenetration region and a deformation region. The result of applying s_i is that the interpenetration region is compressed so that contact is maintained without interpenetration occurring (see Figure 22.25). The effect of s_i is attenuated to zero within the propagation region so that the volume outside of the two regions is not deformed.

Given two skeletal elements generating fields $f_1(p)$ and $f_2(p)$, the surface around each one is calculated as

$$\begin{aligned} f_1(p) + s_1(p) &= 0, \\ f_2(p) + s_2(p) &= 0. \end{aligned}$$

We need to generate a surface common to both elements (dotted line in Figure 22.25), i.e., where they share a solution in the interpenetration region for some p in that region:

$$\begin{aligned} s_1(p) - f_1(p) &= \text{iso}, \\ s_2(p) - f_2(p) &= \text{iso}. \end{aligned} \tag{22.7}$$

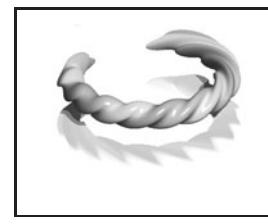


Figure 22.23. Three隐式圆柱体，扭曲在一起，锥形和弯曲。图片由Erwin提供DeGroot.

22.6.3 Bend

弯曲也沿一个长轴施加。（见图22.23。）对于下面的弯曲示例，弯曲速率以单位长度的弧度为k，弯曲的轴为 $(x_0, 1/k)$ ，角度θ定义为 $(x - x_0) / k$ 。绕z的弯曲是

$$w(x, y, z) = \begin{cases} -\sin(\theta) * (y - 1/k) + x_0 \\ \cos(\theta) * (y - 1/k) + 1/k \\ z \end{cases}$$

22.7 精确的接触建模

精确接触建模（PCM）是一种在接触情况下变形隐式表面基元的方法，同时保持与C1的精确接触表面

连续性（Gascuel, 1993）。PCM非常重要，因为它是一种简单而自动的方式来显示模型如何对其环境做出反应。用非隐式方法不能这么容易地做到这一点（见图22.24）。

PCM是通过包含一个变形函数s(p)来实现的，该函数对每个点返回的字段值进行修改。对于每对对象，碰撞是

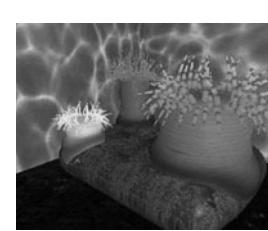


图22.24。海葵变形为隐岩。图片由Mai Nur和X.Liang提供。

首先使用边界框测试检测到。一旦确定可能发生碰撞，就应用PCM。计算局部几何变形项 s_i 并将其添加到隐式函数 f_i 中。碰撞物体的体积分为互穿区域和变形区域。应用 s_i 的结果是插入区域被压缩，以便保持接触而不发生插入（见图22.25）。 s_i 的效应在传播区域内衰减到零，使得两个区域之外的体积不变形。

给定产生场 $f_1(p)$ 和 $f_2(p)$ 的两个骨架单元，每个骨架单元周围的表面计算如下

$$\begin{aligned} f_1(p) + s_1(p) &= 0, \\ f_2(p) + s_2(p) &= 0. \end{aligned}$$

我们需要生成一个两个元素共同的表面（图22.25中的虚线），即它们在插入区域中为该区域中的某些 p 共享一个解：

$$\begin{aligned} s_1(p) - f_1(p) &= \text{iso}, \\ s_2(p) - f_2(p) &= \text{iso}. \end{aligned} \tag{22.7}$$

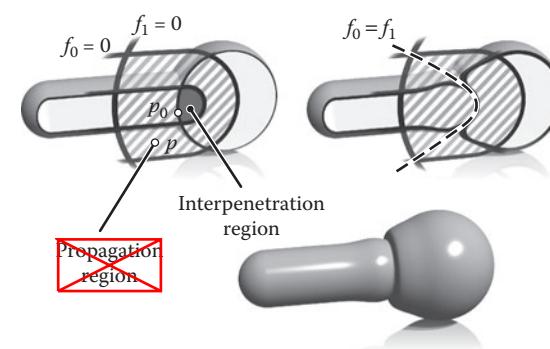


Figure 22.25. A 2D slice through objects in collision showing the various regions and PCM deformation. *Image courtesy Erwin DeGroot.*

Intuitively, the deeper within object 1 that object 2 penetrates, the higher the implicit value of object 1 and thus the more that object 2 will be compressed.

The function, s_i is defined to produce a smooth junction at the boundary of the interpenetration region, in other words where $s_i = 0$ but its derivative is greater than zero. From here to the boundary of the propagation region, s_i is used to attenuate the propagation to zero. The *nearest* point on the interpenetration region boundary p_0 is found by following the gradient.

Within the propagation region $s_i(p) = h_i(r)$, where $p = (x, y, z)$ is the point whose implicit value is being calculated and $r = \|p - p_0\|$ (see Figure 22.26). The value of r_i , set by the user, defines the size of the propagation region; no deformation occurs beyond this region. To control how much the objects inflate in the propagation region, the user provides a value for the parameter α . The maximum

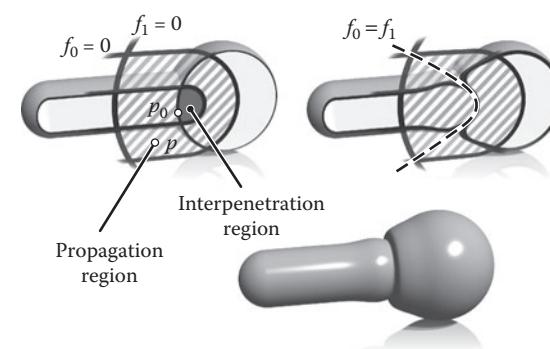


Figure 22.25. A 2D slice through objects in collision showing the various regions and PCM deformation. *Image courtesy Erwin DeGroot.*

Intuitively, the deeper within object 1 that object 2 penetrates, the higher the implicit value of object 1 and thus the more that object 2 will be compressed.

The function, s_i is defined to produce a smooth junction at the boundary of the interpenetration region, in other words where $s_i = 0$ but its derivative is greater than zero. From here to the boundary of the propagation region, s_i is used to attenuate the propagation to zero. The *nearest* point on the interpenetration region boundary p_0 is found by following the gradient.

Within the propagation region $s_i(p) = h_i(r)$, where $p = (x, y, z)$ is the point whose implicit value is being calculated and $r = \|p - p_0\|$ (see Figure 22.26). The value of r_i , set by the user, defines the size of the propagation region; no deformation occurs beyond this region. To control how much the objects inflate in the propagation region, the user provides a value for the parameter α . The maximum

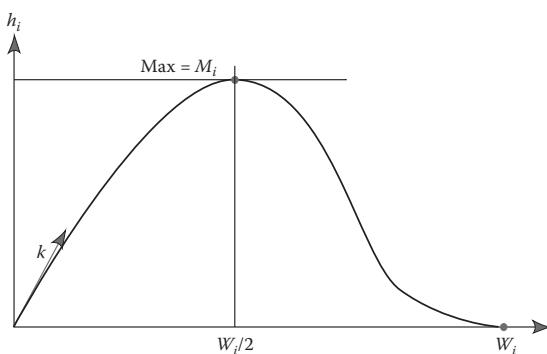


Figure 22.26. The function, $h_i(r)$ is the value of the deformation function w_i in the propagation region.

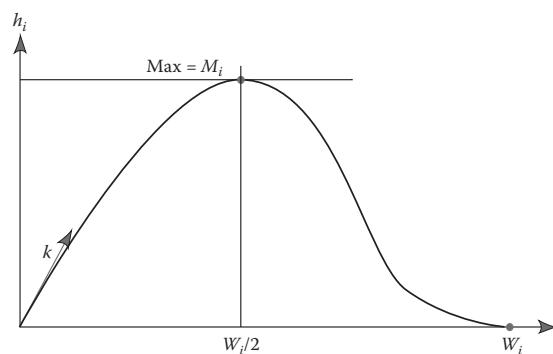


Figure 22.26. The function, $h_i(r)$ is the value of the deformation function w_i in the propagation region.

value of h_i is M_i . The current minimum of s_i is negative in the interpenetration region and is given as $s_{i\min}$, where $M_i = -\alpha_i s_{i\min}$. Thus an object will be compressed in the interpenetration region and will inflate in the propagation region. The equation for h_i is formed in two parts by two cubic polynomials that are designed to join at $r = r_i/2$, where the slope is zero:

$$\begin{aligned} c &= \frac{4(w_i k - 4M_i)}{w_i^3}, \\ d &= \frac{4(3M_i - w_i k)}{w_i^2}, \\ h_i(r) &= cr^3 + dr^2 + kr \quad \text{if } r \in [0, w_i/2], \\ h_i(r) &= \frac{4M_i}{w_i^3}(r - w_i)^2(4r - w_i)^3 \quad \text{if } r \in [w_i/2, w_i]. \end{aligned}$$

It is desirable that we have C^1 -continuity as we move from the interpenetration to the propagation region. Thus, $h'_i(0) = k$ in Figure 22.26, is the directional derivative of s_i at the junction (marked as p_0 in Figure 22.25). As indicated in Equation (22.7), $s_i = -f_i$ in the interpenetration region, thus:

$$k = \|\nabla(f_i, p_0)\|$$

PCM is only an approximation to a properly deformed surface, but it is an attractive algorithm due to its simplicity.

22.8 The BlobTree

The *BlobTree* is a method that employs a tree structure that extended the CSG tree to include various blending operations using skeletal primitives (Wyvill et al., 1999). A system with similar capabilities, the *Hyperfun* project, used a specialized language to describe F-rep objects (Adzhiev et al., 1999).

In the BlobTree system, models are defined by expressions that combine implicit primitives and the operators \cup (union), \cap (intersection), $-$ (difference), $+$ (blend), \diamond (super-elliptic blend), and w (warp). The BlobTree is not only the data structure built from these expressions but also a way of visualizing the structure of the models. The operators listed above are binary with the exception of warp, which is a unary operator. In general it is more efficient to use n-ary rather than binary operators. The BlobTree incorporates affine transformations as nodes so that it is also a scene graph and primitives (e.g., skeletons) form the leaf nodes.

h_i 的值是 M_i 。在互穿区域中， s_i 的电流最小值为负，并以 $s_{i\min}$ 给出，其中 $M_i = -\alpha_i s_{i\min}$ 。因此，物体将在传播区域中被压缩，并在传播区域中膨胀。 H_i 的方程由两个三次多项式组成，这些多项式被设计为在 $r=r_i/2$ 处连接，其中斜率为零：

$$\begin{aligned} c &= \frac{4(w_i k - 4M_i)}{w_i^3}, \\ d &= \frac{4(3M_i - w_i k)}{w_i^2}, \\ h_i(r) &= cr^3 + dr^2 + kr \quad \text{if } r \in [0, w_i/2], \\ h_i(r) &= \frac{4M_i}{w_i^3}(r - w_i)^2(4r - w_i)^3 \quad \text{if } r \in [w_i/2, w_i]. \end{aligned}$$

当我们从interpenetration移动到传播区域时我们希望有 C^1 -连续性.因此,图22.26中的 $i'(0) = k$, 是 s_i 在交界处的方向导数 (在图22.25中标记为 p_0)。如等式 (22.7) 所示, 在互穿区域中 $s_i = f_i$, 因此:

$$k = \|\nabla(f_i, p_0)\|$$

PCM只是一个适当变形表面的近似值, 但由于其简单性, 它是一个有吸引力的算法。

22.8 The BlobTree

BlobTree是一种使用树结构的方法，该树结构扩展了CSG树，以包括使用骨架基元的各种混合操作(Wyvill et al. 1999).具有类似功能的系统Hyperfun项目使用specialized语言来描述F-rep对象(Adzhiev et al. 1999).

在BlobTree系统中，模型由结合implicit基元和运算符 \cup (union)、 \cap (intersection)、 $-$ (difference)、 $+$ (blend)、 \diamond (super-ellipticblend)和 w (warp)的表达式定义。BlobTree不仅是从这些表达式构建的数据结构，也是可视化模型结构的一种方式。上面列出的运算符是二进制的，但warp除外，warp是一元运算符。一般来说，使用n-ary而不是二元运算符更有效。BlobTree将仿射变换合并为节点，因此它也是一个场景图，基元（例如骨架）构成叶节点。

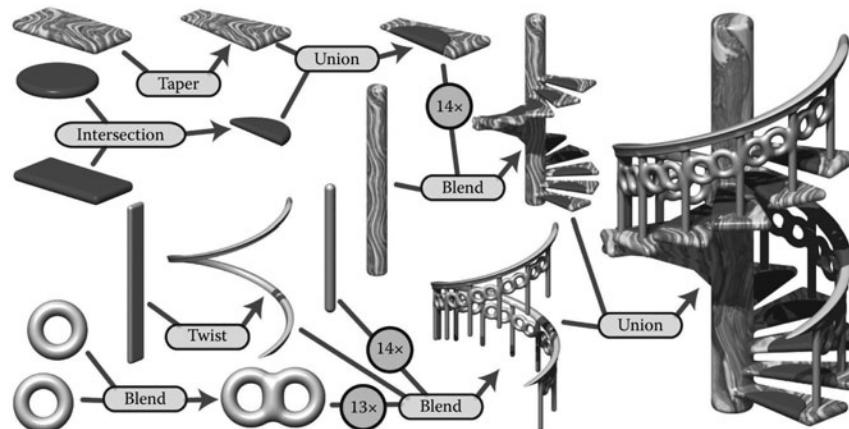


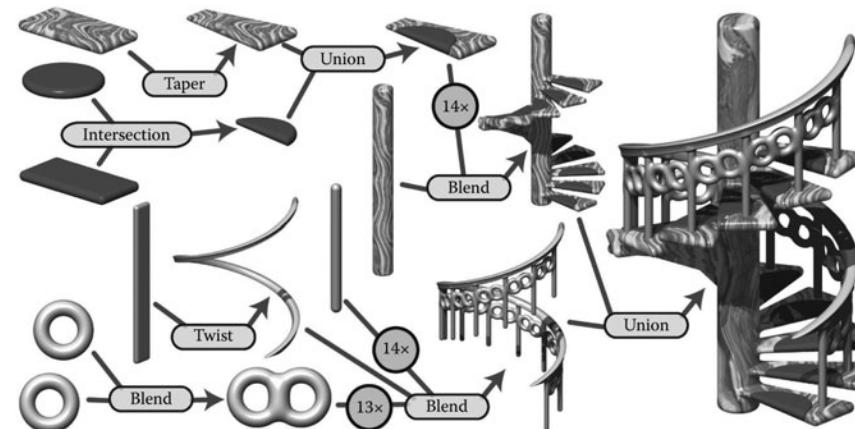
Figure 22.27. BlobTree. The spiral staircase is built from a central textured cylinder to which the stairs and the railing are blended. The railing is comprised of a series of cylinders blended with two circle (torus) primitives, blended together and further blended with a vertical cylinder. The BlobTree is also a scene graph and instancing nodes repeat the various parts transformed by the appropriate matrices. Each stair is made from a tapered polygon primitive (that becomes an offset surface); intersection and union nodes combine the inflated disk with the stair.

22.8.1 Traversing the BlobTree

An example of a BlobTree including the Barr warps and CSG operations is shown in Figure 22.27. Other nodes can include 2D texturing (Schmidt, Grimm, & Wyvill, 2006), precise contact modeling, as well as animation and other attributes. The traversal of the BlobTree is in essence very simple. All that is required to render the object either by polygonizing or ray tracing is to find the implicit value of any point (and the corresponding gradient). This can be done by traversing the tree. Polygonization and ray-tracing algorithms need to evaluate the implicit field function at a large number of points in space. The function $f(\mathcal{N}, M)$ returns the field value for the node \mathcal{N} at the point M , which depends on the type of the node. The values \mathcal{L} and \mathcal{R} indicate that the left or right branch of the tree is explored. The algorithm below is written (for simplicity) as if the tree were binary:

function $f(\mathcal{N}, M)$:

- primitive: $f(M)$;
 - warp: $f(\mathcal{L}(\mathcal{N}), w(M))$;
 - blend: $f(\mathcal{L}(\mathcal{N}), M) + f(\mathcal{R}(\mathcal{N}), M)$;
 - union: $\max(f(\mathcal{L}(\mathcal{N}), M), f(\mathcal{R}(\mathcal{N}), M))$



布洛布特里。螺旋楼梯是由一个中央纹理圆柱建造的，楼梯和栏杆是混合的。栏杆由一系列与两个圆形（环面）基元混合的圆柱体组成，混合在一起并进一步与垂直圆柱体混合。BlobTree也是一个场景图，实例化节点重复由适当的矩阵转换的各个部分。每个楼梯由锥形多边形基元（成为偏移表面）制成；相交和联合节点将膨胀的磁盘与楼梯结合在一起。

22.8.1 遍历BlobTree

包括Barrwarps和CSG操作的BlobTree示例如图22.27所示。其他节点可以包括2D纹理（Schmidt Grimm & Wyvill 2006）、精确接触建模以及动画和其他属性。BlobTree的遍历本质上非常简单。通过多边形化或光线追踪来重新定义对象所需要的只是找到任何点的隐式值（以及相应的梯度）。这可以通过遍历树来完成。多边形化和光线跟踪算法需要评估空间中大量点处的隐式场函数。函数 $f(N, M)$ 返回节点 N 在点 M 处的字段值，该值取决于节点的类型。值L和R表示探索树的左分支或右分支。

下面的算法被写成（为了简单起见），就好像树是二进制的一样：

function $f(\mathcal{N}, M)$:

- primitive: $f(M)$;
 - warp: $f(\mathcal{L}(\mathcal{N}), w(M))$;
 - blend: $f(\mathcal{L}(\mathcal{N}), M) + f(\mathcal{R}(\mathcal{N}), M)$;
 - union: $\max(f(\mathcal{L}(\mathcal{N}), M), f(\mathcal{R}(\mathcal{N}), M))$;



Figure 22.28. “Spiral Stairs.” A complex BlobTree implicit model created in Erwin DeGroot’s BlobTree.net system.

- intersection: $\min(f(\mathcal{L}(\mathcal{N}), M), f(\mathcal{R}(\mathcal{N}), M))$;
- difference: $\min(f(\mathcal{L}(\mathcal{N}), M), -f(\mathcal{R}(\mathcal{N}), M))$.

A complex BlobTree model showing many of the features that have been integrated is shown in Figure 22.28.



Figure 22.29. Outlines are inflated. *Image courtesy Erwin DeGroot.*

22.9 Interactive Implicit Modeling Systems

Early sketch-based modeling systems, such as *Teddy* (Igarashi, Matsuoka, & Tanaka, 1999), used a few drawn strokes from the user to infer a polygonal model in 3-space. With better hardware and improved algorithms, sketch-based implicit modeling systems are now possible. *Shapeshop* uses implicit sweep surfaces to manufacture 3D strokes from 2D user strokes and also preserves the hierarchy of the BlobTree unlike the early systems that produced homogeneous meshes (Schmidt, Wyvill, Sousa, & Jorge, 2005). This enables a user to produce complex models of arbitrary topology from a few simple strokes. The margin figures show a closed drawn stroke (Figure 22.29) inflated into a an implicit sweep and a second sweep (Figure 22.30) that has a smaller sweep object subtracted using CSG.



“螺旋楼梯。”在ErwinDeGroot's中创建的复杂BlobTree隐式模型BlobTree.net系统。

- intersection: $\min(f(\mathcal{L}(\mathcal{N}), M), f(\mathcal{R}(\mathcal{N}), M))$;
- difference: $\min(f(\mathcal{L}(\mathcal{N}), M), -f(\mathcal{R}(\mathcal{N}), M))$.

一个复杂的BlobTree模型显示了许多已集成的特性，如图22.28所示。

22.9 交互式隐式建模系统

早期基于草图的建模系统，如*Teddy* (Igarashi, Matsuoka, & Tanaka, 1999)，使用用户的一些绘制笔划来推断3空间中的多边形模型。有了更好的硬件和改进的算法，基于草图的隐式建模系统现在成为可能。*Shapeshop*使用隐式扫描曲面来



Figure 22.29. 大纲是膨胀。图像礼貌ErwinDeGroot.

从2D用户笔划中制造3D笔划，并且还保留了BlobTree的层次结构，不像产生同质网格的早期系统 (Schmidt, Wyvill, Sousa, & Jorge, 2005)。这使用户能够从几个简单的笔画中生成任意拓扑的复杂模型。边距图显示了一个封闭的绘制笔划 (图22.29) 膨胀成一个隐式扫描和第二个扫描 (图22.30)，该扫描对象使用CSG减去了一个较小的扫描对象。

One of the improvements that made this possible is a caching system that uses a fixed 3D grid of implicit values at each node of the BlobTree representing the values found by traversing the tree below the node (Schmidt, Wyvill, & Galin, 2005). If the value of some point p is required at node N , a value may be returned without traversing the tree below N , provided that part of the tree is unaltered. Instead, an interpolation scheme (see Chapter 9) is used to find a value for p . This scheme speeds up traversal for complex BlobTrees and is one factor in enabling a system to run at interactive rates.

The next generation of implicit modeling systems will exploit hardware and software advances to be able to handle more and more complex hierarchical models interactively. A more complex Shapeshop example is shown in Figure 22.31.



Figure 22.31. “The Next Step.” A complex BlobTree implicit model created interactively in Ryan Schmidt’s Shapeshop by artist Corien Clapwijk (Andusan).

使这成为可能的改进之一是一个缓存系统，它在BlobTree的每个节点处使用隐含值的固定3d网格，表示通过遍历节点下方的树找到的值（Schmidt, Wyvill, & Galin, 2005）。如果在节点N处需要某个点p的值，则可以返回一个值，而不遍历N以下的树，前提是树的一部分是未改变的。相反，使用插值方案（参见第9章）来查找p的值。该方案加快了复杂BlobTree的遍历速度，并且是使系统能够以交互速率运行的一个因素。



Figure 22.30. BlobTree operations can be applied, e.g., CSG difference. *Image courtesy Erwin DeGroot.*

下一代隐式建模系统将利用硬件和软件的进步，以便能够以交互方式处理越来越复杂的分层模型。

一个更复杂的Shapeshop示例如图所示



“下一步。”艺术家CorienClapwijk (Andusan) 在RyanSchmidt的Shapeshop中交互创建的复杂BlobTree隐式模型。

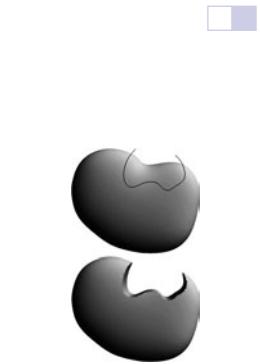


图22.30。可以应用BlobTree操作，例如CSG差异。图片由ErwinDeGroot提供。

Exercises

1. In an implicit surface modeling system the fall-off filter function is defined as

$$f(r) = \begin{cases} 0, & r > R, \\ 1 - r/R, & \text{otherwise,} \end{cases}$$

where R is a constant. A point primitive placed at $(-1, 0)$ and another at $(1, 0)$ are rendered to show the $f = 0.5$ iso-surface. The value R , the distance where the potential due to the point falls to zero in both cases, is 1.5.

Calculate the potential at the point $(0, 0)$ and at $+0.5$ intervals until the point $(2.5, 0)$. Sketch the 0.5 contour and the contour at which the field falls to zero.

2. Why are the ambiguous cases in the polygonization algorithm considered to be a sampling problem?
3. Calculate the error involved in using linear interpolation to estimate the intersection of an implicit surface and a cubic voxel.
4. Design an implicit primitive function using the skeleton of your choice. The function must take as input a point and return an implicit value and also the gradient at that point.

Exercises

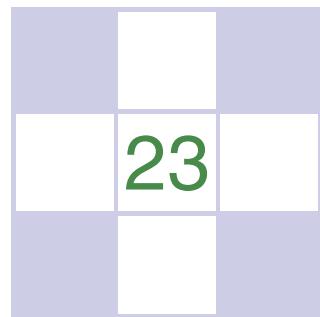
1. 在隐式曲面建模系统中，衰减滤波器函数定义为

$$f(r) = \begin{cases} 0, & r > R, \\ 1 - r/R, & \text{otherwise,} \end{cases}$$

其中R是常数。一个放置在 $(-1, 0)$ 处的点基元和另一个放置在 $(1, 0)$ 处的点基元被渲染以显示 $f=0.5$ iso-表面。值R，在这两种情况下由于点的电位下降到零的距离，是1.5。

计算点 $(0, 0)$ 和 $+0.5$ 处的电位。5个区间，直到点 $(2.5, 0)$ 。勾画 0.5 轮廓和场落到零处的轮廓。

2. 为什么多边形化算法中的模糊情况被认为是抽样问题？
3. 计算使用线性插值来估计隐式表面和立方体素的交点所涉及的误差。
4. 使用您选择的骨架设计隐式基元函数。该函数必须将一个点作为输入，并返回一个隐式值和该点的梯度。

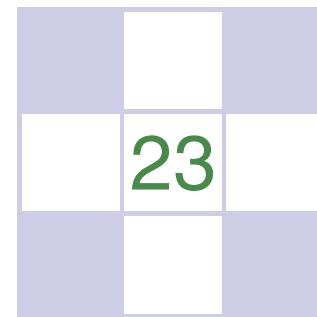


Global Illumination

Many surfaces in the real world receive most or all of their incident light from other reflective surfaces. This is often called *indirect lighting* or *mutual illumination*. For example, the ceilings of most rooms receive little or no illumination directly from luminaires (light-emitting objects). The direct and indirect components of illumination are shown in Figure 23.1.

Although accounting for the interreflection of light between surfaces is straightforward, it is potentially costly because all surfaces may reflect any given surface, resulting in as many as $O(N^2)$ interactions for N surfaces. Because the entire global database of objects may illuminate any given object, accounting for indirect illumination is often called the *global illumination* problem.

There is a rich and complex literature on solving the global illumination problem (e.g., Appel, 1968; Goral, Torrance, Greenberg, & Battaile, 1984; Cook et al.,



全局照明

现实世界中的许多表面接收来自其他反射表面的大部分或全部入射光。这通常被称为间接照明或相互照明。例如，大多数房间的天花板很少或没有直接从灯具（发光物体）获得照明。照明的直接和间接成分如图23.1所示。

虽然解释表面之间光线的互反射是直接的，但它可能会产生成本，因为所有表面都可能反射任何给定的表面，从而导致 N 个表面的 $O(N^2)$ 相互作用。因为对象的整个全局数据库可能照亮任何给定的对象，所以考虑间接照明通常被称为全局照明问题。

关于解决全局照明问题有丰富而复杂的文献（例如，Appel, 1968; Goral, Torrance, Greenberg, & Battaile, 1984; Cook et al.,

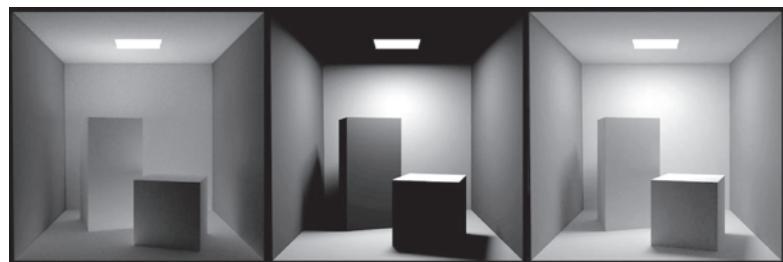
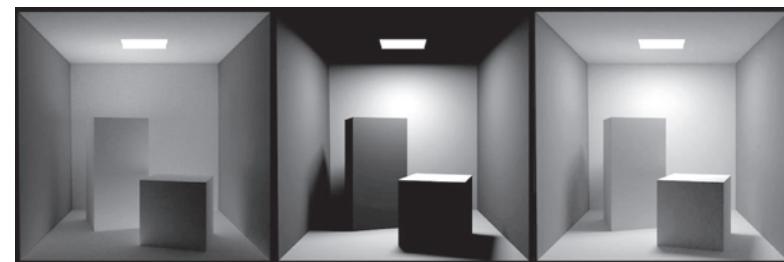


Figure 23.1. In the left and middle images, the indirect and direct lighting, respectively, are separated out. On the right, the sum of both components is shown. Global illumination algorithms account for both the direct and the indirect lighting.



在左图像和中间图像中，分别将间接照明和直接照明分开。在右侧，显示了两个分量的总和。全局照明算法既考虑了直接照明也考虑了间接照明。

1984; Immel et al., 1986; Kajiya, 1986; Malley, 1988). In this chapter, we discuss two algorithms as examples: particle tracing and path tracing. The first is useful for walkthrough applications such as maze games, and as a component of batch rendering. The second is useful for realistic batch rendering. Then we discuss separating out “direct” lighting where light takes exactly once bounce between luminaire and camera.

23.1 Particle Tracing for Lambertian Scenes

Recall the transport equation from Section 18.2:

$$L_s(\mathbf{k}_o) = \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i.$$

The geometry for this equation is shown in Figure 23.2. When the illuminated point is Lambertian, this equation reduces to:

$$L_s = \frac{R}{\pi} \int_{\text{all } \mathbf{k}_i} L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i,$$

where R is the diffuse reflectance. One way to approximate the solution to this equation is to use finite element methods. First, we break the scene into N surfaces each with unknown surface radiance L_i , reflectance R_i , and emitted radiance E_i . This results in the set of N simultaneous linear equations

$$L_i = E_i + \frac{R_i}{\pi} \sum_{j=1}^N k_{ij} L_j,$$

where k_{ij} is a constant related to the original integral representation. We then solve this set of linear equations, and we can render N constant-colored polygons. This finite element approach is often called *radiosity*.

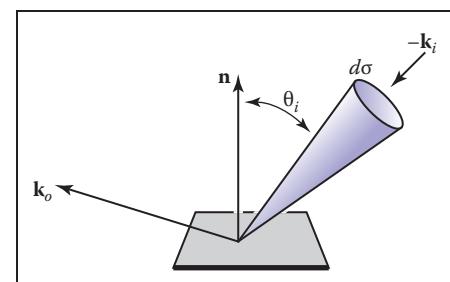


Figure 23.2. The geometry for the transport equation in its directional form.

1984;Immel等人。, 1986;Kajiya, 1986;Malley, 1988)。在本章中, 我们将讨论两种算法作为示例: 粒子跟踪和路径跟踪。第一个是有用的演练应用程序, 如迷宫游戏, 并作为批量渲染的一个组件。第二个对于逼真的批量渲染非常有用。然后, 我们讨论分离出“直接”照明, 其中光线恰好在灯具和相机之间反弹。

23.1 朗伯场景的粒子追踪

回想一下第18.2节中的运输方程:

$$L_s(\mathbf{k}_o) = \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i.$$

该方程的几何形状如图23.2所示。当照明点是朗伯时, 这个方程减少到:

$$L_s = \frac{R}{\pi} \int_{\text{all } \mathbf{k}_i} L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i,$$

其中 R 是漫反射率。近似该方程解的一种方法是使用有限元方法。首先, 我们将场景分解为 N 个表面, 每个表面具有未知的表面辐射度 L_i , 反射率 R_i 和发射辐射度 E_i 。这就得到了 N 个联立线性方程组

$$L_i = E_i + \frac{R_i}{\pi} \sum_{j=1}^N k_{ij} L_j,$$

其中 k_{ij} 是与原始积分表示相关的常数。然后我们求解这组线性方程组, 我们可以渲染 N 个恒定颜色的多边形。这种有限元方法通常被称为辐射度。

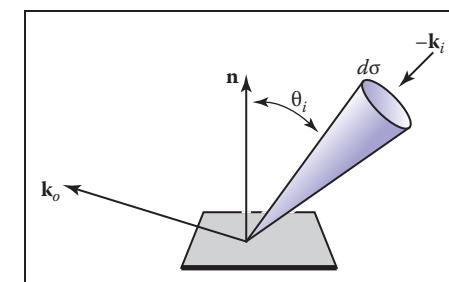


图23.2。运输方程的几何形状以其方向形式。



An alternative method to radiosity is to use a statistical simulation approach by randomly following light “particles” from the luminaire through the environment. This is a type of *particle tracing*. There are many algorithms that use some form of particle tracing; we will discuss a form of particle tracing that deposits light in the textures on triangles. First, we review some basic radiometric relations. The radiance L of a Lambertian surface with area A is directly proportional to the incident power per unit area:

$$L = \frac{\Phi}{\pi A}, \quad (23.1)$$

where Φ is the outgoing power from the surface. Note that in this discussion, all radiometric quantities are either spectral or RGB, depending on the implementation. If the surface has emitted power Φ_e , incident power Φ_i , and reflectance R , then this equation becomes

$$L = \frac{\Phi_e + R\Phi_i}{\pi A}.$$

If we are given a model with Φ_e and R specified for each triangle, we can proceed luminaire by luminaire, firing power in the form of particles from each luminaire. We associate a texture map with each triangle to store accumulated radiance, with all texels initialized to

$$L = \frac{\Phi_e}{\pi A}.$$

If a given triangle has area A and n_t texels, and it is hit by a particle carrying power ϕ , then the radiance of that texel is incremented by

$$\Delta L = \frac{n_t \phi}{\pi A}.$$

Once a particle hits a surface, we increment the radiance of the texel it hits, probabilistically decide whether to reflect the particle, and if we reflect it we choose a direction and adjust its power.

Note that we want the particle to terminate at some point. For each surface we can assign a reflection probability p to each surface interaction. A natural choice would be to let $p = R$ as it is with light in nature. The particle would then scatter around the environment, not losing or gaining any energy until it is absorbed. This approach works well when the particles carry a single wavelength (Walter, Hubbard, Shirley, & Greenberg, 1997). However, when a spectrum or RGB triple is carried by the ray as is often implemented (Jensen, 2001), there is no single R and some compromise for the value of p should be chosen. The power ϕ' for reflected particles should be adjusted to account for the possible extinction of the particles:

$$\phi' = \frac{R\phi}{p}.$$



辐射度的另一种方法是使用统计模拟方法，通过随机跟踪来自灯具的光“粒子”穿过环境。这是一种粒子跟踪。有许多算法使用某种形式的粒子跟踪；我们将讨论一种形式的粒子跟踪，它将光沉积在三角形上的纹理中。首先，我们回顾一些基本的辐射关系。面积为 a 的朗伯表面的辐射 L 与单位面积的入射功率成正比：

$$L = \frac{\Phi}{\pi A}, \quad (23.1)$$

其中 Φ 是来自表面的传出功率。请注意，在本讨论中，所有辐射量都是光谱或RGB，这取决于实现。如果表面具有发射功率 Φ_e ，入射功率 Φ_i 和反射率 R ，则该方程变为

$$L = \frac{\Phi_e + R\Phi_i}{\pi A}.$$

如果给出每个三角形指定 Φ_e 和 R 的模型，我们可以按灯具进行灯具，以每个灯具的粒子形式发射功率。我们将纹理映射与每个三角形关联起来，以存储累积的亮度，所有的纹素都初始化为

$$L = \frac{\Phi_e}{\pi A}.$$

如果给定的三角形具有面积 A 和 n_t 纹素，并且它被携带功率 ϕ 的粒子击中，则该纹素的辐射度由

$$\Delta L = \frac{n_t \phi}{\pi A}.$$

一旦一个粒子击中一个表面，我们增加它击中的纹素的亮度，灵活地决定是否反射粒子，如果我们反射它，我们选择一个方向和调整它的力量。

请注意，我们希望粒子在某个点终止。对于每个表面，我们可以为每个表面相互作用分配反射概率 p 。一个自然的选择是让 $p=R$ ，因为它是与自然光。然后，粒子会在环境中散射，在被吸收之前不会损失或获得任何能量。当粒子携带单一波长时，这种方法效果很好（Walter, Hubbard, Shirley, & Greenberg, 1997）。然而，当光谱或RGB三重被射线携带时（Jensen, 2001），没有单一的 R ，应该选择 p 值的一些折衷。应调整反射粒子的功率 ϕ' ，以考虑粒子可能的消光：

$$\phi' = \frac{R\phi}{p}.$$

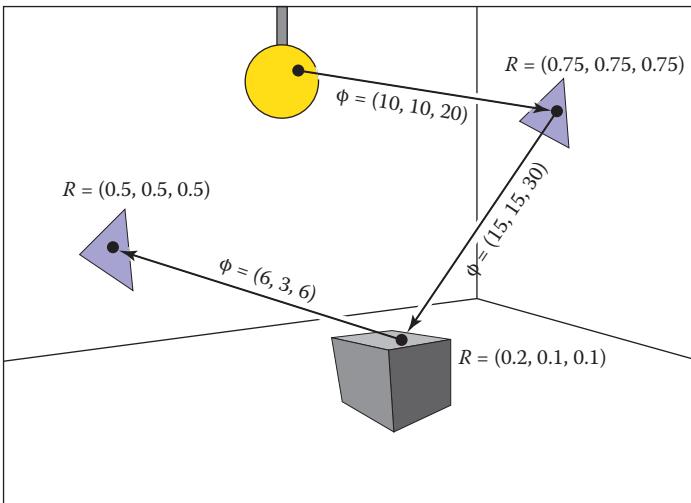


Figure 23.3. The path of a particle that survives with probability 0.5 and is absorbed at the last intersection. The RGB power is shown for each path segment.

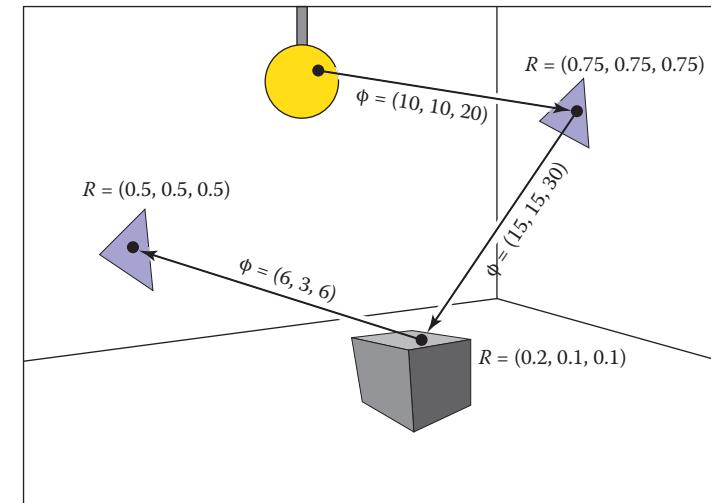
Note that p can be set to any positive constant less than one, and that this constant can be different for each interaction. When $p > R$ for a given wavelength, the particle will gain power at that wavelength, and when $p < R$ it will lose power at that wavelength. The case where it gains power will not interfere with convergence because the particle will stop scattering and be terminated at some point as long as $p < 1$. For the remainder of this discussion we set $p = 0.5$. The path of a single particle in such a system is shown in Figure 23.3.

A key part to this algorithm is that we scatter the light with an appropriate distribution for Lambertian surfaces. As discussed in Section 14.4.1, we can find a vector with a cosine (Lambertian) distribution by transforming two canonical random numbers (ξ_1, ξ_2) as follows:

$$\mathbf{a} = \left(\cos(2\pi\xi_1)\sqrt{\xi_2}, \sin(2\pi\xi_1)\sqrt{\xi_2}, \sqrt{1-\xi_2} \right). \quad (23.2)$$

Note that this assumes the normal vector is parallel to the z -axis. For a triangle, we must establish an orthonormal basis with \mathbf{w} parallel to the normal vector. We can accomplish this as follows:

$$\begin{aligned}\mathbf{w} &= \frac{\mathbf{n}}{\|\mathbf{n}\|}, \\ \mathbf{u} &= \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|}, \\ \mathbf{v} &= \mathbf{w} \times \mathbf{u},\end{aligned}$$



以概率0.5生存并在最后一个交叉点被吸收的粒子的路径。RGB功率显示为每个路径段。

注意， p 可以被设置为小于一的任何正常数，并且该常数对于每个相互作用可以是不同的。当 $p>r$ 对于给定波长时，粒子将在该波长处获得功率，而当 $p<R$ 时它将在该波长处失去功率。它获得功率的情况不会干扰convergence，因为只要 $p<1$ ，粒子就会停止散射并在某个点终止。对于本讨论的其余部分，我们设置 $p=0$ 。5.这种系统中单个粒子的路径如图23.3所示。

该算法的一个关键部分是我们以朗伯表面的适当分布散射光线。正如第14.4.1节所讨论的，我们可以通过转换两个规范随机数 (ξ_1, ξ_2) 来找到一个余弦 (Lambertian) 分布的向量，如下所示：

$$\mathbf{a} = \left(\cos(2\pi\xi_1)\sqrt{\xi_2}, \sin(2\pi\xi_1)\sqrt{\xi_2}, \sqrt{1-\xi_2} \right). \quad (23.2)$$

请注意，这假设法向量平行于 z 轴。对于三角形，我们必须建立一个正交基， \mathbf{w} 平行于法向量。我们可以做到这一点如下：

$$\begin{aligned}\mathbf{w} &= \frac{\mathbf{n}}{\|\mathbf{n}\|}, \\ \mathbf{u} &= \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|}, \\ \mathbf{v} &= \mathbf{w} \times \mathbf{u},\end{aligned}$$

where \mathbf{p}_i are the vertices of the triangle. Then, by definition, our vector in the appropriate coordinates is

$$\mathbf{a} = \cos(2\pi\xi_1)\sqrt{\xi_2}\mathbf{u} + \sin(2\pi\xi_1)\sqrt{\xi_2}\mathbf{v} + \sqrt{1-\xi_2}\mathbf{w}. \quad (23.3)$$

In pseudocode our algorithm for $p = 0.5$ and one luminaire is:

```

for (Each of  $n$  particles) do
    RGB  $\phi = \Phi/n$ 
    compute uniform random point  $\mathbf{a}$  on luminaire
    compute random direction  $\mathbf{b}$  with cosine density
    done = false
    while not done do
        if (ray  $\mathbf{a} + t\mathbf{b}$  hits at some point  $\mathbf{c}$ ) then
            add  $n_t R\phi / (\pi A)$  to appropriate texel
            if ( $\xi_1 > 0.5$ ) then
                 $\phi = 2R\phi$ 
            a =  $\mathbf{c}$ 
            b = random direction with cosine density
        else
            done = true

```

Here ξ_i are canonical random numbers. Once this code has run, the texture maps store the radiance of each triangle and can be rendered directly for any viewpoint with no additional computation.

23.2 Path Tracing

While particle tracing is well suited to precomputation of the radiances of diffuse scenes, it is problematic for creating images of scenes with general BRDFs or scenes that contain many objects. The most straightforward way to create images of such scenes is to use *path tracing* (Kajiya, 1986). This is a probabilistic method that sends rays from the eye and traces them back to the light. Often path tracing is used only to compute the indirect lighting. Here we will present it in a way that captures all lighting, which can be inefficient. This is sometimes called *brute force* path tracing. In Section 23.3, more efficient techniques for direct lighting can be added.

In path tracing, we start with the full transport equation:

$$L_s(\mathbf{k}_o) = L_e(\mathbf{k}_o) + \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i.$$

其中 \mathbf{p}_i 是三角形的顶点。然后，根据定义，我们在适当坐标中的向量是

$$\mathbf{a} = \cos(2\pi\xi_1)\sqrt{\xi_2}\mathbf{u} + \sin(2\pi\xi_1)\sqrt{\xi_2}\mathbf{v} + \sqrt{1-\xi_2}\mathbf{w}. \quad (23.3)$$

在伪代码中，我们的算法为 $p=0.5$ 和一个灯具是：

RGB $\phi = \Phi/n$ 计算灯具上的均匀随机点 \mathbf{a} 计算余弦密度的随机方向 \mathbf{b}

如果 (射线 $\mathbf{a} + t\mathbf{b}$ 在某个点 \mathbf{c} 击中) 则将 $n_t R\phi / (\pi A)$ 添加到适当的 texel if ($\xi_1 > 0.5$) 则 $\phi = 2R\phi$ 带余弦密度的随机方向。

完成=真

这里 ξ_i 是规范随机数。运行此代码后，纹理贴图会存储每个三角形的亮度，并且可以直接为任何视点渲染，无需额外计算。

23.2 路径跟踪

虽然粒子跟踪非常适合对漫反射场景的辐射进行预算，但对于创建具有一般BRDF的场景或包含许多对象的场景的图像来说，这是有问题的。创建此类场景图像的最直接方法是使用路径跟踪 (Kajiya, 1986)。这是一种概率方法，它从眼睛发送光线并将其追踪回光线。路径跟踪通常仅用于计算间接照明。在这里，我们将以捕获所有照明的方式呈现它，这可能是低效的。这有时称为蛮力路径跟踪。在第23.3节中，可以添加更有效的直接照明技术。

在路径跟踪中，我们从完整的传输方程开始：

$$L_s(\mathbf{k}_o) = L_e(\mathbf{k}_o) + \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i.$$

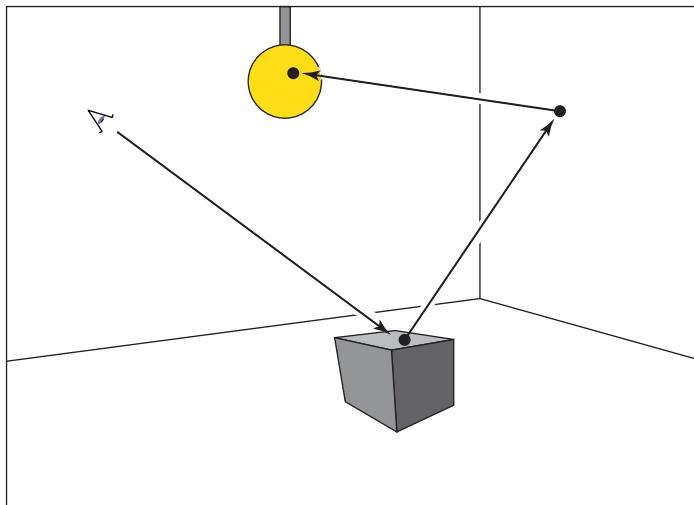


Figure 23.4. In path tracing, a ray is followed through a pixel from the eye and scattered through the scene until it hits a luminaire.

We use Monte Carlo integration to approximate the solution to this equation for each viewing ray. Recall from Section 14.3, that we can use random samples to approximate an integral:

$$\int_{x \in S} g(x) d\mu \approx \frac{1}{N} \sum_{i=1}^N \frac{g(x_i)}{p(x_i)},$$

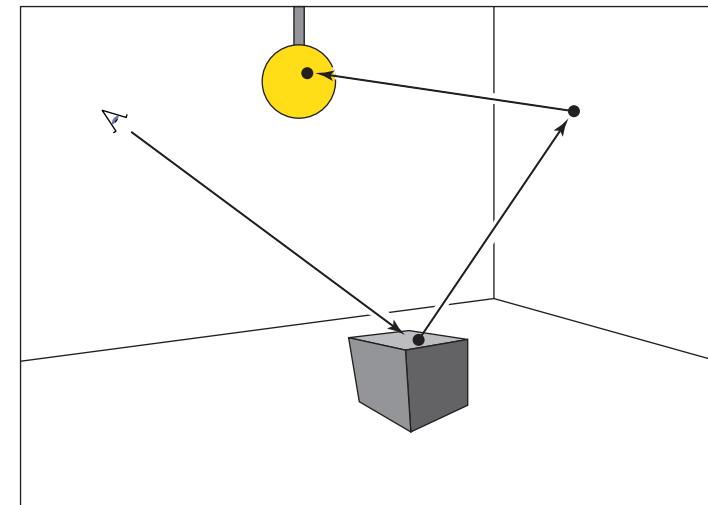
where the x_i are random points with probability density function p . If we apply this directly to the transport equation with $N = 1$ we get

$$L_s(\mathbf{k}_o) \approx L_e(\mathbf{k}_o) + \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i}{p(\mathbf{k}_i)}.$$

So, if we have a way to select random directions \mathbf{k}_i with a known density p , we can get an estimate. The catch is that $L_f(\mathbf{k}_i)$ is itself an unknown. Fortunately, we can apply recursion and use a statistical estimate for $L_f(\mathbf{k}_i)$ by sending a ray in that direction to find the surface seen in that direction. We end when we hit a luminaire and L_e is nonzero (Figure 23.4). This method assumes lights have zero reflectance, or we would continue to recurse.

In the case of a Lambertian BRDF ($\rho = R/\pi$), we can use a cosine density function:

$$p(\mathbf{k}_i) = \frac{\cos \theta_i}{\pi}.$$



在路径追踪中，光线从眼睛穿过一个像素，然后散射在场景中，直到它击中灯具。

我们使用蒙特卡罗积分来近似每个观察射线的该方程的解。回想一下第14.3节，我们可以使用随机样本来近似积分：

$$\int_{x \in S} g(x) d\mu \approx \frac{1}{N} \sum_{i=1}^N \frac{g(x_i)}{p(x_i)},$$

其中 x_i 是具有概率密度函数

的随机点。如果我们将其直接应用于n=1的传输方程，我们得到

$$L_s(\mathbf{k}_o) \approx L_e(\mathbf{k}_o) + \frac{\rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i}{p(\mathbf{k}_i)}.$$

所以，如果我们有办法选择具有已知密度p的随机方向ki，我们可以得到一个估计。美中不足的是，Lf (ki) 本身就是一个未知。幸运的是，我们可以应用递归并通过向该方向发送射线来使用Lf (ki) 的统计估计来找到该方向上看到的表面。当我们击中一个灯具并且Le非零时，我们结束（图23.4）。此方法假设光源的反射率为零，否则我们将继续递归。

在朗伯BRDF ($\rho=r\pi$) 的情况下，我们可以使用余弦密度函数：

$$p(\mathbf{k}_i) = \frac{\cos \theta_i}{\pi}.$$



A direction with this density can be chosen according to Equation (23.3). This allows some cancellation of cosine terms in our estimate:

$$L_s(\mathbf{k}_o) \approx L_e(\mathbf{k}_o) + RL_f(\mathbf{k}_i).$$

In pseudocode, such a path tracer for Lambertian surfaces would operate just like the ray tracers described in Chapter 4, but the *raycolor* function would be modified:

```
RGB raycolor(ray a + tb, int depth)
if (ray hits at some point c ) then
    RGB c = L_e(-b)
    if (depth < maxdepth) then
        compute random direction d
        return c + R raycolor(c + sd, depth+1)
    else
        return background color
```

This will result in a very noisy image unless either large luminaires or very large numbers of samples are used. Note the color of the luminaires must be well above one (sometimes thousands or tens of thousands) to make the surfaces have final colors near one, because only those rays that hit a luminaire by chance will make a contribution, and most rays will contribute only a color near zero. To generate the random direction d , we use the same technique as we do in particle tracing (see Equation (23.2)).

In the general case, we might want to use spectral colors or use a more general BRDF. In practice, we should have the material class contain member functions to compute a random direction as well as compute the p associated with that direction. This way materials can be added transparently to an implementation.

23.3 Accurate Direct Lighting

This section presents a more physically based method of direct lighting than Chapter 10. These methods will be useful in making global illumination algorithms more efficient. The key idea is to send shadow rays to the luminaires as described in Chapter 4, but to do so with careful bookkeeping based on the transport equation from the previous chapter. The global illumination algorithms can be adjusted to make sure they compute the direct component exactly once. For example, in particle tracing, particles coming directly from the luminaire would not be logged, so the particles would only encode indirect lighting. This makes



具有该密度的方向可根据等式(23.3)选择。这允许在我们的估计余弦项的一些取消:

$$L_s(\mathbf{k}_o) \approx L_e(\mathbf{k}_o) + RL_f(\mathbf{k}_i).$$

在伪代码中，这样的朗伯曲面路径跟踪器将像第4章中描述的光线跟踪器一样运行，但raycolor函数将被修改:

```
RGB raycolor(raya+tb intdepth)if(r
ayhitsatsomepointc)thenRGBc=L
e(-b)if(depth<maxdepth)then
计算随机方向d返回c+R射线颜色(c+sd 深
度+1)
else
    返回背景颜色
```

这将导致一个非常嘈杂的图像，除非使用大的灯具或非常大量的样品。请注意，灯具的颜色必须远高于一（有时数千或数万），以使表面具有接近一的最终颜色，因为只有那些偶然击中灯具的光线才会做出贡献，而大多数光线只为了生成随机方向d，我们使用与粒子跟踪相同的技术（参见方程 (23.2)）。

在一般情况下，我们可能希望使用光谱颜色或使用更一般的BRDF。在实践中，我们应该让材质类包含成员函数来计算随机方向以及计算与该方向相关的p。通过这种方式，材料可以透明地添加到实现中。

23.3 精确的直接照明

与第10章相比，本节介绍了一种更基于物理的直接照明方法。这些方法将有助于提高全局照明算法的效率。关键思想是将阴影射线发送到灯具，如第4章所述，但要根据上一章的传输方程仔细记帐。可以调整全局照明算法，以确保它们精确计算一次直接分量。例如，在粒子追踪中，直接来自灯具的粒子不会被记录，因此粒子只会编码间接照明。这使得

nice looking shadows much more efficiently than computing direct lighting in the context of global illumination.

23.3.1 Mathematical Framework

To calculate the direct light from one *luminaire* (light emitting object) onto a non-emitting surface, we solve a form of the transport equation from Section 18.2:

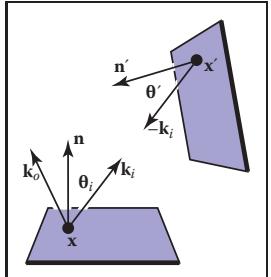


Figure 23.5. The direct lighting terms for Equation (23.4).

Recall that L_e is the emitted radiance of the source, v is a visibility function that is equal to 1 if x “sees” x' and zero otherwise, and the other variables are as illustrated in Figure 23.5.

If we are to sample Equation (23.4) using Monte Carlo integration, we need to pick a random point x' on the surface of the luminaire with density function p (so $x' \sim p$). Just plugging into Equation (14.5) with one sample yields

$$L_s(x, k_o) \approx \frac{\rho(k_i, k_o) L_e(x', -k_i) v(x, x') \cos \theta_i \cos \theta'}{p(x') \|x - x'\|^2}. \quad (23.5)$$

If we pick a uniform random point on the luminaire, then $p = 1/A$, where A is the area of the luminaire. This gives

$$L_s(x, k_o) \approx \frac{\rho(k_i, k_o) L_e(x', -k_i) v(x, x') A \cos \theta_i \cos \theta'}{\|x - x'\|^2}. \quad (23.6)$$

We can use Equation (23.6) to sample planar (e.g., rectangular) luminaires in a straightforward fashion. We simply pick a random point on each luminaire.

The code for one luminaire is:

```
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 - epsilon) then
    return rho(k_i, k_o) L_e(x', -k_i) (n . d) (-n' . d) / \|d\|^4
else
    return 0
```

The above code needs some extra tests such as clamping the cosines to zero if they are negative. Note that the term $\|d\|^4$ comes from the distance squared term

好看的阴影比在全局照明环境下计算直接照明更有效。

23.3.1 数学框架

为了计算从一个灯具（发光物体）到非拟合表面的直射光，我们解决了第18.2节中的传输方程的一种形式：

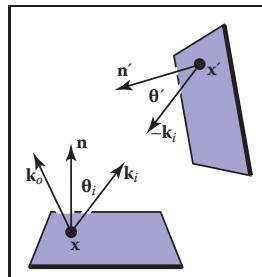


Figure 23.5. 直接的
lighting terms for Equa-
tion (23.4).

$$L_s(x, k_o) = \int_{\text{all } x'} \frac{\rho(k_i, k_o) L_e(x', -k_i) v(x, x') \cos \theta_i \cos \theta'}{\|x - x'\|^2} dA'. \quad (23.4)$$

回想一下， L_e 是源的发射辐射， v 是一个可见性函数，如果“看到” x' ，则等于1，否则为零，其他变量如图23.5所示。

如果我们要使用蒙特卡罗积分对方程 (23.4) 进行采样，我们需要以密度函数 $p(x' \in p)$ 在灯具表面选取一个随机点 x' 。只需用一个样品产量插入公式 (14.5)

$$L_s(x, k_o) \approx \frac{\rho(k_i, k_o) L_e(x', -k_i) v(x, x') \cos \theta_i \cos \theta'}{p(x') \|x - x'\|^2}. \quad (23.5)$$

如果我们在灯具上选择一个均匀的随机点，那么 $p=1/A$ ，其中 A 是灯具的面积。这给

$$L_s(x, k_o) \approx \frac{\rho(k_i, k_o) L_e(x', -k_i) v(x, x') A \cos \theta_i \cos \theta'}{\|x - x'\|^2}. \quad (23.6)$$

我们可以使用公式 (23.6) 以简单的方式对平面（例如矩形）灯具进行采样。我们只需在每个灯具上随机选择一个点。一个灯具的代码是：

```
color directLight(x ko n) pickrandompointx'
withnormalvectorn'onlightd=x' xki=d d
if(ray x+td hasnohitsfort<1- )then
```

```
i o e i
||^4
else
return 0
```

上面的代码需要一些额外的测试，例如如果余弦为负，则将余弦一位为零。请注意，术语 d^4 来自距离平方项

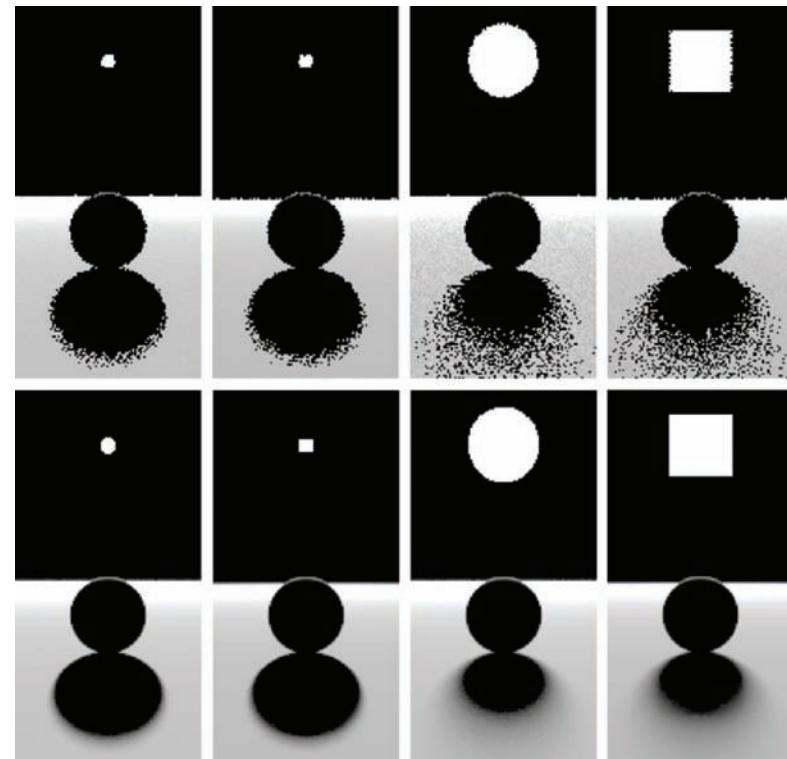


Figure 23.6. Various soft shadows on a backlit sphere with a square and an area light source. Top: 1 sample. Bottom: 100 samples. Note that the shape of the light source is less important than its size in determining shadow appearance.

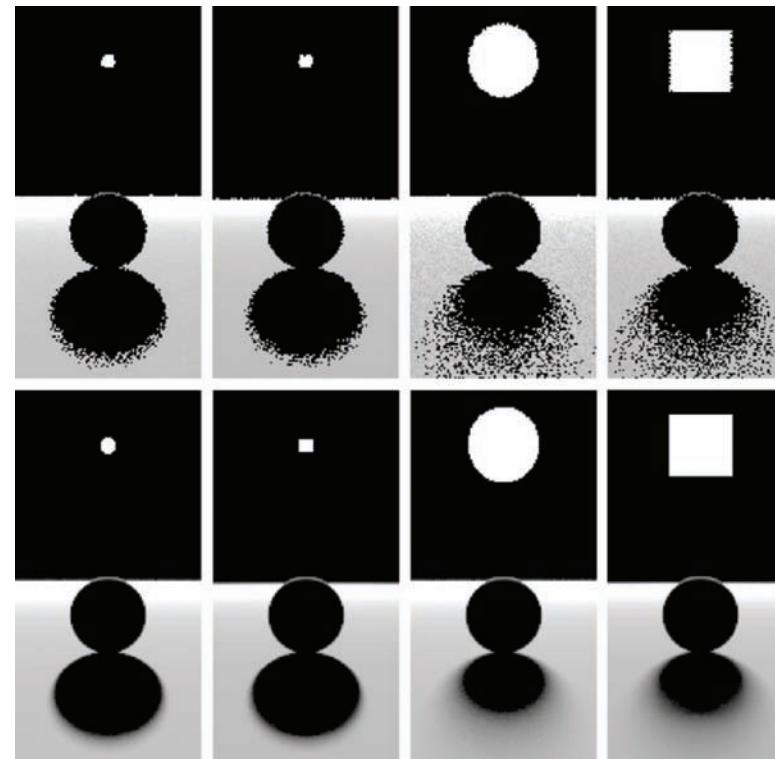
and the two cosines, e.g., $\mathbf{n} \cdot \mathbf{d} = \|\mathbf{d}\| \cos \theta$ because \mathbf{d} is not necessarily a unit vector.

Several examples of soft shadows are shown in Figure 23.6.

23.3.2 Sampling a Spherical Luminaire

Though a sphere with center \mathbf{c} and radius R can be sampled using Equation (23.6), this sampling will yield a very noisy image because many samples will be on the back of the sphere, and the $\cos \theta'$ term varies so much. Instead, we can use a more complex $p(\mathbf{x}')$ to reduce noise.

The first nonuniform density we might try is $p(\mathbf{x}') \propto \cos \theta'$. This turns out to be just as complicated as sampling with $p(\mathbf{x}') \propto \cos \theta' / \|\mathbf{x}' - \mathbf{x}\|^2$, so we instead discuss that here. We observe that sampling on the luminaire this way is the



具有正方形和面积光源的背光球体上的各种柔和阴影。上图：1个样品。底部：100个样品。请注意，光源的形状在确定阴影外观方面不如其大小重要。

和两个余弦，例如， $n \cdot d = d \cos \theta$ ，因为 d 不一定是单位向量。

软阴影的几个示例如图23.6所示。

23.3.2 对球形灯具进行取样

虽然可以使用公式 (23.6) 对中心 c 和半径 R 的球体进行采样，但这种采样将产生非常嘈杂的图像，因为许多样本将位于球体的背面，并且 $\cos \theta'$ 项变化很大。相反，我们可以使用更复杂的 $p(\mathbf{x}')$ 来降低噪声。

我们可能尝试的第一个非均匀密度是 $p(\mathbf{x}') \in \cos \theta'$ 。事实证明，这与用 $p(\mathbf{x}') \in \cos \theta' \in \mathbf{x}' \in 2$ 采样一样复杂，所以我们在这里讨论这个问题。我们观察到，以这种方式对灯具进行采样是

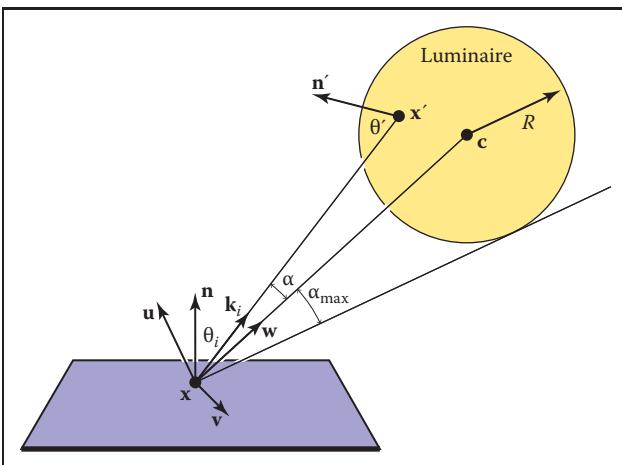


Figure 23.7. Geometry for direct lighting at point x from a spherical luminaire.

same as using a constant density function $q(\mathbf{k}_i) = \text{const}$ defined in the space of directions subtended by the luminaire as seen from x . We now use a coordinate system defined with x at the origin, and a right-handed orthonormal basis with $\mathbf{w} = (\mathbf{c} - \mathbf{x})/\|\mathbf{c} - \mathbf{x}\|$, and $\mathbf{v} = (\mathbf{w} \times \mathbf{n})/\|(\mathbf{w} \times \mathbf{n})\|$ (see Figure 23.7). We also define (α, ϕ) to be the azimuthal and polar angles with respect to the uvw coordinate system.

The maximum α that includes the spherical luminaire is given by

$$\alpha_{\max} = \arcsin\left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right) = \arccos\sqrt{1 - \left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right)^2}.$$

Thus, a uniform density (with respect to solid angle) within the cone of directions subtended by the sphere is just the reciprocal of the solid angle $2\pi(1 - \cos \alpha_{\max})$ subtended by the sphere:

$$q(\mathbf{k}_i) = \frac{1}{2\pi \left(1 - \sqrt{1 - \left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right)^2}\right)}.$$

And we get

$$\begin{bmatrix} \cos \alpha \\ \phi \end{bmatrix} = \begin{bmatrix} 1 - \xi_1 + \xi_1 \sqrt{1 - \left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right)^2} \\ 2\pi\xi_2 \end{bmatrix}.$$

This gives us the direction \mathbf{k}_i . To find the actual point, we need to find the first point on the sphere in that direction. The ray in that direction is just $(\mathbf{x} + t\mathbf{k}_i)$,

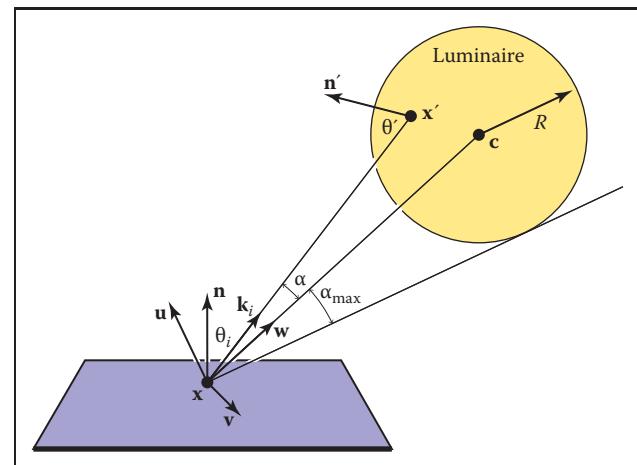


图23.7。用于从球形灯具在点x处直接照明的几何形状。

与使用恒定密度函数 $q(\mathbf{k}_i) = \text{const}$ 定义在灯具从 x 看到的方向空间中相同。我们现在使用以 x 为原点定义的坐标系，以及以 $\mathbf{w} = (\mathbf{c}-\mathbf{x})/\|\mathbf{c}-\mathbf{x}\|$ 和 $\mathbf{v} = (\mathbf{w} \times \mathbf{n})/\|(\mathbf{w} \times \mathbf{n})\|$ （见图23.7）的右旋正交基。我们还将 (α, ϕ) 定义为相对于 uvw 坐标系的方位角和极角。

包括球形灯具的最大 α 由下式给出

$$\alpha_{\max} = \arcsin\left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right) = \arccos\sqrt{1 - \left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right)^2}.$$

因此，在由球体子化的方向锥内的均匀密度（相对于实心角）只是由球体子化的实心角 $2\pi(1 - \cos \alpha_{\max})$ 的倒数：

$$q(\mathbf{k}_i) = \frac{1}{2\pi \left(1 - \sqrt{1 - \left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right)^2}\right)}.$$

我们得到了

$$\begin{bmatrix} \cos \alpha \\ \phi \end{bmatrix} = \begin{bmatrix} 1 - \xi_1 + \xi_1 \sqrt{1 - \left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|}\right)^2} \\ 2\pi\xi_2 \end{bmatrix}.$$

这给了我们 \mathbf{k}_i 的方向。要找到实际点，我们需要找到该方向球体上的第一个点。那个方向的射线刚好 $(\mathbf{x} + t\mathbf{k}_i)$

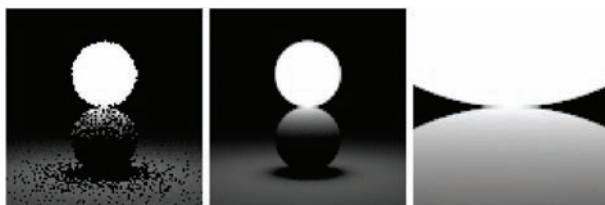


Figure 23.8. A sphere with $L_e = 1$ touching a sphere of reflectance 1. Where the two spheres touch, the reflective sphere should have $L(x') = 1$. Left: 1 sample. Middle: 100 samples. Right: 100 samples, close-up.

where \mathbf{k}_i is given by

$$\mathbf{k}_i = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \begin{bmatrix} \cos \phi \sin \alpha \\ \sin \phi \sin \alpha \\ \cos \alpha \end{bmatrix}.$$

We must also calculate $p(\mathbf{x}')$, the probability density function with respect to the area measure (recall that the density function q is defined in solid angle space). Since we know that q is a valid probability density function using the ω measure, and we know that $d\Omega = dA(\mathbf{x}') \cos \theta' / \|\mathbf{x}' - \mathbf{x}\|^2$, we can relate any probability density function $q(\mathbf{k}_i)$ with its associated probability density function $p(\mathbf{x}')$:

$$q(\mathbf{k}_i) = \frac{p(\mathbf{x}') \cos \theta'}{\|\mathbf{x}' - \mathbf{x}\|^2}. \quad (23.7)$$

So we can solve for $p(\mathbf{x}')$:

$$p(\mathbf{x}') = \frac{\cos \theta'}{2\pi \|\mathbf{x}' - \mathbf{x}\|^2 \left(1 - \sqrt{1 - \left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|} \right)^2} \right)}.$$

A good debugging case for this is shown in Figure 23.8.

23.3.3 Nondiffuse Luminaries

There is no reason the luminance of the luminaire cannot vary with both direction and position. For example, it can vary with position if the luminaire is a television. It can vary with direction for car headlights and other directional sources. Little in our analysis need change from the previous sections, except that $L_e(\mathbf{x}')$ must change to $L_e(\mathbf{x}', -\mathbf{k}_i)$. The simplest way to vary the intensity with direction is to use a Phong-like pattern with respect to the normal vector \mathbf{n}' . To avoid using an exponent in the term for the total light output, we can use the form

$$L_e(\mathbf{x}', -\mathbf{k}_i) = \frac{(n+1)E(\mathbf{x}')}{2\pi} \cos^{(n-1)\theta'},$$



Le=1的球体接触反射率1的球体。在两个球体接触的地方，反射球体应该有 $L(x') = 1$ 。左：1个样品。中间：100个样品。右：100个样品，特写。

其中 \mathbf{k}_i 由

$$\mathbf{k}_i = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \begin{bmatrix} \cos \phi \sin \alpha \\ \sin \phi \sin \alpha \\ \cos \alpha \end{bmatrix}.$$

我们还必须计算 $p(\mathbf{x}')$ ，即相对于面积度量的概率密度函数（回想一下密度函数 q 是在立体角空间中定义的）。由于我们知道 q 是使用 ω 度量的有效概率密度函数，并且我们知道 $d\Omega = dA(\mathbf{x}') \cos \theta' / \|\mathbf{x}' - \mathbf{x}\|^2$ ，我们可以将任何概率密度函数 $q(\mathbf{k}_i)$ 与其相关联的概率密度函：

$$q(\mathbf{k}_i) = \frac{p(\mathbf{x}') \cos \theta'}{\|\mathbf{x}' - \mathbf{x}\|^2}. \quad (23.7)$$

所以我们可以求解 $p(\mathbf{x}')$ ：

$$p(\mathbf{x}') = \frac{\cos \theta'}{2\pi \|\mathbf{x}' - \mathbf{x}\|^2 \left(1 - \sqrt{1 - \left(\frac{R}{\|\mathbf{x} - \mathbf{c}\|} \right)^2} \right)}.$$

一个很好的调试案例如图23.8所示。

23.3.3 Nondiffuse Luminaries

没有理由灯具的亮度不能随方向和位置而变化。例如，如果灯具是电视，它可以随位置而变化。它可以随汽车前灯和其他方向源的方向而变化。在我们的分析中，除了 $L_e(\mathbf{x}')$ 必须更改为 $L_e(\mathbf{x}', -\mathbf{k}_i)$ 之外，我们不需要改变前面的部分。改变强度与方向的最简单方法是使用相对于法向量 \mathbf{n}' 的Phong样图案。为了避免在总光输出项中使用指数，我们可以使用以下形式

$$L_e(\mathbf{x}', -\mathbf{k}_i) = \frac{(n+1)E(\mathbf{x}')}{2\pi} \cos^{(n-1)\theta'},$$

where $E(x')$ is the *radiant exitance* (power per unit area) at point x' , and n is the Phong exponent. You get a diffuse light for $n = 1$. If the light is nonuniform across its area, e.g., as a television set is, then E will not be a constant.

Frequently Asked Questions

- My pixel values are no longer in some sensible zero-to-one range. What should I display?

You should use one of the *tone reproduction* techniques described in Chapter 21.

- What global illumination techniques are used in practice?

For batch rendering of complex scenes, path tracing with one level of reflection is often used. Path tracing is often augmented with a particle tracing preprocess as described in Jensen's book in the chapter notes. For walkthrough games, some form of world-space preprocess is often used, such as the particle tracing described in this chapter. For scenes with very complicated specular transport, an elegant but involved method, Metropolis Light Transport (Veach & Guibas, 1997) may be the best choice.

- How does the ambient component relate to global illumination?

For diffuse scenes, the radiance of a surface is proportional to the product of the irradiance at the surface and the reflectance of the surface. The ambient component is just an approximation to the irradiance scaled by the inverse of π . So although it is a crude approximation, there can be some methodology to guessing it (M. F. Cohen, Chen, Wallace, & Greenberg, 1988), and it is probably more accurate than doing nothing, i.e., using zero for the ambient term. Because the indirect irradiance can vary widely within a scene, using a different constant for each surface can be used for better results rather than using a global ambient term.

- Why do most algorithms compute direct lighting using traditional ray tracing?

Although global illumination algorithms automatically compute direct lighting, and it is, in fact, slightly more complicated to make them compute only indirect lighting, it is usually faster to compute direct lighting separately. There are three reasons for this. First, indirect lighting tends to be smooth compared to

其中 $E(x')$ 是 x 点的辐射系数（单位面积的功率）， n 是Phong指数。你得到 $n=1$ 的漫射光。如果光在其区域内是不均匀的，例如，作为电视机，则 E 将不是常数。

常见问题

- 我的像素值不再在一些合理的零到一范围内。我应该显示什么？

您应该使用第21章中描述的音调再现技术之一。

- *实践中使用了哪些全局照明技术？

对于复杂场景的批量渲染，通常使用具有一级反射的路径跟踪。路径跟踪通常使用粒子跟踪预处理来增强，如Jensen在章节注释中的书中所述。对于演练游戏，经常使用某种形式的世界空间预处理，例如本章中描述的粒子跟踪。对于镜面传输非常复杂的场景，一种优雅但复杂的方法，MetropolisLightTransport (Veach & Guibas, 1997) 可能是最佳选择。

- *环境分量如何与全局照明相关？

对于漫反射场景，表面的辐射度与表面的辐照度和表面的反射率的乘积成正比。环境分量只是由 π 的倒数缩放的辐照度的近似值。因此，尽管它是一个粗略的近似值，但可以有一些方法来猜测它 (M.F.Cohen, Chen, Wallace, & Greenberg, 1988)，它可能比无所事事更准确，即使用零作为环境项。由于间接辐照度在场景中变化很大，因此对每个表面使用不同的常数可以获得更好的结果，而不是使用全局环境项。

- *为什么大多数算法使用传统的光线追踪来计算直接照明？

虽然全局照明算法会自动计算直接照明，而且实际上，使它们只计算间接照明稍微复杂一些，但单独计算直接照明通常会更快。这有三个原因。首先，间接照明相比于



Figure 23.9. A comparison between a rendering and a photo. *Image courtesy Suman Pattanaik and the Cornell Program of Computer Graphics.*

direct lighting (see Figure 23.1) so coarser representations can be used, e.g., low-resolution texture maps for particle tracing. The second reason is that light sources tend to be small, and it is rare to hit them by chance in a “from the eye” method such as path tracing, while direct shadow rays are efficient. The third reason is that direct lighting allows stratified sampling, so it converges rapidly compared to unstratified sampling. The issue of stratification is the reason that shadow rays are used in Metropolis Light Transport despite the stability of its default technique for dealing with direct lighting as just one type of path to handle.

- How artificial is it to assume ideal diffuse and specular behavior?

For environments that have only matte and mirrored surfaces, the Lambertian/specular assumption works well. A comparison between a rendering using that assumption and a photograph is shown in Figure 23.9.

- How many shadow rays are needed per pixel?

Typically between 16 and 400. Using narrow penumbra, a large ambient term (or a large indirect component), and a masking texture (Ferwerda, Shirley, Pattanaik, & Greenberg, 1997) can reduce the number needed.

- How do I sample something like a filament with a metal reflector where much of the light is reflected from the filament?

Typically, the whole light is replaced by a simple source that approximates its aggregate behavior. For viewing rays, the complicated source is used. So a car

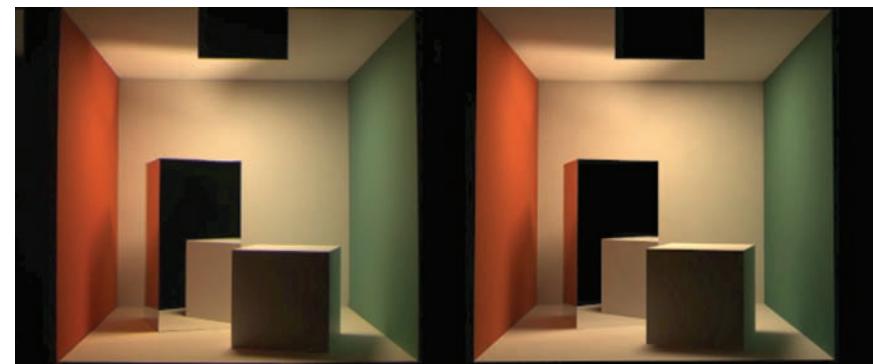


图23.9。渲染和照片之间的比较。图像由Suman Pattanaik和计算机图形学的康奈尔程序提供。

直接照明（见图23.1），因此可以使用较粗的表示，例如用于粒子跟踪的低分辨率纹理图。第二个原因是光源往往很小，并且在路径追踪等“从眼睛”方法中偶然击中它们是罕见的，而直接阴影射线是有效的。第三个原因是直接照明允许分层采样，因此与未分层采样相比，它会快速收敛。分层问题是阴影射线被用于大都市光传输的原因，尽管它的默认技术的稳定性处理直接照明只是一种类型的路径来处理。

- 假设理想的漫反射和镜面行为有多人为？

对于只有哑光表面和镜面表面的环境，朗伯镜面假设效果很好。使用该假设的渲染与照片之间的比较如图23.9所示。

- 每个像素需要多少阴影射线？

通常在16和400之间。使用窄半影，大的环境项（或大的间接分量）和掩蔽纹理（Ferwerda, Shirley, Pattanaik, & Greenberg, 1997）可以减少所需的数量。

- 如何使用金属反射器对灯丝进行采样，其中大部分光线从灯丝反射？

通常，整个光源被近似其聚集行为的简单光源所取代。对于查看射线，使用复杂的源。所以一辆车

headlight would look complex to the viewer, but the lighting code might see simple disk-shaped lights.

- Isn't something like the sky a luminaire?

Yes, and you can treat it as one. However, such large light sources may not be helped by direct lighting; the brute-force techniques are likely to work better.

Notes

Global illumination has its roots in the fields of heat transfer and illumination engineering as documented in *Radiosity: A Programmer's Perspective* (Ashdown, 1994). Other good books related to global illumination include *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, & Bekaert, 2002), and *Physically Based Rendering* (Pharr & Humphreys, 2004). The probabilistic methods discussed in this chapter are from *Monte Carlo Techniques for Direct Lighting Calculations* (Shirley, Wang, & Zimmerman, 1996).

Exercises

1. For a closed environment, where every surface is a diffuse reflector and emitter with reflectance R and emitted radiance E , what is the total radiance at each point? Hint: for $R = 0.5$ and $E = 0.25$ the answer is 0.5. This is an excellent debugging case.
2. Using the definitions from Chapter 18, verify Equation (23.1).
3. If we want to render a typically sized room with textures at centimeter-square resolution, approximately how many particles should we send to get an average of about 1000 hits per texel?
4. Develop a method to take random samples with uniform density from a disk.
5. Develop a method to take random samples with uniform density from a triangle.
6. Develop a method to take uniform random samples on a “sky dome” (the inside of a hemisphere).

前照灯对观察者来说看起来很复杂，但照明代码可能会看到简单的圆盘形灯。

- *像天空这样的东西不是灯具吗？

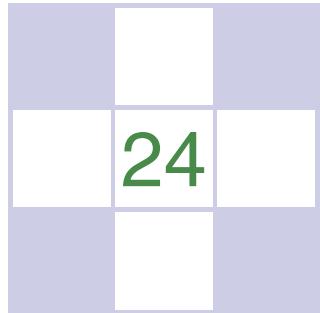
是的，你可以把它当作一个。但是，直接照明可能无法帮助这种大型光源；蛮力技术可能会更好地工作。

Notes

如Radiosity:AProgrammer'SPerspective (Ashdown, 1994) 所述，全球照明起源于传热和照明工程领域。与全球照明相关的其他好书包括Radiosity和GlobalIllumination (M.F.Cohen & Wallace, 1993) , Radiosity和RealisticImageSynthesis (Sillion & Puech, 1994) , PrinciplesOfDigitalImageSynthesis (Glassner, 1995) , RealisticImageSynthesisUsingPhotonMapping (Jensen, 2001) , AdvancedGlobalIllumination (Dutré, Bala, & Bekaert, 2002) 和PhysicalBasedRendering (Pharr & Humphreys, 2004) 。本章讨论的概率方法来自MonteCarloTechniquesForDirectLightingCalculations (Shirley, Wang, & Zimmerman, 1996) 。

Exercises

- 1.对于封闭环境，其中每个表面都是反射率R和发射辐射度E的漫反射器和发射器，每个点的总辐射度是多少？提示：对于R=0.5和E=0.25答案是0.5。这是一个很好的调试案例。
- 2.使用第18章的定义，验证方程（23.1）。
- 3.如果我们想以厘米级的分辨率渲染一个通常大小的房间，那么我们应该发送多少个粒子来获得平均每个纹素1000次点击？
- 4.开发一种从盘中取具有均匀密度的随机样品的方法。
- 5.开发一种从三角形中提取密度均匀的随机样本的方法。
- 6.开发一种在“天空穹顶”（半球内部）上采取均匀随机样本的方法。



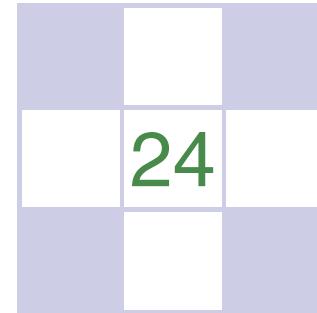
Reflection Models

As we discussed in Chapter 18, the reflective properties of a surface can be summarized using the BRDF (Nicodemus, Richmond, Hsia, Ginsberg, & Limperis, 1977; Cook & Torrance, 1982). In this chapter, we discuss some of the most visually important aspects of material properties and a few fairly simple models that are useful in capturing these properties. There are many BRDF models in use in graphics, and the models presented here are meant to give just an idea of nondiffuse BRDFs.

24.1 Real-World Materials

Many real materials have a visible structure at normal viewing distances. For example, most carpets have easily visible pile that contributes to appearance. For our purposes, such structure is not part of the material property but is, instead, part of the geometric model. Structure whose details are invisible at normal viewing distances, but which do determine macroscopic material appearance, are part of the material property. For example, the fibers in paper have a complex appearance under magnification, but they are blurred together into an homogeneous appearance when viewed at arm's length. This distinction between microstructure that is folded into BRDF is somewhat arbitrary and depends on what one defines as "normal" viewing distance and visual acuity, but the distinction has proven quite useful in practice.

In this section, we define some categories of materials. Later in the chapter, we present reflection models that target each type of material. In the notes at the



反射模型

正如我们在第18章中所讨论的，表面的反射特性可以用BRDF来概括（Nicodemus, Richmond, Hsia, Ginsberg, & Limperis, 1977; Cook&Torrance, 1982）。在本章中，我们将讨论材料属性的一些视觉上最重要的方面，以及一些在捕获这些属性时很有用的相当简单的模型。在图形中使用了许多BRDF模型，这里介绍的模型只是为了给出一个非差异Brdf的概念。

24.1 Real-World Materials

许多真实材料在正常观察距离下具有可见结构。对于ex充足，大多数地毯都有容易看到的桩，有助于外观。对于我们的目的，这种结构不是材料属性的一部分，而是几何模型的一部分。结构的细节在正常的观察距离是不可见的，但它确实决定了宏观材料的外观，是材料属性的一部分。例如，纸中的纤维在放大下具有复杂的外观，但是当以手臂的长度观察时，它们一起模糊成均匀的外观。折叠成BRDF的微结构之间的这种区分有些随意，取决于人们定义的"正常"观看距离和视觉敏锐度，但这种区分在实践中已被证明非常有用。

在本节中，我们定义了一些材料类别。在本章的后面，我们将介绍针对每种类型材料的反射模型。在笔记中

end of the chapter, some models that account for more exotic materials are also discussed.

24.1.1 Smooth Dielectrics and Metals

Dielectrics are clear materials that refract light; their basic properties were summarized in Chapter 4. Metals reflect and refract light much like dielectrics, but they absorb light very, very quickly. Thus, only very thin metal sheets are transparent at all, e.g., the thin gold plating on some glass objects. For a smooth material, there are only two important properties:

1. How much light is reflected at each incident angle and wavelength.
2. What fraction of light is absorbed as it travels through the material for a given distance and wavelength.

The amount of light transmitted is whatever is not reflected (a result of energy conservation). For a metal, in practice, we can assume all the light is immediately absorbed. For a dielectric, the fraction is determined by the constant used in Beer's Law as discussed in Chapter 13.

The amount of light reflected is determined by the *Fresnel equations* as discussed in Chapter 4. These equations are straightforward, but cumbersome. The main effect of the Fresnel equations is to increase the reflectance as the incident angle increases, particularly near grazing angles. This effect works for transmitted light as well. These ideas are shown diagrammatically in Figure 24.1. Note that the light is repeatedly reflected and refracted as shown in Figure 24.2. Usually only one or two of the reflected images is easily visible.

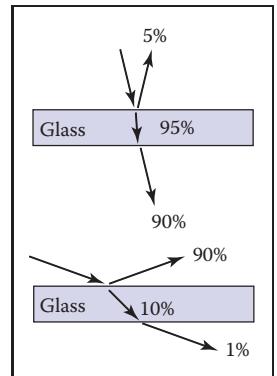


Figure 24.1. The amount of light reflected and transmitted by glass varies with the angle.

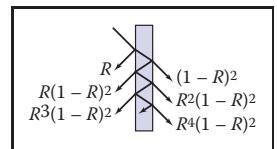


Figure 24.2. Light is repeatedly reflected and refracted by glass, with the fractions of energy shown.

24.1.2 Rough Surfaces

If a metal or dielectric is roughened to a small degree, but not so small that diffraction occurs, then we can think of it as a surface with *microfacets* (Cook & Torrance, 1982). Such surfaces behave specularly at a closer distance, but viewed at a further distance seem to spread the light out in a distribution. For a metal, an example of this rough surface might be brushed steel, or the "cloudy" side of most aluminum foil.

For dielectrics, such as a sheet of glass, scratches or other irregular surface features make the glass blur the reflected and transmitted images that we can normally see clearly. If the surface is heavily scratched, we call it *translucent* rather than transparent. This is a somewhat arbitrary distinction, but it is usually clear whether we would consider a glass translucent or transparent.

在本章的结尾，还讨论了一些占更多奇异材料的模型。

24.1.1 光滑的电介质和金属

电介质是折射光的透明材料；它们的基本性质在第4章中进行了总结。金属像电介质一样反射和折射光，但它们吸收光的速度非常快。因此，只有非常薄的金属片是透明的，例如，在一些玻璃物体上镀薄金。对于光滑的材料，只有两个重要的属性：

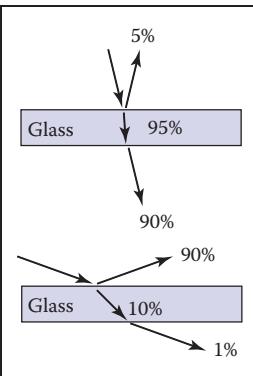
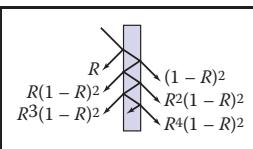


图24.1。玻璃反射和透射的光量随角度而变化。



光被玻璃反复反射和重新破碎，显示出能量的压裂。

1. 在每个入射角度和波长处反射多少光。

2. 当光在给定的距离和波长内穿过材料时，吸收了多少部分的光。

透射的光量是没有反射的（节能的结果）。对于金属，在实践中，我们可以假设所有的光立即被吸收。对于电介质，分数由啤酒定律中使用的常数决定，如第13章所讨论的。

反射的光量由第4章中讨论的菲涅耳方程确定。这些方程很简单，但很麻烦。菲涅耳方程的主要作用是随着入射角的增加而增加反射率，特别是在掠射角附近。这种效果也适用于透射光。这些想法如图24.1所示。请注意，如图24.2所示，光被反复反射和折射。通常只有一个或两个反射图像是容易可见的。

24.1.2 粗糙表面

如果金属或电介质被粗糙化到很小程度，但不是那么小以至于发生衍射，那么我们可以将其视为具有微表面的表面（Cook & Torrance, 1982）。这样的表面在较近的距离处表现得像镜面，但在更远的距离处观察似乎将光以分布散开。对于金属，这种粗糙表面的一个例子可能是拉丝钢，或大多数铝箔的“混浊”侧。

对于电介质，例如一片玻璃，划痕或其他不规则的表面特征使玻璃模糊了我们通常可以清楚地看到的反射和透射图像。如果表面被严重划伤，我们称之为半透明而不是透明。这是一个有点随意的区别，但通常很清楚我们是否会考虑玻璃半透明或透明。



24.1.3 Diffuse Materials

A material is *diffuse* if it is matte, i.e., not shiny. Many surfaces we see are diffuse, such as most stones, paper, and unfinished wood. To a first approximation, diffuse surfaces can be approximated with a Lambertian (constant) BRDF. Real diffuse materials usually become somewhat specular for grazing angles. This is a subtle effect, but can be important for realism.

24.1.4 Translucent Materials

Many thin objects, such as leaves and paper, both transmit and reflect light diffusely. For all practical purposes no clear image is transmitted by these objects. These surfaces can add a hue shift to the transmitted light. For example, red paper is red because it filters out non-red light for light that penetrates a short distance into the paper, and then scatters back out. The paper also transmits light with a red hue because the same mechanisms apply, but the transmitted light makes it all the way through the paper. One implication of this property is that the transmitted coefficient should be the same in both directions.

24.1.5 Layered Materials

Many surfaces are composed of “layers” or are dielectrics with embedded particles that give the surface a diffuse property (Phong, 1975). The surface of such materials reflects specularly as shown in Figure 24.3, and thus obeys the Fresnel equations. The light that is transmitted is either absorbed or scattered back up to the dielectric surface where it may or may not be transmitted. That light that is transmitted, scattered, and then retransmitted in the opposite direction forms a diffuse “reflection” component.

Note that the diffuse component also is attenuated with the degree of the angle, because the Fresnel equations cause reflection back into the surface as the angle increases as shown in Figure 24.4. Thus, instead of a constant diffuse BRDF, one that vanishes near the grazing angle is more appropriate.

24.2 Implementing Reflection Models

A BRDF model, as described in Section 18.1.6, will produce a rendering which is more physically based than the rendering we get from point light sources and Phong-like models. Unfortunately, real BRDFs are typically quite complicated and cannot be deduced from first principles. Instead, they must either be measured



24.1.3 扩散材料

如果材料是哑光的，即没有光泽，则它是漫反射的。我们看到的许多表面都是弥散的，例如大多数石头，纸张和未完成的木材。对于第一个近似值，漫反射表面可以用朗伯（常数）BRDF近似。对于放牧角度，真正的漫反射材质通常会变得有些镜面。这是一种微妙的效果，但对于现实主义来说可能很重要。

24.1.4 半透明材料

许多薄的物体如树叶和纸都能扩散地透射和反射光。出于所有实际目的，这些物体不会传输清晰的图像。这些表面可以为透射光添加色调偏移。例如，红纸是红色的，因为它过滤掉非红光的光穿透一小段距离进入纸张，然后散射回来。纸张也透射红色色调的光，因为同样的机制适用，但透射的光使它一直通过纸张。此属性的一个含义是，透射系数在两个方向上应该是相同的。

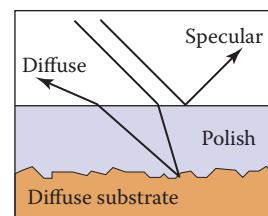


Figure 24.3. Light hitting a layered surface can be reflected specularly, or it can be transmitted and then scatter diffusely off the substrate.

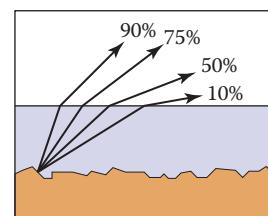
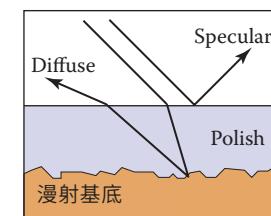


Figure 24.4. The light scattered by the substrate is less and less likely to make it out of the surface as the angle relative to the surface normal increases.



撞击在层状表面上的光可以镜面反射，或者它可以透射然后扩散地散射离开衬底。

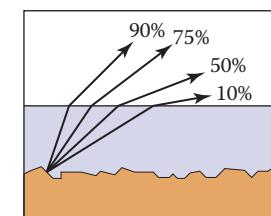


图24.4。随着相对于表面法线的角度增加，由衬底散射的光越来越不太可能使其脱离表面。

24.2 实现反射模型

一个BRDF模型，如第18.1.6节所述，将产生一个比我们从点光源和类似Phong的模型得到的渲染更基于物理的渲染。不幸的是，真正的BRDFs通常非常复杂，不能从第一原则推导出来。相反，它们必须被测量

and directly approximated from raw data, or they must be crudely approximated in an empirical fashion. The latter empirical strategy is what is usually done, and the development of such approximate models is still an area of research. This section discusses several desirable properties of such empirical models.

First, physical constraints imply two properties of a BRDF model. The first constraint is energy conservation:

$$\text{for all } \mathbf{k}_i, R(\mathbf{k}_i) = \int_{\text{all } \mathbf{k}_o} \rho(\mathbf{k}_i, \mathbf{k}_o) \cos \theta_o d\sigma_o \leq 1.$$

If you send a beam of light at a surface from any direction \mathbf{k}_i , then the total amount of light reflected over all directions will be at most the incident amount. The second physical property we expect all BRDFs to have is reciprocity:

$$\text{for all } \mathbf{k}_i, \mathbf{k}_o, \rho(\mathbf{k}_i, \mathbf{k}_o) = \rho(\mathbf{k}_o, \mathbf{k}_i).$$

Second, we want a clear separation between diffuse and specular components. The reason for this is that, although there is a mathematically clean delta function formulation for ideal specular components, delta functions must be implemented as special cases in practice. Such special cases are only practical if the BRDF model clearly indicates what is specular and what is diffuse.

Third, we would like intuitive parameters. For example, one reason the Phong model has enjoyed such longevity is that its diffuse constant and exponent are both clearly related to the intuitive properties of the surface, namely surface color and highlight size.

Finally, we would like the BRDF function to be amenable to Monte Carlo sampling. Recall from Chapter 14 that an integral can be sampled by N random points $x_i \sim p$ where p is defined with the same measure as the integral:

$$\int f(x) d\mu \approx \frac{1}{N} \sum_{j=1}^N \frac{f(x_j)}{p(x_j)}.$$

Recall from Section 18.2 that the surface radiance in direction \mathbf{k}_o is given by a transport equation:

$$L_s(\mathbf{k}_o) = \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i.$$

If we sample directions with pdf $p(\mathbf{k}_i)$ as discussed in Chapter 23, then we can approximate the surface radiance with samples:

$$L_s(\mathbf{k}_o) \approx \frac{1}{N} \sum_{j=1}^N \frac{\rho(\mathbf{k}_j, \mathbf{k}_o) L_f(\mathbf{k}_j) \cos \theta_j}{p(\mathbf{k}_j)}.$$

直接从原始数据中近似，或者它们必须以经验的方式粗略地近似。后一种经验策略是通常所做的，而这种近似模型的开发仍然是一个研究领域。本节讨论此类经验模型的几个理想属性。

首先，物理约束意味着BRDF模型的两个属性。第一个约束是节能：

$$\text{对于所有 } \mathbf{k}_i, R(\mathbf{k}_i) = \int_{\text{all } \mathbf{k}_o} \rho(\mathbf{k}_i, \mathbf{k}_o) \cos \theta_o d\sigma_o \leq 1.$$

如果从任何方向 \mathbf{k}_i 在表面处发送一束光，那么在所有方向上反射的光的总量将至多为入射量。我们期望所有BRDFs都具有的第二个物理性质是互惠性：

$$\text{对于所有 } \mathbf{k}_i, \mathbf{k}_o, \rho(\mathbf{k}_i, \mathbf{k}_o) = \rho(\mathbf{k}_o, \mathbf{k}_i).$$

其次，我们希望在漫反射和镜面成分之间进行清晰的分离。这样做的原因是，尽管对于理想的镜面分量存在数学上干净的delta函数公式，但delta函数在实践中必须作为特殊情况实现。只有当BRDF模型清楚地表明什么是镜面和什么是漫反射时，这种特殊情况才实用。

第三，我们想要直观的参数。例如，Phong模型享有如此长寿的一个原因是其漫射常数和指数都与表面的直观属性（即表面颜色和高光尺寸）明显相关。

最后，我们希望BRDF函数适用于蒙特卡罗采样。回顾第14章，积分可以由 N 个随机点 $x_i \in p$ 采样，其中 p 定义为与积分相同的度量：

$$\int f(x) d\mu \approx \frac{1}{N} \sum_{j=1}^N \frac{f(x_j)}{p(x_j)}.$$

从第18.2节中可以看出， \mathbf{k}_o 方向的表面辐射度由运输方程给出：

$$L_s(\mathbf{k}_o) = \int_{\text{all } \mathbf{k}_i} \rho(\mathbf{k}_i, \mathbf{k}_o) L_f(\mathbf{k}_i) \cos \theta_i d\sigma_i.$$

如果我们像第23章中所讨论的那样用 $p(\mathbf{k}_i)$ 对方向进行采样，那么我们可以用样本近似地表辐射：

$$L_s(\mathbf{k}_o) \approx \frac{1}{N} \sum_{j=1}^N \frac{\rho(\mathbf{k}_j, \mathbf{k}_o) L_f(\mathbf{k}_j) \cos \theta_j}{p(\mathbf{k}_j)}.$$



This approximation will converge for any p that is nonzero where the integrand is nonzero. However, it will only converge well if the integrand is not very large relative to p . Ideally, $p(\mathbf{k})$ should be approximately shaped like the integrand $\rho(\mathbf{k}_j, \mathbf{k}_o)L_f(\mathbf{k}_j)\cos\theta_j$. In practice, L_f is complicated, and the best we can accomplish is to have $p(\mathbf{k})$ shaped somewhat like $\rho(\mathbf{k}, \mathbf{k}_o)L_f(\mathbf{k})\cos\theta$.

For example, if the BRDF is Lambertian, then it is constant and the “ideal” $p(\mathbf{k})$ is proportional to $\cos\theta$. Because the integral of p must be one, we can deduce the leading constant:

$$\int_{\text{all } \mathbf{k} \text{ with } \theta < \pi/2} C \cos\theta d\sigma = 1.$$

This implies that $C = 1/\pi$, so we have

$$p(\mathbf{k}) = \frac{1}{\pi} \cos\theta.$$

An acceptably efficient implementation will result as long as p doesn’t get too small when the integrand is nonzero. Thus, the constant pdf will also suffice:

$$p(\mathbf{k}) = \frac{1}{2\pi}.$$

This emphasizes that many pdfs may be acceptable for a given BRDF model.

24.3 Specular Reflection Models

For a metal, we typically specify the reflectance at normal incidence $R_0(\lambda)$. The reflectance should vary according to the Fresnel equations, and a good approximation is given by (Schlick, 1994a)

$$R(\theta, \lambda) = R_0(\lambda) + (1 - R_0(\lambda))(1 - \cos\theta)^5.$$

This approximation allows us to just set the normal reflectance of the metal either from data or by eye.

For a dielectric, the same formula works for reflectance. However, we can set $R_0(\lambda)$ in terms of the refractive index $n(\lambda)$:

$$R_0(\lambda) = \left(\frac{n(\lambda) - 1}{n(\lambda) + 1} \right)^2.$$

Typically, n does not vary with wavelength, but for applications where dispersion is important, n can vary. The refractive indices that are often useful include water ($n = 1.33$), glass ($n = 1.4$ to $n = 1.7$), and diamond ($n = 2.4$).



该近似值将收敛于被积函数非零的任何非零 p 。但是，只有当积分相对于 p 不是很大时，它才会很好地收敛。理想情况下， $p(\mathbf{k})$ 应该近似形状为积分数 $\rho(\mathbf{k}_j, \mathbf{k}_o)L_f(\mathbf{k}_j)\cos\theta_j$ 。在实践中， L_f 是复杂的，我们可以 accomplish 最好的是有 $p(\mathbf{k})$ 形状有点像 $\rho(\mathbf{k}, \mathbf{k}_o)L_f(\mathbf{k})\cos\theta$ 。

例如，如果BRDF是朗伯，那么它是恒定的，“理想” $p(\mathbf{k})$ 与 $\cos\theta$ 成正比。因为 p 的积分必须是一，我们可以推导出前导常数：

$$\int_{\theta < \pi/2 \text{ 的所有 } \mathbf{k}} C \cos\theta d\sigma = 1.$$

这意味着 $C=1/\pi$ ，所以我们有

$$p(\mathbf{k}) = \frac{1}{\pi} \cos\theta.$$

只要当被积函数非零时， p 不会变得太小，就会产生可接受的高效实现。因此，常数pdf也就足够了：

$$p(\mathbf{k}) = \frac{1}{2\pi}.$$

这强调了许多pdf对于给定的BRDF模型可能是可接受的。

24.3 镜面反射模型

对于金属，我们通常指定法向入射时的反射率 $R_0(\lambda)$ 。反射率应该根据菲涅耳方程而变化，并且由 (Schlick, 1994a) 给出了一个很好的近似值

$$R(\theta, \lambda) = R_0(\lambda) + (1 - R_0(\lambda))(1 - \cos\theta)^5.$$

这个近似值允许我们根据数据或眼睛设置金属的正常反射率。

对于电介质，相同的公式适用于反射率。但是，我们可以设置以折射率 $n(\lambda)$ 计的 $r_0(\lambda)$ ：

$$R_0(\lambda) = \left(\frac{n(\lambda) - 1}{n(\lambda) + 1} \right)^2.$$

通常， n 不随波长而变化，但对于色散很重要的应用， n 可以变化。通常有用的折射率包括水 ($n=1.33$)，玻璃 ($n=1.4$ 到 $n=1.7$)，以及菱形 ($n=2.4$)。

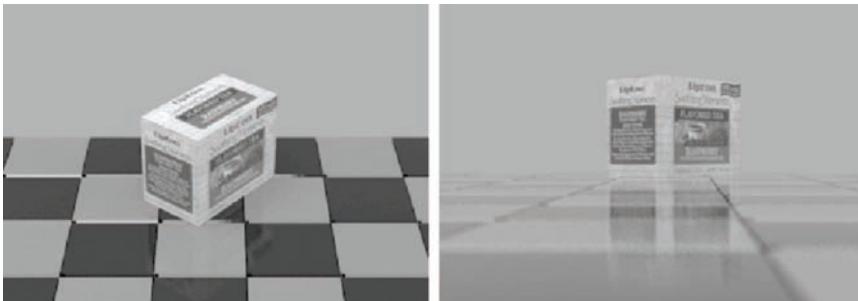


Figure 24.5. Renderings of polished tiles using coupled model. These images were produced using a Monte Carlo path tracer. The sampling distribution for the diffuse term is $\cos \theta / \pi$.

24.4 Smooth-Layered Model

Reflection in matte/specular materials, such as plastics or polished woods, is governed by Fresnel equations at the surface and by scattering within the subsurface. An example of this reflection can be seen in the tiles in the renderings in Figure 24.5. Note that the blurring in the specular reflection is mostly vertical due to the compression of apparent bump spacing in the view direction. This effect causes the vertically streaked reflections seen on lakes on windy days; it can either be modeled using explicit microgeometry and a simple smooth-surface reflection model or by a more general model that accounts for this asymmetry.

We could use the traditional Lambertian-specular model for the tiles, which linearly mixes specular and Lambertian terms. In standard radiometric terms, this can be expressed as

$$\rho(\theta, \phi, \theta', \phi' \lambda) = \frac{R_d(\lambda)}{\pi} + R_s \rho_s(\theta, \phi, \theta', \phi'),$$

where $R_d(\lambda)$ is the hemispherical reflectance of the matte term, R_s is the specular reflectance, and ρ_s is the normalized specular BRDF (a weighted Dirac delta function on the sphere). This equation is a simplified version of the BRDF where R_s is independent of wavelength. The independence of wavelength causes a highlight that is the color of the luminaire, so a polished rather than a metal appearance will be achieved. Ward (G. J. Ward, 1992) suggests to set $R_d(\lambda) + R_s \leq 1$ in order to conserve energy. However, such models with constant R_s fail to show the increase in specularity for steep viewing angles. This is the key point: in the real world the relative proportions of matte and specular appearance change with the viewing angle.

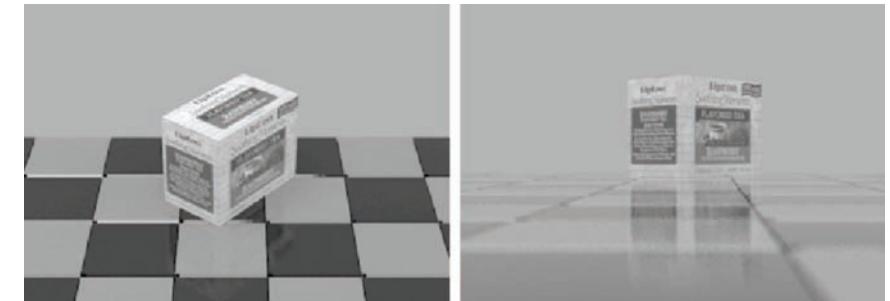


图24.5。使用耦合模型抛光瓷砖的效果图。这些图像是使用蒙特卡罗路径示踪剂产生的。漫射项的采样分布为 $\cos\theta/\pi$ 。

24.4 Smooth-Layered Model

哑光镜面材料（如塑料或抛光木材）中的反射由表面的菲涅耳方程和次表面内的散射控制。这种反射的一个例子可以在图24.5的效果图中的图块中看到。请注意，镜面反射中的模糊主要是垂直的，这是由于在视图方向上明显的凹凸间距的压缩。这种效应导致大风天在湖泊上看到的垂直条纹反射；它可以使用显式微观测量和简单的光滑表面反射模型进行建模，也可以使用解释这种不对称性的更

我们可以对图块使用传统的朗伯镜面模型，它将镜面和朗伯项线性混合在一起。在标准辐射术语中，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 $R_d(\lambda)$ 是哑光项的半球反射率， R_s 是镜面反射率， ρ_s 是归一化镜面BRDF（球体上的加权狄拉克delta函数）。该方程是BRDF的简化版本，其中 R_s 与波长无关。波长的独立性导致了一个亮点，即灯具的颜色，因此将实现抛光而不是金属外观。Ward(G.J.Ward 1992)建议设定 $R_d(\lambda)+R_s \leq 1$ 以节约能源。然而，具有恒定 R_s 的此类模型未能显示对于陡峭视角的镜面增加。这是关键点：在现实世界中，哑光和镜面外观的相对比例随视角而变化。

One way to simulate the change in the matte appearance is to explicitly dampen $R_d(\lambda)$ as R_s increases (Shirley, 1991):

$$\rho(\theta, \phi, \theta', \phi', \lambda) = R_f(\theta)\rho_s(\theta, \phi, \theta', \phi') + \frac{R_d(\lambda)(1 - R_f(\theta))}{\pi},$$

where $R_f(\theta)$ is the Fresnel reflectance for a polish-air interface. The problem with this equation is that it is not reciprocal, as can be seen by exchanging θ and θ' ; this changes the value of the matte damping factor because of the multiplication by $(1 - R_f(\theta))$. The specular term, a scaled Dirac delta function, is reciprocal, but this does not make up for the non-reciprocity of the matte term. Although this BRDF works well, its lack of reciprocity can cause some rendering methods to have ill-defined solutions.

We now present a model that produces the matte/specular tradeoff while remaining reciprocal and energy conserving. Because the key feature of the new model is that it couples the matte and specular scaling coefficients, it is called a *coupled* model (Shirley, Smits, Hu, & Lafourture, 1997).

Surfaces which have a glossy appearance are often a clear dielectric, such as polyurethane or oil, with some subsurface structure. The specular (mirror-like) component of the reflection is caused by the smooth dielectric surface and is independent of the structure below this surface. The magnitude of this specular term is governed by the Fresnel equations.

The light that is not reflected specularly at the surface is transmitted through the surface. There, either it is absorbed by the subsurface, or it is reflected from a pigment or a subsurface and transmitted back through the surface of the polish. This transmitted light forms the matte component of reflection. Since the matte component can only consist of the light that is transmitted, it will naturally decrease in total magnitude for increasing angle.

To avoid choosing between physically plausible models and models with good qualitative behavior over a range of incident angles, note that the Fresnel equations that account for the specular term, $R_f(\theta)$, are derived directly from the physics of the dielectric-air interface. Therefore, the problem must lie in the matte term. We could use a full-blown simulation of subsurface scattering as implemented, but this technique is both costly and requires detailed knowledge of subsurface structure, which is usually neither known nor easily measurable. Instead, we can modify the matte term to be a simple approximation that captures the important qualitative angular behavior shown in Figure 24.4.

Let us assume that the matte term is not Lambertian, but instead is some other function that depends only on θ , θ' and λ : $\rho_m(\theta, \theta', \lambda)$. We discard behavior that depends on ϕ or ϕ' in the interest of simplicity. We try to keep the formulas reasonably simple because the physics of the matte term is complicated and

模拟哑光外观变化的一种方法是显式阻尼
 $R_d(\lambda)$ 随着 R_s 的增加 (Shirley, 1991) :

$$\rho(\theta, \phi, \theta', \phi', \lambda) = R_f(\theta)\rho_s(\theta, \phi, \theta', \phi') + \frac{R_d(\lambda)(1 - R_f(\theta))}{\pi},$$

其中 $R_f(\theta)$ 是波兰-空气界面的菲涅耳反射率。这个方程的问题是它不是倒数的，通过交换 θ 和 θ' 可以看出；这改变了哑光阻尼因子的值，因为乘以 $(1 - R_f(\theta))$ 。镜面项，一个缩放的狄拉克德尔塔函数，是互易的，但这并不能弥补磨砂项的非互易性。虽然这种BRDF工作得很好，但它缺乏互惠性会导致一些渲染方法具有不明确的解决方案。

我们现在提出一个模型，产生哑光镜面折衷，同时重新主导互惠和节能。因为新模型的关键特征是它耦合了哑光和镜面缩放系数，所以它被称为耦合模型 (Shirley, Smits, Hu, & Lafourture, 1997)。

具有光泽外观的表面通常是清晰的电介质，如聚氨酯或油，具有一些次表面结构。反射的镜面（镜面状）分量是由光滑的介电表面引起的，并且独立于该表面下面的结构。此镜面项的大小由菲涅耳方程控制。

面处未镜面反射的光透射通过所述表面。在那里，或者它被亚表面吸收，或者它从颜料或亚表面反射并通过 polish 的表面传回。这种透射光形成反射的哑光分量。由于哑光组件只能由透射的光组成，因此随着角度的增加，它的总幅度自然会减小。

为了避免在物理上似是而非的模型和在一定入射角范围内具有良好定性行为的模型之间进行选择，请注意，占镜面项 $r_f(\theta)$ 的菲涅耳等离子直接来自介质-空气界面的物理场。因此，问题必须出在短期内。我们可以使用完整的次表面散射模拟来实现，但这种技术既昂贵又需要详细的次表面结构知识，这通常既不知道也不容易测量。相反，我们可以将哑光项修改为一个简单的近似值，它捕获了图 24.4 所示的重要定性角行为。

让我们假设哑光项不是朗伯项，而是其他一些仅依赖于 θ , θ' 和 λ 的函数： $\rho_m(\theta, \theta', \lambda)$ 。为了简单起见，我们放弃了依赖于 ϕ 或 ϕ' 的行为。我们试图保持 formulas 合理简单，因为哑光术语的物理学是复杂的，并且

sometimes requires unknown parameters. We expect the matte term to be close to constant, and roughly rotationally symmetric (He et al., 1992).

An obvious candidate for the matte component $\rho_m(\theta, \theta', \lambda)$ that will be reciprocal is the *separable* form $kR_m(\lambda)f(\theta)f(\theta')$ for some constant k and matte reflectance parameter $R_m(\lambda)$. We could merge k and $R_m(\lambda)$ into a single term, but we choose to keep them separated because this makes it more intuitive to set $R_m(\lambda)$ —which must be between 0 and 1 for all wavelengths. Separable BRDFs have been shown to have several computational advantages, thus we use the separable model:

$$\rho(\theta, \phi, \theta', \phi', \lambda) = R_f(\theta)\rho_s(\theta, \phi, \theta', \phi') + kR_m(\lambda)f(\theta)f(\theta').$$

We know that the matte component can only contain energy not reflected in the surface (specular) component. This means that for $R_m(\lambda) = 1$, the incident and reflected energy are the same, which suggests the following constraint on the BRDF for each incident θ and λ :

$$R_f(\theta) + 2\pi k f(\theta) \int_0^{\frac{\pi}{2}} f(\theta') \cos \theta' \sin \theta' d\theta' = 1. \quad (24.1)$$

We can see that $f(\theta)$ must be proportional to $(1 - R_f(\theta))$. If we assume that matte components that absorb some energy have the same directional pattern as this ideal, we get a BRDF of the form

$$\rho(\theta, \phi, \theta', \phi', \lambda) = R_f(\theta)\rho_s(\theta, \phi, \theta', \phi') + kR_m(\lambda)[1 - R_f(\theta)][1 - R_f(\theta')].$$

We could now insert the full form of the Fresnel equations to get $R_f(\theta)$, and then use energy conservation to solve for constraints on k . Instead, we will use the approximation discussed in Section 24.1.1. We find that

$$f(\theta) \propto (1 - (1 - \cos \theta)^5).$$

Applying Equation (24.1) yields

$$k = \frac{21}{20\pi(1 - R_0)}. \quad (24.2)$$

The full coupled BRDF is then

$$\begin{aligned} \rho(\theta, \phi, \theta', \phi', \lambda) = & \\ & [R_0 + (1 - \cos \theta)^5(1 - R_0)]\rho_s(\theta, \phi, \theta', \phi') + \\ & kR_m(\lambda)[1 - (1 - \cos \theta)^5][1 - (1 - \cos \theta')^5]. \end{aligned} \quad (24.3)$$

有时需要未知参数。我们期望哑光项接近恒定，并且大致旋转对称 (He et al. 1992).

将被重新ciprocal的哑光分量 $\rho_m(\theta, \theta', \lambda)$ 的明显候选者是对于一些常数 k 和哑光反射率参数 $R_m(\lambda)$ 的可分离形式 $kR_m(\lambda)f(\theta)f(\theta')$ 。我们可以将 k 和 $R_m(\lambda)$ 合并为一个项，但我们选择将它们分开，因为这使得设置 $R_m(\lambda)$ 更直观—所有波长的 R_m 必须在 0 和 1 之间。可分离的BRDFs 已被证明具有几个计算优势，因此我们使用separable模型：

$$\rho(\theta, \phi, \theta', \phi', \lambda) = R_f(\theta)\rho_s(\theta, \phi, \theta', \phi') + kR_m(\lambda)f(\theta)f(\theta').$$

我们知道，哑光组件只能包含未在表面（镜面）组件中反射的能量。这意味着对于 $R_m(\lambda) = 1$ ，入射和反射能量是相同的，这表明每个入射 θ 和 λ 对 BRDF 的以下约束：

$$R_f(\theta) + 2\pi k f(\theta) \int_0^{\frac{\pi}{2}} f(\theta') \cos \theta' \sin \theta' d\theta' = 1. \quad (24.1)$$

我们可以看到 $f(\theta)$ 必须与 $(1 - R_f(\theta))$ 成比例。如果我们假设吸收一些能量的哑光组件具有与此理想相同的方向图案，我们得到形式的BRDF

$$\rho(\theta, \phi, \theta', \phi', \lambda) = R_f(\theta)\rho_s(\theta, \phi, \theta', \phi') + kR_m(\lambda)[1 - R_f(\theta)][1 - R_f(\theta')].$$

我们现在可以插入菲涅耳方程的完整形式以获得 $R_f(\theta)$ ，然后使用能量守恒求解 k 的约束。相反，我们将使用第 24.1.1 节中讨论的近似值，我们发现

$$f(\theta) \propto (1 - (1 - \cos \theta)^5).$$

应用方程(24.1)收益率

$$k = \frac{21}{20\pi(1 - R_0)}. \quad (24.2)$$

然后将全耦合的BRDF

$$\begin{aligned} \rho(\theta, \phi, \theta', \phi', \lambda) = & \\ & [R_0 + (1 - \cos \theta)^5(1 - R_0)]\rho_s(\theta, \phi, \theta', \phi') + \\ & kR_m(\lambda)[1 - (1 - \cos \theta)^5][1 - (1 - \cos \theta')^5]. \end{aligned} \quad (24.3)$$



The results of running the coupled model is shown in Figure 24.5. Note that for the high viewpoint, the specular reflection is almost invisible, but it is clearly visible in the low-angle photograph image, where the matte behavior is less obvious.

For reasonable values of refractive indices, R_0 is limited to approximately the range 0.03 to 0.06 (the value $R_0 = 0.05$ was used for Figure 24.5). The value of R_s in a traditional Phong model is harder to choose, because it typically must be tuned for viewpoint in static images and tuned for a particular camera sequence for animations. Thus, the coupled model is easier to use in a “hands-off” mode.

24.5 Rough-Layered Model

The previous model is fine if the surface is smooth. However, if the surface is not ideal, some spread is needed in the specular component. An extension of the coupled model to this case is presented here (Ashikhmin & Shirley, 2000). At a given point on a surface, the BRDF is a function of two directions, one in the direction toward the light and one in the direction toward the viewer. We would like to have a BRDF model that works for “common” surfaces, such as metal and plastic, and has the following characteristics:

1. **Plausible.** As defined by Lewis (R. R. Lewis, 1994), this refers to the BRDF obeying energy conservation and reciprocity.
2. **Anisotropy.** The material should model simple anisotropy, such as seen on brushed metals.
3. **Intuitive parameters.** For material, such as plastics, there should be parameters R_d for the substrate and R_s for the normal specular reflectance as well as two roughness parameters n_u and n_v .
4. **Fresnel behavior.** Specularity should increase as the incident angle decreases.
5. **Non-Lambertian diffuse term.** The material should allow for a diffuse term, but the component should be non-Lambertian to assure energy conservation in the presence of Fresnel behavior.
6. **Monte Carlo friendliness.** There should be some reasonable probability density function that allows straightforward Monte Carlo sample generation for the BRDF.



运行耦合模型的结果如图24.5所示。请注意，对于高视点，镜面反射几乎不可见，但在低角度照片图像中清晰可见，其中哑光行为不太明显。

对于折射率的合理值， R_0 被限制为近似范围0. 03至0. 06(值 $R_0=0. 05$ 用于图24.5)。传统Phong模型中的 R_s 值较难选择，因为它通常必须针对静态图像中的视点进行调整，并针对动画的特定摄像机序列进行调整。因此，耦合模型更容易在“放手”模式下使用。

24.5 Rough-Layered Model

如果表面光滑，则以前的模型很好。但是，如果表面不理想，则需要在镜面组件中进行一些铺展。这里介绍了耦合模型对这种情况的扩展 (Ashikhmin & Shirley, 2000)。在表面上的给定点处，BRDF是两个方向的函数，一个方向朝向光线，另一个方向朝向观看者。我们希望有一个BRDF模型，适用于“常见”表面，如金属和塑料，并具有以下特征：

- 1.似是而非。正如Lewis (R.R.Lewis, 1994) 所定义的，这是指BRDF服从节能和互惠。
- 2.各向异性。材料应该模拟简单的各向异性，如在拉丝金属上看到的。
- 3.直观的参数。对于塑料等材料，应该有衬底的参数 R_d 和正常镜面反射率的参数 R_s 以及两个粗糙度参数 n_u 和 n_v 。
- 4.菲涅耳行为。镜面度应随着入射角的减小而增大。
- 5.非朗伯漫术语。材料应该允许漫射项，但组件应该是非朗伯，以确保在存在菲涅耳行为的情况下能量守恒。
- 6.蒙特卡罗友好。应该有一些合理的概率密度函数，允许直接生成BRDF的蒙特卡罗样本。

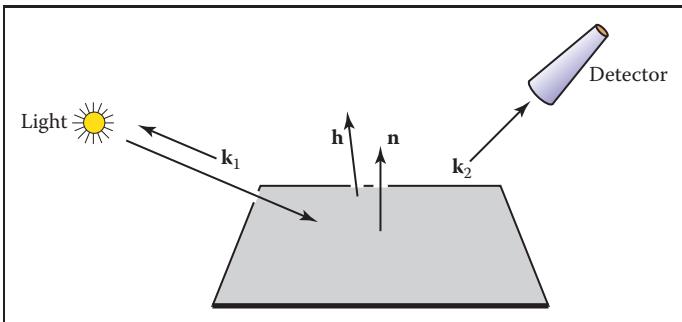


Figure 24.6. Geometry of reflection. Note that \mathbf{k}_1 , \mathbf{k}_2 , and \mathbf{h} share a plane, which usually does not include \mathbf{n} .

A BRDF with these properties is a Fresnel-weighted, Phong-style cosine lobe model that is anisotropic.

We again decompose the BRDF into a specular component and a diffuse component (Figure 24.6). Accordingly, we write our BRDF as the classical sum of two parts:

$$\rho(\mathbf{k}_1, \mathbf{k}_2) = \rho_s(\mathbf{k}_1, \mathbf{k}_2) + \rho_d(\mathbf{k}_1, \mathbf{k}_2), \quad (24.4)$$

where the first term accounts for the specular reflection (this will be presented in the next section). While it is possible to use the Lambertian BRDF for the diffuse term $\rho_d(\mathbf{k}_1, \mathbf{k}_2)$ in our model, we will discuss a better solution in Section 24.5.2 and how to implement the model in Section 24.5.3. Readers who just want to implement the model should skip to that section.

24.5.1 Anisotropic Specular BRDF

To model the specular behavior, we use a Phong-style specular lobe but make this lobe anisotropic and incorporate Fresnel behavior while attempting to preserve the simplicity of the initial model. This BRDF is

$$\rho(\mathbf{k}_1, \mathbf{k}_2) = \frac{\sqrt{(n_u + 1)(n_v + 1)}}{8\pi} \frac{(\mathbf{n} \cdot \mathbf{h})^{n_u} \cos^2 \phi + n_v \sin^2 \phi}{(\mathbf{h} \cdot \mathbf{k}_i) \max(\cos \theta_i, \cos \theta_o)} F(\mathbf{k}_i \cdot \mathbf{h}). \quad (24.5)$$

Again we use Schlick's approximation to the Fresnel equation:

$$F(\mathbf{k}_i \cdot \mathbf{h}) = R_s + (1 - R_s)(1 - (\mathbf{k}_i \cdot \mathbf{h}))^5, \quad (24.6)$$

where R_s is the material's reflectance for the normal incidence. Because $\mathbf{k}_i \cdot \mathbf{h} = \mathbf{k}_o \cdot \mathbf{h}$, this form is reciprocal. We have an empirical model whose terms are

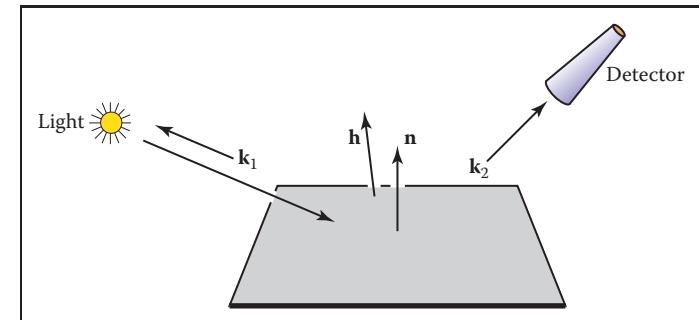


图24.6。反射的几何形状。请注意， \mathbf{k}_1 ， \mathbf{k}_2 和 \mathbf{h} 共享一个平面，该平面通常不包括 \mathbf{n} 。

具有这些属性的BRDF是各向异性的菲涅耳加权、Phong式余弦瓣模型。

我们再次将BRDF分解为镜面组件和漫射共聚体（图24.6）。因此，我们将BRDF写为两个部分的经典总和：

$$\rho(\mathbf{k}_1, \mathbf{k}_2) = \rho_s(\mathbf{k}_1, \mathbf{k}_2) + \rho_d(\mathbf{k}_1, \mathbf{k}_2), \quad (24.4)$$

其中第一个项说明镜面反射（这将在下一节中介绍）。虽然可以在我们的模型中对扩散项 $\rho_d(\mathbf{k}_1, \mathbf{k}_2)$ 使用朗伯BRDF，但我们在第24.5.2节中讨论更好的解决方案，以及如何在第24.5.3节中实现模型。只想实现模型的读者应该跳到该部分。

24.5.1 Anisotropic Specular BRDF

为了模拟镜面行为，我们使用了一个Phong风格的镜面瓣，但使这个瓣各向异性，并结合菲涅尔行为，同时试图保持初始模式的简单性。这个BRDF是

$$\rho(\mathbf{k}_1, \mathbf{k}_2) = \frac{\sqrt{(n_u + 1)(n_v + 1)}}{8\pi} \frac{(\mathbf{n} \cdot \mathbf{h})^{n_u} \cos^2 \phi + n_v \sin^2 \phi}{(\mathbf{h} \cdot \mathbf{k}_i) \max(\cos \theta_i, \cos \theta_o)} F(\mathbf{k}_i \cdot \mathbf{h}). \quad (24.5)$$

我们再次使用Schlick对菲涅耳方程的近似：

$$F(\mathbf{k}_i \cdot \mathbf{h}) = R_s + (1 - R_s)(1 - (\mathbf{k}_i \cdot \mathbf{h}))^5, \quad (24.6)$$

其中 R_s 是材料对法线入射的反射率。因为 $\mathbf{k}_i \cdot \mathbf{h} = \mathbf{k}_o \cdot \mathbf{h}$ ，所以这种形式是倒数的。我们有一个经验模型，其术语是

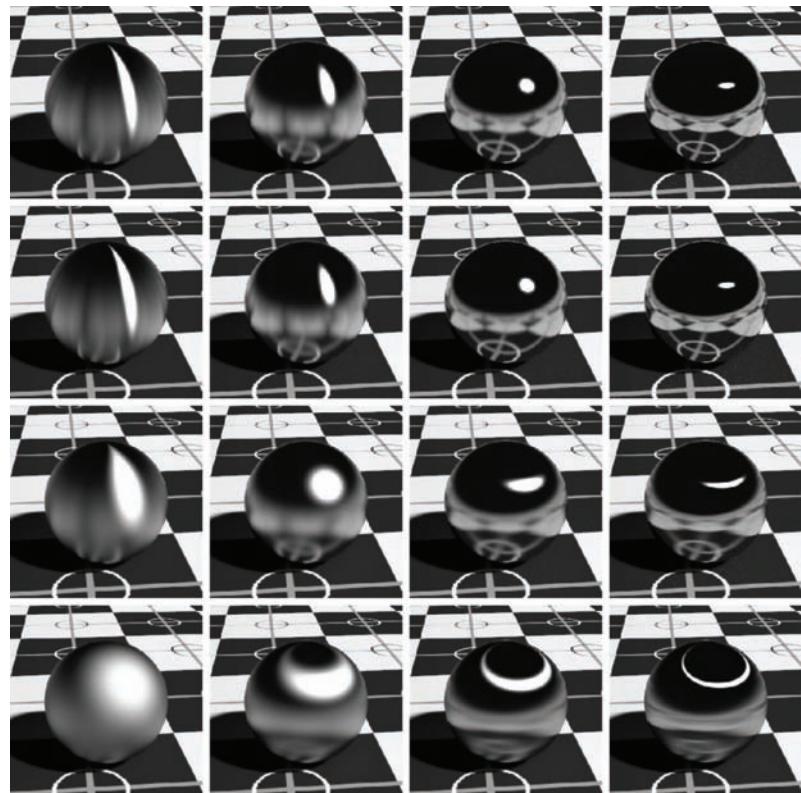


Figure 24.7. Metallic spheres for exponents 10, 100, 1000, and 10,000 increasing both left to right and top to bottom.

chosen to enforce energy conservation and reciprocity. A full rationalization for the terms is given in the paper by Ashikhmin, listed in the chapter notes.

The specular BRDF of Equation (24.5) is useful for representing metallic surfaces where the diffuse component of reflection is very small. Figure 24.7 shows a set of metal spheres on a texture-mapped Lambertian plane. As the values of parameters n_u and n_v change, the appearance of the spheres shift from rough metal to almost perfect mirror, and from highly anisotropic to the more familiar Phong-like behavior.

24.5.2 Diffuse Term for the Anisotropic Phong Model

It is possible to use a Lambertian BRDF together with the anisotropic specular term; this is done for most models, but it does not necessarily conserve energy. A

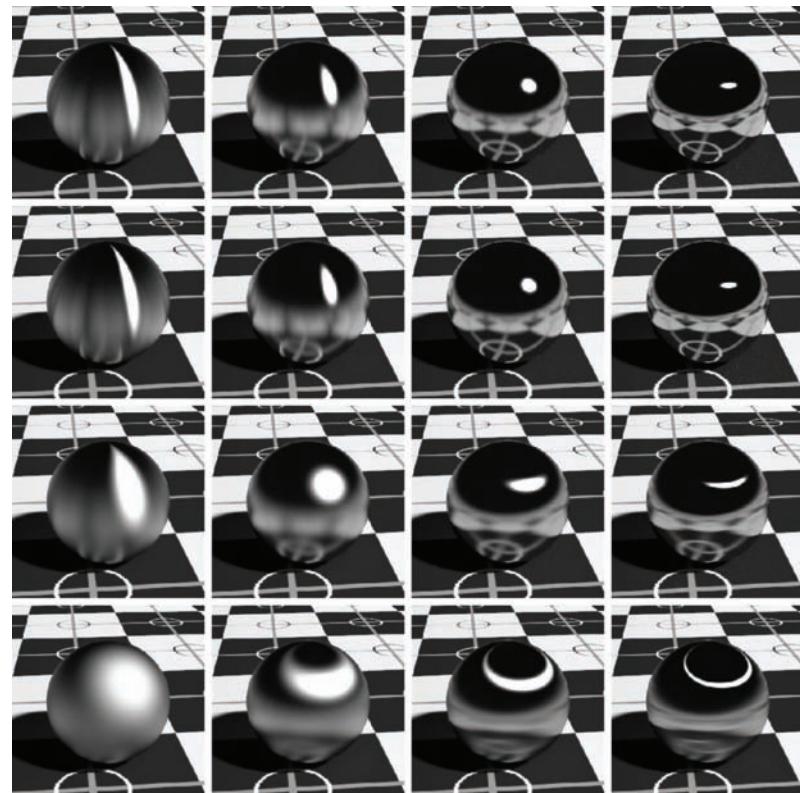


图24.7。指数10、100、1000和10 000的金属球体从左到右和从上到下增加。

选择执行节能和互惠。Ashikhmin在章节注释中列出的论文中给出了这些术语的全面合理化。

方程(24.5)的镜面BRDF可用于表示反射漫射分量非常小的金属表面。图24.7显示了纹理映射的朗伯平面上的一组金属球体。随着参数 n_u 和 n_v 值的变化，球体的外观从粗糙的金属转变为近乎完美的镜面，从高度各向异性转变为更熟悉的Phong样行为。

24.5.2 各向异性Phong模型的扩散项

可以将朗伯BRDF与各向异性镜面项一起使用；这对大多数模型都是如此，但不一定能节约能源。A

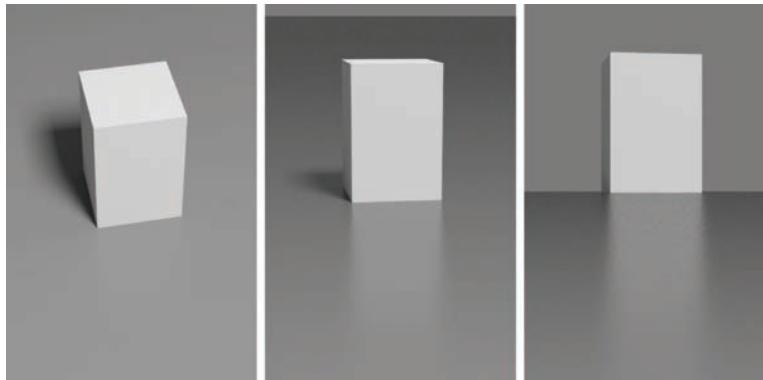


Figure 24.8. Three views for $n_u = n_v = 400$ and a diffuse substrate. Note the change in intensity of the specular reflection.

better approach is a simple angle-dependent form of the diffuse component which accounts for the fact that the amount of energy available for diffuse scattering varies due to the dependence of the specular term's total reflectance on the incident angle. In particular, diffuse color of a surface disappears near the grazing angle, because the total specular reflectance is close to one. This well-known effect cannot be reproduced with a Lambertian diffuse term and is therefore missed by most reflection models.

Following a similar approach to the coupled model, we can find a form of the diffuse term that is compatible with the anisotropic Phong lobe:

$$\rho_d(\mathbf{k}_1, \mathbf{k}_2) = \frac{28R_d}{23\pi} (1 - R_s) \left(1 - \left(1 - \frac{\cos \theta_i}{2}\right)^5\right) \left(1 - \left(1 - \frac{\cos \theta_o}{2}\right)^5\right). \quad (24.7)$$

Here R_d is the diffuse reflectance for normal incidence, and R_s is the Phong lobe coefficient. An example using this model is shown in Figure 24.8.

24.5.3 Implementing the Model

Recall that the BRDF is a combination of diffuse and specular components:

$$\rho(\mathbf{k}_1, \mathbf{k}_2) = \rho_s(\mathbf{k}_1, \mathbf{k}_2) + \rho_d(\mathbf{k}_1, \mathbf{k}_2). \quad (24.8)$$

The diffuse component is given in Equation (24.7); the specular component is given in Equation (24.5). It is not necessary to call trigonometric functions to

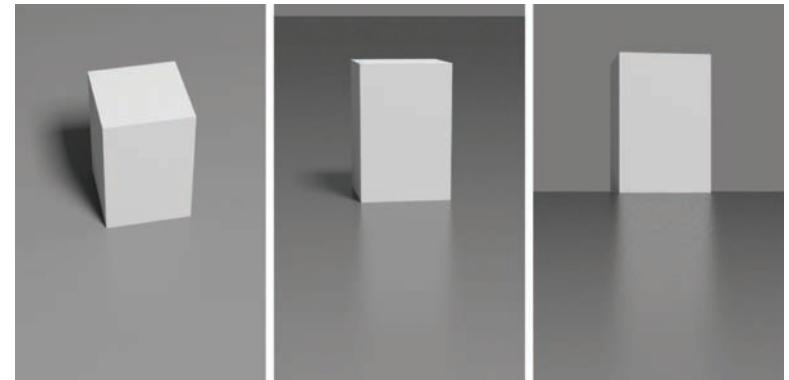


图24.8。Nu=nv=400的三个视图和一个漫射衬底。注意镜面反射强度的变化。

更好的方法是漫射分量的简单角度依赖形式，该形式考虑了由于镜面项的总反射率对inci凹痕角的依赖性，可用于漫散射的能量量而变化的事实。特别是，表面的漫射颜色在掠射角附近消失，因为总镜面反射率接近一。这种众所周知的effect不能用朗伯漫射项再现，因此大多数反射模型都忽略了它。

按照耦合模型的类似方法，我们可以找到一种与各向异性Phong瓣兼容的扩散项形式：

$$\rho_d(\mathbf{k}_1, \mathbf{k}_2) = \frac{28R_d}{23\pi} (1 - R_s) \left(1 - \left(1 - \frac{\cos \theta_i}{2}\right)^5\right) \left(1 - \left(1 - \frac{\cos \theta_o}{2}\right)^5\right).$$

(24.7) 这里Rd是正常入射的漫反射率，Rs是Phonglobe系数。使用该模

型的示例如图24.8所示。

24.5.3 实施模型

回想一下，BRDF是漫反射和镜面组件的组合：

$$\rho(\mathbf{k}_1, \mathbf{k}_2) = \rho_s(\mathbf{k}_1, \mathbf{k}_2) + \rho_d(\mathbf{k}_1, \mathbf{k}_2). \quad (24.8)$$

漫射分量在等式(24.7)中给出；镜面分量在等式(24.5)中给出。没有必要调用三角函数来

compute the exponent, so the specular BRDF can be written:

$$\rho(\mathbf{k}_1, \mathbf{k}_2) = \frac{\sqrt{(n_u + 1)(n_v + 1)}}{8\pi} (\mathbf{n} \cdot \mathbf{h})^{\frac{(n_u(\mathbf{h} \cdot \mathbf{u})^2 + n_v(\mathbf{h} \cdot \mathbf{v})^2)/(1 - (\mathbf{h} \cdot \mathbf{n})^2)}{(h \cdot k_i) \max(\cos \theta_i, \cos \theta_o)}} F(\mathbf{k}_i \cdot \mathbf{h}). \quad (24.9)$$

In a Monte Carlo setting, we are interested in the following problem: given \mathbf{k}_1 , generate samples of \mathbf{k}_2 with a distribution whose shape is similar to the cosine-weighted BRDF. Note that greatly undersampling a large value of the integrand is a serious error, while greatly oversampling a small value is acceptable in practice. The reader can verify that the densities suggested below have this property.

A suitable way to construct a pdf for sampling is to consider the distribution of half vectors that would give rise to our BRDF. Such a function is

$$p_h(\mathbf{h}) = \frac{\sqrt{(n_u + 1)(n_v + 1)}}{2\pi} (\mathbf{n}\mathbf{h})^{n_u \cos^2 \phi + n_v \sin^2 \phi}, \quad (24.10)$$

where the constants are chosen to ensure it is a valid pdf.

We can just use the probability density function $p_h(\mathbf{h})$ of Equation (24.10) to generate a random \mathbf{h} . However, to evaluate the rendering equation, we need both a reflected vector \mathbf{k}_o and a probability density function $p(\mathbf{k}_o)$. It is important to note that if you generate \mathbf{h} according to $p_h(\mathbf{h})$ and then transform to the resulting \mathbf{k}_o :

$$\mathbf{k}_o = -\mathbf{k}_i + 2(\mathbf{k}_i \cdot \mathbf{h})\mathbf{h}, \quad (24.11)$$

the density of the resulting \mathbf{k}_o is **not** $p_h(\mathbf{k}_o)$. This is because of the difference in measures in \mathbf{h} and \mathbf{k}_o . So the actual density $p(\mathbf{k}_o)$ is

$$p(\mathbf{k}_o) = \frac{p_h(\mathbf{h})}{4(\mathbf{k}_i \cdot \mathbf{h})}. \quad (24.12)$$

Note that in an implementation where the BRDF is known to be this model, the estimate of the rendering equation is quite simple as many terms cancel out.

It is possible to generate an \mathbf{h} vector whose corresponding vector \mathbf{k}_o will point inside the surface, i.e., $\cos \theta_o < 0$. The weight of such a sample should be set to zero. This situation corresponds to the specular lobe going below the horizon and is the main source of energy loss in the model. Clearly, this problem becomes progressively less severe as n_u, n_v become larger.

The only thing left now is to describe how to generate \mathbf{h} vectors with the pdf of Equation (24.10). We will start by generating \mathbf{h} with its spherical angles in the range $(\theta, \phi) \in [0, \frac{\pi}{2}] \times [0, \frac{\pi}{2}]$. Note that this is only the first quadrant of the hemisphere. Given two random numbers (ξ_1, ξ_2) uniformly distributed in $[0, 1]$, we can choose

$$\phi = \arctan \left(\sqrt{\frac{n_u + 1}{n_v + 1}} \tan \left(\frac{\pi \xi_1}{2} \right) \right), \quad (24.13)$$

计算指数，因此可以写出镜面BRDF：

(h·k_i)max(cosθ_i cosθ_o)F(k_i·h)。 (24.9) 在蒙特卡罗设置中，我们对以下问题感兴趣：给定k1，生成具有形状类似于余弦加权BRDF的分布的k2的样本。请注意，对被积函数的大值进行大幅欠采样是严重的误差，而对小值进行大幅过采样在实践中是可接受的。读者可以验证下面建议的密度是否具有此属性。

构建用于采样的pdf的合适方法是考虑产生BRDF的半向量的分布。这样的功能是

$$p_h(\mathbf{h}) = \frac{\sqrt{(n_u + 1)(n_v + 1)} (\mathbf{n}\mathbf{h})^{n_u \cos 2\phi + n_v \sin 2\phi}}{2\pi} \quad (24.10)$$

选择常量以确保它是有效的pdf。

我们可以只使用方程 (24.10) 的概率密度函数 $p_h(\mathbf{h})$ 来生成随机 \mathbf{h} 。然而，为了评估渲染方程，我们需要一个反射向量 \mathbf{k}_o 和一个概率密度函数 $p(\mathbf{k}_o)$ 。重要的是要注意，如果您根据 $p_h(\mathbf{h})$ 生成 \mathbf{h} ，然后转换为生成的 \mathbf{k}_o ：

$$\mathbf{k}_o = -\mathbf{k}_i + 2(\mathbf{k}_i \cdot \mathbf{h})\mathbf{h}, \quad (24.11)$$

所得 \mathbf{k}_o 的密度不是 $p_h(\mathbf{k}_o)$ 。这是因为 \mathbf{h} 和 \mathbf{k}_o 中的度量差异。所以实际密度 $p(\mathbf{k}_o)$ 为

$$p(\mathbf{k}_o) = \frac{p_h(\mathbf{h})}{4(\mathbf{k}_i \cdot \mathbf{h})}. \quad (24.12)$$

请注意，在已知BRDF是此模型的实现中，渲染方程的估计非常简单，因为许多项会抵消。

可以生成一个 \mathbf{h} 向量，其相应的向量 \mathbf{k}_o 将指向表面内部，即 $\cos \theta_o < 0$ 。这样的样品的重量应设置为零。这种情况对应于镜面瓣低于地平线，是模型中能量损失的主要来源。显然，随着 n_u, n_v 变得更大，这个问题变得越来越不严重。

现在唯一剩下的就是描述如何用方程 (24.10) 的pdf生成 \mathbf{h} 个向量。我们将从生成 \mathbf{h} 开始，其球面角在 (θ, ϕ) 范围内 $[0, \pi/2]$ 。请注意，这只是半球的第一象限。给定两个均匀分布在 $[0, 1]$ 中的随机数 (ξ_1, ξ_2) ，我们可以选择

$$\phi = \arctan \frac{n_u + 1}{n_v + 1} \tan \frac{\pi \xi_1}{2}, \quad (24.13)$$

and then use this value of ϕ to obtain θ according to

$$\cos \theta = (1 - \xi_2)^{1/(n_u \cos^2 \phi + n_v \sin^2 \phi + 1)}. \quad (24.14)$$

To sample the entire hemisphere, we use the standard manipulation where ξ_1 is mapped to one of four possible functions depending on whether it is in $[0, 0.25]$, $[0.25, 0.5]$, $[0.5, 0.75]$, or $[0.75, 1.0]$. For example, for $\xi_1 \in [0.25, 0.5]$, find $\phi(1 - 4(0.5 - \xi_1))$ via Equation (24.13), and then “flip” it about the $\phi = \pi/2$ axis. This ensures full coverage and stratification.

For the diffuse term, use a simpler approach and generate samples according to a cosine distribution. This is sufficiently close to the complete diffuse BRDF to substantially reduce variance of the Monte Carlo estimation.

Frequently Asked Questions

- My images look too smooth, even with a complex BRDF. What am I doing wrong?

BRDFs only capture subpixel detail that is too small to be resolved by the eye. Most real surfaces also have some small variations, such as the wrinkles in skin, that can be seen. If you want true realism, some sort of texture or displacement map is needed.

- How do I integrate the BRDF with texture mapping?

Texture mapping can be used to control any parameter on a surface. So any kinds of colors or control parameters used by a BRDF should be programmable.

- I have very pretty code except for my material class. What am I doing wrong?

You are probably doing nothing wrong. Material classes tend to be the ugly thing in everybody’s programs. If you find a nice way to deal with it, please let me know! My own code uses a shader architecture (Hanrahan & Lawson, 1990) which makes the material include much of the rendering algorithm.

Notes

There are many BRDF models described in the literature, and only a few of them have been described here. Others include (Cook & Torrance, 1982; He

然后用 ϕ 的这个值得到 θ 根据

$$\cos\theta=(1-\infty^2)^{1/(n_u \cos^2 \phi + n_v \sin^2 \phi + 1)} \quad (24.14)$$

为了对整个半球进行采样，我们使用标准操作，其中 ξ_1 映射到四个可能的函数之一，具体取决于它是否在 $[0, 0.25]$ $[0.25, 0.5]$ $[0.5, 0.75]$ ，或 $[0.75, 1.0]$ 。例如，对于 $\xi_1 \in [0.25, 0.5]$ ，求 $\phi(1 - 4(0.5 - \xi_1))$ 经由等式(24.13)，然后关于 $\phi=\pi/2$ 轴“翻转”它。这确保了完全覆盖和分层。

对于扩散项，使用更简单的方法并根据余弦分布生成样本。这足够接近完全漫射BRDF，以显著降低蒙特卡洛估计的方差。

常见问题

- 我的图像看起来太光滑，即使是复杂的BRDF。我做错了什么？

BRDFs只捕获太小而无法被眼睛分辨的亚像素细节。大多数真实的表面也有一些小的变化，如皮肤的皱纹，可以看到。如果你想要真正的现实主义，需要某种纹理或位移图。

- 如何将BRDF与纹理映射集成？

纹理映射可用于控制曲面上的任何参数。因此，BRDF使用的任何颜色或控制参数都应该是可编程的。

- 除了我的材料类，我有非常漂亮的代码。我做错了什么？

你可能没有做错什么。在每个人的程序中，材料类往往是丑陋的东西。如果你找到一个很好的方法来处理它，请让我知道！我自己的代码使用着色器架构 (Hanrahan & Lawson, 1990)，这使得材质包含了大部分渲染算法。

Notes

文献中描述了许多BRDF模型，这里仅描述了其中的少数几个模型。其他包括 (Cook & Torrance, 1982; He



et al., 1992; G. J. Ward, 1992; Oren & Nayar, 1994; Schlick, 1994a; Lafortune, Foo, Torrance, & Greenberg, 1997; Stam, 1999; Ashikhmin, Premožec, & Shirley, 2000; Ershov, Kolchin, & Myszkowski, 2001; Matusik, Pfister, Brand, & McMillan, 2003; Lawrence, Rusinkiewicz, & Ramamoorthi, 2004; Stark, Arvo, & Smits, 2005). The desired characteristics of BRDF models is discussed in *Making Shaders More Physically Plausible* (R. R. Lewis, 1994).

Exercises

1. Suppose that instead of the Lambertian BRDF we used a BRDF of the form $C \cos^a \theta_i$. What must C be to conserve energy?
2. The BRDF in Exercise 1 is not reciprocal. Can you modify it to be reciprocal?
3. Something like a highway sign is a *retroreflector*. This means that the BRDF is large when \mathbf{k}_i and \mathbf{k}_o are near each other. Make a model inspired by the Phong model that captures retroreflection behavior while being reciprocal and conserving energy.



等人。1992;G.J.Ward 1992;Oren&Nayar 1994;Schlick 1994a;Lafortune Foo Torrance &Greenberg 1997;Stam 1999;Ashikhmin Premožec &Shirley 2000;Ershov Kolchin &Myszkowski 2001;Matusik Pfister Brand &McMillan 2003;Lawrence rusinkiewicz, &Ramamoorthi, 2004;Stark, Arvo, &Smits, 2005)。BRDF模型的期望特性在使着色器更具物理合理性 (R.R.Lewis, 1994) 中进行了讨论。

Exercises

- 1.假设我们使用 $C\cos\theta_i$ 形式的BRDF而不是LambertianBRDF。C必须是什么才能节约能源?
- 2.练习1中的BRDF不是对等的。你能把它修改成对等的吗?
- 3.类似高速公路标志的东西是回射器。这意味着当 \mathbf{k}_i 和 \mathbf{k}_o 彼此靠近时BRDF很大。制作一个受Phong模型启发的模型，该模型捕获了逆向反射行为，同时具有互惠性和节约能源。



Computer Graphics in Games

Of all the applications of computer graphics, computer and video games attract perhaps the most attention. The graphics methods selected for a given game have a profound effect, not only on the game engine code, but also on the art asset creation, and even sometimes on the *gameplay*, or core game mechanics.

Although game graphics rely on the material in all of the preceding chapters, two chapters are particularly germane. Games need to make highly efficient use of graphics hardware, so an understanding of the material in Chapter 17 is important.

In this chapter, I will detail the specific considerations that apply to graphics in game development, from the platforms on which games run to the game production process.

25.1 Platforms

Here, I use the term *platform* to refer to a specific combination of hardware, operating system, and API (application programming interface) for which a game is designed. Games run on a large variety of platforms, ranging from virtual machines used for browser-based games to dedicated game consoles using specialized hardware and APIs.

643



游戏中的计算机图形学

在计算机图形学的所有应用中，计算机和视频游戏可能吸引了最多关注。为给定游戏选择的图形方法具有深远的影响，不仅对游戏引擎代码，而且对艺术资产创建，甚至有时对游戏玩法或核心游戏机制。

虽然游戏图形依赖于前面所有章节的材料，但两章特别是日耳曼。游戏需要高效地利用图形硬件，因此了解第17章中的材料非常重要。

在本章中，我将详细介绍应用于游戏开发中的图形ic的具体考虑因素，从游戏运行的平台到游戏制作过程。

25.1 Platforms

在这里，我使用术语平台来指代为其设计游戏的硬件，操作系统和API（应用程序编程接口）的特定组合。游戏在各种平台上运行，从用于基于浏览器的游戏的虚拟机到使用专用硬件和Api的专用游戏机。

643

In the past, it was common for games to be designed for a single platform. The increasing cost of game development has made this rare; *multiplatform* game development is now the norm. The incremental increase in development cost to support multiple platforms is more than repaid by a potential doubling or tripling of the customer base.

Some platforms are quite loosely defined. For example, when developing a game for the Windows PC platform, the developer must account for a very large variety of possible hardware configurations. Games are even expected to run (and run well) on PC configurations that did not exist when the game was developed! This is only possible due to the abstractions afforded by the APIs defining the Windows platform.

One way in which developers account for wide variance in graphics performance is by *scaling*—adjusting graphics quality in response to system capabilities. This can ensure reasonable performance on low-end systems, while still achieving competitive visuals on high-performance systems. This adjustment is sometimes done automatically by profiling the system performance, but more often this control is left in the hands of the user, who can best judge his personal preferences for quality versus speed. Display resolution is easiest to adjust, followed by antialiasing quality. It is also fairly common to offer several quality levels for visual effects such as shadows and motion blur, including the option of turning the effect off entirely.

Differences in graphics performance can be so large that some machines may not run the game at a playable frame rate, even with the lowest quality settings; for this reason PC game developers publish minimum and recommended machine specifications for each game.

As platforms, game consoles are strictly defined. When developing a game for, e.g., Nintendo's Wii console, the developer knows exactly what hardware the game will run on. If the platform's hardware implementation is changed (often done to reduce manufacturing costs), the console manufacturer must ensure that the new implementation behaves *exactly* like the previous one, including timing and performance. This is not to say that the console developer's task is easy; console APIs tend to be much less abstract and closer to the underlying hardware. This gives console development its own set of difficulties. In some sense, multiplatform development (which commonly includes at least two different console platforms and often Windows as well) is the hardest of all, since the multiplatform game developer has neither the assurance of a fixed platform or the convenience of a single high-level API.

Browser-based *virtual machines* such as Adobe Flash are an interesting class of game platforms. Although such virtual machines run on a wide class of hard-

在过去，游戏设计为单一平台是很常见的。游戏开发成本的增加使这种情况变得罕见；多平台游戏开发现在是常态。支持多个平台的开发成本的增量增加超过了潜在的客户群的两倍或三倍。

有些平台的定义相当松散。例如，在为Windows PC平台开发游戏时，开发人员必须考虑各种可能的硬件配置。游戏甚至有望在游戏开发时不存在的PC配置上运行（并且运行良好）！这是唯一可能的，因为由定义

Windows平台。

开发人员考虑图形性能差异的一种方式是通过缩放—调整图形质量以响应系统capabilities。这可以确保在低端系统上的合理性能，同时仍然在高性能系统上实现有竞争力的视觉效果。这种调整有时是通过分析系统性能自动完成的，但更多的是这种控制留在用户手中，用户可以最好地判断他对质量与速度的个人偏好。显示分辨率最容易调整，但由于抗锯齿质量而降低。为阴影和运动模糊等视觉效果提供多个质量级别也是相当常见的，包括完全关闭效果的选项。

图形性能的差异可能如此之大，以至于某些机器可能无法以可玩的帧速率运行游戏，即使是最低质量的设置；为此，PC游戏开发人员发布了每个游戏的最

作为平台，游戏机是严格定义的。在开发游戏时，例如任天堂的Wii游戏机，开发人员确切地知道游戏将在什么硬件上运行。如果更改平台的硬件实现（通常是为了降低成本），控制台制造商必须确保新实现的行为与前一个实现完全相同，包括时序和性能。这并不是说控制台开发人员的任务很容易；console API往往不那么抽象，更接近底层硬件。这给控制台开发带来了自己的一系列困难。从某种意义上说，多平台开发（通常包括至少两个不同的控制台平台，通常也包括Windows）是最难的，因为多平台游戏开发人员既没有固定平台的保证，也没有

基于浏览器的虚拟机（如Adobe Flash）是一类有趣的游戏平台。虽然这样的虚拟机运行在一个宽类的硬

ware from personal computers to mobile phones, the high degree of abstraction provided by the virtual machine results in a stable and unified development platform. The relative ease of development for these platforms and the huge pool of potential customers makes them increasingly attractive to game developers. However, these platforms are defined by the lowest common denominator of the supported hardware, and virtual machines have lower performance than native code on any given platform. For these reasons, such platforms are best suited to games with modest graphics requirements.

Platforms can also be characterized by their openness to development, which is a business or legal distinction rather than a technical one. For example, Windows is open in the sense that development tools are widely available, and there are no gatekeepers controlling access to the marketplace of Windows games. Apple's iPhone is a somewhat more restricted platform in that all applications need to pass a certification process and certain classes of applications are banned outright. Consoles are the most restrictive game platforms, where access to the development tools is tightly controlled. This is opening up somewhat with the introduction of online console game marketplaces, which tend to be more open. A particularly interesting example is Microsoft's Xbox LIVE Community Games service, where the development tools are freely available and the "gatekeeping" is performed primarily by peer review. Games distributed through this service must use a virtual machine platform provided by Microsoft for security reasons.

The game platform determines many elements of the game experience. For example, PC gamers use keyboard and mouse, while console gamers use specialized game controllers. Many console games support multiple players on the same console, either sharing a screen or providing a window for each player. Due to the difficulty of sharing keyboard and mouse, this type of play is not found on PC. A handheld game system will have a different control scheme than a touch-screen phone, etc.

Although game platforms vary widely, some common trends can be discerned. Most platforms have multiple processing cores, divided between general-purpose (CPU) and graphics-specific (GPU). Performance gains over time are due mostly to increases in core count; gains in individual core performance are modest. As GPU cores grow in generality, the lines between GPU and CPU cores are increasingly blurred. Storage capacity tends to increase at a slower rate than processing power, and communication bandwidth (between cores as well as between each core and storage) grows at a slower pace still.

从个人电脑到手机，虚拟机提供的高度抽象化，形成了稳定统一的开发平台形式。这些平台的开发相对容易，以及庞大的潜在客户群，使它们对游戏开发者越来越有吸引力。但是，这些平台由支持的硬件的最低共同点定义，并且虚拟机的性能低于任何给定平台上的本机代码。由于这些原因，这些平台最适合具有适度图形要求的游戏。

平台还可以以其对开发的开放性为特征，这是业务或法律上的区别，而不是技术上的区别。例如，Windows在开发工具广泛可用的意义上是开放的，并且没有看门人控制对Windows游戏市场的访问。Apple的iPhone是一个更受限制的平台，因为所有应用程序都需要通过认证过程，并且某些类别的应用程序被禁止。游戏机是限制性最强的游戏平台，对开发工具的访问受到严格控制。随着在线控制台游戏市场的生产，这在某种程度上是开放的，而在线控制台游戏市场往往更加开放。一个特别有趣的例子是微软的XboxLIVE社区游戏服务，其中开发工具是免费提供的，“守门员”主要由同行评审执行。出于安全原因，通过此服务分发的游戏必须使用Microsoft提供的虚拟机平台。

游戏平台决定了游戏体验的许多元素。例如，PC游戏玩家使用键盘和鼠标，而控制台游戏玩家使用特殊的ized游戏控制器。许多主机游戏支持同一台主机上的多个玩家，要么共享一个屏幕，要么为每个玩家提供一个窗口。由于共享键盘和鼠标的困难，这种类型的游戏在PC上找不到。掌上游戏系统将具有与触摸屏电话等不同的控制方案。

虽然游戏平台差异很大，但可以看出一些常见的趋势。大多数平台都有多个处理内核，分为通用(CPU)和图形专用(GPU)。随着时间的推移，性能的提高主要是由于核心数量的增加；个别核心性能的提高是适度的。随着GPU核心的普遍增长，GPU和CPU核心之间的界限逐渐模糊。存储容量的增长速度往往比处理能力慢，通信带宽（内核之间以及每个内核与存储之间）的增长速度仍然较慢。

25.2 Limited Resources

One of the primary challenges of game graphics is the need to manage multiple pools of limited resources. Each platform imposes its own constraints on hardware resources such as processing time, storage, and memory bandwidth. At a higher level, development resources also need to be managed; there is a fixed-size team of programmers, artists, and game designers with limited time to complete the game, hopefully without working *too* much overtime! This needs to be taken into account when deciding which graphics techniques to adopt.

25.2.1 Processing Time

Early game developers only had to worry about budgeting a single processor. Current game platforms contain multiple CPU and GPU cores. These processors need to be carefully synchronized to avoid deadlocks or excessive stalls.

Since the time consumed by a single rendering command is highly variable, graphics processors are decoupled from the rest of the system via a *command buffer*. This buffer acts as a queue; commands are deposited on one end and the GPU reads rendering commands from the other. Increasing the size of this buffer decreases the chances of GPU starvation. It is fairly common for games to buffer an entire frame's worth of rendering commands before sending them to the GPU; this guarantees that GPU starvation does not occur. However, this approach requires reserving enough storage space for two full frame's worth of commands (the GPU works on one, while the CPU deposits commands in the other). It also increases the latency between the user's input and the display, which can be problematic for fast-paced games.

Processing budgets are determined by the *frame rate*, which is the frequency at which the frame buffer is refreshed with new renderings of the scene. On fixed platforms (such as consoles), the frame rate experienced by the user is essentially the same one seen by the game developer, so fairly strict frame-rate limits can be imposed. Most games target a frame rate of 30 frames per second (fps); in games where response latency is especially important, the target is often 60 fps. On highly variable platforms (such as PCs), the frame-rate budgets are (by necessity) defined more loosely.

The required frame rate gives the graphics programmer a fixed budget per frame to work with. In the case of a 30 fps target, the CPU cores have 33 milliseconds to gather inputs, process the game logic, perform any physical simulations, traverse the scene description, and send the rendering commands to the graphics

25.2 资源有限

游戏图形的主要挑战之一是需要管理多个有限资源池。每个平台对硬件资源（如处理时间、存储和内存带宽）施加自己的约束。在更高层次上，开发资源也需要管理；有一个固定规模的程序员，艺术家和游戏设计师团队，时间有限，可以完成游戏，希望不会加班太多！在决定采用哪种图形技术时，需要考虑到这一点。

25.2.1 处理时间

早期的游戏开发者只需要担心预算单个处理器。
当前的游戏平台包含多个CPU和GPU内核。这些专业处理器需要小心同步，以避免死锁或过多的推位。

由于单个渲染命令消耗的时间是高度可变的，图形处理器通过命令缓冲区与系统的其余部分分离。此缓冲区充当队列；命令存储在一端，GPU从另一端读取渲染命令。增加此缓冲区的大小会降低GPU饥饿的可能性。游戏在将渲染命令发送到GPU之前缓冲整个帧的渲染命令是相当常见的；这可以保证GPU不会出现饥饿。但是，这种方法需要为两个完整帧的命令预留足够的存储空间（GPU在一个上运行，而CPU在另一个上存储命令）。它还增加了用户输入和显示之间的延迟，这对于快节奏的游戏来说可能是有问题的。

处理预算由帧速率决定，帧速率是用场景的新渲染刷新帧缓冲区的频率。在固定平台（如控制台）上，用户体验到的帧速率基本上与游戏开发者看到的帧速率相同，因此可以施加相当严格的帧速率限制。大多数游戏的目标帧速率为每秒30帧（fps）；在响应延迟特别重要的游戏中，目标通常为60fps。在高度可变的平台（如PC）上，帧速率预算（根据需要）定义得更宽松。

所需的帧速率为图形程序员提供了每帧的固定预算。对于一个30fps的目标，CPU内核有33毫秒的on来收集输入、处理游戏逻辑、执行任何物理模拟、遍历场景描述并将渲染命令发送到图形

hardware. In parallel, other tasks such as audio and network processing must be handled, with their own required response times. While this is happening, the GPU is typically executing the graphics commands submitted during the previous frame.

In most cases, CPU cores are a *homogeneous* resource; all cores are the same, and any of them are equally well suited to a given workload (there are some exceptions, such as the Cell processor used in Sony's PLAYSTATION 3 console).

In contrast, GPUs contain a *heterogeneous* mix of resources, each specialized to a certain set of tasks. Some of these resources consist of fixed-function hardware (for triangle rasterization, alpha blending, and texture sampling), and some are programmable cores. On older GPUs, programmable cores were further differentiated into vertex and pixel processing cores; newer GPU designs have *unified shader cores* which can execute any of the programmable shader types.

Such heterogeneous resources are budgeted separately. Typically, at any point, only one resource type will be the bottleneck, and the others will have excess capacity. On the one hand, this is good, since this capacity can be leveraged to improve visual quality without decreasing performance. On the other hand, it makes it harder to improve performance, since decreasing usage of any of the non-bottleneck resources will have no effect. Even decreasing usage of the bottleneck resource may only improve performance slightly, depending on the degree of utilization of the “next bottleneck.”

25.2.2 Storage

Game platforms, like any modern computing system, possess multi-stage *storage hierarchies*, with smaller, faster memory types at the top and larger, slower storage at the bottom. This arrangement is borne of engineering necessity, although it does complicate life for the developer. Most platforms include optical disc storage, which is extremely slow and is used mostly for delivery. On platforms such as Windows, a lengthy installation process is performed once to move all data from the optical disc onto the hard drive, which is significantly faster. The optical disc is never used again (except as an anti-piracy measure). On console platforms, this is less common, although it does sometimes happen when a hard drive is guaranteed to be present, as on Sony's PLAYSTATION 3 console. More often, the hard drive (if present) is only used as a cache for the optical disc.

The next step up the memory hierarchy is RAM, which on many platforms is divided into general system RAM and VRAM (video RAM) which benefits from a high-speed interface to the graphics hardware. A game level may be too large to

硬件。同时，其他任务，如音频和网络处理，必须处理，与他们自己所需的响应时间。当这种情况发生时，GPU通常执行在前一帧期间提交的图形命令。

在大多数情况下，CPU内核是一种同质资源；所有内核都是相同的，其中任何内核都同样非常适合给定的工作负载（有一些例外，例如索尼PLAYSTATION3控制台中使用相比之下，GPU包含异构的资源组合，每个资源都专门用于特定的任务集。其中一些资源由固定功能硬件（用于三角形光栅化、alpha混合和纹理采样）组成，还有一些是可编程内核。在较旧的GPU上，可编程内核进一步区分为顶点和像素处理内核；较新的GPU设计具有统一的着色器内核，可以执行任何可编程着色器类型。这种异质资源是单独预算的。通常，在任何时候，只有一种资源类型将成为瓶颈，而其他资源类型将具有多余的capacity。一方面，这很好，因为可以利用这种容量来提高视觉质量而不会降低性能。另一方面，它使提高性能变得更加困难，因为减少任何非瓶颈资源的使用都不会产生影响。即使减少bottleneck资源的使用也可能只会略微提高性能，具体取决于“下一个瓶颈”的利用程度。

25.2.2 Storage

游戏平台与任何现代计算系统一样，拥有多阶段storage层次结构，顶部有更小，更快的内存类型，底部有更大，更慢的存储。这种安排是出于工程上的需要 尽管它确实使开发人员的生活复杂化。大多数平台包括光盘存储，这是非常慢的，主要用于交付。在windows等其他形式上，执行一次冗长的安装过程以将所有数据从光盘移动到硬盘驱动器上，这明显更快。光盘不再使用（除了作为反盗版措施）。在console平台上，这种情况不太常见，尽管有时会发生硬盘驱动器保证存在时，就像索尼的PLAYSTATION3控制台一样。更常见的是，硬盘驱动器（如果存在）仅用作光盘的缓存。

存储器层次结构的下一步是RAM，在许多平台上，RAM分为通用系统RAM和VRAM（视频RAM），后者受益于与图形硬件的高速接口。游戏级别可能太大而不能

fit in RAM, in which case the game developer needs to manage moving the data in and out of RAM as needed. On platforms such as Windows, virtual memory is often used for this. On console platforms, custom data streaming and caching systems are typically employed.

Finally, both the CPU and GPU boast various kinds of on-chip memory and caches. These are extremely small and fast and are usually managed by the graphics API.

Graphics resources take up a lot of memory, so they are a primary focus of storage budgets in game development. Textures are usually the greatest memory consumers, followed by geometry (vertex data), and finally other types of graphics data such as animations. Not all memory can be used for graphics—audio also takes up a fair bit, and game logic may use sizeable data structures. As in the case of processing time, budgeting tends to be somewhat looser on Windows, where the exact amount of memory present on the user's system is unknown and virtual memory covers a multitude of sins. In contrast, memory budgeting on console platforms is quite strict—often the lead programmer keeps track of memory on a spreadsheet and a programmer requiring more memory for their system needs to beg, borrow, or steal it from someone else.

The various levels of the memory hierarchy differ not only in size, but also in access speed. This has two separate dimensions: *latency* and *bandwidth*.

Latency is the time that elapses between a storage access request and its final fulfillment. This varies from a few clock cycles (for on-chip cache) to millions of clock cycles (for data residing on optical disc). Latency is usually an issue for read access (although write latency can also be an issue if the result needs to be read back from memory soon after). In some cases, the read request is *blocking*, which means that the processor core that submitted the read can do nothing else until the request is fulfilled. In other cases, the read is *non-blocking*; the processing core can submit the read request, do other types of processing, and then use the results of the read after it has arrived. Texture accesses by the GPU are an example of non-blocking reads; an important aspect of GPU design is to find ways to “hide” texture read latency by performing unrelated computations while the texture read is being fulfilled.

For this latency hiding to work, there must be a sufficient amount of computation relative to texture accesses. This is an important consideration for the shader writer; the optimal mix of computation vs. texture access keeps changing (in favor of more computation) as memory fails to keep up with increases in processing power.

Bandwidth refers to the maximum rate of transfer to and from storage. It is typically measured in gigabytes per second.

适合RAM，在这种情况下，游戏开发人员需要根据需要管理将数据移入和移出RAM。在Windows等平台上，虚拟内存通常用于此。在控制台平台上，通常使用自定义数据流和缓存系统。

最后，CPU和GPU都拥有各种片上内存和缓存。这些是非常小的和快速的，通常由图形API管理。

图形资源占用大量内存，因此它们是游戏开发中存储预算的主要焦点。纹理通常是最大的内存消耗者，其次是几何（顶点数据），最后是其他类型的图形数据，如动画。并非所有的内存都可以用于图形-音频也占用了相当多的位，游戏逻辑可能会使用相当大的数据结构。与处理时间的情况一样，预算在Windows上往往有些宽松，其中用户系统上存在的确切内存量是未知的，虚拟内存覆盖了大量的罪恶。相比之下，控制台平台上的内存预算非常严格—通常首席程序员会在电子表格上跟踪内存，而需要更多内存的程序员需要向其他人乞求，借用或窃取它。

内存层次结构的各个层次不仅在大小上不同，而且在访问速度上也不同。这有两个独立的维度：延迟和带宽。

延迟是存储访问请求与其最终实现之间经过的时间。这从几个时钟周期（对于片上高速缓存）到数百万个时钟周期（对于驻留在光盘上的数据）不等。延迟通常是读访问的一个问题（尽管写延迟也可能是一个问题，如果结果需要很快从内存中读回）。在某些情况下，读取请求是阻塞的，这意味着提交读取的处理器内核在满足请求之前不能执行任何其他操作。在其他情况下，读取是非阻塞的；处理核心可以提交读取请求，做其他类型的处理，然后在到达后使用读取的结果。GPU的纹理访问是非阻塞读取的一个例子；GPU设计的一个重要方面是通过在纹理读取完成时执行不相关的计算来找到“隐藏”纹理读取延迟的方法。

为了使这种延迟隐藏起作用，相对于纹理访问必须有足够的计算量。这是着色器编写器的一个重要因素；由于内存无法跟上处理能力的增加，计算与纹理访问的最佳组合不断变化（有利于更多的计算）。

带宽是指传输到和从存储的最大速率。它通常以每秒千兆字节为单位。



25.2.3 Development Resources

Besides hardware resources, such as processing power and storage space, the game graphics programmer also has to contend with a different kind of limited resource—the time of his teammates! When selecting graphics techniques, the engineering resources needed to implement each technique must be taken into account, as well as any tools necessary to compute the input data (in many cases, tools can take significantly more time than implementing the technique itself). Perhaps most importantly, the impact on artist productivity must be taken into account. Most graphics techniques use assets created by game artists, who comprise by far the largest part of most modern game teams. The graphics programmer must foster the artist's productivity and creativity, which will ultimately determine the visual quality of the game.

25.3 Optimization Techniques

Making wise use of these limited resources is the primary challenge of the game graphics programmer. To this end, various optimization techniques are commonly employed.

In many games, pixel shader processing is a primary bottleneck. Most GPUs contain hierarchical depth-culling hardware which can avoid executing pixel shaders on occluded surfaces. To make good use of this hardware, opaque objects can be rendered back-to-front. Alternatively, optimal depth-culling usage can be achieved by performing a *depth prepass*, i.e., rendering all the opaque objects into the depth buffer (without any color output or pixel shaders) before rendering the scene normally. This does incur some overhead (due to the need to render every object twice), but in many cases the performance gain is worth it.

The fastest way to render an object is to not render it at all; thus any method of discerning early on that an object is occluded can be useful. This saves not only pixel processing but also vertex processing and even CPU time that would be spent submitting the object to the graphics API. View frustum culling (see Section 8.4.1) is universally employed, but in many games it is not sufficient. High-level occlusion culling algorithms are often used, utilizing data structures such as PVS (potentially visible sets) or BSP (binary spatial partitioning) trees to quickly narrow down the pool of potentially visible objects.

Even if an object is visible, it may be at such a distance that most of its detail can be removed without apparent effect. LOD (level-of-detail) algorithms render different representations of an object based on distance (or other factors, such



25.2.3 发展资源

除了硬件资源，如处理能力和存储空间，游戏图形程序员还必须与另一种有限的资源竞争—他的队友的时间！在选择图形技术时，必须考虑实现每种技术所需的工程资源，以及计算输入数据所需的任何工具（在许多情况下，工具可能比实现技术本身花费更多时间）。也许最重要的是，必须考虑到对艺术家生产力的影响。大多数图形技术使用由游戏美术师创建的资源，这些美术师构成了大多数现代游戏团队的最大部分。图形程序员必须培养艺术家的生产力和创造力，这将最终决定游戏的视觉质量。

25.3 优化技术

明智地利用这些有限的资源是游戏图形程序员的主要挑战。为此，通常采用各种优化技术。

在许多游戏中，像素着色器处理是一个主要瓶颈。大多数GPU都包含分层深度剔除硬件，可以避免在遮挡表面上执行像素着色器。为了充分利用这个硬件，不透明的对象可以背对背地呈现。或者，可以通过执行深度预处理来实现最佳深度剔除使用，即在正常渲染场景之前将所有不透明对象渲染到深度缓冲区（没有任何颜色输出或像素着色器）。这确实会产生一些开销（由于需要渲染每个对象两次），但在许多情况下，性能增益是值得的。

渲染对象的最快方法是根本不渲染它；因此，任何早期识别对象被遮挡的方法都是有用的。这不仅节省了像素处理，还节省了顶点处理，甚至节省了将对象提交到图形API所花费的CPU时间。视图截头剔除（见第8.4.1节）是普遍使用的，但在许多游戏中它是不够的。通常使用高级遮挡剔除算法，利用PVS（潜在可见集）或BSP（二进制空间分区）树等数据结构来快速缩小潜在可见对象的池。

即使一个物体是可见的，它可能在这样的距离，它的大部分细节可以被删除没有明显的效果。Lod（详细级别）算法基于距离（或其他因素，例如

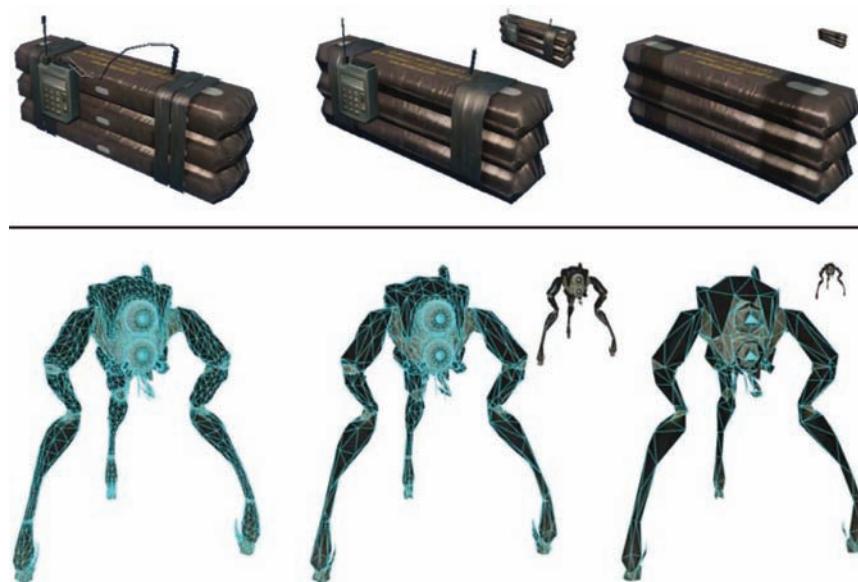


Figure 25.1. Two examples of game objects at a varying level of detail. The small inset images show the relative sizes at which the simplified models might be used. *Upper row of images courtesy Crytek; lower row courtesy Valve Corp.*

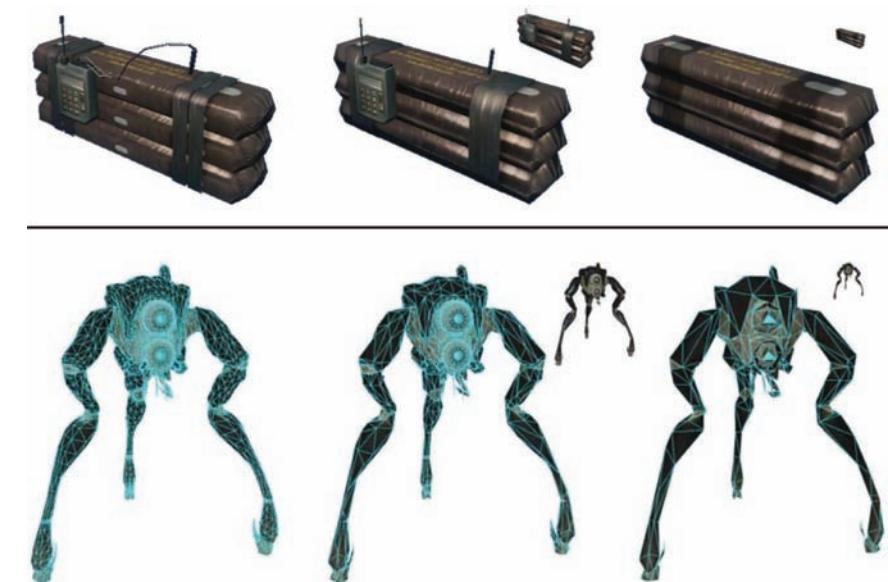
as screen coverage or importance). This can save significant processing, vertex processing in particular. Examples can be seen in Figure 25.1.

In many cases, processing can be performed before the game even starts. The results of such *preprocessing* can be stored and used each frame, thus speeding up the game. This is most commonly employed for lighting, where global illumination algorithms are utilized to compute lighting throughout the scene and store it in lightmaps and other data structures for later use.

25.4 Game Types

Since game requirements vary widely, the selection of graphics techniques is driven by the exact type of game being developed.

The allocation of processing time depends strongly on the frame rate. Currently, most console games tend to target 30 frames per second, since this enables much higher graphics quality. However, certain game types with fast gameplay require very low latency, and such games typically render at 60 frames per second.



不同细节级别的游戏对象的两个例子。小插图显示了简化模型可能使用的相对大小。上排图像礼貌Crytek;下排礼貌阀门公司。

作为屏幕覆盖或重要性）。这可以节省显着的处理，特别是顶点处理。例子可以在图25.1中看到。

在许多情况下，可以在游戏甚至开始之前执行处理。这样的预处理的结果可以每帧存储和使用，从而加快游戏速度。这是最常用的照明，全局照明国家算法被用来计算整个场景的照明，并将其存储在光图和其他数据结构供以后使用。

25.4 游戏类型

由于游戏要求差异很大，因此图形技术的选择取决于正在开发的游戏的确切类型。

处理时间的分配强烈地取决于帧速率。然而，大多数游戏机游戏倾向于以每秒30帧为目标，因为这可以实现更高的图形质量。但是，某些具有快速游戏性的游戏类型需要非常低的延迟，此类游戏通常以每秒60帧的速度渲染。



This includes music games such as *Guitar Hero* and first-person shooters such as *Call of Duty*.

The frame rate determines the available time to render the scene. The composition of the scene itself also varies widely from game to game. Most games have a division between *background geometry* (scenery, mostly static) and *foreground geometry* (characters and dynamic objects). These are handled differently by the rendering engine. For example, background geometry will often have lightmaps containing precomputed lighting, which is not feasible for foreground objects. Precomputed lighting is typically applied to foreground objects via some type of volumetric representation which can take account of the changing position of each object over time.

Some games have relatively enclosed environments, where the camera remains largely in place. The purest examples are fighting games such as the *Street Fighter* series, but this is also true to some extent for games such as *Devil May Cry* and *God of War*. These games have cameras that are not under direct player control, and the game play tends to move from one enclosed environment to another, spending a significant amount of playing time in each. This allows the game developer to lavish large amounts of resources (processing, storage, and artist time) on each room or enclosed environment, resulting in very high levels of graphics fidelity.

Other games have extremely large worlds, where the player can move about freely. This is most true for “sandbox games” such as the *Grand Theft Auto* series and online role-playing games such as *World of Warcraft*. Such games pose great challenges to the graphics developer, since resource allocation is very difficult when during each frame the player can see a large extent of the world. Further complicating things, the player can freely go to some formerly distant part of the world and observe it from up close. Such games typically have changing time of day, which makes precomputation of lighting difficult at best, if not impossible.

Most games, such as first-person shooters, are somewhere between the two extremes. The player can see a fair amount of scenery each frame, but movement through the game world is somewhat constrained. Many games also have a fixed time of day for each game level, for ease of lighting precomputation.

The number of foreground objects rendered also varies widely between game types. Real-time strategy games such as the *Command and Conquer* series often have many dozens, if not hundreds, of units visible on screen. Other types of games have more limited quantities of visible characters, with fighting games at the opposite extreme, where only two characters are visible, each rendered with extremely high detail. A distinction must be drawn between the number of characters visible at any time (which affects budgeting of processing time) and



这包括音乐游戏，如吉他英雄和第一人称射击游戏，如使命召唤。

帧速率确定渲染场景的可用时间。场景本身的composition也因游戏而异。大多数游戏都有背景几何（风景，大部分是静态的）和前景几何（人物和动态对象）之间的划分。这些由渲染引擎以不同的方式处理。例如，背景几何体通常具有包含预算光照的光照贴图，这对于前景对象来说是不可行的。预算光照通常通过某种类型的体积表示应用于前景对象，这种表示可以考虑每个对象随时间变化的位置。

有些游戏有相对封闭的环境 在那里相机重新主要到位.最纯粹的例子是格斗游戏，如街头霸王系列，但对于DevilMayCry和GodOfWar等游戏来说，这在某种程度上也是如此。这些游戏的摄像机不在直接玩家控制下，游戏往往从一个封闭的环境移动到另一个封闭的环境，在每个环境中花费大量的游戏时间。这使得游戏developer在每个房间或封闭的环境中浪费大量的资源（处理，存储和艺术家时间），从而产生非常高的图形保真度。

其他游戏有非常大的世界，玩家可以自由移动。对于侠盗猎车手系列等“沙盒游戏”和魔兽世界等在线角色扮演游戏来说，情况最为如此。这样的游戏给图形开发者带来了很大的挑战，因为资源分配是非常困难的，当在每个帧中玩家可以看到很大程度的世界。进一步复杂化的事情，玩家可以自由地去一些以前遥远的世界的一部分，并从近距离观察它。这样的游戏通常有一天中不断变化的时间，这使得照明的预算至多是困难的，如果不是不可能的话。大多数游戏，如第一人称射击游戏，都介于两个极端之间。玩家可以在每帧中看到相当数量的场景，但在游戏世界中的移动受到一定的限制。许多游戏也有一个固定的时间为每个游戏级别，以方便照明预算。

渲染的前景对象的数量在游戏类型之间也有很大差异。实时战略游戏，如命令与征服系列通常有许多几十个，如果不是数百个，在屏幕上可见的单位。其他类型的游戏的可见角色数量更有限，而格斗游戏则处于相反的极端，只有两个角色可见，每个角色都以极高的细节呈现。必须区分在任何时间可见的字符数（这会影响处理时间的预算）和



Figure 25.2. Crysis exemplifies the realistic and detailed graphics expected of first-person shooters.
Image courtesy Crytek.

the number of *unique* characters which can potentially be visible at short notice (which affects storage budgets).

The type or *genre* of game also determines audience expectations of the graphics. For example, first-person shooters have historically had very high levels of graphics fidelity, and this expectation drives the graphics design when developing new games in that genre; see Figure 25.2. On the other hand, puzzle games have typically had relatively simplistic graphics, so most game developers will not invest large amounts of programming or art resources into developing photorealistic graphics for such games.

Although most games aim for a photorealistic look, a few do attempt more stylized rendering. One interesting example of this is *Okami*, which can be seen in Figure 25.3.

The management of development resources also differs by game type. Most games have a closed development cycle of one to two years, which ends after the game ships. Recently it has become common to have downloadable content (DLC), which can be purchased after the game ships, so some development resources need to be reserved for that. Persistent-world online games have a never-ending development process where new content is continually being generated, at least as long as the game is economically viable (which may be a period of decades).



图25.2。孤岛危机举例说明了第一人称射击游戏所期望的逼真和详细的图形。
图像礼貌Crytek.

在短时间内可能可见的唯一字符数（这会影响存储预算）。

游戏的类型或类型也决定了观众对图形的期望。例如，第一人称射击游戏在历史上具有非常高的图形保真度，这种期望在开发该类型的新游戏时推动了图形设计；见图25.2。另一方面，益智游戏通常具有相对简单的图形，因此大多数游戏开发人员不会投入大量编程或艺术资源来开发此类游戏的逼真图形。

虽然大多数游戏的目标是一个真实的外观，一些尝试更程式化的渲染。一个有趣的例子是*Okami*，可以在图25.3中看到。

开发资源的管理也因游戏类型而异。大多数游戏都有一到两年的封闭开发周期，在游戏出货后结束。最近，拥有可下载内容（DLC）变得很普遍，这些内容可以在游戏发布后购买，因此需要为此保留一些开发资源。持久性世界网络游戏有一个永无止境的发展过程，其中不断产生新的内容，至少只要游戏在经济上可行（可能是几十年的时间）。



Figure 25.3. An example of highly stylized, non-photorealistic rendering from the game *Okami*.
Image courtesy Capcom Entertainment, Inc.

The creative exploitation of the specific requirements and restrictions of a particular game is the hallmark of a skilled game graphics programmer. A good example is the game *LittleBigPlanet*, which has a “two-and-a-half-dimensional” game world comprising a small number of two-dimensional layers, as well as a noninteractive background. The graphics quality of this game is excellent, driven by the use of unusual rendering techniques specialized to this type of environment; see Figure 25.4.

25.5 The Game Production Process

The game production process starts with the basic game design or concept. In some cases (such as sequels), the basic gameplay and visual design is clear, and only incremental changes are made. In the case of a new game type, extensive prototyping is needed to determine gameplay and design. Most cases sit somewhere in the middle, where there are some new gameplay elements and the visual design is somewhat open. After this step there may be a *greenlight* stage where



Figure 25.3. 从游戏Okami高度程式化，非真实感渲染的一个例子。
图片由CapcomEntertainment Inc.提供。

对特定游戏的特定要求和限制的创造性开发是一个熟练的游戏图形程序员的标志。一个很好的例子是游戏LittleBigPlanet，它有一个包含少量二维图层的“二维半维”游戏世界，以及一个非交互式背景。这个游戏的图形质量非常好，这是由于使用了专门用于这种环境的不寻常的渲染技术所驱动的；见图25.4。

25.5 游戏制作过程

游戏制作过程从基本的游戏设计或概念开始。在某些情况下（如续集），基本的游戏玩法和视觉设计是明确的，并且只进行增量更改。在新游戏类型的情况下，需要广泛的原型来确定游戏性和设计。大多数情况下坐在中间的某个地方，那里有一些新的游戏元素，视觉设计有些开放。在这一步之后，可能会有一个绿灯阶段。



Figure 25.4. The *LittleBigPlanet* developers took care to choose techniques that fit the game's constraints, combining them in unusual ways to achieve stunning results. *LittleBigPlanet* © 2007 Sony Computer Entertainment Europe. Developed by Media Molecule. *LittleBigPlanet* is a trademark of Sony Computer Entertainment Europe.

some early demo or concept is shown to the game publisher to get approval (and funding!) for the game.

The next step is typically *pre-production*. While other teams are working on finishing up the last game, a small core team works on making any needed changes to the game engine and production tool chain, as well as working out the rough details of any new gameplay elements. This core team is working under a strict deadline. After the existing game ships and the rest of the team comes back from a well-deserved vacation, the entire tool chain and engine must be ready for them. If the core team misses this deadline, several dozen developers may be left idle—an extremely expensive proposition!

Full production is the next step, with the entire team creating art assets, designing levels, tweaking gameplay, and implementing further changes to the game engine. In a perfect world, everything done during this process would be used in the final game, but in reality there is an iterative nature to game development which will result in some work being thrown out and redone. The goal is to minimize this with careful planning and prototyping.

When the game is functionally complete, the final stage begins. The term *alpha release* usually refers to the version which marks the start of extensive internal testing, *beta release* to the one which marks the start of extensive external testing, and *gold release* to the final release submitted to the console manufacturer,



图25.4。 LittleBigPlanet开发人员注意选择适合游戏约束的技术，以不同寻常的方式将它们结合起来以获得惊人的结果。LittleBigPlanet©2007索尼电脑娱乐欧洲.由MediaMolecule开发。LittleBigPlanet是索尼电脑娱乐欧洲.

一些早期的演示或概念显示给游戏发行商以获得批准（和资金！）的游戏。

下一步通常是预生产。当其他团队正在完成最后一个游戏时，一个小型核心团队正在对游戏引擎和生产工具链进行任何必要的更改，以及制定任何新游戏元素的粗略细节。这个核心团队正在严格的最后期限下工作。在现有的游戏船只和团队的其他成员从当之无愧的假期回来之后，整个工具链和引擎必须为他们做好准备。如果核心团队错过了这个最后期限，几十个开发人员可能会闲置—这是一个非常昂贵的提议！

下一步是全面制作，整个团队创建艺术资产，取消签级别，调整游戏玩法，并对游戏引擎进行进一步更改。在一个完美的世界里，在这个过程中所做的一切都将在最后的游戏中使用，但实际上游戏开发具有迭代的性质，这将导致一些工作被抛出和重做。我们的目标是通过仔细的规划和原型来最大限度地模仿这一点。

当游戏功能完成后，最后阶段开始。术语alpha版本通常是指标志着广泛内部测试开始的版本，beta版本是指标志着广泛外部测试开始的版本，黄金版本是指提交给控制台制造商的最终版本

but different companies have slightly varying definitions of these terms. In any case, testing, or *quality assurance* (QA) is an important part of this phase, and it involves testers at the game development studio, at the publisher, at the console manufacturer, and possibly external QA contractors as well. These various rounds of testing result in bug reports which are submitted back to the game developers and worked on until the next release.

After the game ships, most of the developers go on vacation for a while, but a small team may have to stay to work on patches or downloadable content. In the meantime, a small core team has been working on pre-production for the next game.

Art asset creation is an aspect of game production that is particularly relevant to graphics development, so I will go into it in some detail.

25.5.1 Asset Creation

While the exact process of art asset creation varies from game to game, the outline I give here is fairly representative. In the past, a single artist would create an entire asset from start to finish, but this process is now much more specialized, involving people with different skill sets working on each asset at various times. Some of these stages have clear dependencies (for example, a character cannot be animated until it is rigged and cannot be rigged before it is modeled). Most game developers have well-defined approval processes, where the art director or a lead artist signs off on each stage before the asset is sent on to the next. Ideally an asset proceeds through each stage exactly once, but in practice changes may be made that require resubmission.

Initial Modeling

Typically the art asset creation process starts by modeling the object geometry. This step is performed in a general-purpose modeling package such as Maya, MAX or Softimage. The modeled geometry will be passed directly to the game engine, so it is important to minimize vertex count while preserving good silhouettes. Character meshes must also be constructed so as to be amenable to animation.

In this stage, a two-dimensional surface parameterization for textures is usually created. It is important that this parameterization be highly continuous, since discontinuities require vertex duplication and may cause filtering artifacts. An example of a mesh with its associated texture parameterization is shown in Figure 25.5.

但不同的公司对这些术语的定义略有不同。无论如何，测试或质量保证 (QA) 是这个阶段的重要组成部分，它涉及游戏开发工作室，发行商，控制台制造商以及可能的外部QA承包商的测试人员。这些不同的测试结果的错误报告提交给游戏开发者，并一直工作到下一个版本。

游戏发布后，大多数开发人员都会去度假一段时间，但一个小团队可能不得不留下来处理补丁或可下载的内容。与此同时，一个小型核心团队一直在为下一场比赛进行前期制作。

美术资源创建是游戏制作的一个方面，与图形开发特别相关，因此我将详细介绍它。

25.5.1 资产创造

虽然艺术资产创建的确切过程因游戏而异，但我在这里给出的大纲相当具有代表性。过去，单个美工师会从头到尾创建整个资产，但现在这个过程更加专业化，涉及具有不同技能的人员在不同时间处理每个资产。其中一些阶段具有明确的依赖关系（例如，角色在被操纵之前不能动画，并且在被建模之前不能被操纵）。大多数游戏开发者都有明确的审批流程，在资产发送到下一个阶段之前，艺术总监或主要艺术家在每个阶段签名。理想情况下，资产在每个阶段只进行一次，但在实践中可能会发生需要重新提交的变化。

初始建模

通常，艺术资源创建过程从对象几何体建模开始。此步骤在Maya、MAX或Softimage等通用建模包中执行。建模后的几何体将直接传递到游戏引擎，因此在保持良好的silhouettes的同时最小化顶点计数非常重要。还必须构建角色网格，以便适应动画。

在这个阶段，通常会创建纹理的二维表面参数化。重要的是，这种参数化是高度连续的，因为不连续性需要顶点重复并且可能导致过滤伪影。具有相关纹理参数化的网格的示例如图Figure 25.5所示。

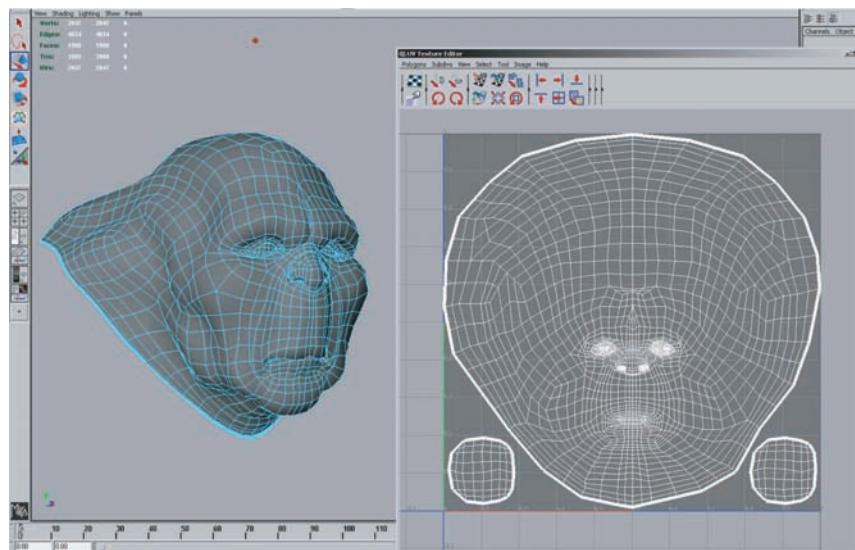


Figure 25.5. A mesh being modeled in Maya, with associated texture parameterization. *Image courtesy Keith Bruns.*

Texturing

In the past, texturing was a straightforward process of painting a color texture, typically in Photoshop. Now, specialized detail modeling packages such as ZBrush or Mudbox are commonly used to sculpt fine surface detail. Figures 25.6 and 25.7 show an example of this process.

If this additional detail were to be represented with actual geometry, millions of triangles would be needed. Instead, the detail is commonly “baked” into a normal map which is applied onto the original, coarse mesh, as shown in Figures 25.8 and 25.9.

Besides normal maps, multiple textures containing surface properties such as diffuse color, specular color, and smoothness (specular power) are also created. These are either painted directly on the surface in the detail modeling application, or in a two-dimensional application such as Photoshop. All of these texture maps use the surface parameterization defined in the initial modeling phase. When the texture is painted in a two-dimensional painting application, the artist must frequently switch between the painting application and some other application which can show a three-dimensional rendering of the object with the texture applied. This iterative process is illustrated in Figures 25.10, 25.11, 25.12, and 25.13.

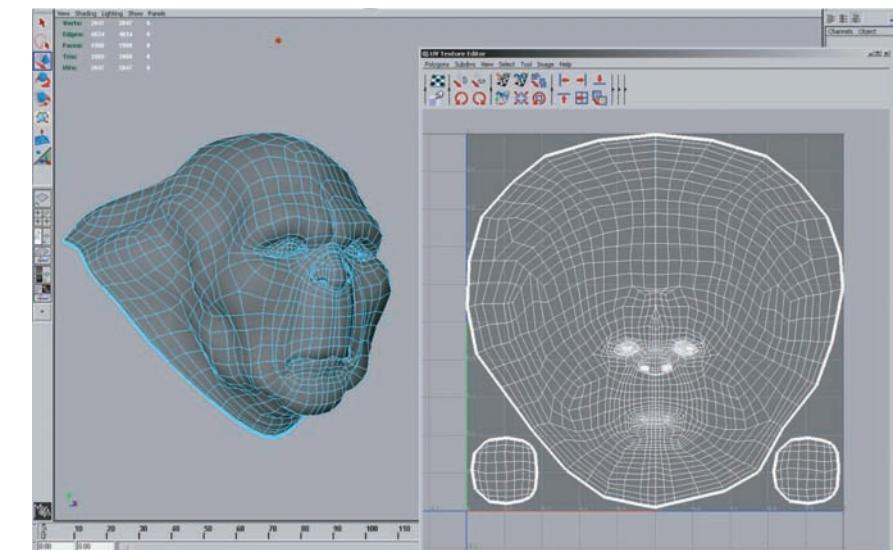


图25.5。 在Maya中建模的网格，并使用相关的纹理参数化。图片由基思·布伦斯提供。

Texturing

在过去，纹理化是绘制颜色纹理的简单过程，通常在Photoshop中使用。现在，专门的细节建模包（如ZBrush或Mudbox）通常用于雕刻精细的表面细节。图25.6和25.7显示了这个过程的一个例子。

如果要用实际几何来表示这个额外的细节，则需要数百万个三角形。相反，细节通常被“烘焙”到一个normal映射中，该映射被应用到原始的粗网格上，如图25.8和图25.9所示。

除了法线贴图之外，还会创建包含漫反射颜色、高光颜色和平滑度（高光强度）等表面属性的多个纹理。它们要么在细节建模应用程序中直接绘制在表面上，要么在Photoshop等二维应用程序中绘制。所有这些纹理贴图都使用初始建模阶段中定义的曲面参数化。当在二维绘画应用程序中绘制纹理时，艺术家必须在绘画应用程序和一些其他应用程序之间快速切换，这些应用程序可以显示应用纹理的对象的三维图25.10、25.11、25.12和25.13说明了这个迭代过程。

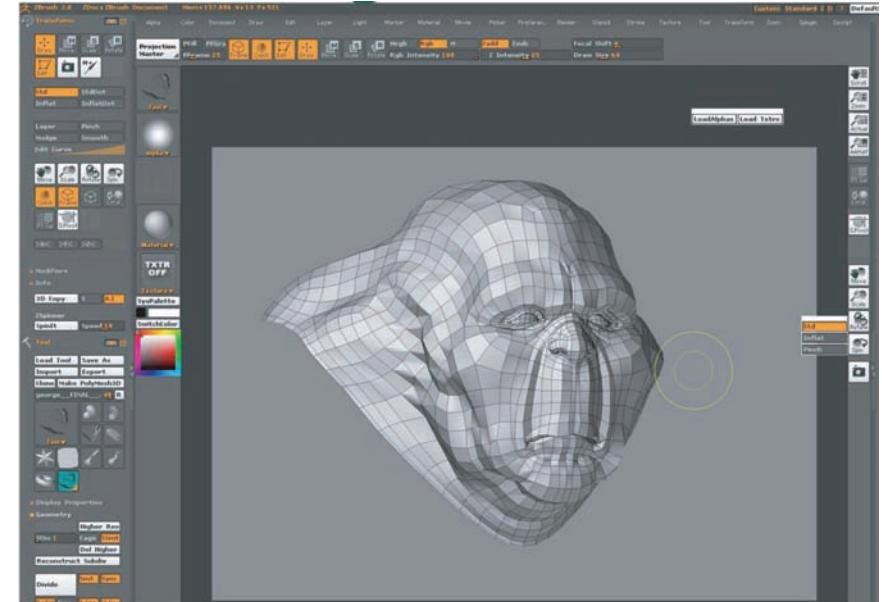


Figure 25.6. The mesh from Figure 25.5 has been brought into ZBrush for detail modeling. *Image courtesy Keith Bruns.*

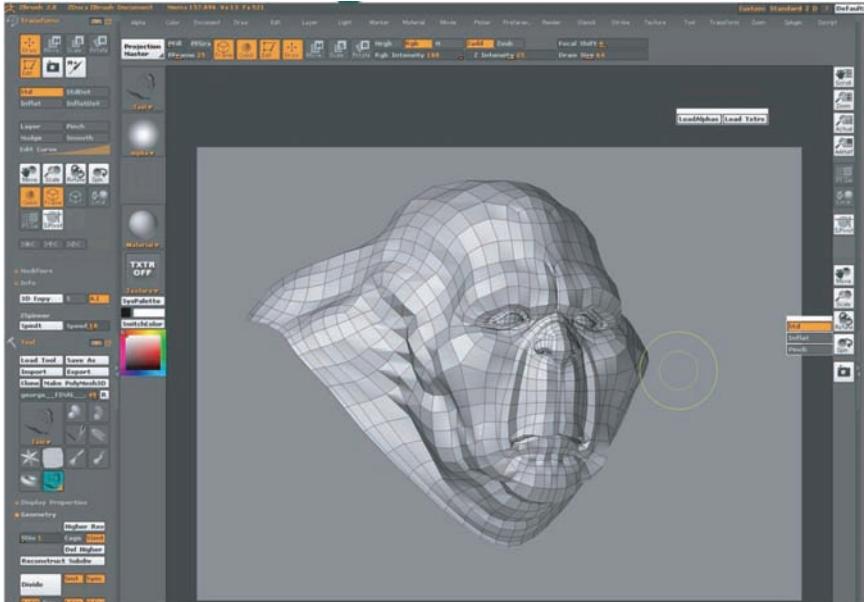


图25.6。图25.5中的网格已被带入ZBrush中进行细节建模。图片由基思·布伦斯提供。

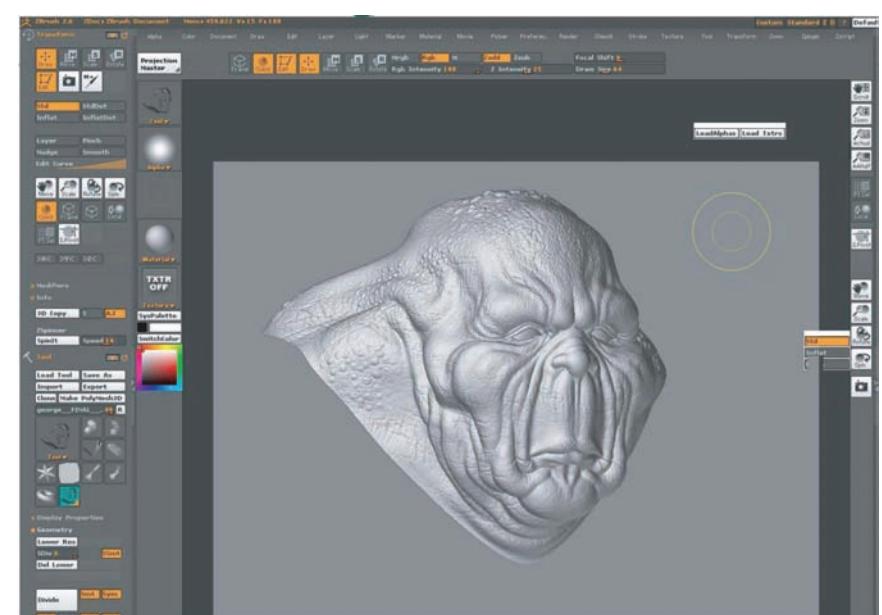


Figure 25.7. The mesh from Figure 25.6, with fine detail added to it in ZBrush. *Image courtesy Keith Bruns.*

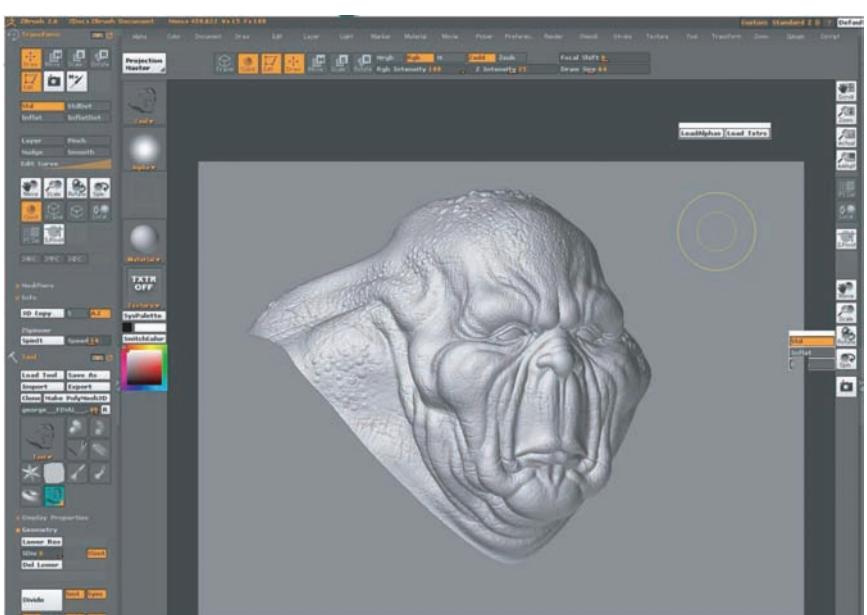


图25.7。图25.6中的网格，在ZBrush中添加了精细细节。图片提供Keith Bruns.



Figure 25.8. A visualization (in ZBrush) of the mesh from Figure 25.6, rendered with a normal map derived from the detailed mesh in Figure 25.7. The bottom of the figure shows the interface for ZBrush’s “Zmapper” tool, which was used to derive the normal map. *Image courtesy Keith Bruns.*



图25.6中网格的可视化（在ZBrush中），用图25.7中详细网格的法线贴图渲染。图的底部显示了ZBrush的“Zmapper”工具的界面，该工具用于派生法线贴图。图片由基思·布伦斯提供。

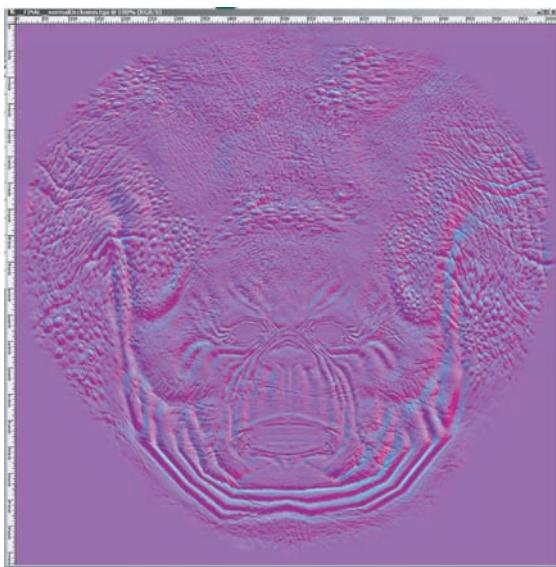


Figure 25.9. The normal map used in Figure 25.8. In this image, the red, green, and blue channels of the texture contain the X, Y, and Z coordinates of the surface normals. *Image courtesy Keith Bruns.*

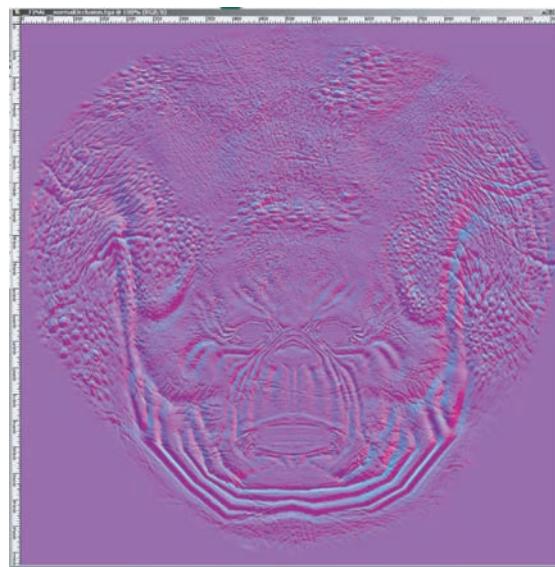


图25.9。图25.8中使用的法线贴图。在此图像中，纹理的红色、绿色和蓝色通道包含表面法线的X、Y和Z坐标。图片由基思·布伦斯提供。

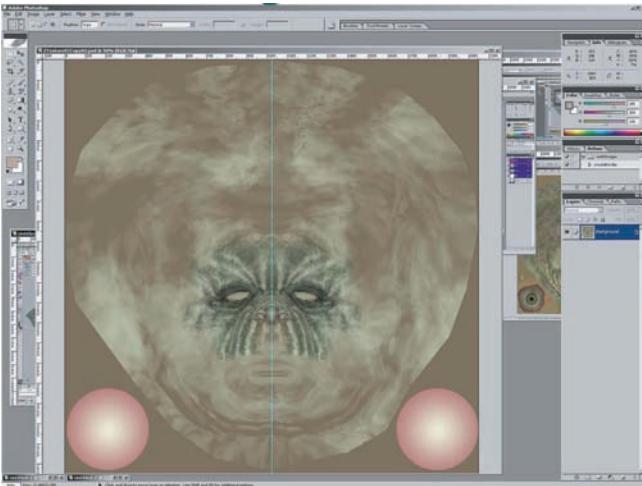


Figure 25.10. An early version of a diffuse color texture for the mesh from Figure 25.8, shown in Photoshop. *Image courtesy Keith Bruns.*

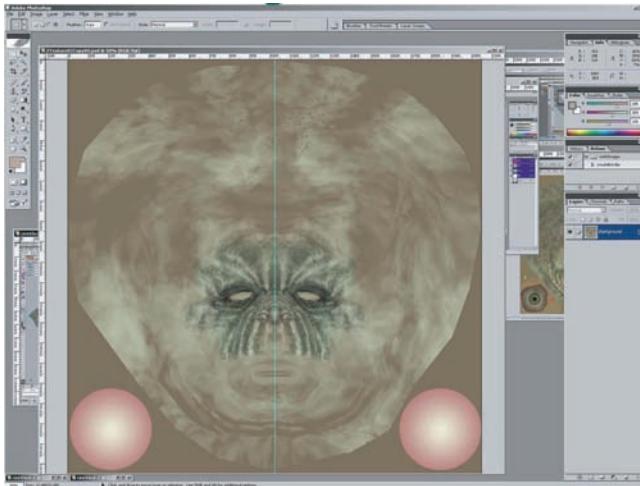


Figure 25.10. 图25.8中网格的漫反射颜色纹理的早期版本，如下所示
Photoshop。图片由基思·布伦斯提供。

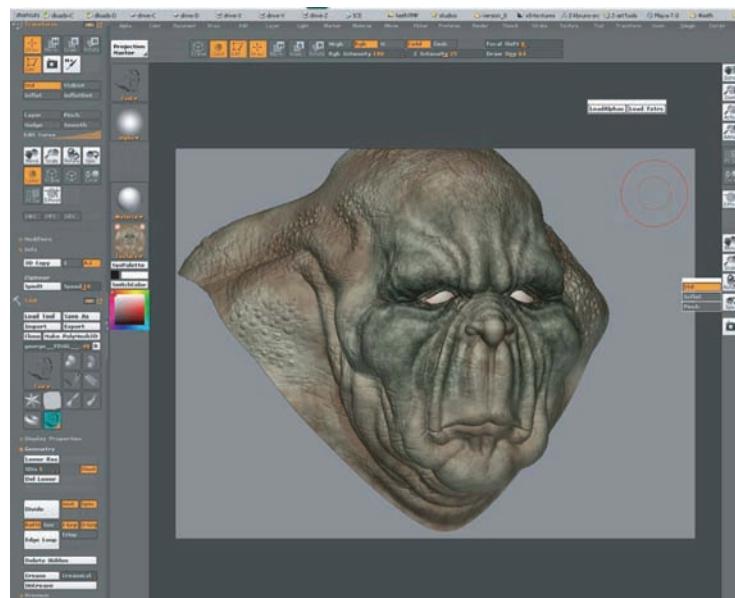


Figure 25.11. A rendering (in ZBrush) of the mesh with normal map and early diffuse color texture (from Figure 25.10) applied. *Image courtesy Keith Bruns.*

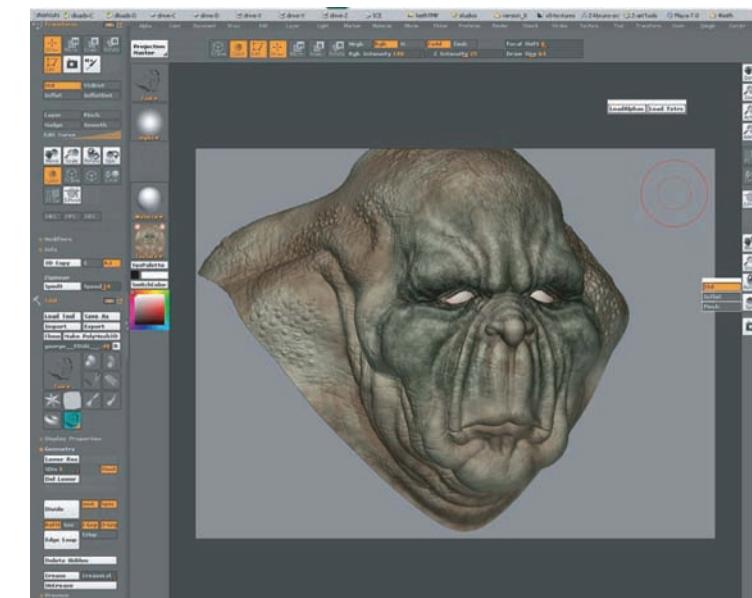


图25.11。 应用法线贴图和早期漫反射颜色纹理（来自图25.10）的网格的渲染（在ZBrush中）。图片由基思·布伦斯提供。



Figure 25.12. Final version of the color texture from Figure 25.10. *Image courtesy Keith Bruns.*



图25.12。最终版本的颜色纹理从图25.10。图片由基思·布伦斯提供。

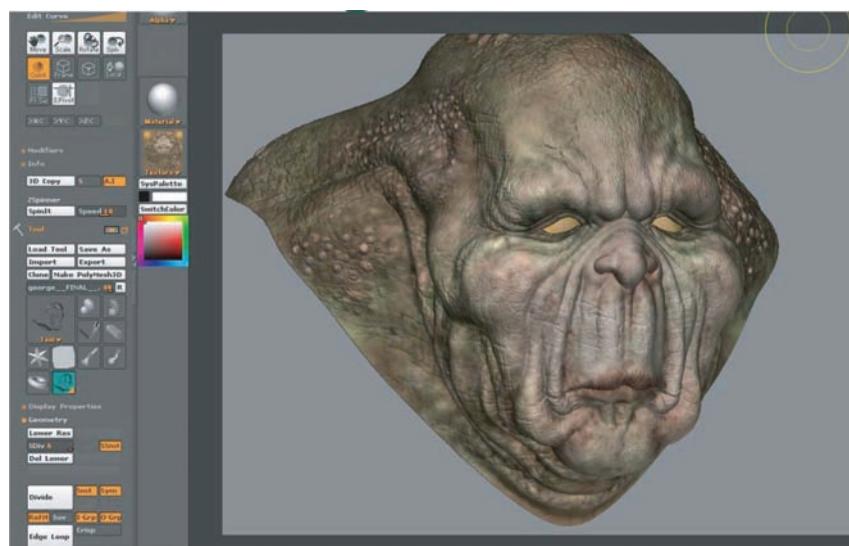
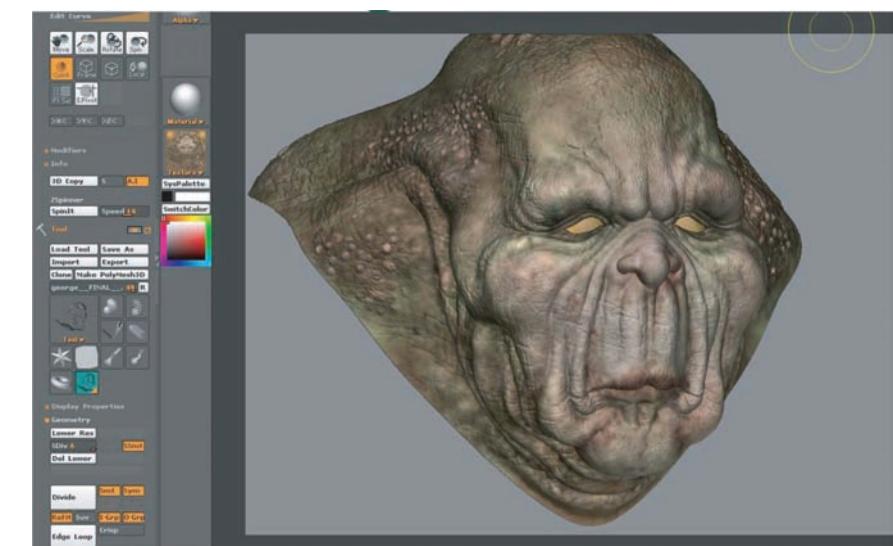


Figure 25.13. Rendering of the mesh with normal map and final color texture (from Figure 25.12) applied. *Image courtesy Keith Bruns.*



应用法线贴图和最终颜色纹理的网格渲染（来自图25.12）。图片由基思·布伦斯提供。

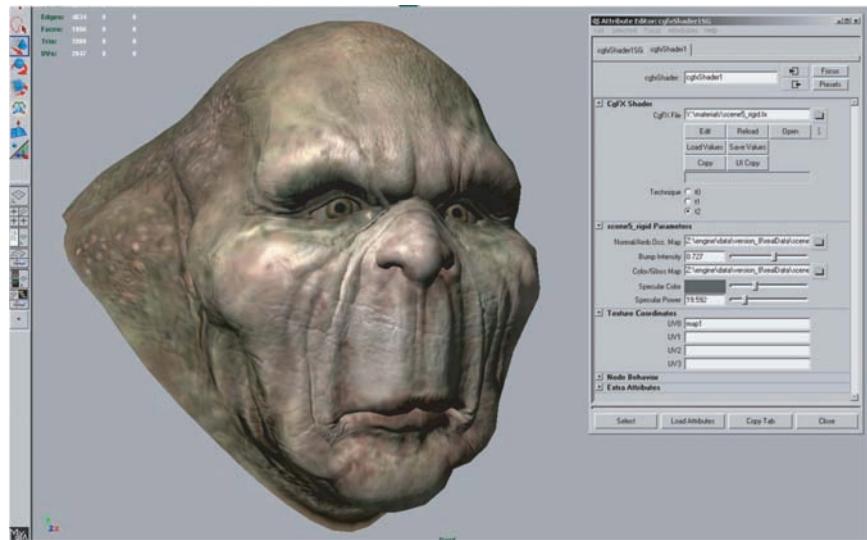


Figure 25.14. Shader configuration in Maya. The interface on the right is used to select the shader, assign textures to shader inputs, and set the values of non-texture shader inputs (such as the “Specular Color” and “Specular Power” sliders). The rendering on the left is updated dynamically while these properties are modified, enabling immediate visual feedback. *Image courtesy Keith Bruns.*

Shading

Shaders are typically applied in the same application used for initial modeling. In this process, a shader (from the set of shaders defined for that game) is applied to the mesh. The various textures resulting from the detail modeling stage are applied as inputs to this shader, using the surface parameterization defined during initial modeling. Various other shader inputs are set via visual experimentation (“tweaking”); see Figure 25.14.

Lighting

In the case of background scenery, lighting artists will typically start their work after modeling, texturing, and shading have been completed. Light sources are placed and their effect computed in a preprocessing step. The results of this process are stored in lightmaps for later use by the rendering engine.

Animation

Character meshes undergo several additional steps related to animation. The primary method used to animate game characters is *skinning*. This requires a *rig*, consisting of a hierarchy of transform nodes that is attached to the character, a

图25.14。 Maya中的着色器配置。右侧的界面用于选择着色器，将纹理分配给着色器输入，以及设置非纹理着色器输入的值（例如“镜面颜色”和“镜面功率”滑块）。在修改这些属性时，左侧的渲染会动态更新，从而实现即时的视觉反馈。图片由基思·布伦斯提供。

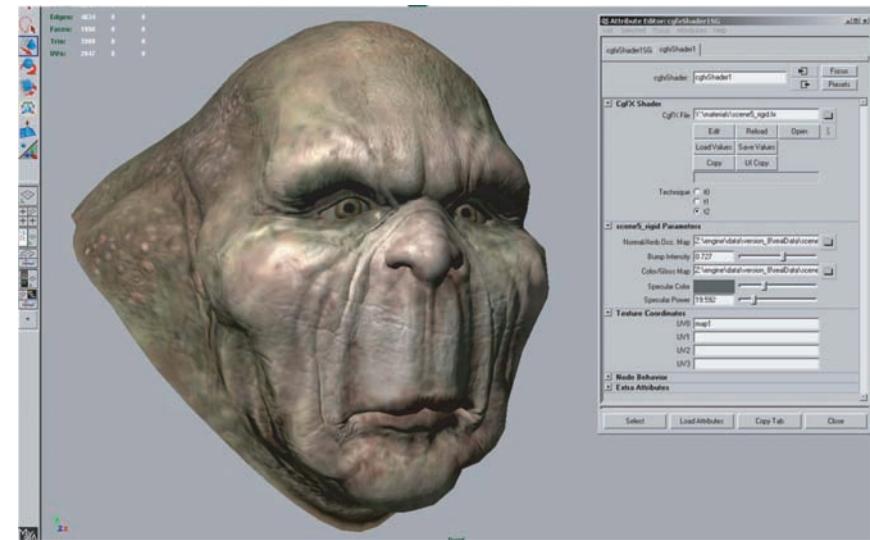


图25.14。 Maya中的着色器配置。右侧的界面用于选择着色器，将纹理分配给着色器输入，以及设置非纹理着色器输入的值（例如“镜面颜色”和“镜面功率”滑块）。在修改这些属性时，左侧的渲染会动态更新，从而实现即时的视觉反馈。图片由基思·布伦斯提供。

Shading

着色器通常应用于用于初始建模的相同应用程序中。在此过程中，着色器（来自为该游戏定义的着色器集）应用于网格。从细节建模阶段产生的各种纹理作为输入应用到这个着色器，使用在初始建模期间定义的表面参数化。通过视觉实验（“调整”）设置各种其他着色器输入；见图25.14。

Lighting

对于背景场景，照明艺术家通常会在建模、纹理和着色完成后开始他们的工作。在预处理步骤中放置光源并计算其效果。此过程的结果存储在光照贴图中，供渲染引擎以后使用。

Animation

角色网格体经历了与动画相关的几个额外步骤。用于动画游戏角色的主要方法是换肤。这需要一个rig，由附加到角色的变换节点的层次结构组成，

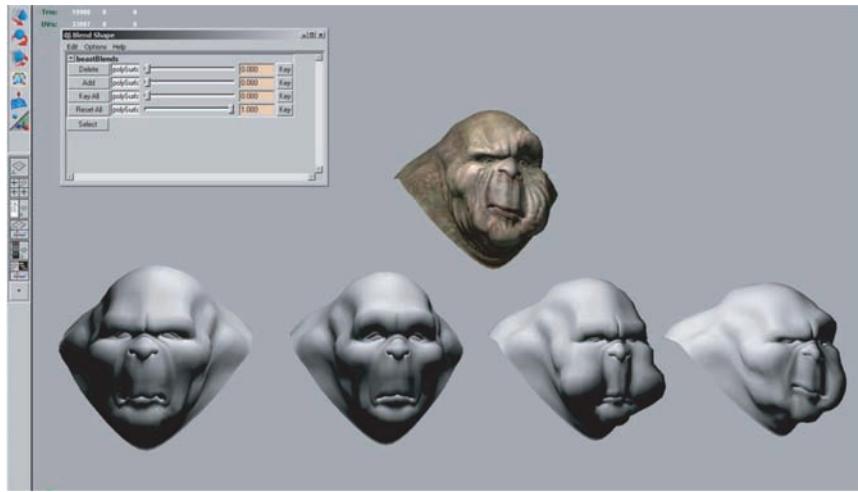


Figure 25.15. Morph target interface in Maya. The bottom row shows four different morph targets, and the model at the top shows the effects of combining several morph targets together. The interface at the upper left is used to control the degree to which each morph target is applied. *Image courtesy Keith Bruns.*

process known as *rigging*. The area of effect of each transform node is painted onto a subset of mesh vertices. Finally, animators create animations that move, rotate, and scale these transform nodes, “dragging” the mesh behind them.

A typical game character will have many dozens of animations, corresponding to different modes of motion (walking, running, turning) as well as different actions such as attacks. In the case of a main character, the number of animations can be in the hundreds. Transitions between different animations also need to be defined.

For facial animation, another technique, called *morph targets* is sometimes employed. In this technique, the mesh vertices are directly manipulated to deform the mesh. Different copies of the deformed mesh are stored (e.g., for different facial expressions) and combined by the game engine at runtime. The creation of morph targets is shown in Figure 25.15.

Notes

There is a huge amount of information on real-time rendering and game programming available, both in books and online. Here are some resources I can recommend from personal familiarity.

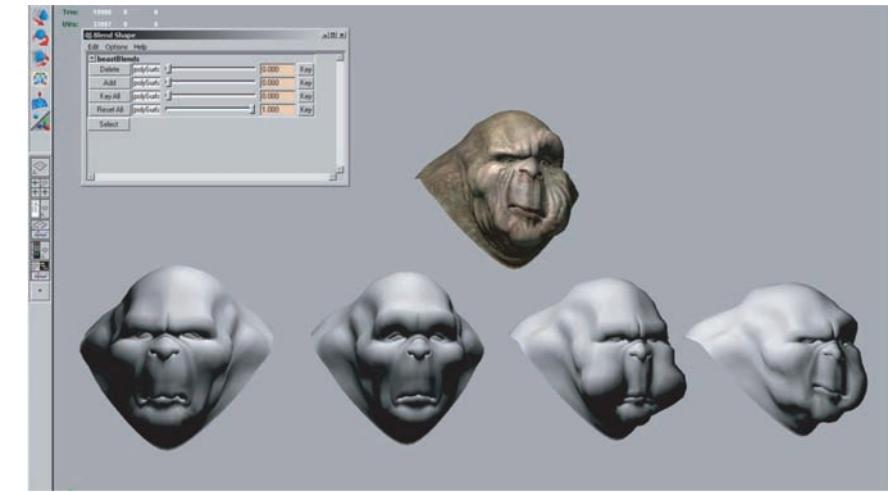


图25.15。Maya中的变形目标界面。底部一行显示了四个不同的变形目标，顶部的模型显示了将多个变形目标组合在一起的效果。左上方的界面用于控制每个变形目标应用的程度。图片由基思·布伦斯提供。

被称为索具的过程。每个变换节点的效果区域被绘制到网格顶点的子集上。最后，动画师创建移动、旋转和缩放这些变换节点的动画，“拖动”它们后面的网格。

一个典型的游戏角色会有许多几十个动画，对应于不同的运动模式（步行，跑步，转弯）以及不同的动作，如攻击。在一个主要角色的情况下，动画的数量可以是数百个。还需要定义不同动画之间的过渡。

对于面部动画，有时会使用另一种称为变形目标的技术。在这种技术中，直接操作网格顶点以使网格变形。变形网格的不同副本被存储（例如，用于不同的面部表情），并在运行时由游戏引擎组合。变形目标的创建如图25.15所示。

Notes

无论是书籍还是在线，都有大量关于实时渲染和游戏编程的信息。以下是我可以从个人熟悉中推荐的一些资源。



Game Developer Magazine is a good source of information on game development, as are slides from the talks given at the annual *Game Developers Conference* (GDC) and Microsoft's *Gamefest* conference. The *GPU Gems* and *ShaderX* book series also contain good information—all of the former and the first two of the latter are also available online.

Eric Lengyel's *Mathematics for 3D Game Programming & Computer Graphics*, now in its second edition, is a good reference for the various types of math used in graphics and games. A specific area of game programming that is closely related to graphics is collision detection, for which Christer Ericson's *Real-Time Collision Detection* is the definitive resource.

Since its first edition in 1999, Eric Haines and Tomas Akenine-Möller's *Real-Time Rendering* has endeavored to cover this fast-growing field in a thorough manner. As a longtime fan of this book, I was glad to have the opportunity to be a coauthor on the third edition, which came out in mid-2008.

Reading is not enough—make sure you play a variety of games regularly to get a good idea of the requirements of various game types, as well as the current state of the art.

Exercises

1. Examine the visuals of two dissimilar games. What differences can you deduce in the graphics requirements of these two games? Analyze the effect on rendering time, storage budgets, etc.



游戏开发者杂志是一个很好的信息来源，游戏开发者协会（GDC）和微软Gamefest会议上的演讲幻灯片也是如此。GPU Gems和ShaderX系列书籍也包含很好的信息—所有前者和后者的前两个也可以在线获得。

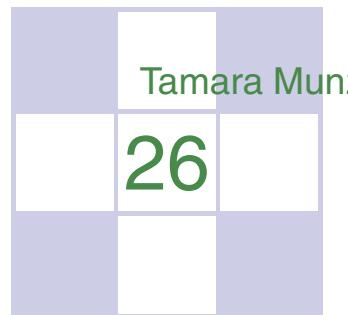
Eric Lengyel的Mathematics for 3D Game Programming & Computer Graphics，现已进入第二版，是图形和游戏中使用的各种类型数学的好参考。与图形密切相关的游戏编程的一个特定领域是碰撞检测，克里斯特·埃里克森的实时碰撞检测是最终的资源。

自1999年第一版以来，Eric Haines和Tomas Akenine-Möller的Real Time Rendering努力以彻底的方式复盖这个快速增长的领域。作为这本书的长期粉丝，我很高兴有机会成为第三版的合着者，该版于2008年年中出版。

阅读是不够的—确保你经常玩各种游戏，以了解各种游戏类型的要求，以及当前的艺术状态。

Exercises

1. 检查两个不同游戏的视觉效果。你可以在这两款游戏的图形要求中推断出哪些差异？分析对渲染时间、存储预算等的影响。

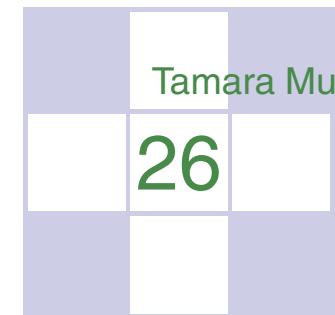


Visualization

A major application area of computer graphics is *visualization*, where computer-generated images are used to help people understand both spatial and nonspatial data. Visualization is used when the goal is to augment human capabilities in situations where the problem is not sufficiently well defined for a computer to handle algorithmically. If a totally automatic solution can completely replace human judgment, then visualization is not typically required. Visualization can be used to generate new hypotheses when exploring a completely unfamiliar dataset, to confirm existing hypotheses in a partially understood dataset, or to present information about a known dataset to another audience.

Visualization allows people to offload cognition to the perceptual system, using carefully designed images as a form of *external memory*. The human visual system is a very high-bandwidth channel to the brain, with a significant amount of processing occurring in parallel and at the pre-conscious level. We can thus use external images as a substitute for keeping track of things inside our own heads. For an example, let us consider the task of understanding the relationships between a subset of the topics in the splendid book *Gödel, Escher, Bach: The Eternal Golden Braid* (Hofstadter, 1979); see Figure 26.1.

When we see the dataset as a text list, at the low level we must read words and compare them to memories of previously read words. It is hard to keep track of just these dozen topics using cognition and memory alone, let alone the hundreds of topics in the full book. The higher-level problem of identifying neighborhoods, for instance finding all the topics two hops away from the target topic *Paradoxes*, is very difficult.



Visualization

计算机图形学的一个主要应用领域是可视化，其中计算机生成的图像用于帮助人们理解空间和非空间数据。当目标是在问题没有足够好地定义计算机以算法处理的情况下增强人类能力时，使用可视化。如果完全自动的解决方案可以完全取代人类的判断，那么通常不需要可视化。可视化可用于在探索完全陌生的数据集时生成新的假设，确认部分理解的数据集中的现有假设，或向其他受众呈现有关已知数据集的信息。

可视化允许人们将认知卸载到感知系统，我们将精心设计的图像作为外部记忆的一种形式。人类视觉系统是一个非常高带宽的大脑通道，大量的处理并行发生和预先意识水平。因此，我们可以使用外部图像作为跟踪我们自己头脑中的事物的替代品。举个例子，让我们考虑理解精彩着作Godel, Escher, Bach: TheEternalGoldenBraid (Hofstadter, 1979) 中主题子集之间关系的任务;见图26.1。

当我们把数据集视为文本列表时，在低级别，我们必须阅读单词并将它们与以前阅读过的单词的记忆进行比较。单凭认知和记忆很难跟踪这十几个主题，更不用说整本书中的数百个主题了。识别邻域的更高层次的问题，例如找到距离目标主题悖论两跳的所有主题，是非常困难的。

Infinity - Lewis Carroll	Epimenides - Self-ref
Infinity - Zeno	Epimenides - Tarski
Infinity - Paradoxes	Tarski - Epimenides
Infinity - Halting problem	Halting problem - Decision procedures
Zeno - Lewis Carroll	Halting problem - Turing
Paradoxes - Lewis Carroll	Lewis Carroll - Wordplay
Paradoxes - Epimenides	Epimenides - Truth vs. provability
Paradoxes - Self-ref	Tarski - Undecidability

Figure 26.1. Keeping track of relationships between topics is difficult using a text list.

Figure 26.2 shows an external visual representation of the same dataset as a node-link graph, where each topic is a node and the linkage between two topics is shown directly with a line. Following the lines by moving our eyes around the image is a fast low-level operation with minimal cognitive load, so higher-level neighborhood finding becomes possible. The placement of the nodes and the routing of the links between them was created automatically by the *dot* graph drawing program (Gansner, Koutsofios, North, & Vo, 1993).

We call the mapping of dataset attributes to a visual representation a *visual encoding*. One of the central problems in visualization is choosing appropriate encodings from the enormous space of possible visual representations, taking into account the characteristics of the human perceptual system, the dataset in question, and the task at hand.

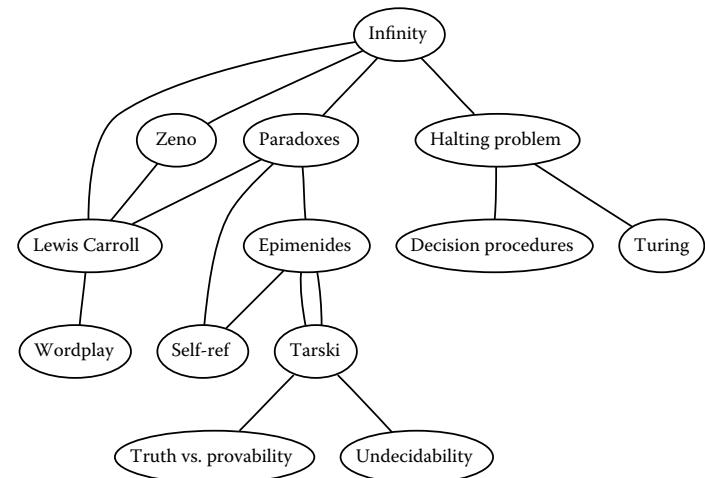


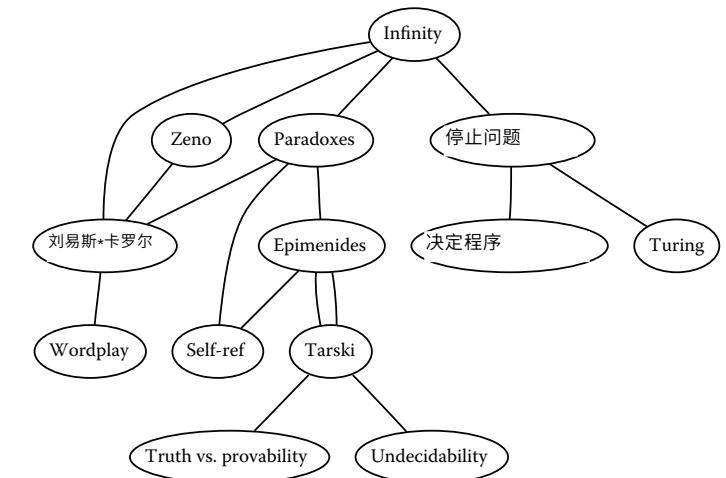
Figure 26.2. Substituting perception for cognition and memory allows us to understand relationships between book topics quickly.

无限刘易斯卡罗尔	Epimenides - Self-ref
Infinity - Zeno	Epimenides - Tarski
无限悖论无限停止问题	Tarski - Epimenides
芝诺*刘易斯*卡罗尔	停止问题决定程序
悖论刘易斯*卡罗尔	停止问题图灵刘易斯卡
Paradoxes - Epimenides	罗尔文字游戏
Paradoxes - Self-ref	Tarski - Truth vs. provability
	Tarski - Undecidability

图26.1。使用文本列表很难跟踪主题之间的关系。

图26.2显示了同一数据集的外部可视化表示作为节点链接图，其中每个主题是一个节点，两个主题之间的链接直接用一条线显示。通过在图像周围移动我们的眼睛来跟踪线条是一种快速的低级操作，具有最小的认知负荷，因此可以实现更高层次的邻域查找。节点的放置和它们之间的链接的路由是由点图绘制程序自动创建的（Gansner, Koutsofios, North, & Vo, 1993）。

我们将数据集属性到可视化表示的映射称为可视化编码。可视化的核心问题之一是从可能的视觉表示的巨大空间中选择适当的编码，同时考虑到人类感知系统的特征，所讨论的数据集和手头的任务。



用感知代替认知和记忆使我们能够快速理解书籍主题之间的关系。



26.1 Background

26.1.1 History

People have a long history of conveying meaning through static images, dating back to the oldest known cave paintings from over thirty thousand years ago. We continue to visually communicate today in ways ranging from rough sketches on the back of a napkin to the slick graphic design of advertisements. For thousands of years, cartographers have studied the problem of making maps that represent some aspect of the world around us. The first visual representations of abstract, nonspatial datasets were created in the 18th century by William Playfair (Friendly, 2008).

Although we have had the power to create moving images for over one hundred and fifty years, creating dynamic images interactively is a more recent development only made possible by the widespread availability of fast computer graphics hardware and algorithms in the past few decades. Static visualizations of tiny datasets can be created by hand, but computer graphics enables interactive visualization of large datasets.

26.1.2 Resource Limitations

When designing a visualization system, we must consider three different kinds of limitations: computational capacity, human perceptual and cognitive capacity, and display capacity.

As with any application of computer graphics, computer time and memory are limited resources and we often have hard constraints. If the visualization system needs to deliver interactive response, then it must use algorithms that can run in a fraction of a second rather than minutes or hours.

On the human side, memory and attention must be considered as finite resources. Human memory is notoriously limited, both for long-term recall and for shorter-term working memory. Later in this chapter, we discuss some of the power and limitations of the low-level visual attention mechanisms that carry out massively parallel processing of the visual field. We store surprisingly little information internally in visual working memory, leaving us vulnerable to *change blindness*, the phenomenon where even very large changes are not noticed if we are attending to something else in our view (Simons, 2000). Moreover, vigilance is also a highly limited resource; our ability to perform visual search tasks degrades quickly, with far worse results after several hours than in the first few minutes (Ware, 2000).



26.1 Background

26.1.1 History

人们通过静态图像传达意义的悠久历史，可以追溯到三万多年前已知最古老的洞穴壁画。我们今天继续以视觉传达的方式，从餐巾纸背面的粗略草图到光滑的广告平面设计。几千年来，制图师一直在研究制作代表我们周围世界某些方面的地图的问题。抽象，非空间数据集的第一个视觉表示是由WilliamPlayfair (Friendly, 2008) 在18世纪创建的。

虽然我们已经拥有创建动态图像的能力超过一年和五十年，但交互式创建动态图像是最近的发展，只有在过去几十年中快速计算机图形硬件和算法的可以手工创建微小数据集的静态可视化，但计算机图形可以实现大型数据集的交互式可视化。

26.1.2 资源限制

在设计可视化系统时，我们必须考虑三种不同的限制：计算能力、人类感知和认知能力以及显示能力。

与计算机图形学的任何应用一样，计算机时间和内存是有限的资源，我们经常有硬性限制。如果可视化系统需要提供交互式响应，那么它必须使用可以在几分之一秒而不是几分钟或几小时内运行的算法。

在人的方面，记忆和注意力必须被视为有限的来源。众所周知，人类的记忆是有限的，无论是长期记忆还是短期工作记忆。在本章的后面，我们讨论了对视场进行大规模并行处理的低级视觉注意机制的一些力量和局限性。我们在视觉工作记忆中内部存储的形式令人惊讶地很少，使我们容易受到变化失明的影响，这种现象即使是非常大的变化也不会被注意到，如果我们此外，vigilance也是一个高度有限的资源；我们执行视觉搜索任务的能力很快就会下降，几个小时后的结果比最初几分钟（Ware, 2000）差得多。

Display capacity is a third kind of limitation to consider. Visualization designers often “run out of pixels,” where the resolution of the screen is not large enough to show all desired information simultaneously. The *information density* of a particular frame is a measure of the amount of information encoded versus the amount of unused space. There is a tradeoff between the benefits of showing as much as possible at once, to minimize the need for navigation and exploration, and the costs of showing too much at once, where the user is overwhelmed by visual clutter.

26.2 Data Types

Many aspects of a visualization design are driven by the type of the data that we need to look at. For example, is it a table of numbers, or a set of relations between items, or inherently spatial data such as a location on the Earth’s surface or a collection of documents?

We start by considering a table of data. We call the rows *items* of data and the columns are *dimensions*, also known as *attributes*. For example, the rows might represent people, and the columns might be names, age, height, shirt size, and favorite fruit.

We distinguish between three types of dimensions: quantitative, ordered, and categorical. *Quantitative* data, such as age or height, is numerical and we can do arithmetic on it. For example, the quantity of 68 inches minus 42 inches is 26 inches. With *ordered* data, such as shirt size, we cannot do full-fledged arithmetic, but there is a well-defined ordering. For example, large minus medium is not a meaningful concept, but we know that medium falls between small and large. *Categorical* data, such as favorite fruit or names, does not have an implicit ordering. We can only distinguish whether two things are the same (apples) or different (apples vs. bananas).

Relational data, or *graphs*, are another data type where *nodes* are connected by *links*. One specific kind of graph is a *tree*, which is typically used for hierarchical data. Both nodes and edges can have associated attributes. The word *graph* is unfortunately overloaded in visualization. The node-link graphs we discuss here, following the terminology of graph drawing and graph theory, could also be called *networks*. In the field of statistical graphics, *graph* is often used for *chart*, as in the line charts for time-series data shown in Figure 26.10.

Some data is inherently spatial, such as geographic location or a field of measurements at positions in three-dimensional space as in the MRI or CT scans used by doctors to see the internal structure of a person’s body. The information as-

显示容量是需要考虑的第三种限制。可视化设计人员经常“用完像素”，其中屏幕的分辨率不足以同时显示所有所需的信息。特定帧的信息密度是编码的信息量与未使用空间量的度量。在一次显示尽可能多的好处之间存在权衡，以最大限度地减少导航和探索的需要，以及一次显示太多的成本，其中用户被视觉混乱所淹没。

26.2 数据类型

可视化设计的许多方面都是由我们需要查看的数据类型驱动的。例如，它是一个数字表，还是一组项目之间的关系，还是固有的空间数据，例如地球表面上的位置或文档集合？

我们首先考虑一个数据表。我们将数据的行称为项，列称为维度，也称为属性。例如，行可能代表人，列可能是姓名、年龄、身高、衬衫大小和最喜欢的水果。

我们区分三种类型的维度：定量，有序和分类。定量数据，如年龄或身高，是数字的，我们可以对其进行算术运算。例如，68英寸减去42英寸的数量是26英寸。有了有序的数据，比如衬衫尺寸，我们不能做全面的排序，但有一个明确定义的排序。例如，大减中不是一个有意义的概念，但我们知道中介于小和大之间。类别数据（如喜爱的水果或名称）没有隐式排序。我们只能区分两件事是相同的（苹果）还是不同的（苹果与香蕉）。

关系数据或图形是节点通过链接连接的另一种数据类型。一种特定类型的图是树，它通常用于分层数据。节点和边都可以具有相关联的属性。不幸的是，wordgraph在可视化中超载了。我们在这里讨论的节点链接图，遵循图绘制和图理论的术语，也可以称为网络。在统计图形领域，图形通常用于图表，如图26.10所示的时间序列数据的折线图。

一些数据本质上是空间的，例如地理位置或三维空间位置的mea保证范围，如医生用于查看人体内部结构的MRI或CT扫描。信息为



sociated with each point in space may be an unordered set of scalar quantities, or indexed vectors, or tensors. In contrast, nonspatial data can be visually encoded using spatial position, but that encoding is chosen by the designer rather than given implicitly in the semantics of the dataset itself. This choice is one of the most central and difficult problems of visualization design.

26.2.1 Dimension and Item Count

The number of data dimensions that need to be visually encoded is one of the most fundamental aspects of the visualization design problem. Techniques that work for a low-dimensional dataset with a few columns will often fail for very high-dimensional datasets with dozens or hundreds of columns. A data dimension may have hierarchical structure, for example with a time series dataset where there are interesting patterns at multiple temporal scales.

The number of data items is also important: a visualization that performs well for a few hundred items often does not scale to millions of items. In some cases the difficulty is purely algorithmic, where a computation would take too long; in others it is an even deeper perceptual problem that even an instantaneous algorithm could not solve, where visual clutter makes the representation unusable by a person. The range of possible values within a dimension may also be relevant.

26.2.2 Data Transformation and Derived Dimensions

Data is often transformed from one type to another as part of a visualization pipeline for solving the domain problem. For example, an original data dimension might be made up of quantitative data: floating point numbers that represent temperature. For some tasks, like finding anomalies in local weather patterns, the raw data might be used directly. For another task, like deciding whether water is an appropriate temperature for a shower, the data might be transformed into an ordered dimension: hot, warm, or cold. In this transformation, most of the detail is aggregated away. In a third example, when making toast, an even more lossy transformation into a categorical dimension might suffice: burned or not burned.

The principle of transforming data into *derived dimensions*, rather than simply visually encoding the data in its original form, is a powerful idea. In Figure 26.10, the original data was an ordered collection of time-series curves. The transformation was to cluster the data, reducing the amount of information to visually encode to a few highly meaningful curves.



与空间中的每个点相关联的可能是一组无序的标量，或索引向量或张量。相反，非空间数据可以使用空间位置进行可视化编码，但该编码由设计者选择，而不是在数据集本身的语义中隐式给定。这种选择是可视化设计最核心和最困难的问题之一。

26.2.1 尺寸和项目计数

需要可视化编码的数据维度数量是可视化设计问题最基本的因素之一。对于具有几列的低维数据集工作的技术，对于具有数十列或数百列的非常高维数据集通常会失败。数据维度可能具有分层结构，例如具有时间序列数据集，其中在多个时间尺度上存在有趣的模式。

数据项的数量也很重要：对于几百个项目表现良好的可视化通常不会扩展到数百万个项目。在某些情况下，困难纯粹是算法，计算需要太长时间；在其他情况下，这是一个更深层次的感知问题，即使是瞬时算法也无法解决，其中视觉混乱使得表示维度内的可能值的范围也可能是相关的。

26.2.2 数据转换和派生维度

数据通常作为解决域问题的可视化管道的一部分从一种类型转换到另一种类型。例如，原始数据维度可能由定量数据组成：表示温度的浮点数。对于某些任务，例如查找本地天气模式中的异常，原始数据可能会直接使用。对于另一项任务，例如确定水是否适合淋浴的温度，数据可能会转换为有序维度：热，暖或冷。在这种变换中，大部分细节被聚集起来。在第三个例子中，在制作toast时，向分类维度进行更有损耗的转换可能就足够了：烧掉或不烧掉。将数据转换为派生维度的原则，而不是简单地以原始形式直观地对数据进行编码，这是一个强大的想法。在图26.10中，原始数据是时间序列曲线的有序集合。这种转换是对数据进行聚类，减少了视觉编码到一些高度有意义的曲线的信息量。

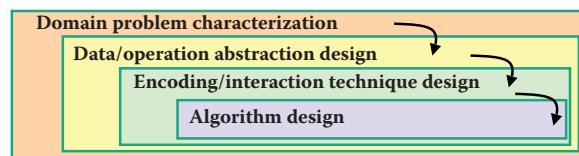


Figure 26.3. Four nested layers of validation for visualization.

26.3 Human-Centered Design Process

The visualization design process can be split into a cascading set of layers, as shown in Figure 26.3. These layers all depend on each other; the output of the level above is input into the level below.

26.3.1 Task Characterization

A given dataset has many possible visual encodings. Choosing which visual encoding to use can be guided by the specific needs of some intended user. Different questions, or *tasks*, require very different visual encodings. For example, consider the domain of software engineering. The task of understanding the coverage of a test suite is well supported by the Tarantula interface shown in Figure 26.11. However, the task of understanding the modular decomposition of the software while refactoring the code might be better served by showing its hierarchical structure more directly as a node-link graph.

Understanding the requirements of some target audience is a tricky problem. In a human-centered design approach, the visualization designer works with a group of target users over time (C. Lewis & Rieman, 1993). In most cases, users know they need to somehow view their data but cannot directly articulate their needs as clear-cut tasks in terms of operations on data types. The iterative design process includes gathering information from the target users about their problems through interviews and observation of them at work, creating prototypes, and observing how users interact with those prototypes to see how well the proposed solution actually works. The software engineering methodology of requirements analysis can also be useful (Kovitz, 1999).

26.3.2 Abstraction

After the specific domain problem has been identified in the first layer, the next layer requires abstracting it into a more generic representation as operations on



图26.3。用于可视化的四个嵌套验证层。

26.3 以人为本的设计过程

可视化设计过程可以拆分为一组级联的层，如图26.3所示。这些层都相互依赖；上面的电平的输出被输入到下面的电平中。

26.3.1 任务表征

给定的数据集有许多可能的视觉编码。选择使用哪种视觉en编码可以由某些预期用户的特定需求指导。不同的问题或任务需要非常不同的视觉编码。例如，考虑软件工程领域。图26.11所示的狼蛛接口很好地支持了解测试套件复盖范围的任务。然而，在重构代码的同时理解软件的模块化分解的任务可能会更好地通过将其分层结构更直接地显示为节点链接图来实现。

了解一些目标受众的要求是一个棘手的问题。

在以人为中心的设计方法中，可视化设计师随着时间的推移与一群目标用户一起工作（C.Lewis & Rieman, 1993）。在大多数情况下，用户知道他们需要以某种方式查看他们的数据，但不能直接将他们的需求作为数据类型操作方面的明确任务。迭代设计过程包括通过访谈和在工作中观察目标用户的问题，从目标用户那里收集有关他们问题的信息，创建原型，并观察用户如何与这些原型交互，以了解所提出的解决方案实际工作的情况。需求分析的软件工程方法也很有用（Kovitz, 1999）。

26.3.2 抽象

在第一层确定了特定的域问题后，下一层需要将其抽象为更通用的表示形式，作为对

the data types discussed in the previous section. Problems from very different domains can map to the same visualization abstraction. These generic operations include sorting, filtering, characterizing trends and distributions, finding anomalies and outliers, and finding correlation (Amar, Eagan, & Stasko, 2005). They also include operations that are specific to a particular data type, for example following a path for relational data in the form of graphs or trees.

This abstraction step often involves data transformations from the original raw data into derived dimensions. These derived dimensions are often of a different type than the original data: a graph may be converted into a tree, tabular data may be converted into a graph by using a threshold to decide whether a link should exist based on the field values, and so on.

26.3.3 Technique and Algorithm Design

Once an abstraction has been chosen, the next layer is to design appropriate visual encoding and interaction techniques. Section 26.4 covers the principles of visual encoding, and we discuss interaction principles in Sections 26.5. We present techniques that take these principles into account in Sections 26.6 and 26.7.

A detailed discussion of visualization algorithms is unfortunately beyond the scope of this chapter.

26.3.4 Validation

Each of the four layers has different validation requirements.

The first layer is designed to determine whether the problem is correctly characterized: is there really a target audience performing particular tasks that would benefit from the proposed tool? An immediate way to test assumptions and conjectures is to observe or interview members of the target audience, to ensure that the visualization designer fully understands their tasks. A measurement that cannot be done until a tool has been built and deployed is to monitor its adoption rate within that community, although of course many other factors in addition to utility affect adoption.

The next layer is used to determine whether the abstraction from the domain problem into operations on specific data types actually solves the desired problem. After a prototype or finished tool has been deployed, a field study can be carried out to observe whether and how it is used by its intended audience. Also, images produced by the system can be analyzed both qualitatively and quantitatively.

The purpose of the third layer is to verify that the visual encoding and interaction techniques chosen by the designer effectively communicate the chosen

上一节中讨论的数据类型。来自非常不同领域的问题可以映射到相同的可视化抽象。这些通用操作包括排序，过滤，表征趋势和分布，查找异常值和异常值以及查找相关性 (Amar, Eagan, & Stasko, 2005)。它们还包括特定于特定数据类型的操作，例如遵循图形或树形式的关系数据路径。

此抽象步骤通常涉及从原始原始数据到派生维度的数据转换。这些派生维度通常与原始数据的类型不同：可以将图形转换为树，可以使用阈值将表格数据转换为图形，以根据字段值决定链接是否应该存在，等等。

26.3.3 技术与算法设计

一旦选择了抽象，下一层是设计适当的视觉编码和交互技术。第26.4节涵盖了视觉编码的原理，我们在第26.5节中讨论了交互原理。我们在第26.6和26.7节中提出了考虑到这些原则的技术。

不幸的是，可视化算法的详细讨论超出了本章的范围。

26.3.4 Validation

这四个层中的每一层都有不同的验证要求。

第一层旨在确定问题是否被正确characterized：是否真的有一个目标受众执行特定的任务，将受益于所提出的工具？测试假设和假设的直接方法是观察或采访目标受众的成员，以确保可视化设计师完全理解他们的任务。在构建和部署工具之前无法进行的测量是监控其在该社区内的采用率，尽管除了效用之外，当然还有许多其他因素会影响采用。

下一层用于确定从域问题抽象为对特定数据类型的操作是否实际上解决了期望的问题。部署原型或成品工具后，可以进行现场研究，以观察其目标受众是否以及如何使用它。此外，该系统产生的图像可以定性和定量分析。

第三层的目的是验证设计人员选择的可视化编码和interaction技术是否有有效地传达了所选择的

abstraction to the users. An immediate test is to justify that individual design choices do not violate known perceptual and cognitive principles. Such a justification is necessary but not sufficient, since visualization design involves many tradeoffs between interacting choices. After a system is built, it can be tested through formal laboratory studies where many people are asked to do assigned tasks so that measurements of the time required for them to complete the tasks and their error rates can be statistically analyzed.

A fourth layer is employed to verify that the algorithm designed to carry out the encoding and interaction choices is faster or takes less memory than previous algorithms. An immediate test is to analyze the computational complexity of the proposed algorithm. After implementation, the actual time performance and memory usage of the system can be directly measured.

26.4 Visual Encoding Principles

We can describe visual encodings as graphical elements, called *marks*, that convey information through visual channels. A zero-dimensional mark is a point, a one-dimensional mark is a line, a two-dimensional mark is an area, and a three-dimensional mark is a volume. Many *visual channels* can encode information, including spatial position, color, size, shape, orientation, and direction of motion. Multiple visual channels can be used to simultaneously encode different data dimensions; for example, Figure 26.4 shows the use of horizontal and vertical spatial position, color, and size to display four data dimensions. More than one channel can be used to redundantly code the same dimension, for a design that displays less information but shows it more clearly.

26.4.1 Visual Channel Characteristics

Important characteristics of visual channels are distinguishability, separability, and popout.

Channels are not all equally distinguishable. Many psychophysical experiments have been carried out to measure the ability of people to make precise distinctions about information encoded by the different visual channels. Our abilities depend on whether the data type is quantitative, ordered, or categorical. Figure 26.5 shows the rankings of visual channels for the three data types. Figure 26.6 shows some of the default mappings for visual channels in the Tableau/Polaris system, which take into account the data type.

抽象给用户。一个直接的测试是证明个人的设计选择不违反已知的感知和认知原则。这样的解释是必要的，但还不够，因为可视化设计涉及交互选择之间的许多权衡。系统建成后，可以通过正式的实验室研究进行测试，要求许多人完成指定的任务，以便对他们完成任务所需的时间及其错误率进行统计分析。

第四层用于验证设计用于执行编码和交互选择的算法比以前的算法更快或占用更少的内存。一个直接的测试是分析所提出的算法的计算复杂性。实施后，可以直接测量系统的实际时间性能和内存使用情况。

26.4 视觉编码原理

我们可以将视觉编码描述为图形元素，称为标记，通过视觉渠道传达信息。零维标记是点，一维标记是线，二维标记是区域，三维标记是体积。许多视觉通道可以编码信息，包括空间位置，颜色，大小，形状，方向和运动方向。多个视觉通道可用于同时编码不同的数据维度；例如，图26.4示出了使用水平和垂直空间位置、颜色和大小来显示四个数据维度。多个通道可用于对同一维度进行冗余编码，以实现显示更少信息但显示更清晰的设计。

26.4.1 视觉通道特性

视觉通道的重要特征是可区分性、可分性和弹出性。

渠道并不都是同样可区分的。已经进行了许多心理物理实验来衡量人们对由不同视觉通道编码的信息进行精确区分的能力。我们的能力取决于数据类型是定量的、有序的还是分类的。图26.5显示了三种数据类型的可视通道排名。图26.6显示了TableauPolaris系统中可视通道的一些默认映射，其中考虑了数据类型。

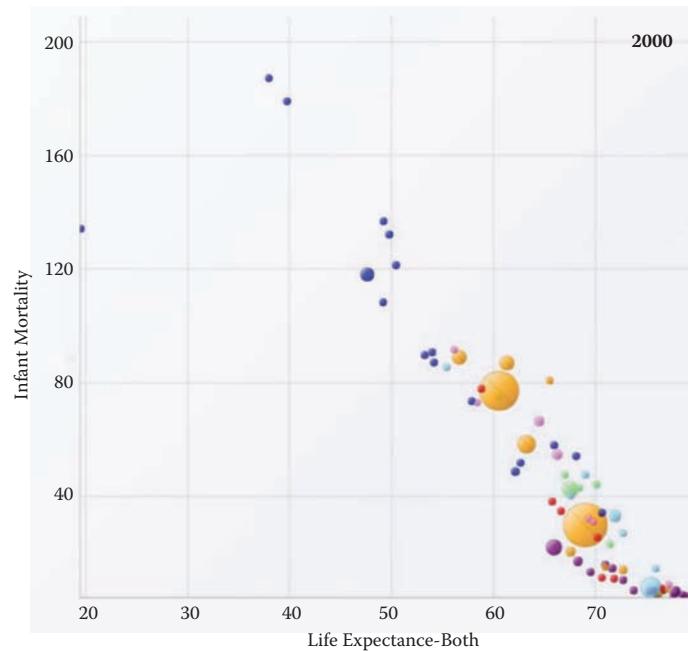
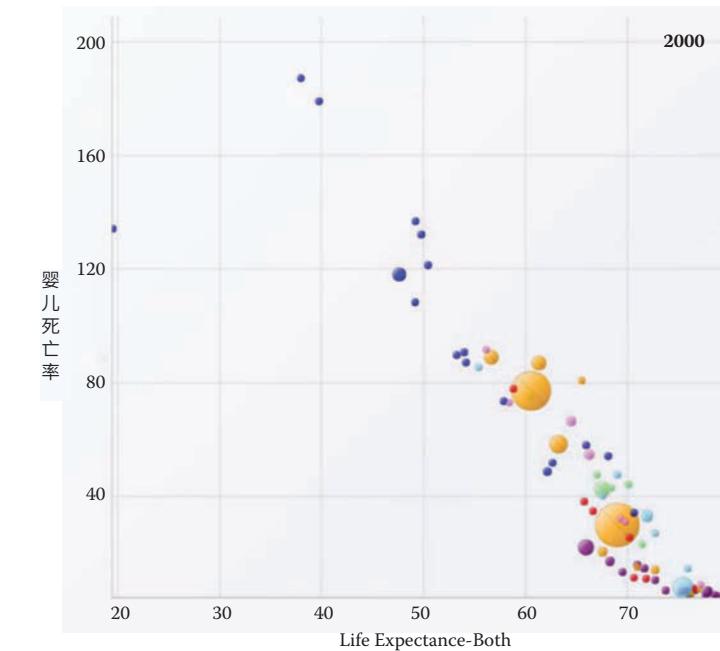


Figure 26.4. The four visual channels of horizontal and vertical spatial position, color, and size are used to encode information in this scatterplot chart. *Image courtesy George Robertson* (Robertson, Fernandez, Fisher, Lee, & Stasko, 2008), © IEEE 2008.



水平和垂直空间位置、颜色和大小的四个视觉通道用于在此散点图图像中编码信息，George Robertson(Robertson Fernandez Fisher Lee & Stasko 2008) ©IEEE2008.

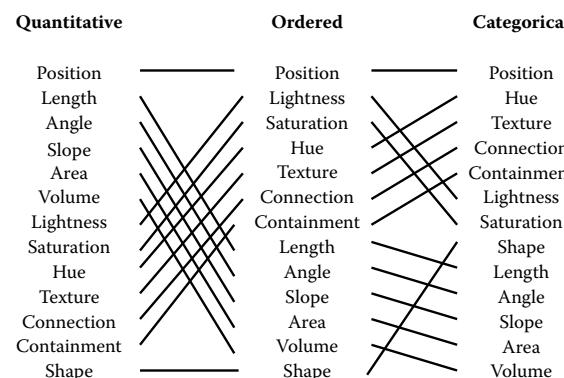


Figure 26.5. Our ability to perceive information encoded by a visual channel depends on the type of data used, from most accurate at the top to least at the bottom. Redrawn and adapted from (Mackinlay, 1986).

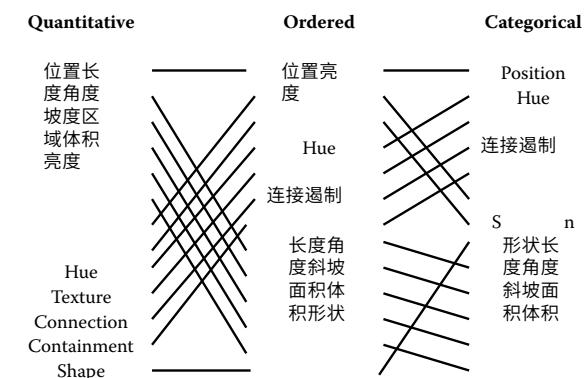


图26.5。我们感知由视觉通道编码的信息的能力取决于所使用的数据类型，从最准确的顶部到至少在底部。重绘改编自 (Mackinlay, 1986)。

Property	Ordinal/nominal mapping	Quantitative mapping
Shape	○ □ + × * ◇ △	
Size	● ● ● ●	● ● ● ● ● ● ● ● ● ●
Orientation	— ↗ ↘ ↖ ↙ ↖ ↗	— ↗ ↘ ↖ ↙ ↖ ↗ ↘ ↘ ↖
Color		

Figure 26.6. The Tableau/Polaris system default mappings for four visual channels according to data type. *Image courtesy Chris Stolte* (Stolte, Tang, & Hanrahan, 2008), © 2008 IEEE.

Spatial position is the most accurate visual channel for all three types of data, and it dominates our perception of a visual encoding. Thus, the two most important data dimensions are often mapped to horizontal and vertical spatial positions.

However, the other channels differ strongly between types. The channels of length and angle are highly discriminable for quantitative data but poor for ordered and categorical, while in contrast hue is very accurate for categorical data but mediocre for quantitative data

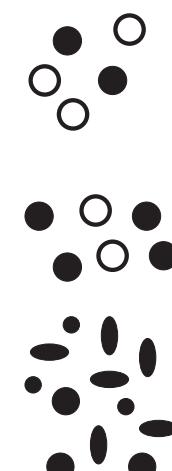
We must always consider whether there is a good match between the dynamic range necessary to show the data dimension and the dynamic range available in the channel. For example, encoding with line width uses a one-dimensional mark and the size channel. There are a limited number of width steps that we can reliably use to visually encode information: a minimum thinness of one pixel is enforced by the screen resolution (ignoring antialiasing to simplify this discussion), and there is a maximum thickness beyond which the object will be perceived as a polygon rather than a line. Line width can work very well to show three or four different values in a data dimension, but it would be a poor choice for dozens or hundreds of values.

Figure 26.7. Color and location are separable channels well suited to encode different data dimensions, but the horizontal size and vertical size channels are automatically fused into an integrated perception of area. *Redrawn after (Ware, 2000).*

Figure 26.7. Color and location are separable channels well suited to encode different data dimensions, but the horizontal size and vertical size channels are automatically fused into an integrated perception of area. *Redrawn after (Ware, 2000).*

TableauPolaris系统根据数据类型为四个可视通道默认映射。图片由ChrisStolte (Stolte, Tang, & Hanrahan, 2008) 提供, ©2008IEEE。

空间位置是所有三种类型数据的最准确的视觉通道，它主宰了我们对视觉编码的感知。因此，两个最有影响力的数据维度经常被映射到水平和垂直空间位置。但是，其他通道在不同类型之间存在很大差异。长度和角度的通道对于定量数据是高度可辨别的，但对于或dered和categorical较差，而相比之下，hue对于categorical数据非常准确，但对于定量数据平庸。



颜色和lo阳离子是可分离的通道，非常适合编码不同的耳鼻喉科数据尺寸，但水平尺寸和vertical尺寸通道自动融合到区域的综合感知中

after (Ware, 2000)

我们必须始终考虑显示数据维度所需的动态范围与通道中可用的动态范围之间是否存在良好的匹配。例如，具有线宽的编码使用一维标记和尺寸通道。我们可以可靠地使用有限数量的宽度步骤来对信息进行视觉编码：屏幕分辨率强制执行一个像素的最小薄度（忽略抗锯齿以简化此讨论），并且存在最大线宽可以很好地在数据维度中显示三个或四个不同的值，但对于数十个或数百个值来说，这是一个糟糕的选择。

一些视觉通道是积分的，在预先意识的水平上融合在一起，因此它们不是视觉编码不同数据维度的好选择。其他是可分离的，在视觉处理过程中它们之间没有交互，并且可以安全地用于编码多个维度。图26.7显示了两个通道对。颜色和位置是高度可分离的。我们可以看到，水平尺寸和垂直尺寸并不是那么容易分开的，因为我们的视觉系统自动将这些整合在一起，形成一个统一的区域感知。尺寸与许多通道相互作用：随着物体的尺寸变小，区分其形状或颜色变得更加困难。



We can selectively attend to a channel so that items of a particular type “pop out” visually, as discussed in Section 20.4.3. An example of visual popout is when we immediately spot the red item amidst a sea of blue ones, or distinguish the circle from the squares. Visual popout is powerful and scalable because it occurs in parallel, without the need for conscious processing of the items one by one. Many visual channels have this popout property, including not only the list above but also curvature, flicker, stereoscopic depth, and even the direction of lighting. However, in general we can only take advantage of popout for one channel at a time. For example, a white circle does not pop out from a group of circles and squares that can be white or black, as shown in Figure 20.46. When we need to search across more than one channel simultaneously, the length of time it takes to find the target object depends linearly on the number of objects in the scene.

26.4.2 Color

Color can be a very powerful channel, but many people do not understand its properties and use it improperly. As discussed in Section 20.2.2, we can consider color in terms of three separate visual channels: hue, saturation, and lightness. Region size strongly affects our ability to sense color. Color in small regions is relatively difficult to perceive, and designers should use bright, highly saturated colors to ensure that the color coding is distinguishable. The inverse situation is true when colored regions are large, as in backgrounds, where low saturation pastel colors should be used to avoid blinding the viewer.

Hue is a very strong cue for encoding categorical data. However, the available dynamic range is very limited. People can reliably distinguish only around a dozen hues when the colored regions are small and scattered around the display. A good guideline for color coding is to keep the number of categories less than eight, keeping in mind that the background and the neutral object color also count in the total.

For ordered data, lightness and saturation are effective because they have an implicit perceptual ordering. People can reliably order by lightness, always placing gray in between black and white. With saturation, people reliably place the less saturated pink between fully saturated red and zero-saturation white. However, hue is not as good a channel for ordered data because it does not have an implicit perceptual ordering. When asked to create an ordering of red, blue, green, and yellow, people do not all give the same answer. People can and do learn conventions, such as green-yellow-red for traffic lights, or the order of colors in the rainbow, but these constructions are at a higher level than pure perception. Ordered data is typically shown with a discrete set of color values.



我们可以有选择地关注一个频道，以便特定类型的项目在视觉上“弹出”，如第20.4.3节所述。视觉弹出的一个例子是当我们立即发现红色项目在蓝色的海洋中，或区分圆圈和正方形。可视化弹出是强大和可扩展的，因为它是并行发生的，而不需要有意识地逐个处理项目。许多视觉通道都具有这种弹出属性，不仅包括上面的列表，还包括曲率，闪烁，立体深度，甚至照明方向。但是，一般来说，我们一次只能利用一个通道的popout。例如，一个白色的圆圈不会从一组可以是白色或黑色的圆圈和正方形中弹出，如图20.46所示。当我们需要同时跨多个通道进行搜索时，找到目标对象所花费的时间长度与场景中对象的数量成线性关系。

26.4.2 Color

颜色可以是一个非常强大的渠道，但很多人不了解它的属性，并使用不当。正如第20.2.2节所讨论的，我们可以从三个独立的视觉通道来考虑颜色：色调、饱和度和亮度。区域大小强烈影响我们感知颜色的能力。小区域的颜色相对难以感知，设计师应该使用明亮的，高度饱和的颜色，以确保颜色编码是可区分的。相反的情况是真实的，当彩色区域很大，如在背景，其中低饱和度柔和的颜色应该使用，以避免使观众失明。

Hue是对分类数据进行编码的一个非常强的提示。然而，可用的动态范围非常有限。当彩色区域很小并且分散在显示器周围时，人们只能可靠地分辨出大约十几种色调。颜色编码的一个很好的准则是保持类别的数量少于八个，记住背景和中性对象颜色也计入总数。

对于有序数据，亮度和饱和度是有效的，因为它们具有隐含的感知排序。人们可以通过轻盈可靠地订购，总是在黑色和白色之间放置灰色。在饱和度下，人们可靠地将不太饱和的粉红色置于完全饱和的红色和零饱和的白色之间。无论如何，hue并不像有序数据的通道那样好，因为它没有隐含的感知排序。当被要求创建红色，蓝色，绿色和黄色的顺序时，人们并不都给出相同的答案。人们可以而且确实学习约定，例如红绿灯的绿色-黄色-红色，或彩虹中的颜色顺序，但这些结构比纯粹的感知处于更高的水平。有序数据通常以一组离散的颜色值显示。

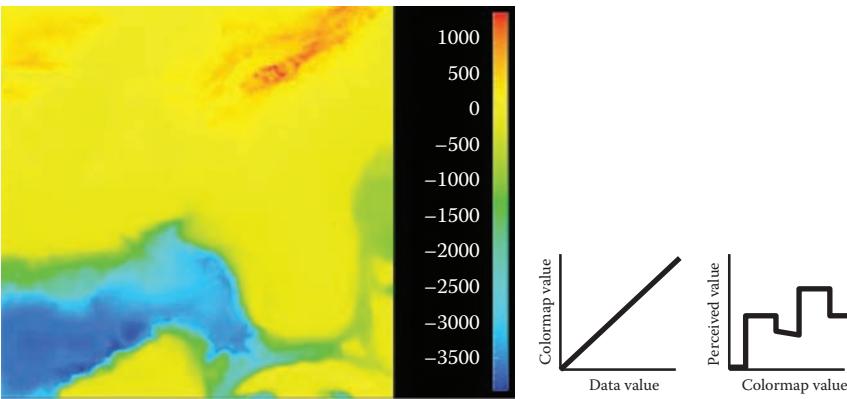


Figure 26.8. The standard rainbow colormap has two defects: it uses hue to denote ordering, and it is not perceptually isolinear. *Image courtesy Bernice Rogowitz.*

Quantitative data is shown with a *colormap*, a range of color values that can be continuous or discrete. A very unfortunate default in many software packages is the rainbow colormap, as shown in Figure 26.8. The standard rainbow scale suffers from three problems. First, hue is used to indicate order. A better choice would be to use lightness because it has an implicit perceptual ordering. Even more importantly, the human eye responds most strongly to luminance. Second, the scale is not perceptually linear: equal steps in the continuous range are not perceived as equal steps by our eyes. Figure 26.8 shows an example, where the rainbow colormap obfuscates the data. While the range from -2000 to -1000 has three distinct colors (cyan, green, and yellow), a range of the same size from -1000 to 0 simply looks yellow throughout. The graphs on the right show that the perceived value is strongly tied to the luminance, which is not even monotonically increasing in this scale.

In contrast, Figure 26.9 shows the same data with a more appropriate colormap, where the lightness increases monotonically. Hue is used to create a semantically meaningful categorization: the viewer can discuss structure in the dataset, such as the dark blue sea, the cyan continental shelf, the green lowlands, and the white mountains.

In both the discrete and continuous cases, colormaps should take into account whether the data is sequential or diverging. The ColorBrewer application (www.colorbrewer.org) is an excellent resource for colormap construction (Brewer, 1999).

Another important issue when encoding with color is that a significant fraction of the population, roughly 10% of men, is red-green color deficient. If a coding using red and green is chosen because of conventions in the target domain, re-

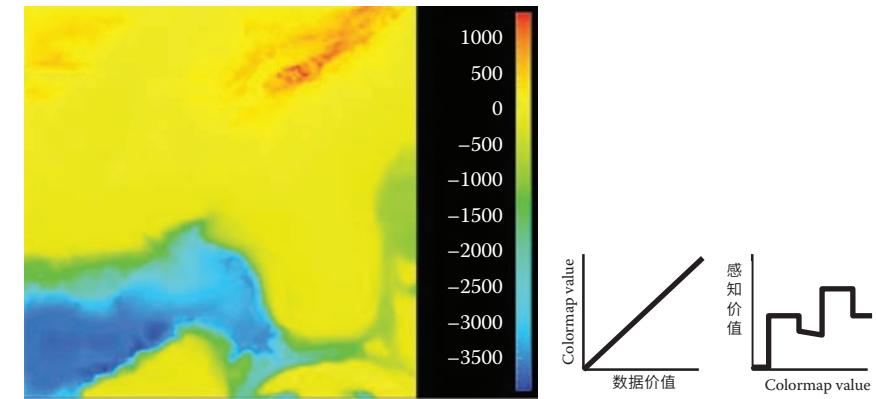


图26.8。 标准的rainbowcolormap有两个缺陷：它使用色调来表示排序，并且在感知上不是孤立的。图片由BerniceRogowitz提供。

定量数据用颜色表显示，颜色值范围可以是连续的或离散的。在许多软件包中，一个非常不幸的默认值是彩虹色表，如图26.8所示。标准彩虹秤有三个问题。首先，色调用于表示顺序。更好的选择是使用轻盈，因为它具有隐含的感知排序。更重要的是，人眼对亮度的反应最强烈。其次，尺度在感知上不是线性的：连续范围内的相等步长不会被我们的眼睛感知为相等步长。图26.8显示了一个示例，其中rainbowcolormap混淆了数据。虽然从 2000 到 1000 的范围有三种不同的颜色（青色，绿色和黄色），但从 1000 到 0 的相同大小的范围在整个范围内看起来都是黄色的。右边的图表显示，感知值与亮度有很强的联系，亮度在这个比例中甚至不是单调增加的。

相比之下，图26.9显示了具有更合适的colormap的相同数据，其中亮度单调增加。色调用于创建语义上有意义的分类：查看者可以讨论数据集中的结构，如深蓝色的海洋、青色的大陆架、绿色的低地和白山。

在离散和连续的情况下，颜色图应该考虑ac计数，无论数据是顺序的还是发散的。彩笔应用(www.colorbrewer.org) 是colormapconstruction (Brewer, 1999) 的绝佳资源。

用颜色编码时的另一个重要问题是，很大一部分人口，大约10%的男性，是红绿颜色不足。如果由于目标域中的约定而选择使用红色和绿色的编码，则

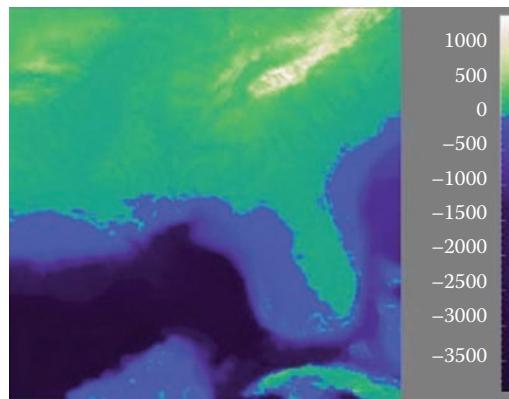


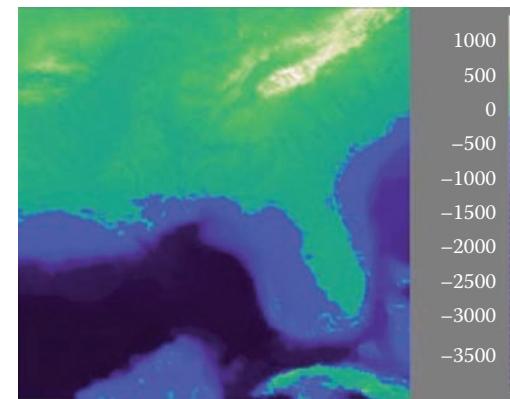
Figure 26.9. The structure of the same dataset is far more clear with a colormap where monotonically increasing lightness is used to show ordering and hue is used instead for segmenting into categorical regions. *Image courtesy Bernice Rogowitz.*

dundantly coding lightness or saturation in addition to hue is wise. Tools such as the website <http://www.vischeck.com> should be used to check whether a color scheme is distinguishable to people with color deficient vision.

26.4.3 2D vs. 3D Spatial Layouts

The question of whether to use two or three channels for spatial position has been extensively studied. When computer-based visualization began in the late 1980s, and interactive 3D graphics was a new capability, there was a lot of enthusiasm for 3D representations. As the field matured, researchers began to understand the costs of 3D approaches when used for abstract datasets (Ware, 2001).

Occlusion, where some parts of the dataset are hidden behind others, is a major problem with 3D. Although hidden surface removal algorithms such as z-buffers and BSP trees allow fast computation of a correct 2D image, people must still synthesize many of these images into an internal mental map. When people look at realistic scenes made from familiar objects, usually they can quickly understand what they see. However, when they see an unfamiliar dataset, where a chosen visual encoding maps abstract dimensions into spatial positions, understanding the details of its 3D structure can be challenging even when they can use interactive navigation controls to change their 3D viewpoint. The reason is once again the limited capacity of human working memory (Plumlee & Ware, 2006).



同一数据集的结构使用颜色表更加清晰，其中使用单色调增加亮度来显示排序，而使用色调来分割为类别区域。图片由BerniceRogowitz提供。

dundantly编码亮度或饱和度除了色调是明智的。工具，如网站<http://www.vischeck.com>应用于检查配色方案是否可与视力有缺陷的人区分。

26.4.3 2D与3d空间布局

对空间位置是否使用两个或三个通道的问题进行了广泛的研究。当基于计算机的可视化在20世纪80年代后期开始，并且交互式3d图形是一种新功能时，人们对3D表示有很大的热情。随着该领域的成熟，研究人员开始了解3d方法在用于抽象数据集时的成本（Ware, 2001）。

虽然隐藏的表面去除算法（如zbuffers和BSP树）允许快速计算正确的2D图像，但人们仍然必须将许多这些图像合成一个内部的心理地图。当人们看着由熟悉的物体制成的逼真场景时，通常他们可以快速理解他们所看到的内容。然而，当他们看到一个不熟悉的数据集（其中所选的视觉编码将抽象维度映射到空间位置）时，即使他们可以使用交互式导航控件来更改其3d视点，也原因再次是人类工作记忆的容量有限（Plumlee & Ware, 2006）。

Another problem with 3D is *perspective distortion*. Although real-world objects do indeed appear smaller when they are further from our eyes, foreshortening makes direct comparison of object heights difficult (Tory, Kirkpatrick, Atkins, & Möller, 2006). Once again, although we can often judge the heights of familiar objects in the real world based on past experience, we cannot necessarily do so with completely abstract data that has a visual encoding where the height conveys meaning. For example, it is more difficult to judge bar heights in a 3D bar chart than in multiple horizontally aligned 2D bar charts.

Another problem with unconstrained 3D representations is that text at arbitrary orientations in 3D space is far more difficult to read than text aligned in the 2D image plane (Grossman, Wigdor, & Balakrishnan, 2007).

Figure 26.10 illustrates how carefully chosen 2D views of an abstract dataset can avoid the problems with occlusion and perspective distortion inherent in 3D views. The top view shows a 3D representation created directly from the original time-series data, where each cross-section is a 2D time-series curve showing power consumption for one day, with one curve for each day of the year along the extruded third axis. Although this representation is straightforward to create, we can only see large-scale patterns such as the higher consumption during working hours and the seasonal variation between winter and summer. To create the 2D linked views at the bottom, the curves were hierarchically clustered, and only aggregate curves representing the top clusters are drawn superimposed in the same 2D frame. Direct comparison between the curve heights at all times of the day is easy because there is no perspective distortion or occlusion. The same color coding is used in the calendar view, which is very effective for understanding temporal patterns.

In contrast, if a dataset consists of inherently 3D spatial data, such as showing fluid flow over an airplane wing or a medical imaging dataset from an MRI scan, then the costs of a 3D view are outweighed by its benefits in helping the user construct a useful mental model of the dataset structure.

26.4.4 Text Labels

Text in the form of labels and legends is a very important factor in creating visualizations that are useful rather than simply pretty. Axes and tick marks should be labeled. Legends should indicate the meaning of colors, whether used as discrete patches or in continuous color ramps. Individual items in a dataset typically have meaningful text labels associated with them. In many cases showing all labels at all times would result in too much visual clutter, so labels can be shown for a subset of the items using label positioning algorithms that show labels at a de-

3D的另一个问题是透视失真。虽然现实世界的objects在离我们眼睛更远的地方确实看起来更小，但预展使得物体高度的直接比较变得困难 (Tory , Kirkpatrick, Atkins, & Möller, 2006)。再一次，虽然我们经常可以根据过去的经验来判断现实世界中熟悉物体的高度，但我们不一定能用完全抽象的数据来判断高度传达意义的视觉编码。例如，在3D条形图中判断条形高度比在多个水平对齐的2D条形图中更困难。

无约束3d表示的另一个问题是，3D空间中arbitrary方向的文本比在2D图像平面中对齐的文本更难以阅读 (Grossman, Wigdor, & Balakrishnan, 2007) 。

图26.10说明了如何仔细选择抽象数据集的2D视图可以避免3d视图中固有的遮挡和透视失真问题。顶视图显示了直接从original时间序列数据创建的3D表示，其中每个横截面都是显示一天功耗的2D时间序列曲线，其中一年中的每一天都有一条曲线沿虽然这种表示很容易创建，但我们只能看到大规模的模式，例如工作时间内的较高消费量以及冬季和夏季之间的季节变化。为了在底部创建2d链接视图，曲线被分层聚类，并且只有表示顶部聚类的aggregate曲线被叠加在同一个2D框架中。一天中所有时间的曲线高度之间的直接比较很容易，因为没有透视失真或遮挡。日历视图中使用相同的颜色编码，这对于理解时间模式非常有效。

相反，如果数据集由固有的3D空间数据组成，例如显示飞机机翼上的流体流动或MRI扫描的医学成像数据集，那么3D视图的成本就会超过它在帮助用户构建数据集结构的有用心理模型方面的好处。

26.4.4 文本标签

标签和图例形式的文本是创建有用而不是简单漂亮的可视化的一个非常重要的因素。轴和刻度线应贴上标签。图例应指示颜色的含义，无论是用作离散色块还是用于连续色带。数据集中的各个项通常具有与其关联的有意义的文本标签。在许多情况下，在任何时候显示所有标签都会导致太多的视觉混乱，因此可以使用标签定位算法为项目的子集显示标签，这些标签可以在de上显示标签-

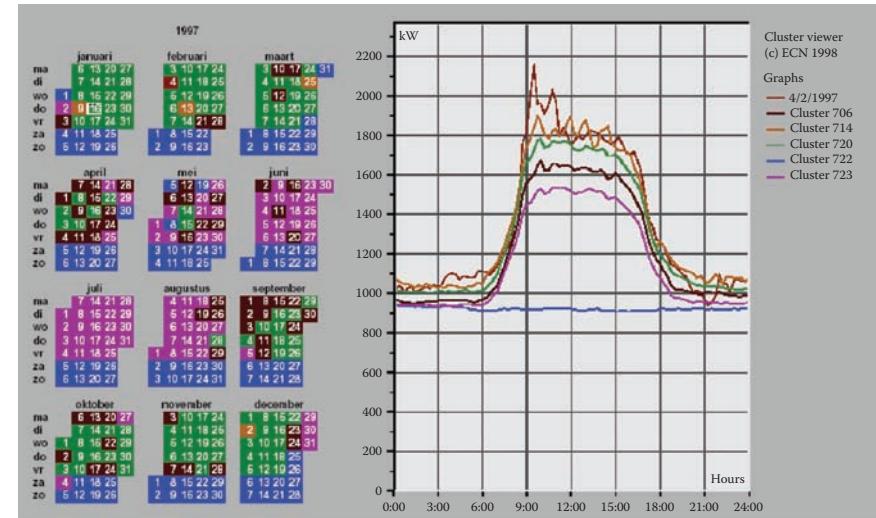
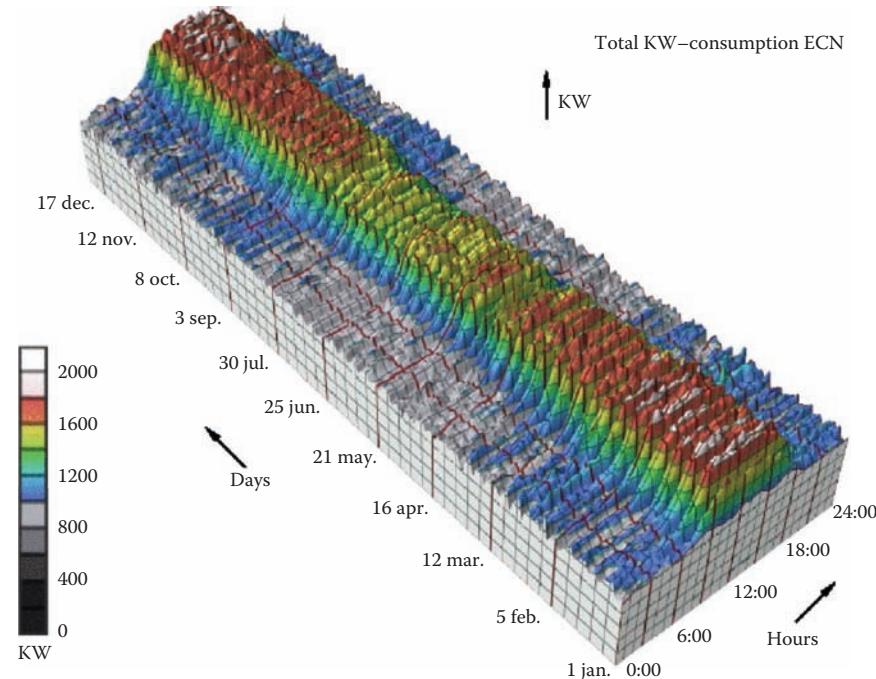
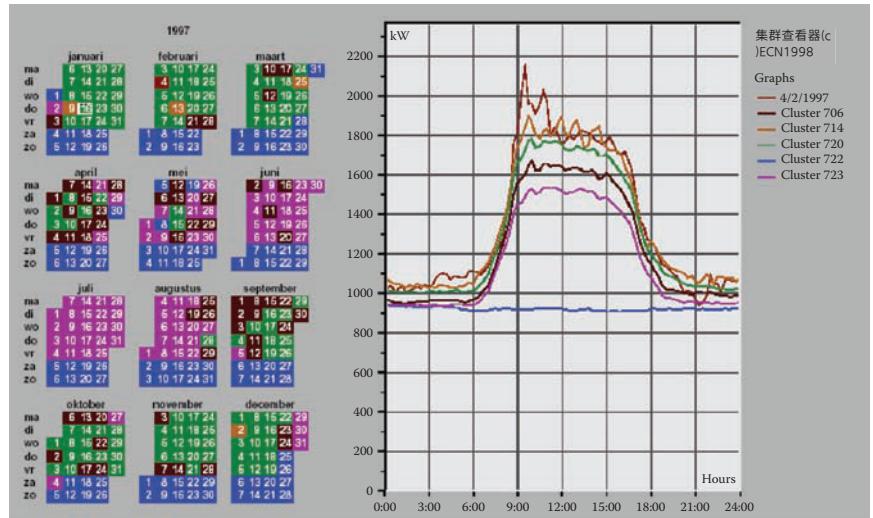
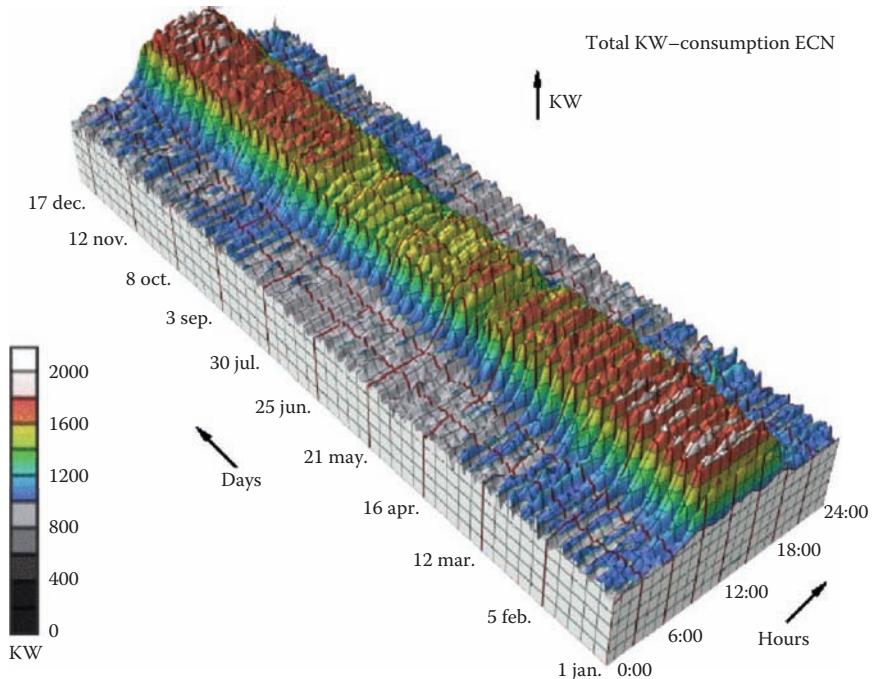


Figure 26.10. Top: A 3D representation of this time series dataset introduces the problems of occlusion and perspective distortion. Bottom: The linked 2D views of derived aggregate curves and the calendar allow direct comparison and show more fine-grained patterns. *Image courtesy Jarke van Wijk (van Wijk & van Selow, 1999), © 1999 IEEE.*



上图：此时间序列数据集的3D表示引入了遮挡和透视失真问题。底部：派生聚合曲线和日历的链接2D视图允许直接比较并显示更细粒度的模式。图片提供Jarke van Wijk (van Wijk & van Selow, 1999)，© 1999 IEEE。

sired density while avoiding overlap (Luboschik, Schumann, & Cords, 2008). A straightforward way to choose the best label to represent a group of items is to use a greedy algorithm based on some measure of label importance, but synthesizing a new label based on the characteristics of the group remains a difficult problem. A more interaction-centric approach is to only show labels for individual items based on an interactive indication from the user.

26.5 Interaction Principles

Several principles of interaction are important when designing a visualization. Low-latency visual feedback allows users to explore more fluidly, for example by showing more detail when the cursor simply hovers over an object rather than requiring the user to explicitly click. Selecting items is a fundamental operation when interacting with large datasets, as is visually indicating the selected set with highlighting. Color coding is a common form of highlighting, but other channels can also be used.

Many forms of interaction can be considered in terms of what aspect of the display they change. Navigation can be considered a change of viewport. Sorting is a change to the spatial ordering; that is, changing how data is mapped to the spatial position visual channel. The entire visual encoding can also be changed.

26.5.1 Overview First, Zoom and Filter, Details on Demand

The influential mantra “Overview first, zoom and filter, details on demand” (Shneiderman, 1996) elucidates the role of interaction and navigation in visualization design. Overviews help the user notice regions where further investigation might be productive, whether through spatial navigation or through filtering. As we discuss below, details can be presented in many ways: with popups from clicking or cursor hovering, in a separate window, and by changing the layout on the fly to make room to show additional information.

26.5.2 Interactivity Costs

Interactivity has both power and cost. The benefit of interaction is that people can explore a larger information space than can be understood in a single static image. However, a cost to interaction is that it requires human time and attention. If the user must exhaustively check every possibility, use of the visualization system

sired密度，同时避免重叠（Luboschik, Schumann, & Cords, 2008）。选择最佳标签来表示一组项目的一种直截了当的方法是使用基于标签重要性的某种度量的贪婪算法，但是基于该组的特征合成新标签仍然是一个难题。更以交互为中心的方法是仅根据来自用户的交互指示显示单个项目的标签。

26.5 互动原则

在设计可视化时，交互的几个原则很重要。低延迟视觉反馈允许用户更流畅地探索，例如，当光标简单地悬停在对象上时显示更多细节，而不是要求用户显式单击。与大型数据集交互时，选择项是一项基本操作，用高亮显示所选集也是如此。颜色编码是突出显示的常见形式，但也可以使用其他通道。

交互的许多形式可以在它们改变显示的哪一方面方面来考虑。导航可以被认为是视口的变化。排序是对空间排序的更改；也就是说，更改数据映射到空间位置视觉通道的方式。整个视觉编码也可以改变。

26.5.1 概述第一，缩放和过滤，按需提供详细信息

有影响力的口头禅“概述第一，缩放和过滤，按需详细信息”(Shneiderman, 1996)阐明了交互和导航在可视化设计中的作用。无论是通过空间导航还是通过过滤，概述都有助于用户注意到可能会产生进一步调查的区域。正如我们在下面讨论的那样，细节可以通过多种方式呈现：通过单击或光标悬停的弹出窗口，在单独的窗口中，以及通过动态更改布局以腾出空间来显示。

26.5.2 交互成本

交互性既有力量，也有成本。交互的好处是人们可以探索比在单个静态图像中可以理解的更大的信息空间。然而，互动的代价是它需要人类的时间和注意力。如果用户必须详尽地检查每一种可能性，使用可视化系统



may degenerate into human-powered search. Automatically detecting features of interest to explicitly bring to the user's attention via the visual encoding is a useful goal for the visualization designer. However, if the task at hand could be completely solved by automatic means, there would be no need for a visualization in the first place. Thus, there is always a tradeoff between finding automatable aspects and relying on the human in the loop to detect patterns.

26.5.3 Animation

Animation shows change using time. We distinguish animation, where successive frames can only be played, paused, or stopped, from true interactive control. There is considerable evidence that animated transitions can be more effective than jump cuts, by helping people track changes in object positions or camera viewpoints (Heer & Robertson, 2007). Although animation can be very effective for narrative and storytelling, it is often used ineffectively in a visualization context (Tversky, Morrison, & Betrancourt, 2002). It might seem obvious to show data that changes over time by using animation, a visual modality that changes over time. However, people have difficulty in making specific comparisons between individual frames that are not contiguous when they see an animation consisting of many frames. The very limited capacity of human visual memory means that we are much worse at comparing memories of things that we have seen in the past than at comparing things that are in our current field of view. For tasks requiring comparison between up to several dozen frames, side-by-side comparison is often more effective than animation. Moreover, if the number of objects that change between frames is large, people will have a hard time tracking everything that occurs (Robertson et al., 2008). Narrative animations are carefully designed to avoid having too many actions occurring simultaneously, whereas a dataset being visualized has no such constraint. For the special case of just two frames with a limited amount of change, the very simple animation of flipping back and forth between the two can be a useful way to identify the differences between them.

26.6 Composite and Adjacent Views

A very fundamental visual encoding choice is whether to have a single composite view showing everything in the same frame or window, or to have multiple views adjacent to each other.



可能退化为人力搜索。自动检测感兴趣的特征以通过可视化编码显式地引起用户的注意，这对于可视化设计人员来说是一个有用的目标。但是，如果手头的任务可以通过自动方式完全解决，那么首先就不需要可视化。因此，在找到可自动化的方面和依靠循环中的人来检测模式之间总是有一个权衡。

26.5.3 Animation

动画显示使用时间的变化。我们将只能播放、暂停或停止连续帧的动画与真正的交互式控制区分开来。有相当多的证据表明，通过帮助人们跟踪物体位置或摄像机视点的变化，动画过渡可以比跳跃切割更有效 (Heer & Robertson, 2007)。虽然动画对于叙事和讲故事非常有效，但它通常在可视化上下文中无效地使用 (Tversky, Morrison, & Betrancourt, 2002)。通过使用动画（一种随时间变化的视觉模式）来显示随时间变化的数据似乎很明显。但是，当人们看到由许多帧组成的动画时，他们很难在不连续的单个帧之间进行具体比较。人类视觉记忆的能力非常有限，这意味着我们在比较过去看到的事物的记忆方面要比比较我们当前视野中的事物差得多。对于需要在多达十几帧之间进行比较的任务，并排比较通常比动画更有效。此外，如果在帧之间变化的对象的数量很大，人们将很难跟踪发生的所有事情(Robertson et al. 2008)。叙事动画经过精心设计，以避免同时发生太多动作，而可视化的数据集没有这样的约束。对于只有两个变化量有限的帧的特殊情况，在两者之间来回翻转的非常简单的动画可以是识别它们之间差异的有用方法。

26.6 复合视图和相邻视图

一个非常基本的视觉编码选择是是否有一个单一的复合视图显示同一帧或窗口中的所有内容，或者有多个视图彼此相邻。

26.6.1 Single Drawing

When there are only one or two data dimensions to encode, then horizontal and vertical spatial position are the obvious visual channel to use, because we perceive them most accurately and position has the strongest influence on our internal mental model of the dataset. The traditional statistical graphics displays of line charts, bar charts, and scatterplots all use spatial ordering of marks to encode information. These displays can be augmented with additional visual channels, such as color and size and shape, as in the scatterplot shown in Figure 26.4.

The simplest possible mark is a single pixel. In *pixel-oriented* displays, the goal is to provide an overview of as many items as possible. These approaches use the spatial position and color channels at a high information density, but preclude the use of the size and shape channels. Figure 26.11 shows the Tarantula software visualization tool (Jones et al., 2002), where most of the screen is devoted to an overview of source code using one-pixel high lines (Eick, Steffen, & Sumner, 1992). The color and brightness of each line shows whether it passed, failed, or had mixed results when executing a suite of test cases.

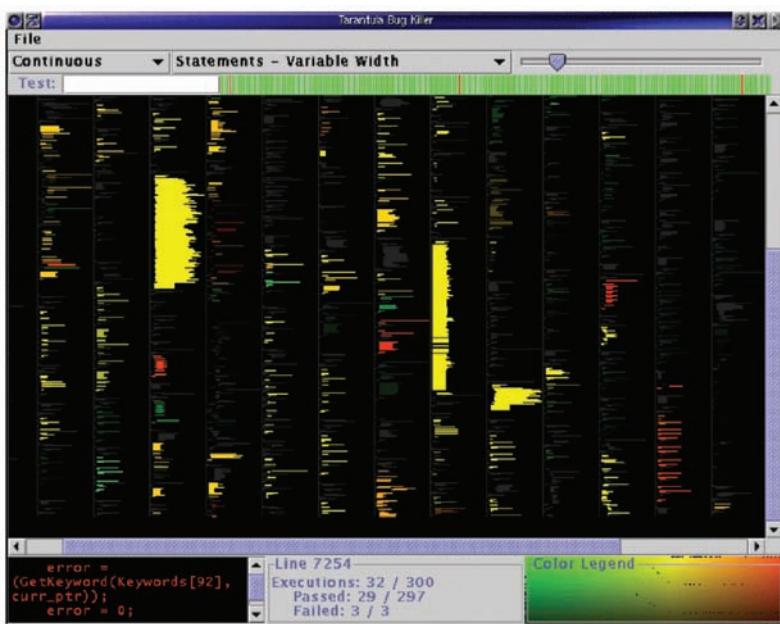
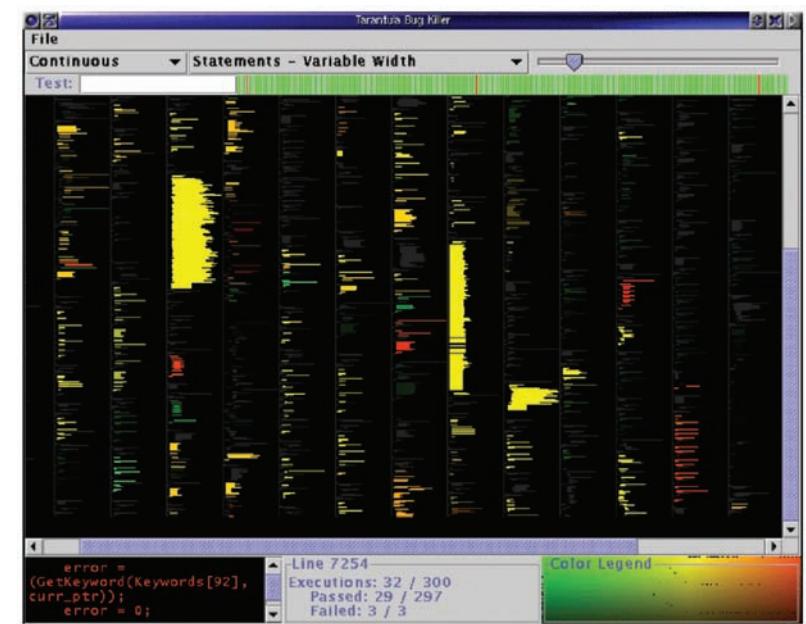


Figure 26.11. Tarantula shows an overview of source code using one-pixel lines color coded by execution status of a software test suite. *Image courtesy John Stasko (Jones, Harrold, & Stasko, 2002).*

26.6.1 单幅绘图

当只有一个或两个数据维度要编码时，水平和垂直空间位置是使用的明显视觉通道，因为我们最准确地感知它们，位置对数据集内部mental模型的影响最传统的直线图、条形图和散点图的统计图形显示都使用标记的空间排序来编码信息。这些显示器可以用额外的视觉通道来增强，例如颜色、大小和形状，如图26.4所示的散点图。

最简单的标记是单个像素。在面向像素的显示器中，目标是提供尽可能多的项目的概述。这些方法在高信息密度下使用空间位置和颜色通道，但用尺寸和形状通道。图26.11显示了狼蛛软件可视化工具(Jones et al., 2002)，其中大部分屏幕都用于使用一像素高线的源代码概述 (Eick, Steffen, & Sumner, 1992)。每行的颜色和亮度显示在执行一组测试用例时，它是否通过、失败或结果好坏参半。



狼蛛显示了一个概述的源代码使用一个像素线颜色编码的执行状态的软件测试套件。图片由JohnStasko提供 (Jones, Harrold, & Stasko, 2002)。

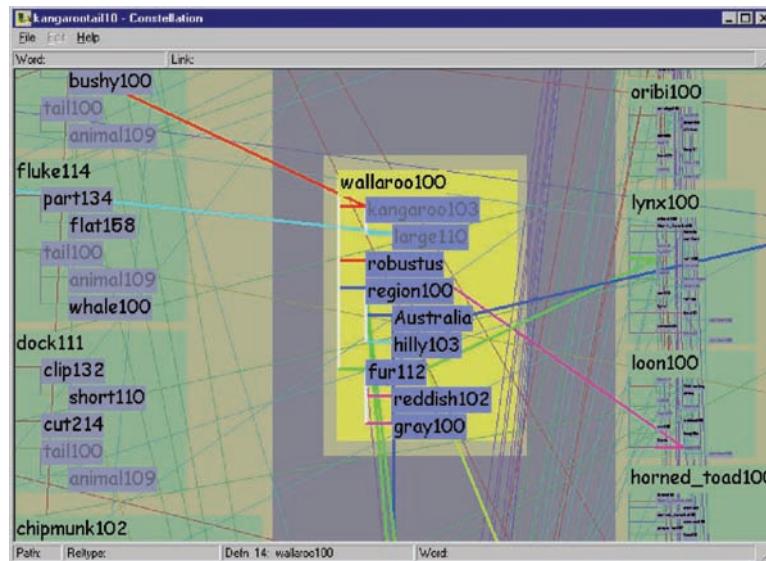


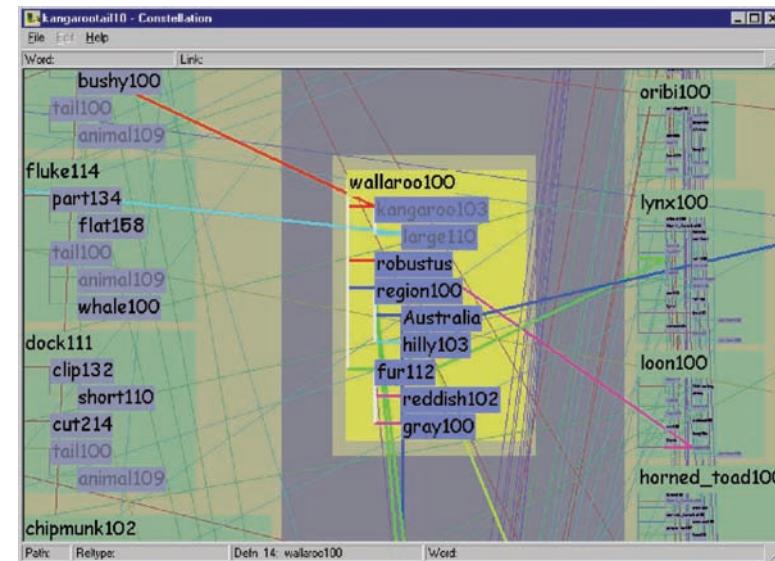
Figure 26.12. Visual layering with size, saturation, and brightness in the Constellation system (Munzner, 2000).

26.6.2 Superimposing and Layering

Multiple items can be superimposed in the same frame when their spatial position is compatible. Several lines can be shown in the same line chart, and many dots in the same scatterplot, when the axes are shared across all items. One benefit of a single shared view is that comparing the position of different items is very easy. If the number of items in the dataset is limited, then a single view will often suffice. Visual layering can extend the usefulness of a single view when there are enough items that visual clutter becomes a concern. Figure 26.12 shows how a redundant combination of the size, saturation, and brightness channels serves to distinguish a foreground layer from a background layer when the user moves the cursor over a block of words.

26.6.3 Glyphs

We have been discussing the idea of visual encoding using simple marks, where a single mark can only have one value for each visual channel used. With more complex marks, which we will call *glyphs*, there is internal structure where sub-regions have different visual channel encodings.



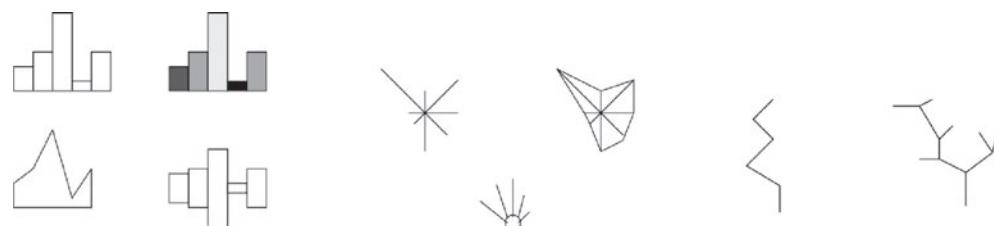
视觉分层与大小、饱和度和亮度星座系统 (Munzner, 2000)。

26.6.2 叠加和分层

当多个项目的空间位置兼容时，它们可以叠加在同一帧中。当轴在所有项目之间共享时，可以在同一折线图中显示多条线，并在同一散点图中显示许多点。单个共享视图的一个好处是比较不同项目的位置非常容易。如果数据集中的项目数量有限，那么单个视图通常就足够了。当有足够的项目使视觉混乱成为一个问题时，视觉分层可以扩展单个视图的有用性。图26.12显示了当用户将光标移动到一个单词块上时，大小、饱和度和亮度通道的冗余组合如何将前景层与背景层区分开来。

26.6.3 Glyphs

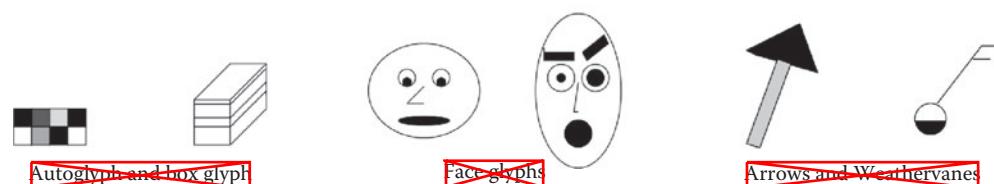
我们一直在讨论使用简单标记进行视觉编码的想法，其中单个标记对于使用的每个视觉通道只能具有一个值。对于更复杂的标记，我们将其称为字形，存在子区域具有不同视觉通道编码的内部结构。



Variations on Profile glyphs

Stars and Anderson/metroglyphs

Sticks and Trees



Autoglyph and box glyph

Face glyphs

Arrows and Weathervanes

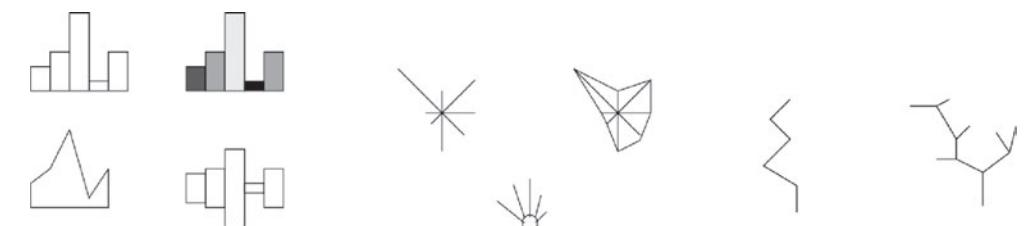
Figure 26.13. Complex marks, which we call *glyphs*, have subsections that visually encode different data dimensions. *Image courtesy Matt Ward (M. O. Ward, 2002).*

Designing appropriate glyphs has the same challenges as designing visual encodings. Figure 26.13 shows a variety of glyphs, including the notorious faces originally proposed by Chernoff. The danger of using faces to show abstract data dimensions is that our perceptual and emotional response to different facial features is highly nonlinear in a way that is not fully understood, but the variability is greater than between the visual channels that we have discussed so far. We are probably far more attuned to features that indicate emotional state, such as eyebrow orientation, than other features, such as nose size or face shape.

Complex glyphs require significant display area for each glyph, as shown in Figure 26.14 where miniature bar charts show the value of four different dimensions at many points along a spiral path. Simpler glyphs can be used to create a global visual texture, the glyph size is so small that individual values cannot be read out without zooming, but region boundaries can be discerned from the overview level. Figure 26.15 shows an example using stick figures of the kind in the upper right in Figure 26.13. Glyphs may be placed at regular intervals, or in data-driven spatial positions using an original or derived data dimension.

26.6.4 ~~Multiple Views~~

We now turn from approaches with only a single frame to those which use multiple views that are linked together. The most common form of linkage is linked



Variations on Profile glyphs

Stars and Anderson/metroglyphs

Sticks and Trees

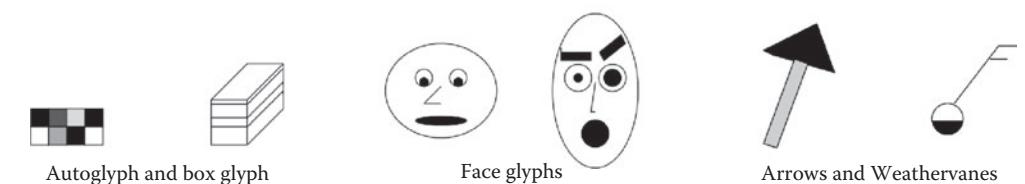


Figure 26.13. Complex marks, which we call *glyphs*, have subsections that visually encode different data dimensions. *Image courtesy Matt Ward (M. O. Ward, 2002).*

Designing appropriate glyphs has the same challenges as designing visual encodings. Figure 26.13 shows a variety of glyphs, including the notorious faces originally proposed by Chernoff. The danger of using faces to show abstract data dimensions is that our perceptual and emotional response to different facial features is highly nonlinear in a way that is not fully understood, but the variability is greater than between the visual channels that we have discussed so far. We are probably far more attuned to features that indicate emotional state, such as eyebrow orientation, than other features, such as nose size or face shape.

Complex glyphs require significant display area for each glyph, as shown in Figure 26.14 where miniature bar charts show the value of four different dimensions at many points along a spiral path. Simpler glyphs can be used to create a global visual texture, the glyph size is so small that individual values cannot be read out without zooming, but region boundaries can be discerned from the overview level. Figure 26.15 shows an example using stick figures of the kind in the upper right in Figure 26.13. Glyphs may be placed at regular intervals, or in data-driven spatial positions using an original or derived data dimension.

26.6.4 Multiple Views

We now turn from approaches with only a single frame to those which use multiple views that are linked together. The most common form of linkage is linked

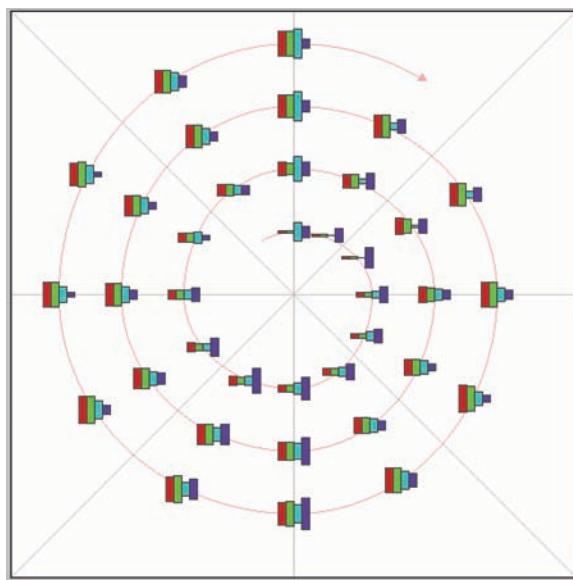


Figure 26.14. Complex glyphs require significant display area so that the encoded information can be read. *Image courtesy Matt Ward, created with the SpiralGlyphics software (M. O. Ward, 2002).*

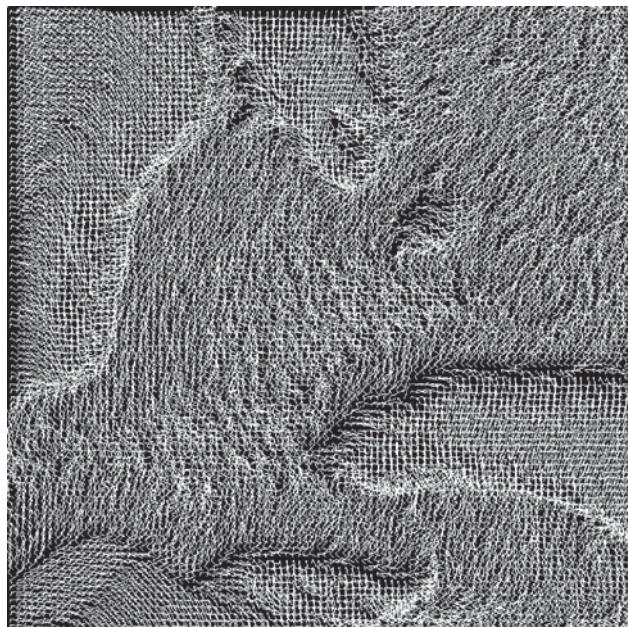


Figure 26.15. A dense array of simple glyphs. *Image courtesy Georges Grinstein (S. Smith, Grinstein, & Bergeron, 1991), © 1991 IEEE.*

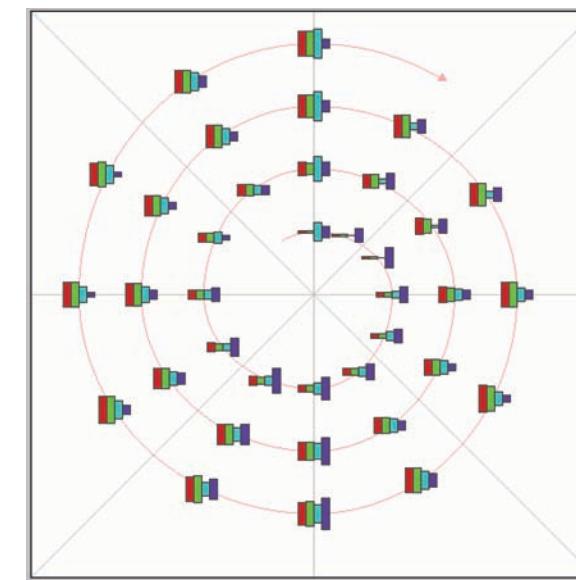


图26.14。复杂的字形需要显着的显示区域，以便可以读取编码的信息。图片由MattWard提供，使用SpiralGlyphics软件创建（M.O.Ward, 2002）。

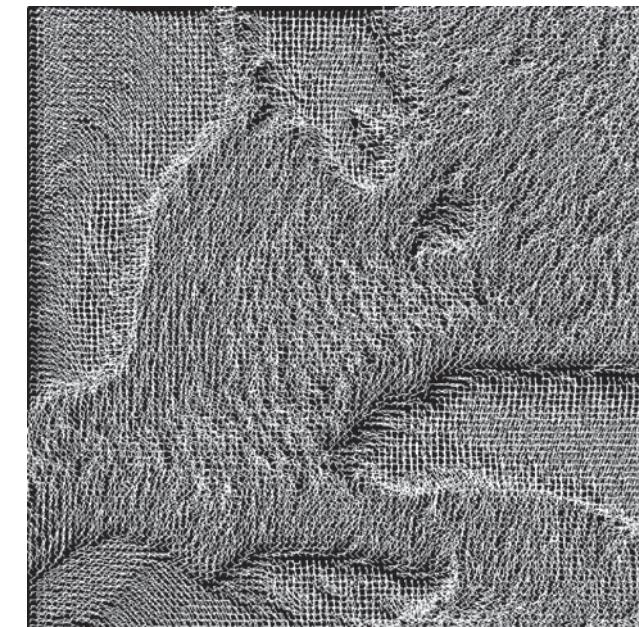


Figure 26.15. 简单字形的密集阵列。图片由GeorgesGrinstein (S.Smith Grinstein, & Bergeron, 1991), © 1991 IEEE.

highlighting, where items selected in one view are highlighted in all others. In linked navigation, movement in one view triggers movement in the others.

There are many kinds of multiple-view approaches. In what is usually called simply the *multiple-view* approach, the same data is shown in several views, each of which has a different visual encoding that shows certain aspects of the dataset most clearly. The power of linked highlighting across multiple visual encodings is that items that fall in a contiguous region in one view are often distributed very differently in the other views. In the *small-multiples* approach, each view has the same visual encoding for different datasets, usually with shared axes between frames so that comparison of spatial position between them is meaningful. Side-by-side comparison with small multiples is an alternative to the visual clutter of superimposing all the data in the same view, and to the human memory limitations of remembering previously seen frames in an animation that changes over time.

The *overview-and-detail* approach is to have the same data and the same visual encoding in two views, where the only difference between them is the level of zooming. In most cases, the overview uses much less display space than the detail view. The combination of overview and detail views is common outside of visualization in many tools ranging from mapping software to photo editing. With a *detail-on-demand* approach, another view shows more information about some selected item, either as a popup window near the cursor or in a permanent window in another part of the display.

Determining the most appropriate spatial position of the views themselves with respect to each other can be as significant a problem as determining the spatial position of marks within a single view. In some systems, the location of the views is arbitrary and left up to the window system or the user. Aligning the views allows precise comparison between them, either vertically, horizontally, or with an array for both directions. Just as items can be sorted within a view, views can be sorted within a display, typically with respect to a derived variable measuring some aspect of the entire view as opposed to an individual item within it.

Figure 26.16 shows a visualization of census data that uses many views. In addition to geographic information, the demographic information for each county includes population, density, gender, median age, percent change since 1990, and proportions of major ethnic groups. The visual encodings used include geographic, scatterplot, parallel coordinate, tabular, and matrix views. The same color encoding is used across all the views, with a legend in the bottom middle. The scatterplot matrix shows linked highlighting across all views, where the blue items are close together in some views and scattered in others. The map in the upper-left corner is an overview for the large detail map in the center. The tabular views allow direct sorting by and selection within a dimension of interest.

突出显示，在一个视图中选择的项目在所有其他视图中突出显示。在链接导航中，一个视图中的移动会触发其他视图中的移动。

多视图方法有很多种。在通常所说的多视图方法中，相同的数据显示在几个视图中，每个视图都有不同的视觉编码，可以最清楚地显示数据集的某些方面。跨多个可视编码的链接突出显示的强大功能是，落在一个视图中连续区域中的项目在其他视图中的分布通常非常不同。在小倍数方法中，每个视图对不同的数据集具有相同的视觉编码，通常在帧之间使用共享轴，以便比较它们之间的空间位置是有意义的。与小倍数的侧边比较是将所有数据叠加在同一视图中的视觉混乱的替代方法，也是记忆动画中先前看到的随时间变化的帧的人类记忆限制的替代方。

概述和细节方法是在两个视图中具有相同的数据和相同的视觉编码，其中它们之间的唯一区别是缩放级别。在大多数情况下，概述使用的显示空间比详细信息视图少得多。在从制图软件到照片编辑的许多工具中，概述视图和细节视图的组合在可视化之外很常见。使用按需详细信息的方法，另一个视图显示有关某些选定项目的更多信息，无论是作为光标附近的弹出窗口还是在显示器的另一部分的永久窗口中。

确定视图本身相对于彼此的最合适的空间位置可能与确定单个视图内标记的空间位置一样是一个重要的问题。在某些系统中，视图的位置是任意的，由窗口系统或用户决定。对齐视图允许它们之间的精确比较，无论是垂直的，水平的，还是两个方向的数组。正如项目可以在视图中排序一样，视图可以在显示中排序，通常是相对于测量整个视图的某个方面的派生变量，而不是其中的单个项目。

图26.16显示了使用许多视图的人口普查数据的可视化。除地理信息外，每个县的人口统计信息还包括人口，密度，性别，年龄中位数，自1990年以来的百分比变化以及主要族群的比例。使用的可视化编码包括地理、散点图、平行坐标、表格和矩阵视图。所有视图都使用相同的颜色编码，底部中间有一个图例。散点图矩阵在所有视图中显示链接突出显示，其中蓝色项目在某些视图中靠近在一起，而在其他视图中分散。左上角的地图是中心大细节地图的概述。表格视图允许在感兴趣的维度中直接排序和选择。

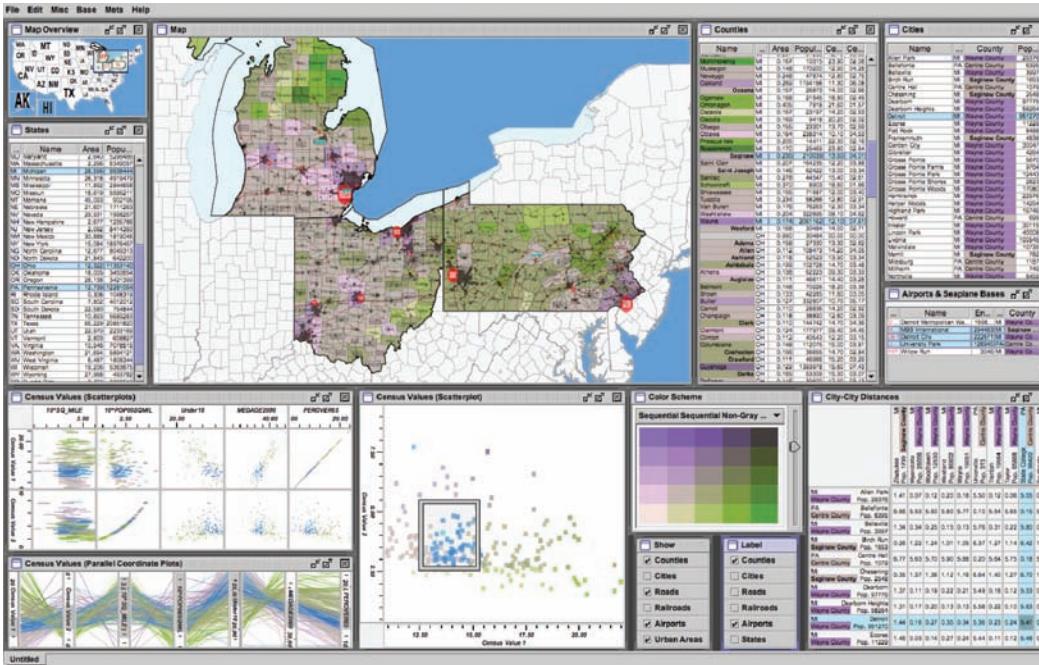


Figure 26.16. The Improvise toolkit was used to create this multiple-view visualization. *Image courtesy Chris Weaver.*

26.7 Data Reduction

The visual encoding techniques that we have discussed so far show all of the items in a dataset. However, many datasets are so large that showing everything simultaneously would result in so much visual clutter that the visual representation would be difficult or impossible for a viewer to understand. The main strategies to reduce the amount of data shown are overviews and aggregation, filtering and navigation, the focus+context techniques, and dimensionality reduction.

26.7.1 Overviews and Aggregation

With tiny datasets, a visual encoding can easily show all data dimensions for all items. For datasets of medium size, an overview that shows information about all items can be constructed by showing less detail for each item. Many datasets have internal or derivable structure at multiple scales. In these cases, a multiscale visual representation can provide many levels of overview, rather than just a single

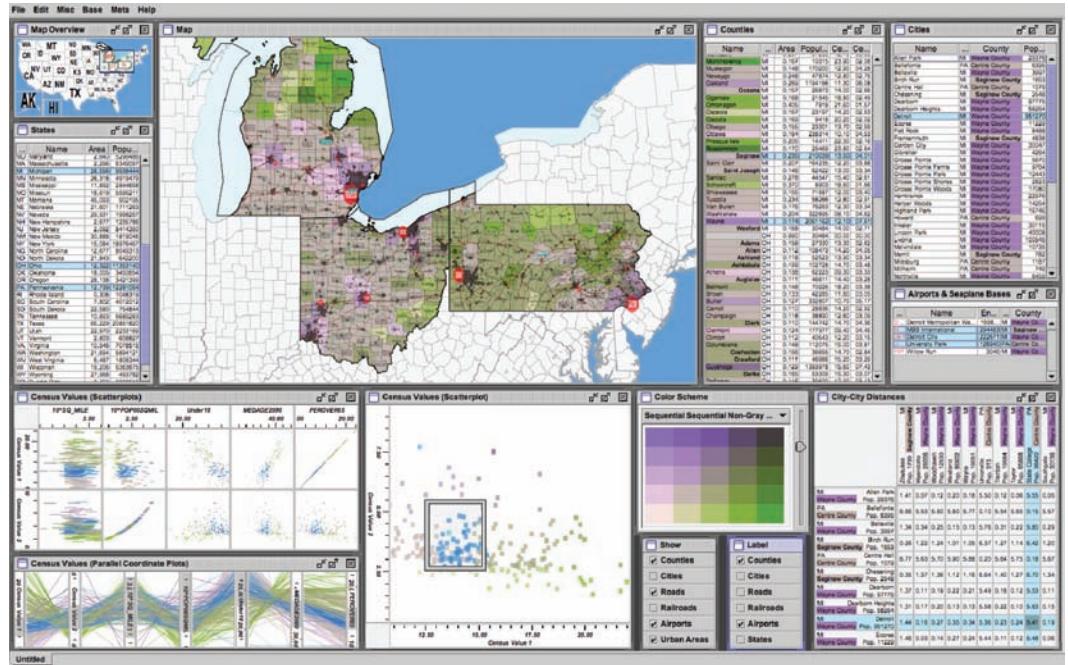


图26.16。即兴工具包被用来创建这个多视图可视化。图片由克里斯·韦弗提供。

26.7 数据缩减

到目前为止，我们讨论的可视化编码技术显示了数据集中的所有项。然而，许多数据集是如此之大，以致于同时显示所有内容将导致如此多的视觉混乱，视觉表示将难以或不可能为观众理解。减少显示的数据量的主要策略是概述和聚合，过滤和导航，焦点+上下文技术和降维。

26.7.1 概述及汇总

对于微小的数据集，可视化编码可以很容易地显示所有项目的所有数据维度。对于中等大小的数据集，可以通过显示每个项目的较少细节来构建显示所有项目信息的概述。许多数据集具有多个尺度的内部或可派生结构。在这些情况下，多尺度可视化表示可以提供许多级别的概述，而不仅仅是单个

level. Overviews are typically used as a starting point to give users clues about where to drill down to inspect in more detail.

For larger datasets, creating an overview requires some kind of visual summarization. One approach to data reduction is to use an *aggregate* representation where a single visual mark in the overview explicitly represents many items.

The challenge of aggregation is to avoid eliminating the interesting signals in the dataset in the process of summarization. In the cartographic literature, the problem of creating maps at different scales while retaining the important distinguishing characteristics has been extensively studied under the name of *cartographic generalization* (Slocum, McMaster, Kessler, & Howard, 2008).

26.7.2 Filtering and Navigation

Another approach to data reduction is to *filter* the data, showing only a subset of the items. Filtering is often carried out by directly selecting ranges of interest in one or more of the data dimensions.

Navigation is a specific kind of filtering based on spatial position, where changing the viewpoint changes the visible set of items. Both geometric and non-geometric zooming are used in visualization. With geometric zooming, the camera position in 2D or 3D space can be changed with standard computer graphics controls. In a realistic scene, items should be drawn at a size that depends on their distance from the camera, and only their apparent size changes based on that distance. However, in a visual encoding of an abstract space, nongeometric zooming can be useful. In *semantic zooming*, the visual appearance of an object changes dramatically based on the number of pixels available to draw it. For instance, an abstract visual representation of a text file could change from a tiny color-coded box with no label to a medium-sized box containing only the filename as a text label to a large rectangle containing a multi-line summary of the file contents. In realistic scenes, objects that are sufficiently far away from the camera are not visible in the images, for example, after they subtend less than one pixel of screen area. With *guaranteed visibility*, one of the original or derived data dimensions is used as a measure of importance, and objects of sufficient importance must have some kind of representation visible in the image plane at all times.

26.7.3 Focus+Context

Focus+context techniques are another approach to data reduction. A subset of the dataset items are interactively chosen by the user to be the focus and are drawn

水平。概述通常用作起点，为用户提供有关在何处深入查看以更详细地检查的线索。

对于较大的数据集，创建概述需要某种视觉总和marization。数据缩减的一种方法是使用聚合表示，其中概述中的单个视觉标记显式表示许多项。

聚合的挑战在于避免在汇总的过程中消除数据集中的有趣信号。在制图文献中，以cartographicgeneralization (Slocum, McMaster, Kessler, & Howard, 2008) 的名义广泛研究了以不同比例创建地图，同时保留重要的去模糊特征的问题。

26.7.2 过滤和导航

数据缩减的另一种方法是过滤数据，仅显示项目的子集。过滤通常是通过直接选择一个或多个数据维度中的感兴趣范围来进行的。

导航是一种基于空间位置的特定过滤，其中改变视点会改变可见的项目集。几何缩放和非几何缩放都用于可视化。通过几何缩放，凸轮在2d或3D空间中的位置可以用标准的计算机图形控制来改变。在一个真实的场景中，项目的绘制尺寸应该取决于它们与相机的距离，并且只有它们的表面尺寸根据这个变化而变化。然而，在抽象空间的视觉编码中，非等距缩放可能很有用。在语义缩放中，对象的视觉外观会根据可用于绘制它的像素数而发生显著变化。例如，文本文件的抽象可视化表示可以从没有标签的微小颜色编码框变为仅包含文件名作为文本标签的中等大小框，变为包含文件内容的多行摘要的大矩形。在现实场景中，距离相机足够远的物体在图像中是不可见的，例如，在它们减去不到一个像素的屏幕区域之后。在保证可见性的情况下，原始或派生数据维度中的一个用作重要性度量，并且具有足够重要性的对象必须具有在图像平面中始终可见的某种表示。

26.7.3 Focus+Context

焦点+上下文技术是数据缩减的另一种方法。用户以交互方式选择数据集项的子集作为焦点并绘制

in detail. The visual encoding also includes information about some or all of the rest of the dataset shown for context, integrated into the same view that shows the focus items. Many of these techniques use carefully chosen distortion to combine magnified focus regions and minified context regions into a unified view.

One common interaction metaphor is a moveable fisheye lens. Hyperbolic geometry provides an elegant mathematical framework for a single radial lens that affects all objects in the view. Another interaction metaphor is to use multiple lenses of different shapes and magnification levels that affect only local regions. Stretch and squish navigation uses the interaction metaphor of a rubber sheet where stretching one region squishes the rest, as shown in Figure 26.17. The borders of the sheet stay fixed so that all items are within the viewport, although many items may be compressed to subpixel size. The fisheye metaphor is not limited to a geometric lens used after spatial layout; it can be used directly on structured data, such as a hierarchical document where some sections are collapsed while others are left expanded.

These distortion-based approaches are another example of nonliteral navigation in the same spirit as nongeometric zooming. When navigating within a large and unfamiliar dataset with realistic camera motion, users can become disori-

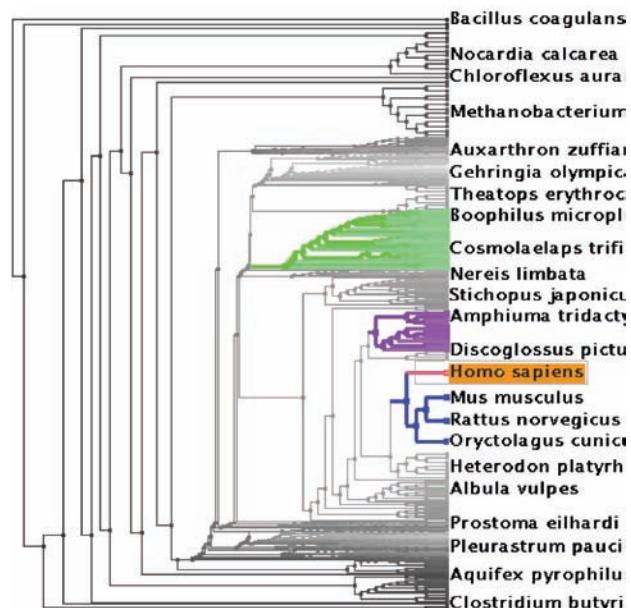
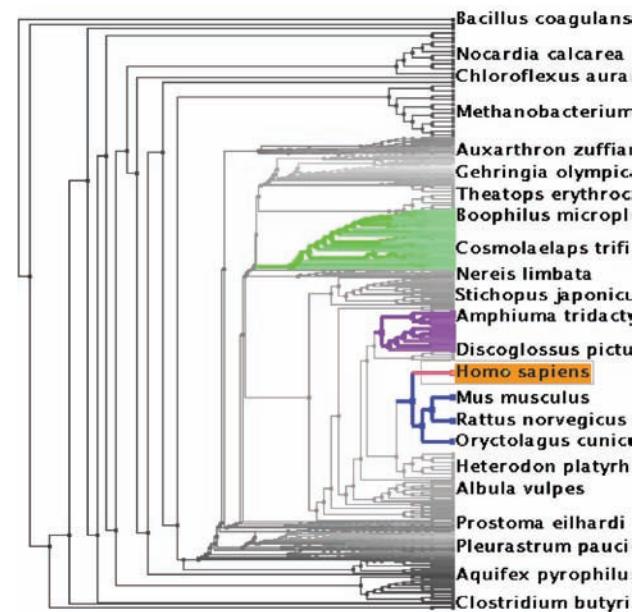


Figure 26.17. The TreeJuxtaposer system features stretch and squish navigation and guaranteed visibility of regions marked with colors (Munzner, Guimbretière, Tasiran, Zhang, & Zhou, 2003).

详细地说。可视化编码还包括有关为上下文显示的数据集其余部分或全部的信息，这些信息集成到显示焦点项目的同一视图中。许多这些技术使用精心选择的失真将放大的焦点区域和缩小的上下文区域组合成一个统一的视图。

一个常见的互动隐喻是可移动的鱼眼镜头。双曲线几何为影响视图中所有对象的单个径向透镜提供了一个优雅的数学框架。另一个交互隐喻是使用不同形状和放大级别的multiple透镜，仅影响局部regions。拉伸和压扁导航使用橡胶片的交互隐喻，其中拉伸一个区域压扁其余区域，如图26.17所示。工作表的边界保持固定，以便所有项目都在视口内，尽管许多项目可能被压缩到亚像素大小。鱼眼隐喻不仅限于空间布局后使用的几何透镜；它可以直接用于结构化数据，例如分层文档，其中一些部分被重叠，而另一些部分则被扩展。

这些基于失真的方法是另一个例子，与非测量缩放的精神相同。使用真实的摄像机运动在大型且不熟悉的数据集内导航时，用户可能会迷失方向



TreeJuxtaposer系统具有拉伸和挤压导航功能，并保证对标有颜色的区域的可见性 (Munzner, Guimbretière, Tasiran, Zhang, & Zhou, 2003)。

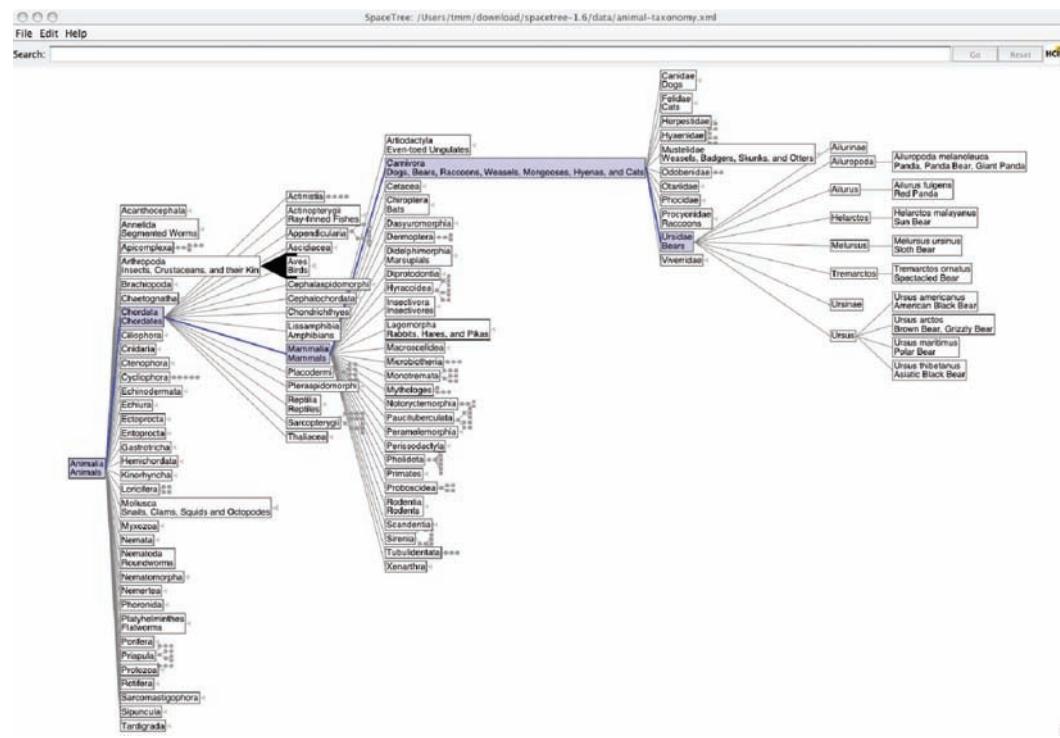


Figure 26.18. The SpaceTree system shows the path between the root and the interactively chosen focus node to provide context (Grosjean, Plaisant, & Bederson, 2002).

ented at high zoom levels when they can see only a small local region. These approaches are designed to provide more contextual information than a single undistorted view, in hopes that people can stay oriented if landmarks remain recognizable. However, these kinds of distortion can still be confusing or difficult to follow for users. The costs and benefits of distortion, as opposed to multiple views or a single realistic view, are not yet fully understood. Standard 3D perspective is a particularly familiar kind of distortion and was explicitly used as a form of focus+context in early visualization work. However, as the costs of 3D spatial layout discussed in Section 26.4 became more understood, this approach became less popular.

Other approaches to providing context around focus items do not require distortion. For instance, the SpaceTree system shown in Figure 26.18 elides most nodes in the tree, showing the path between the interactively chosen focus node and the root of the tree for context.

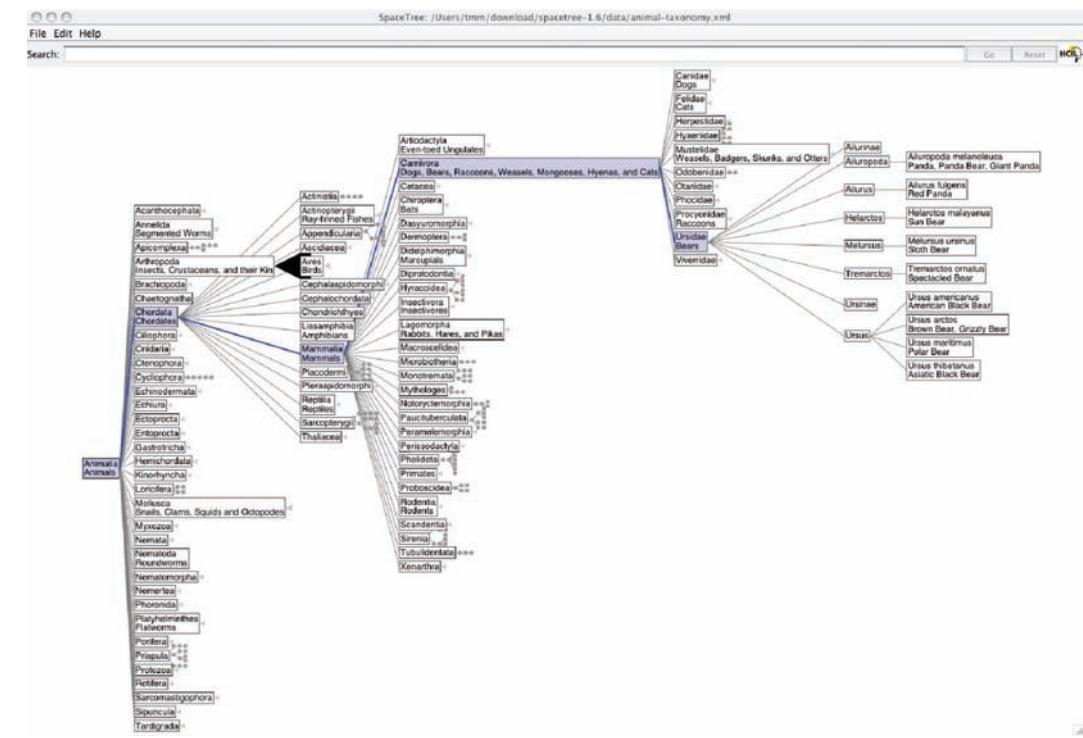


图26.18。 SpaceTree系统显示根和交互选择的焦点节点之间的路径，以提供上下文（Grosjean, Plaisant, & Bederson, 2002）。

当他们只能看到一个局部区域时，在高缩放级别。这些方法旨在提供比单一未失真视图更多的上下文信息，希望如果地标保持可识别，人们可以保持面向。然而，这些失真对于用户来说仍然是令人困惑或难以遵循的。与多个视图或单一现实视图相比，扭曲的成本和收益尚未完全理解。标准3dperspective是一种特别熟悉的失真，在早期可视化工作中被明确用作焦点+上下文的一种形式。然而，随着第26.4节中讨论的3d空间布局成本的理解越来越多，这种方法变得不那么流行。

提供围绕焦点项目的上下文的其他方法不需要dis折磨。例如，图26.18中所示的SpaceTree系统会显示树中的大多数节点，显示交互选择的焦点节点和树根之间的路径，用于上下文。



26.7.4 Dimensionality Reduction

The data reduction approaches covered so far reduce the number of items to draw. When there are many data dimensions, *dimensionality reduction* can also be effective.

With *slicing*, a single value is chosen from the dimension to eliminate, and only the items matching that value for the dimension are extracted to include in the lower-dimensional slice. Slicing is particularly useful with 3D spatial data, for example when inspecting slices through a CT scan of a human head at different heights along the skull. Slicing can be used to eliminate multiple dimensions at once.

With *projection*, no information about the eliminated dimensions is retained; the values for those dimensions are simply dropped, and all items are still shown. A familiar form of projection is the standard graphics perspective transformation which projects from 3D to 2D, losing information about depth along the way. In mathematical visualization, the structure of higher-dimensional geometric objects can be shown by projecting from 4D to 3D before the standard projection to the image plane and using color to encode information from the projected-away dimension. This technique is sometimes called *dimensional filtering* when it is used for nonspatial data.

In some datasets, there may be interesting hidden structure in a much lower-dimensional space than the number of original data dimensions. For instance, sometimes directly measuring the independent variables of interest is difficult or

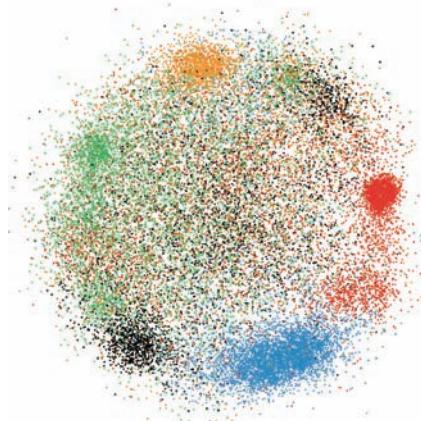


Figure 26.19. Dimensionality reduction with the Glimmer multidimensional scaling approach shows clusters in a document dataset (Ingram, Munzner, & Olano, 2009), © 2009 IEEE.



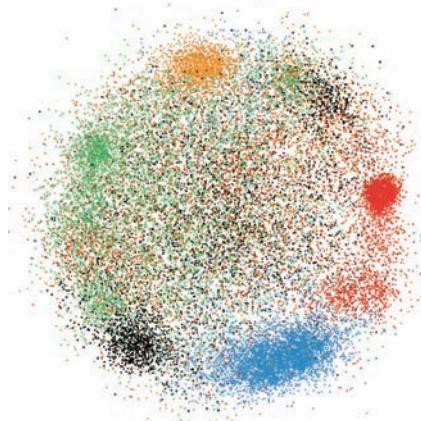
26.7.4 降维

到目前为止所涵盖的数据减少方法减少了要绘制的项目的数量。当有许多数据维度时，降维也可以是有效的。

使用切片时，将从要消除的维度中选择单个值，并且仅提取与该维度的该值匹配的项以包含在较低维切片中。切片对于3D空间数据特别有用，例如，当通过沿头骨不同高度的人头的CT扫描检查切片时。切片可用于一次消除多个维度。

使用投影时，不会保留有关已消除维度的信息；只需删除这些维度的值，并且仍会显示所有项。一种熟悉的投影形式是标准的图形透视转换，它从3D投影到2D，一路上会丢失有关深度的信息。在数学可视化中，更高维度几何对象的结构可以通过在标准投影到图像平面之前从4d投影到3d并使用颜色来编码来自投影-离dimension的信息来示出。当这种技术用于非空间数据时，有时称为维度过滤。

在某些数据集中，在比原始数据维数低得多的维空间中可能存在有趣的隐藏结构。例如，有时直接测量感兴趣的自变量是困难的或



微光多维缩放方法的降维显示文档数据集中的聚类(Ingram, Munzner & Olano 2009) ©2009IEEE.

impossible, but a large set of dependent or indirect variables is available. The goal is to find a small set of dimensions that faithfully represent most of the structure or variance in the dataset. These dimensions may be the original ones, or synthesized new ones that are linear or nonlinear combinations of the originals. Principal component analysis is a fast, widely used linear method. Many nonlinear approaches have been proposed, including multidimensional scaling (MDS). These methods are usually used to determine whether there are large-scale clusters in the dataset; the fine-grained structure in the lower-dimensional plots is usually not reliable because information is lost in the reduction. Figure 26.19 shows document collection in a single scatterplot. When the true dimensionality of the dataset is far higher than two, a matrix of scatterplots showing pairs of synthetic dimensions may be necessary.

26.8 Examples

We conclude this chapter with several examples of visualizing specific types of data using the techniques discussed above.

26.8.1 Tables

Tabular data is extremely common, as all spreadsheet users know. The goal in visualization is to encode this information through easily perceptible visual channels rather than forcing people to read through it as numbers and text. Figure 26.20 shows the Table Lens, a focus+context approach where quantitative values are encoded as the length of one-pixel high lines in the context regions, and shown as numbers in the focus regions. Each dimension of the dataset is shown as a column, and the rows of items can be resorted according to the values in that column with a single click in its header.

The traditional Cartesian approach of a scatterplot, where items are plotted as dots with respect to perpendicular axes, is only usable for two and three dimensions of data. Many tables contain far more than three dimensions of data, and the number of additional dimensions that can be encoded using other visual channels is limited. Parallel coordinates are an approach for visualizing more dimensions at once using spatial position, where the axes are parallel rather than perpendicular and an n -dimensional item is shown as a polyline that crosses each of the n axes once (Inselberg & Dimsdale, 1990; Wegman, 1990). Figure 26.21 shows an eight-dimensional dataset of 230,000 items at multiple levels of detail (Fua, Ward, & Rundensteiner, 1999), from a high-level view at the top to finer detail

不可能，但一大组因变量或间接变量是可用的。目标是找到一小组维度，这些维度忠实地表示数据集中的大部分结构或方差。这些维度可以是原始维度，也可以是原始维度的线性或非线性组合的合成新维度。主成分分析是一种快速，广泛使用的线性方法。已经提出了许多非线性方法，包括多维缩放 (mds)。这些方法通常用于确定数据集中是否存在大规模聚类；较低维图中的细粒度结构通常不可靠，因为信息在减少中丢失。图26.19显示了单个散点图中的文档集合。当数据集的真实维数远远高于两个时，可能需要一个显示合成维数对的散点图矩阵。

26.8 Examples

在本章的结尾，我们用几个使用上面讨论的技术可视化特定类型数据的例子。

26.8.1 Tables

表格数据非常常见，所有电子表格用户都知道。可视化的目标是通过易于感知的视觉通道对这些信息进行编码，而不是强迫人们将其作为数字和文本阅读。图26.20显示了表透镜，一种焦点+上下文方法，其中定量值被编码为上下文区域中一像素高线的长度，并在焦点区域中显示为数字。数据集的每个维度都显示为一列，只需在其标题中单击一下，即可根据该列中的值对项目行进行排序。

散点图的传统笛卡尔方法（其中项目相对于垂直轴绘制为点）仅可用于两个和三个点的数据。许多表包含远不止三个维度的数据，并且可以使用其他视觉通道编码的附加维度的数量是有限的。平行坐标是一种使用空间位置一次可视化更多维度的方法，其中轴是平行而不是垂直的，并且 n 维项目显示为交叉 n 个轴中的每一个一次的折线（Inselberg & Dimsdale, 1990; Wegman, 图26.21显示了一个八维数据集，包含230 000个多层次细节的项目（Fua Ward & Rundensteiner 1999），从顶部的高级视图到更精细的细节

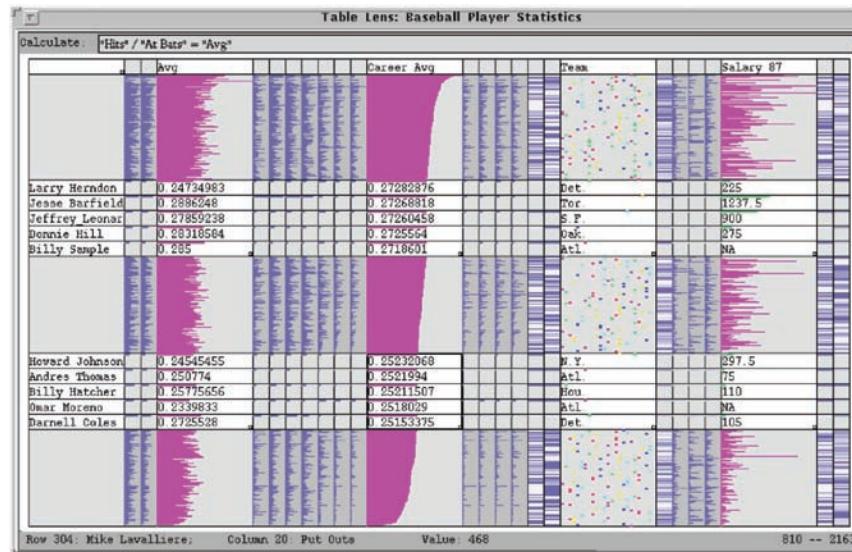


Figure 26.20. The Table Lens provides focus+context interaction with tabular data, immediately reorderable by the values in each dimension column. *Image courtesy Stuart Card (Rao & Card, 1994), © 1994 ACM, Inc. Included here by permission.*

at the bottom. With hierarchical parallel coordinates, the items are clustered and an entire cluster of items is represented by a band of varying width and opacity, where the mean is in the middle and width at each axis depends on the values of the items in the cluster in that dimension. The coloring of each band is based on the proximity between clusters according to a similarity metric.

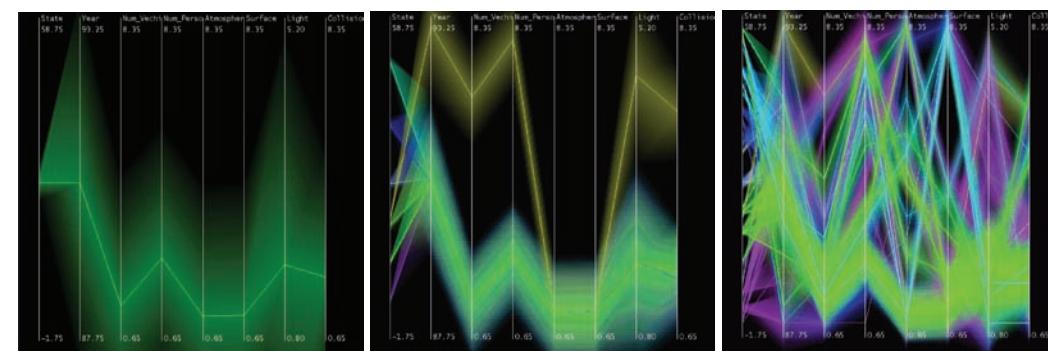
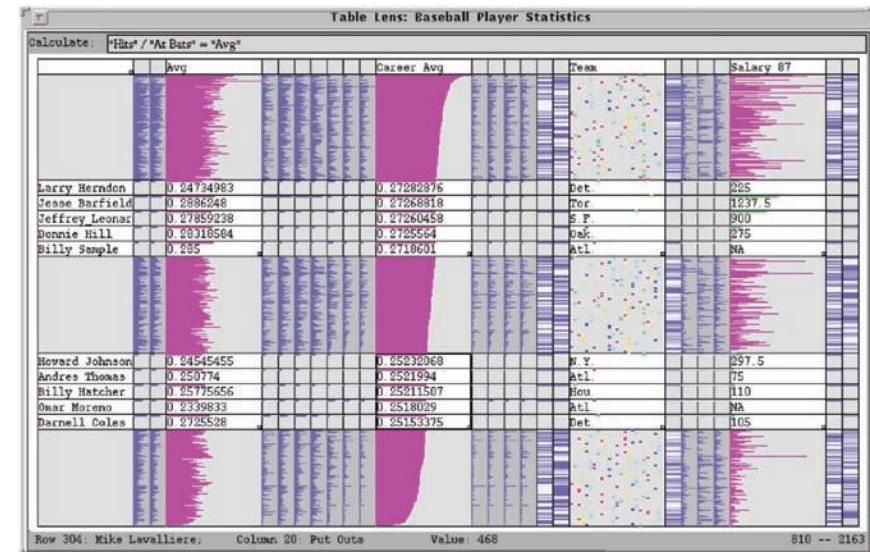


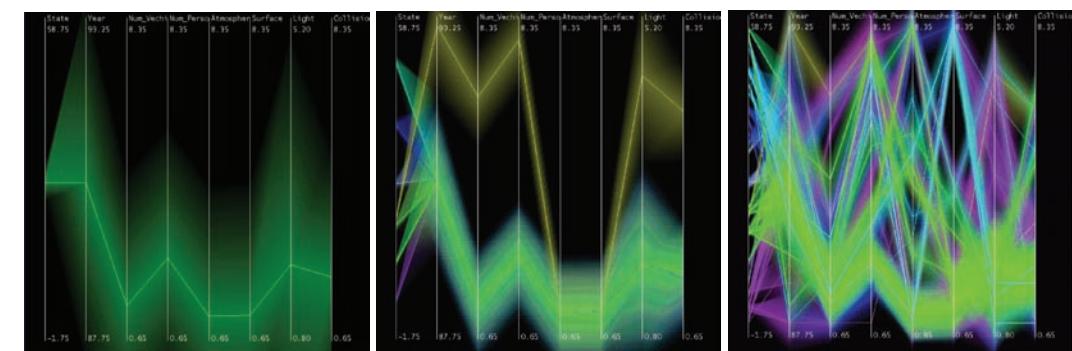
Figure 26.21. Hierarchical parallel coordinates show high-dimensional data at multiple levels of detail. *Image courtesy Matt Ward (Fua et al., 1999), © 1999 IEEE.*



表格透镜提供与表格数据的焦点+上下文交互，可立即通过每个维度列中的值重新排序。图
像礼貌斯图尔特卡 (Rao & Card, 1994) , ©1994ACM, Inc. 经许可包括在这里。

在底部。使用分层平行坐标，项目被聚类，整个项目集群由一个宽度和不透明度变化的带表示，其中平均值在中间，每个轴的宽度取决于该维度中集群中项目的值。每个带的着色基于根据相似性度量的聚类之间的接近度

◦



分层平行坐标以多个细节级别显示高维数据。图像礼貌MattWard (Fua等人。, 1999) , ©1999IEEE.

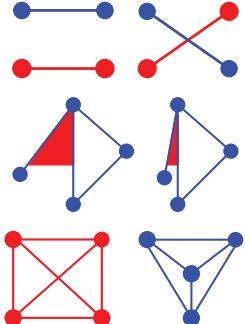


Figure 26.22. Graph layout aesthetic criteria. Top: Edge crossings should be minimized. Middle: Angular resolution should be maximized. Bottom: Symmetry is maximized on the left, whereas crossings are minimized on the right, showing the conflict between the individually NP-hard criteria.

The field of graph drawing is concerned with finding a spatial position for the nodes in a graph in 2D or 3D space and routing the edges between these nodes (Di Battista, Eades, Tamassia, & Tollis, 1999). In many cases the edge-routing problem is simplified by using only straight edges, or by only allowing right-angle bends for the class of *orthogonal* layouts, but some approaches handle true curves. If the graph has directed edges, a layered approach can be used to show hierarchical structure through the horizontal or vertical spatial ordering of nodes, as shown in Figure 26.2.

A suite of aesthetic criteria operationalize human judgments about readable graphs as metrics that can be computed on a proposed layout (Ware, Purchase, Colpys, & McGill, 2002). Figure 26.22 shows some examples. Some metrics should be minimized, such as the number of edge crossings, the total area of the layout, and the number of right-angle bends or curves. Others should be maximized, such as the angular resolution or symmetry. The problem is difficult because most of these criteria are individually NP-hard, and moreover they are mutually incompatible (Brandenburg, 1988).

Many approaches to node-link graph drawing use force-directed placement, motivated by the intuitive physical metaphor of spring forces at the edges drawing together repelling particles at the nodes. Although naive approaches have high time complexity and are prone to being caught in local minima, much work has gone into developing more sophisticated algorithms such as GEM (Frick, Ludwig, & Mehldau, 1994) or IPsep-CoLa (Dwyer, Koren, & Marriott, 2006). Figure 26.23 shows an interactive system using the *r*-PolyLog energy model, where

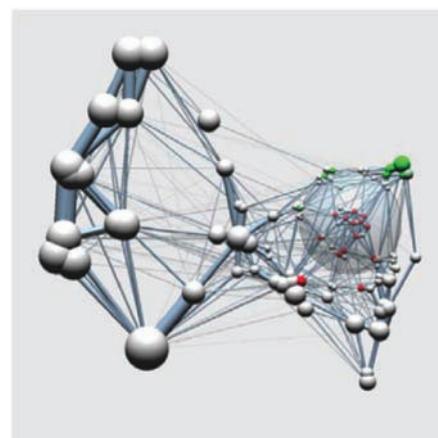
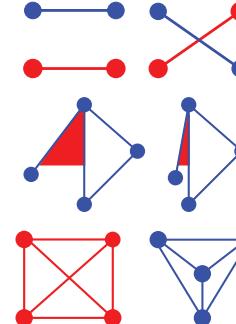


Figure 26.23. Force-directed placement showing a clustered graph with both geometric and semantic fisheye. Image courtesy Jarke van Wijk (van Ham & van Wijk, 2004), © 2004 IEEE.



图形奠定了审美标准。

边缘交叉应该是最小的。中间：应最大限度地提高角解决。底部：
对称性在左边最大，而交叉在右边最小，显示了单独的
NP硬标准之间的冲突。

绘制领域涉及在2d或3D空间中为图中的节点找到空间位置并在这些节点之间路由边 (Di Battista, Eades, Tamassia, & Tollis, 1999)。在许多情况下，通过仅使用直边或仅允许正交布局类的右角弯曲来简化边缘路由问题，但有些方法处理真曲线。如果图形具有有向边，则可以使用分层方法通过节点的水平或垂直空间排序来显示分层结构，如图26.2所示。

一套美学标准将人类对可读图形的判断作为可以在建议的布局上计算的指标 (Ware, Purchase, Colpys, & McGill, 2002)。图26.22显示了一些示例。应尽量减少一些指标，例如边缘交叉的数量、布局的总面积以及直角弯曲或曲线的数量。其他应最大化，例如角分辨率或对称性。这个问题很难，因为大多数这些标准都是单独的NP-hard，而且它们是相互不相容的 (勃兰登堡，1988)。

节点链接图绘制的许多方法都使用力定向放置，其动机是边缘弹簧力的直观物理隐喻将节点上的排斥粒子绘制在一起。虽然天真的方法具有很高的时间复杂性并且容易陷入局部极小值，但许多工作已经投入到开发更复杂的算法中，例如GEM (Frick, Ludwig, & Mehldau, 1994) 或IPsep-CoLa (Dwyer, Koren, & Marriott, 2006)。图26.23显示了使用r-PolyLog能量模型的交互式系统，其中

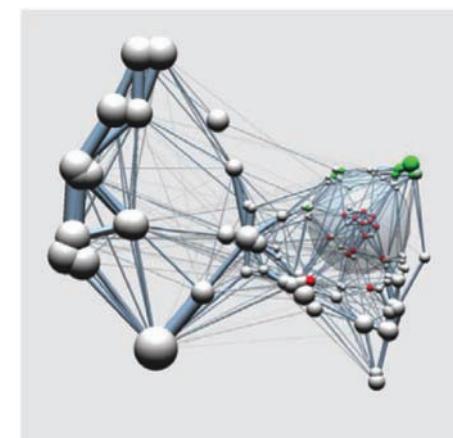


图26.23。力定向放置显示具有几何和语义鱼眼的聚类图。图像礼貌JarkevanWijk (vanHam & vanWijk, 2004)，©2004IEEE。

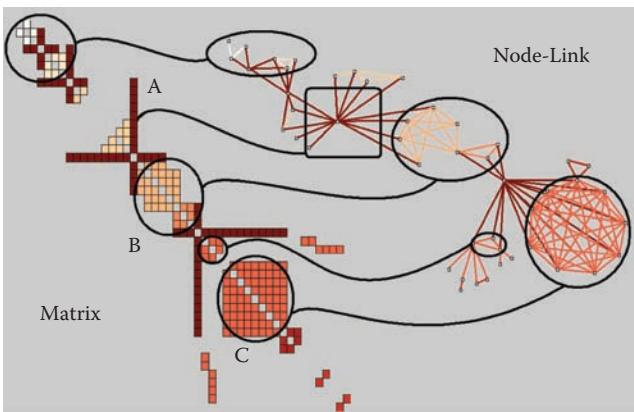


Figure 26.24. Graphs can be shown with either matrix or node-link views. *Image courtesy Jean-Daniel Fekete (Henry & Fekete, 2006), © 2006 IEEE.*

a focus+context view of the clustered graph is created with both geometric and semantic fisheye (van Ham & van Wijk, 2004).

Graphs can also be visually encoded by showing the adjacency matrix, where all vertices are placed along each axis and the cell between two vertices is colored if there is an edge between them. The MatrixExplorer system uses linked multiple views to help social science researchers visually analyze social networks with both matrix and node-link representations (Henry & Fekete, 2006). Figure 26.24 shows the different visual patterns created by the same graph structure in these two views: A represents an actor connecting several communities; B is a community; and C is a clique, or a complete sub-graph. Matrix views do not suffer from cluttered edge crossings, but many tasks including path following are more difficult with this approach.

26.8.3 Trees

Trees are a special case of graphs so common that a great deal of visualization research has been devoted to them. A straightforward algorithm to lay out trees in the two-dimensional plane works well for small trees (Reingold & Tilford, 1981), while a more complex but scalable approach runs in linear time (Buchheim, Jünger, & Leipert, 2002). Figures 26.17 and 26.18 also show trees with different approaches to spatial layout, but all four of these methods visually encode the relationship between parent and child nodes by drawing a link connecting them.

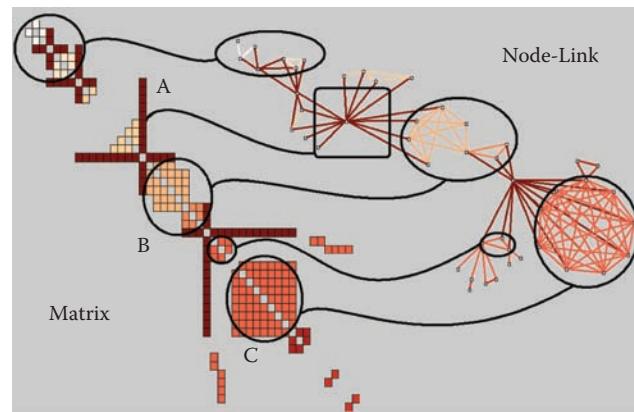


Figure 26.24. 图形可以用矩阵或节点链接视图显示。图片由Jean-Daniel Fekete (Henry & Fekete, 2006), © 2006 IEEE.

使用几何和语义鱼眼创建聚类图的焦点+上下文视图 (vanHam & vanWijk, 2004)。

图形也可以通过显示邻接矩阵进行视觉编码，其中所有顶点沿着每个轴放置，如果两个顶点之间有边，则两个顶点之间的单元格将被着色。MatrixExplorer系统使用链接的多视图来帮助社会科学研究人员直观地分析具有矩阵和节点链接表示的社交网络 (Henry & Fekete, 2006)。图26.24显示了这两个视图中由相同的图结构创建的不同视觉模式：A表示连接几个社区的actor;B是community;C是clique，或完整的子图。矩阵视图不会受到杂乱的边缘交叉的影响，但是使用这种方法，包括路径跟踪在内的许多任务更加困难。

26.8.3 Trees

树是图形的一个特例，如此常见，以至于大量的可视化研究都致力于它们。在二维平面中布置树木的简单算法适用于小树 (Reingold & Tilford, 1981)，而更复杂但可扩展的方法在线性时间内运行 (Buchheim, Junger, & Leipert, 2002)。图26.17和26.18还显示了具有不同空间布局方法的树，但所有四种方法都通过绘制连接它们的链接来直观地编码父节点和子节点之间的关系。

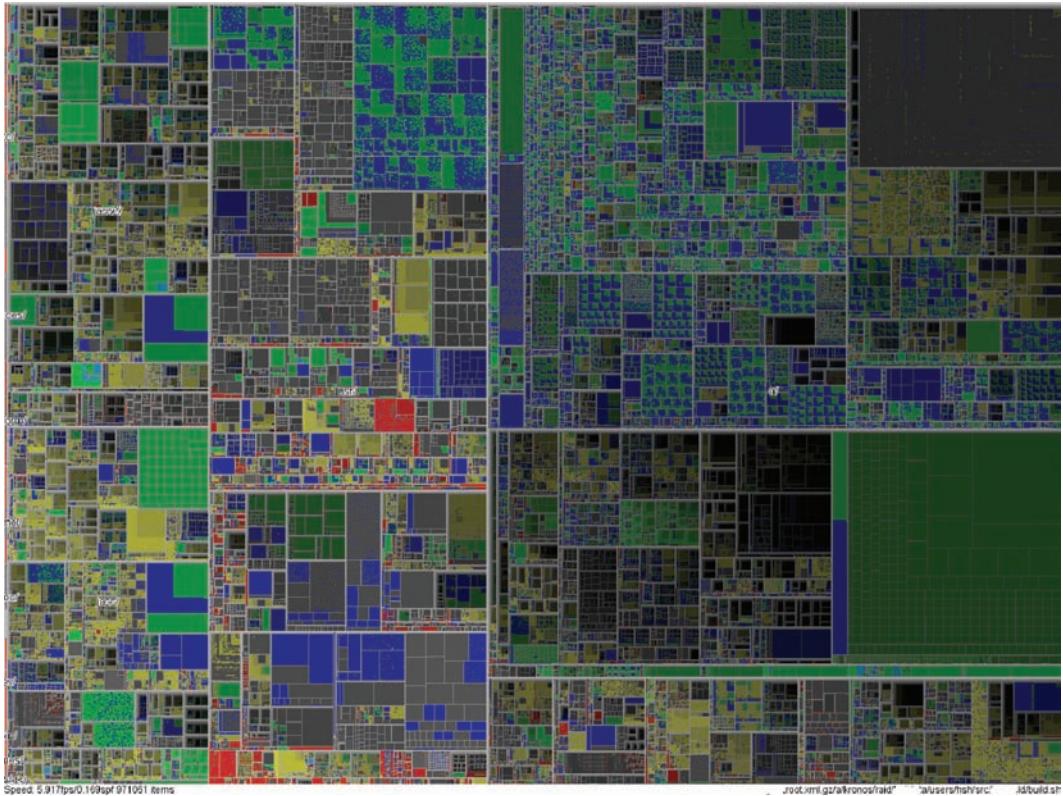


Figure 26.25. Treemap showing a filesystem of nearly one million files. *Image courtesy Jean-Daniel Fekete (Fekete & Plaisant, 2002), © 2002 IEEE.*

Treemaps use containment rather than connection to show the hierarchical relationship between parent and child nodes in a tree (B. Johnson & Shneiderman, 1991). That is, treemaps show child nodes nested within the outlines of the parent node. Figure 26.25 shows a hierarchical filesystem of nearly one million files, where file size is encoded by rectangle size and file type is encoded by color (Fekete & Plaisant, 2002). The size of nodes at the leaves of the tree can encode an additional data dimension, but the size of nodes in the interior does not show the value of that dimension; it is dictated by the cumulative size of their descendants. Although tasks such as understanding the topological structure of the tree or tracing paths through it are more difficult with treemaps than with node-link approaches, tasks that involve understanding an attribute tied to leaf nodes are well supported. Treemaps are space-filling representations that are usually more compact than node-link approaches.

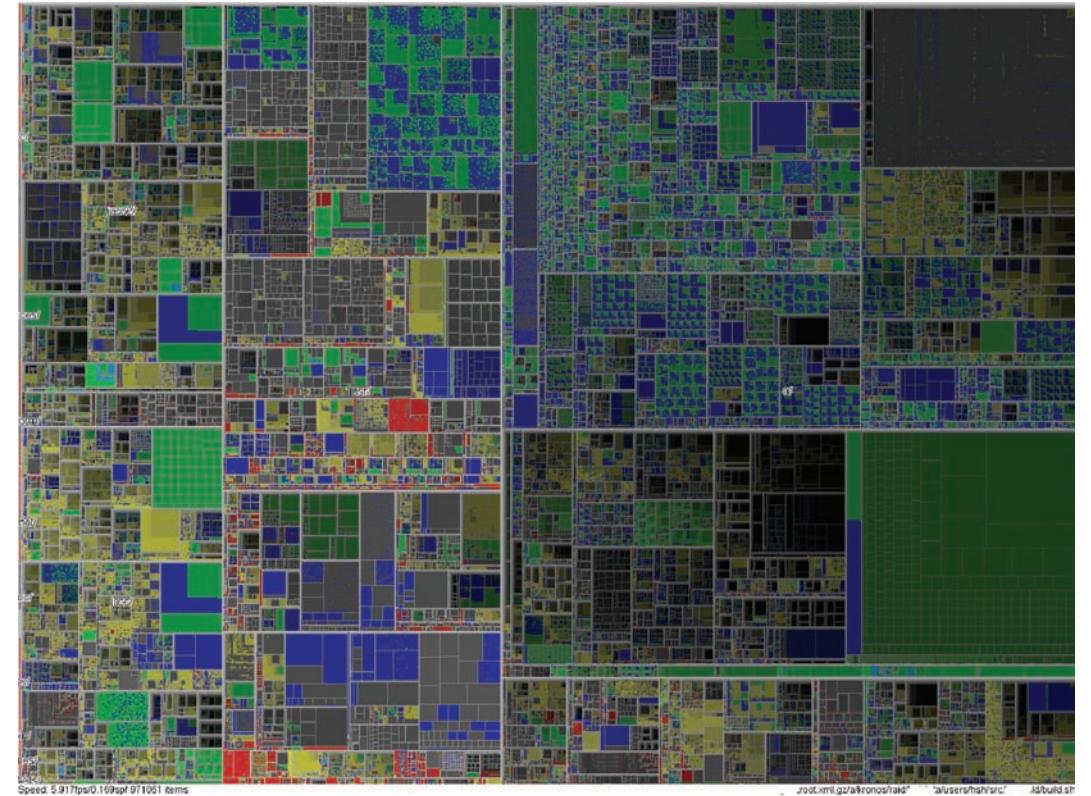


Figure 26.25. 树状图显示了一个将近一百万个文件的文件系统.图片由Jean提供 Daniel Fekete (Fekete & Plaisant, 2002), © 2002 IEEE.

树图使用包含而不是连接来显示树中父节点和子节点之间的层次关系 (B. Johnson & Shneiderman, 1991)。也就是说，树形图显示嵌套在父节点轮廓内的子节点。图26.25显示了一个由近一百万个文件组成的分层文件系统，其中文件大小由矩形大小编码，文件类型由颜色编码 (Fekete & Plaisant, 2002)。树叶节点的大小可以编码一个额外的数据维度，但内部节点的大小并不显示该维度的值；它由它们的后代的累积大小决定。虽然理解树的拓扑结构或跟踪通过树的路径等任务使用树图比使用node-link方法更困难，但涉及理解与叶节点关联的属性的任务得到了很好的支持。树形图是填充空间的表示，通常比节点链接方法更紧凑。



26.8.4 Geographic

Many kinds of analysis such as epidemiology require understanding both geographic and nonspatial data. Figure 26.26 shows a tool for the visual analysis of a cancer demographics dataset that combines many of the ideas described in this chapter (MacEachren, Dai, Hardisty, Guo, & Lengerich, 2003). The top matrix of linked views features small multiples of three types of visual encodings: geographic maps showing Appalachian counties at the lower left, histograms across the diagonal of the matrix, and scatterplots on the upper right. The bottom 2×2 matrix, linking scatterplots with maps, includes the color legend for both. The discrete bivariate sequential colormap has lightness increasing sequentially for each of two complementary hues and is effective for color-deficient people.

26.8.5 Spatial Fields

Most nongeographic spatial data is modeled as a field, where there are one or more values associated with each point in 2D or 3D space. Scalar fields, for example CT or MRI medical imaging scans, are usually visualized by finding isosurfaces or using direct volume rendering. Vector fields, for example, flows in water or air, are often visualized using arrows, streamlines (McLoughlin, Laramee, Peikert, Post, & Chen, 2009), and *line integral convolution* (LIC) (Laramee et al., 2004). Tensor fields, such as those describing the anisotropic diffusion of molecules through the human brain, are particularly challenging to display (Kindlmann, Weinstein, & Hart, 2000).

Frequently Asked Questions

- What conferences and journals are good places to look for further information about visualization?

The IEEE VisWeek conference comprises three subconferences: InfoVis (Information Visualization), Vis (Visualization), and VAST (Visual Analytics Science and Technology). There is also a European EuroVis conference and an Asian PacificVis venue. Relevant journals include IEEE TVCG (Transactions on Visualization and Computer Graphics) and Palgrave Information Visualization.



26.8.4 Geographic

流行病学等许多类型的分析都需要理解地理和非空间数据。图26.26显示了一个癌症人口统计数据集的可视化分析工具，该数据集结合了本章中描述的许多想法 (MacEachren, Dai, Hardisty, Guo, & Lengerich, 2003)。链接视图的顶部矩阵具有三种类型的视觉编码的小倍数：左下方显示阿巴拉契亚县的地理地图，矩阵对角线上的直方图以及右上方的散点图。底部的 2×2 矩阵（将散点图与地图连接起来）包括两者颜色图例。离散双变量顺序颜色表的亮度顺序增加每两个互补色调，是有效的颜色缺陷的人。

26.8.5 空间场

大多数非空间数据被建模为一个字段，其中有一个或多个值与2D或3d空间中的每个点相关联。标量场，例如CT或MRI医学成像扫描，通常通过查找等值面或使用直接体积渲染来可视化。矢量场，例如，在水或空气中的流动，通常使用箭头，流线 (McLoughlin, Laramee, Peikert, Post, & Chen, 2009) 和线积分卷积 (LIC) 可视化 (Larameeetal. 2004).张量场，例如那些描述分子通过人脑的各向异性扩散的场，显示起来特别具有挑战性 (Kindlmann, Weinstein, & Hart, 2000)。

常见问题

- *哪些会议和期刊是寻找有关可视化的进一步信息的好地方？

IEEEVisWeek会议包括三个子会议：InfoVis（信息可视化），Vis（可视化）和VAST（可视化分析科学与技术）。还有一个欧洲EuroVis会议和一个亚洲PacificVis场地。相关期刊包括IEEE TVCG（Transactions on Visualization and Computer Graphics）和Palgrave Information Visualization。

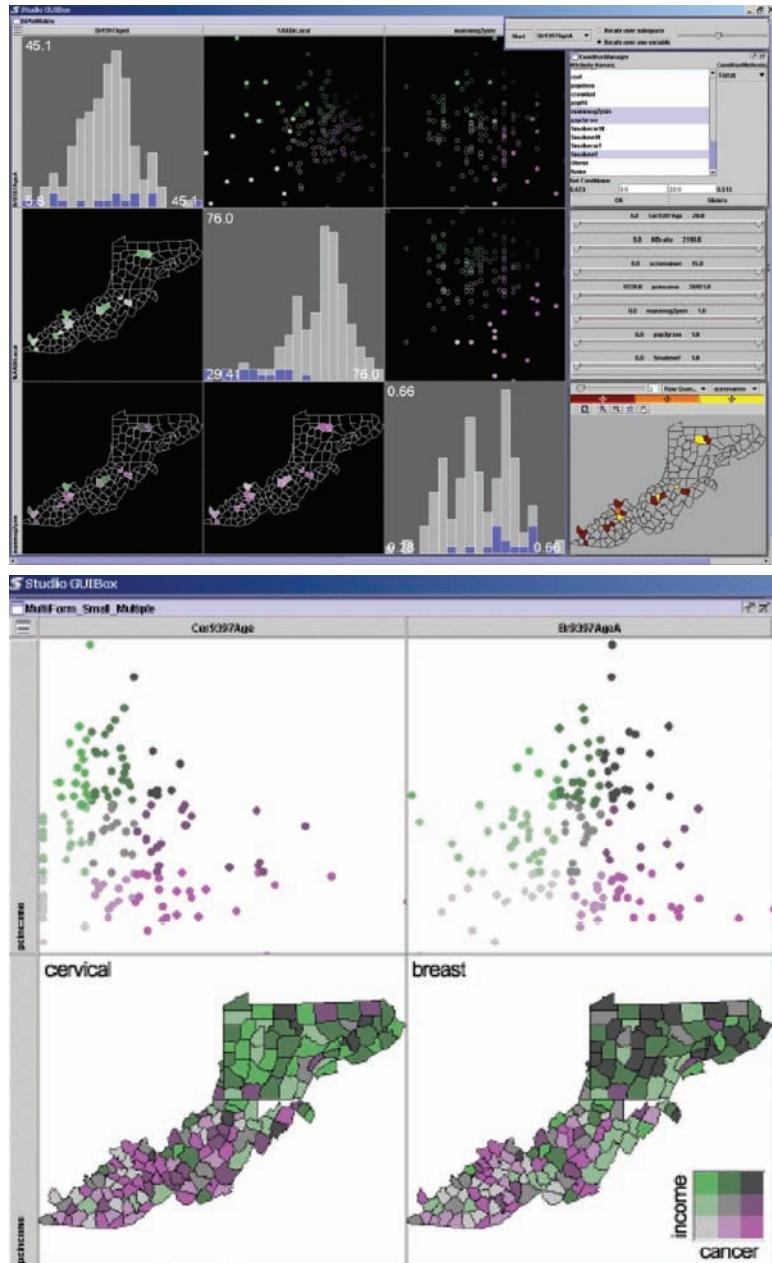
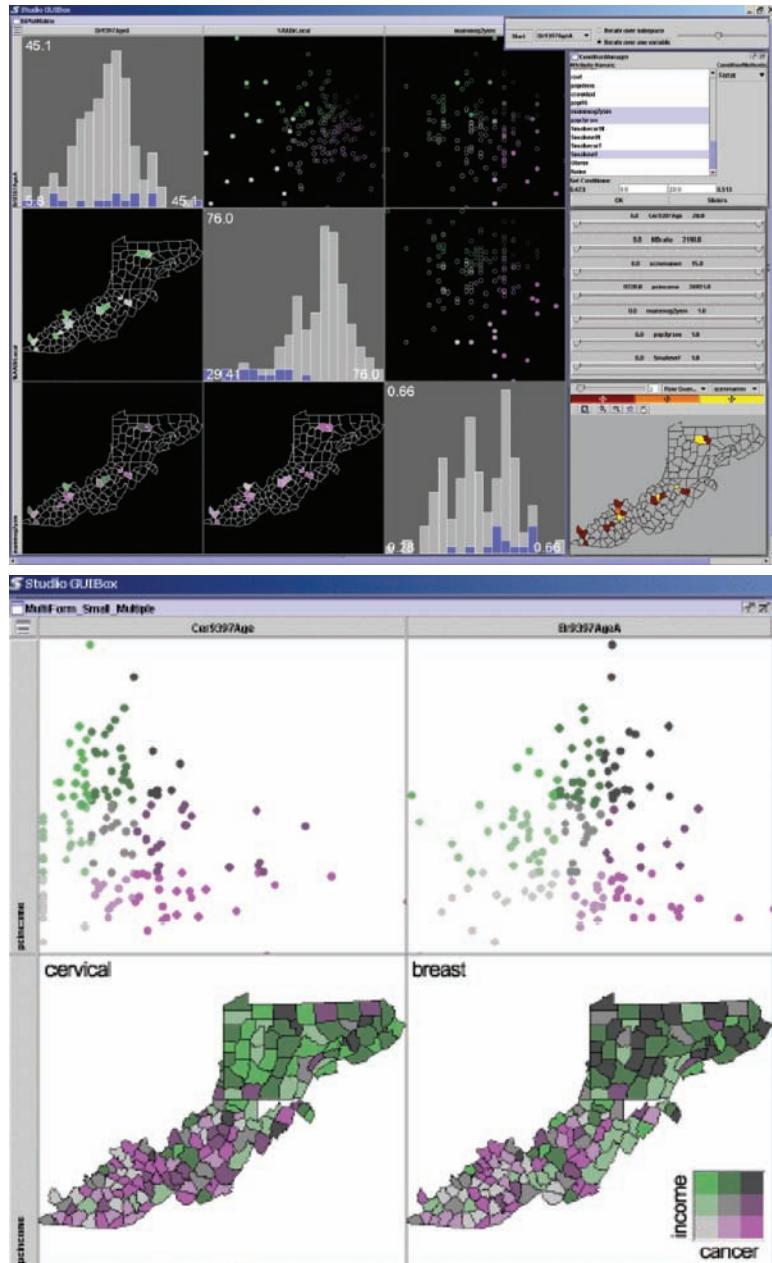


Figure 26.26. Two matrices of linked small multiples showing cancer demographic data (MacEachren et al., 2003), © 2003 IEEE.



显示癌症人口统计数据的链接小倍数的两个矩阵 (MacEachren等。, 2003) , ©2003IEEE.



- What software and toolkits are available for visualization?

The most popular toolkit for spatial data is `vtk`, a C/C++ codebase available at www.vtk.org. For abstract data, the Java-based `prefuse` (<http://www.prefuse.org>) and `Processing` (processing.org) toolkits are becoming widely used. The `ManyEyes` site from IBM Research (www.many-eyes.com) allows people to upload their own data, create interactive visualizations in a variety of formats, and carry on conversations about visual data analysis.



- *有哪些软件和工具包可用于可视化?

最流行的空间数据工具包是 `vtk`, 它是一个c++代码库, 可在www.vtk.org... 对于抽象数据, 基于Java的 `prefuse` (<http://www.prefuse.org>) 和处理 (`processing.org`) 工具包正变得广泛使用。来自IBMResearch的 `ManyEyes` 网站 (www.many-eyes.com) 允许人们上传自己的数据, 以各种格式创建交互式可视化, 并开展有关可视化数据分析的对话。

References

- 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.
- Adzhiev, V., Cartwright, R., Fausett, E., Ossipov, A., Pasko, A., & Savchenko, V. (1999, Sep). Hyperfun Project: A Framework for Collaborative Multidimensional F-rep Modeling. In *Implicit Surfaces '99* (pp. 59–69). Aire-la-Ville, Switzerland: Eurographics Association.
- Akenine-Möller, T., Haines, E., & Hoffman, N. (2008). *Real-Time Rendering* (Third ed.). Wellesley, MA: A K Peters.
- Akkouche, S., & Galin, E. (2001). Adaptive Implicit Surface Polygonization Using Marching Triangles. *Computer Graphics Forum*, 20(2), 67–80.
- Akleman, E., & Chen, J. (1999). Generalized Distance Functions. In *Proceedings of the International Conference on Shape Modeling and Applications* (pp. 72–79). Washington, DC: IEEE Computer Society Press.
- Amanatides, J., & Woo, A. (1987). A Fast Voxel Traversal Algorithm for Ray Tracing. In *Proceedings of Eurographics* (pp. 1–10). Amsterdam: Elsevier Science Publishers.
- Amar, R., Eagan, J., & Stasko, J. (2005). Low-Level Components of Analytic Activity in Information Visualization. In *Proc. IEEE Symposium on Information Visualization (InfoVis)* (pp. 111–117). Washington, DC: IEEE Computer Society Press.
- 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*

References

- Adelson, E. H. (1999). 明度感知和明度幻象。In M.S.Gazzaniga (Ed.) , 新的认知神经科学 (第二版。, 第339–351页) 。剑桥, MA: 麻省理工学院出版社。Adzhiev, V., Cartwright, R., Fausett, E., Ossipov, A., Pasko, A., & Savchenko, V. (1999, Sep) 。Hyperfun项目：一个用于协同多角度F-rep建模的框架。InImplicitSurfaces'99 (第59–69页) 。Aire-laVille, 瑞士: EurographicsAssociation。Akenine-Möller, T., Haines, E., & Hoffman, N. (2008) 。实时渲染 (第三版) 。韦尔斯利, MA: AK彼得斯。Akkouche, S., & Galin, E. (2001) 。使用行进三角形的自适应隐式曲面多边形化。计算机图形论坛, 20 (2) , 67–80。
- Akleman, E., & Chen, J. (1999) 。广义距离函数。InProceedingsOfTheInternationalConferenceonShapeModelingandApplications(pp.72–79)。华盛顿特区: IEEE计算机学会出版社。Amanatides, J., & Woo, A. (1987) 。一种射线快速体素遍历算法
追踪。《欧洲制图学会议记录》(第1–10页)。阿姆斯特丹: 爱思唯尔科学出版社。
- Amar, R., Eagan, J., & Stasko, J. (2005) 。分析的低级成分
信息可视化中的活动。在Proc. IEEESymposiumonInformationVisualization(InfoVis)(第111–117页)。华盛顿特区: IEEE计算机学会出版社。
- 美国国家标准研究所。 (1986).命名和定义
照明工程。ANSI报告 (纽约)。 (ANSIIESRP-161986) 天使, E. (2002))。交互式计算机图形学:OpenGL的自上而下的方法(第三版)。阅读马:艾迪生-韦斯利。
- Appel, A. (1968) 。实体的着色机效果图的一些技术。
在AFIPS春季联合计算会议记录(Vol.32, 第37–45页)。阿菲普斯。Arvo, J. (1995) 。模拟光传输的解析方法 (未发表的博士论文)。Ashdown, I. (1994) 。Radiosity: 程序员的视角。行 峨○
- Ashikhmin, M. (2002) 。高对比度图像的色调映射算法。在 EGRW'02:第13届欧洲制图工作坊会议记录

- (pp. 145–155). Aire-la-Ville, Switzerland: Eurographics Association.
- Ashikhmin, M., Premože, 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.
- Baerentzen, J., & Christensen, N. (2002, May). Volume Sculpting Using the Level-Set Method. In *SMI '02: Proceedings of Shape Modeling International 2002 (SMI '02)* (pp. 175–182). Washington, DC: IEEE Computer Society Press.
- Barr, A. H. (1984). Global and Local Deformations of Solid Primitives. *Proc. SIGGRAPH '84 Computer Graphics*, 18(3), 21–30.
- 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.
- Barthe, L., Dodgson, N. A., Sabin, M. A., Wyvill, B., & Gaildrat, V. (2003). Two-dimensional Potential Fields for Advanced Implicit Modeling Operators. *Computer Graphics Forum*, 22(1), 23–33.
- Barthe, L., Mora, B., Dodgson, N. A., & Sabin, M. A. (2002). Interactive Implicit Modelling based on C1 Reconstruction of Regular Grids. *International Journal of Shape Modeling*, 8(2), 99–117.
- Baumgart, B. (1974, October). *Geometric Modeling for Computer Vision* (Tech. Rep. No. AIM-249). Palo Alto, CA: Stanford University AI Laboratory.
- 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.
- Berns, R. S. (2000). *Billmeyer and Saltzman's Principles of Color Technology* (3rd ed.). New York: John Wiley and Sons.
- Blinn, J. (1982). A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1(3), 235–258.
- Blinn, J. (1996). *Jim Blinn's Corner*. San Francisco, CA: Morgan Kaufmann.
- Blinn, J. F. (1976). Texture and Reflection in Computer Generated Images. *Communications of the ACM*, 19(10), 542–547.
- Bloomenthal, J. (1988). Polygonization of Implicit Surfaces. *Computer Aided Geometric Design*, 4(5), 341–355.
- Bloomenthal, J. (1990). Calculation of Reference Frames Along a Space Curve. In A. Glassner (Ed.), *Graphics Gems* (pp. 567–571). Boston: Academic Press.
- Bloomenthal, J. (1995). *Skeletal Design of Natural Forms* (Unpublished doctoral dissertation). University of Calgary, Canada.
- Bloomenthal, J. (1997). Bulge Elimination in Convolution Surfaces. *Computer Graphics Forum*, 16(1), 31–41.

- (第145–155页)。Aire-la-Ville, 瑞士: Eurographics Association. Ashikhmin, M., Premože, S., & Shirley, P. (2000)。一种基于微表面的BRDF Generator. 在SIGGRAPH的诉讼中 (第65–74页)。Reading, MA: Addison-Wesley Longman.
- Ashikhmin, M., & Shirley, P. (2000)。一个各向异性PhongBRDF模型。图形工具杂志, 5 (2) , 25–32。Baerentzen, J., & Christensen, N. (2002, 5月)。使用级设置的方法。在SMI'02:ProceedingsOfShapeModelingInternational2002(SMI'02)(pp.175–182)中。华盛顿特区: IEEE计算机学会出版社。
- 巴尔, A.H. (1984)。实体基元的全局和局部变形。Proc.SIGGRAPH'84计算机图形学, 18 (3) , 21–30。
- Bartels, R.H., Beatty, J.C., & Barsky, B.A. (1987)。介绍用于计算机图形学和几何建模的样条.旧金山, CA: 摩根考夫曼。
- Barthe, L., Dodgson, N.A., Sabin, M.A., Wyvill, B., & Gaildrat, V. (2003)。高级隐式建模算子的二维势场。计算机图形论坛, 22 (1) , 23–33。
- Barthe, L., Mora, B., Dodgson, N.A., & Sabin, M.A. (2002)。交互式隐式基于规则网格C1重建的建模。国际形状建模杂志, 8 (2) , 99–117。
- Baumgart, B. (1974, 10月)。计算机视觉几何建模 (技术)。代表号A IM-249)。加利福尼亚州帕洛阿尔托: 斯坦福大学人工智能实验室。
- Beck, K., & Andres, C. (2004)。极限编程解释: 拥抱变化 (第二版)。阅读马·艾迪生-韦斯利。
- Berlin, B., & Kay, P. (1969)。基本颜色术语: 它们的普遍性和进化性。坂ヤ医P 伯恩斯, R.S. (2000)。Billmeyer和Saltzman的色彩技术原理 (3rded.)。衍峨○Blinn, J. (1982)。代数曲面绘制的泛化。ACMTransactions on Graphics 1(3) 235–258.Blinn, J. (1996)。吉姆·布林的角落。旧金山, CA: 摩根考夫曼。
- Blinn, J.F. (1976)。计算机生成图像中的纹理和反射.Acm的communications, 19 (10) , 542–547。Bloomenthal, J. (1988)。隐曲面的多边形化。计算机辅助几何设计, 4 (5) , 341–355。
- Bloomenthal, J. (1990)。沿空间曲线计算参考帧。在A.Glassner (Ed.) , 图形宝石 (第567–571页)。波士顿: 学术Press.
- Bloomenthal, J. (1995)。自然形式的骨骼设计 (未发表的博士论文)。加拿大卡尔加里大学。Bloomenthal, J. (1997)。卷积曲面中的隆起消除。电脑图形论坛, 16 (1) , 31–41。

- Bloomenthal, J., & Shoemake, K. (1991). Convolution Surfaces. *Proc. SIGGRAPH '91, Computer Graphics*, 25(4), 251–257.
- Brandenburg, F. J. (1988). Nice Drawing of Graphs are Computationally Hard. In P. Gorney & M. J. Tauber (Eds.), *Visualization in Human-Computer Interaction* (pp. 1–15). Berlin: Springer-Verlag.
- Bresenham, J. E. (1965). Algorithm for Computer Control of a Digital Plotter. *IBM Systems Journal*, 4(1), 25–30.
- Brewer, C. A. (1999). Color Use Guidelines for Data Representation. In *Proc. Section on Statistical Graphics* (pp. 55–60). Alexandria, VA: American Statistical Association.
- Buchheim, C., Jünger, M., & Leipert, S. (2002). Improving Walker's Algorithm to Run in Linear Time. In *GD '02: Revised Papers from the 10th International Symposium on Graph Drawing* (pp. 344–353). London: Springer-Verlag.
- 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, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., & Evans, T. R. (2001). Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 67–76). 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.

- Bloomenthal, J., & Shoemake, K. (1991). 卷积表面。 Proc.SIGGRAPH '91, 计算机图形学, 25 (4) , 251–257。
- 勃兰登堡, F.J. (1988)。漂亮的图形绘制在计算上很难。在P.Gorney & M.J.Tauber (Eds。) , 可视化在人机在teraction (第1–15页) 。柏林: Springer-Verlag。 Bresenham, J.E. (1965)。数字绘图仪计算机控制算法.IBMSystemsJournal, 4 (1) , 25–30。
- 布鲁尔, C.A. (1999)。数据表示的颜色使用准则。在Proc. 统计图形一节(第55–60页)。弗吉尼亚州亚历山大: 美国统计协会。
- Buchheim, C., Junger, M., & Leipert, S. (2002)。改进Walker的算法以线性时间运行。在GD'02:第十届国际绘图研讨会修订论文(第344–353页)。伦敦: SpringerVerlag。
- Campagna, S., Kobbelt, L., & Seidel, H.-P. (1998)。有向边-三角形网格的可缩放表示。图形工具杂志, 3 (4) , 1–12。 Carr, J.C., Beatson, R.K., Cherrie, J.B., Mitchell, T.J., Fright, W.R., McCallum, B.C., & Evans, T.R. (2001)。具有径向基函数的3d对象的重建和表示。在SIGGRAPH'01:ProceedingsOfthe28thAnnualConferenceonComputerGraphicsandInteractiveTechniques(pp.67–76)中。 Chiu, K., Herf, M., Shirley, P., Swamy, S., Wang, C., & Zimmerman, K. (1993)。用于高对比度图像的空间不均匀缩放功能。在ProceedingsOfGraphicsInterface'93(pp.245–253)中。 韦尔斯利 马:一个K彼得斯和加拿大人机通信协会。
- Choudhury, P., & Tumblin, J. (2003)。高对比度的三边滤波器图像和网格。在EGRW'03:第14届欧洲制图讲习班会议记录(第186–196页)。 Aire-la-Ville, 瑞士: 欧洲图形协会。 Cleary, J., Wyvill, B., Birtwistle, G., & Vatti, R. (1983)。平行光线追踪
- 计算机。在Simula用户协会会议记录(第77–80页)。 Cohen, E., Riesenfeld, R.F., & Elber, G. (2001)。用样条进行几何建模:介绍.韦尔斯利, MA : AK彼得斯。
- Cohen, M.F., Chen, S.E., Wallace, J.R., & Greenberg, D.P. (1988)。一种快速辐射图像生成的先进细化方法.InSIGGRAPH'88:ProceedingsOfthe15thAnnualConferenceonComputerGraphicsandInteractiveTechniques(pp.75–84)。 Cohen, M.F., & Wallace, J.R. (1993)。辐射度和逼真的图像合成。剑桥, MA: AcademicPress, Inc.
- Comaniciu, D., & Meer, P. (2002)。MeanShift: 实现有限元空间分析的稳健方法。 IEEE TransactionsonPatternAnalysisandMachineIntelligence 24(5) 603–619.

- Cook, R. L., Carpenter, L., & Catmull, E. (1987). The Reyes Image Rendering Architecture. *Proc. SIGGRAPH '87 Computer Graphics*, 21(4), 95–102.
- Cook, R. L., Porter, T., & Carpenter, L. (1984). Distributed Ray Tracing. *Proc. SIGGRAPH '84, Computer Graphics*, 18(3), 137–145.
- Cook, R. L., & Torrance, K. E. (1982). A Reflectance Model for Computer Graphics. *ACM Transactions on Graphics*, 1(1), 7–24.
- 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.
- da Vinci, L. (1970). *The Notebooks of Leonardo da Vinci* (Vol. 1). Mineola, NY: Dover Press.
- Dachsbacher, C., Vogelsgang, C., & Stamminger, M. (2003). Sequential Point Trees. *ACM Transactions on Graphics*, (Proc. SIGGRAPH 03), 22(3), 657–662.
- 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.
- deGroot, E., & Wyvill, B. (2005). Rayskip: Faster Ray Tracing of Implicit Surface Animations. In *GRAPHITE '05: Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia* (pp. 31–36). New York: ACM Press.
- DeRose, T. (1989). *A Coordinate-Free Approach to Geometric Programming* (Tech. Rep. No. 89-09-16). Seattle, WA: University of Washington.
- Di Battista, G., Eades, P., Tamassia, R., & Tollis, I. G. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. Englewood Cliffs, NJ: Prentice Hall.
- Dinh, H., Slabaugh, G., & Turk, G. (2001). Reconstructing Surfaces Using Anisotropic Basis Functions. In *International Conference on Computer Vision (ICCV) 2001* (pp. 606–613). Washington, DC: IEEE.
- 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.

- Cook, R.L., Carpenter, L., & Catmull, E. (1987) 。的雷耶斯图像渲染架构。 Proc.SIGGRAPH'87ComputerGraphics 21(4) 95–102.
- Cook, R.L., Porter, T., & Carpenter, L. (1984) 。分布式光线追踪。 Proc.SIGGRAPH'84, 计算机图形学, 18 (3) , 137–145。Cook, R.L., & Torrance, K.E. (1982) 。计算机图形学的反射率模型。 AcmTransactions on Graphics, 1 (1) , 7 –24。

Crow, F.C. (1978) 。使用灰度来改进矢量和字符的光栅显示。在SIGGRAPH'78: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques (第1–5页) 中。 呐

Crowe, M.J. (1994) 。矢量分析的历史。米尼奥拉, 纽约: 多佛。达芬奇, L. (1970) 。列奥纳多·达·芬奇的笔记本。1).Mineola, NY: 多佛出版社。

Dachsbaucher, C., Vogelsgang, C., & Stamminger, M. (2003) 。顺序点树。 AcmTransactions on Graphics (Proc.[医]SIGGRAPH03) 22(3) 657 662.D ebevec, P.E., & Malik, J. (1997) 。从照片中恢复高动态范围Radiance 地图。在SIGGRAPH'97:Proceedings Of the 24th Annual Conference on Computer Graphics and Interactive Techniques(pp.369–378)中。阅读 马·艾迪生-韦斯利.DeBoor, C. (1978) 。样条的实用指南。柏林: Springer-Verlag 。 DeBoor, C. (2001) 。样条的实用指南。柏林: Springer-Verlag。 deG root, E., & Wyvill, B. (2005) 。Rayskip: 更快的光线追踪隐式Sur面部动画。在GRAPHITE'05:Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia(pp.31–36)。 呐DeRose, T. (1989) 。一种无坐标的几何编程方法 (技术) 。众议员No.89-09-16)。华盛顿州西雅图: 华盛顿大学。 Di Battista, G., Eades, P., Tamassia, R., & Tollis, I.G. (1999) 。图形绘制:

图形可视化的算法。恩格尔伍悬崖, 新泽西州: 普伦蒂斯 Hall.

Dinh, H., Slabaugh, G., & Turk, G. (2001)。 使用重建表面各向异性基函数。国际计算会议 Vision (ICCV) 2001 (第606–613页)。华盛顿特区: IEEE。

Dobkin, D.P., & Mitchell, D.P. (1993) 。Supersampling模式的随机-边缘差异。在Proceedings Of Graphics Interface (第62–69页) 中。韦尔斯利马:一个K彼得斯和加拿大人机交流协会.Dooley, D., & Cohen, M.F. (1990) 。3d几何模型的自动插图els: 线。 In SI3D'90: Proceedings Of The 1990 Symposium on Interactive 3D Graphics(pp.77–82)。 呐Doran, C. , & Lasenby, A. (2003) 。物理学家的几何代数。剑桥

英国: 剑桥大学出版社。



- Drago, F., Myszkowski, K., Annen, T., & Chiba, N. (2003). Adaptive Logarithmic Mapping for Displaying High Contrast Scenes. *Computer Graphics Forum*, 22(3), 419–426.
- Duchon, J. (1977). Constructive Theory of Functions of Several Variables. In (pp. 85–100). Berlin: Springer-Verlag.
- 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.
- Dwyer, T., Koren, Y., & Marriott, K. (2006). IPsep-CoLa: An Incremental Procedure for Separation Constraint Layout of Graphs. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 06)*, 12(5), 821–828.
- Eberly, D. (2000). *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*. San Francisco, CA: Morgan Kaufmann.
- Eckman, P., & Friesen, W. V. (1978). *Facial Action Coding System*. Palo Alto, CA: Consulting Psychologists Press.
- Eick, S. G., Steffen, J. L., & Sumner, E. E., Jr. (1992). Seesoft-A Tool for Visualizing Line Oriented Software Statistics. *IEEE Trans. Software Eng.*, 18(11), 957–968.
- 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 10th 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.
- Fekete, J.-D., & Plaisant, C. (2002). Interactive Information Visualization of a Million Items. In *Proc. IEEE Symposium on Information Visualization (InfoVis 02)* (pp. 117–124). Washington, DC: IEEE Computer Society Press.
- Ferwerda, J. A., Pattanaik, S., Shirley, P., & Greenberg, D. P. (1996). A Model of Visual Adaptation for Realistic Image Synthesis. In *SIGGRAPH '96*:



- Drago, F., Myszkowski, K., Annen, T., & Chiba, N. (2003). 自适应对数用于显示高对比度场景的映射。计算机图形论坛, 22 (3) , 419–426。
- Duchon, J. (1977). 几个变量的函数的建设性理论。在 (第85–100页)。柏林: Springer-Verlag. Durand, F., & Dorsey, J. (2002). 用于显示高动态范围图像的快速双边滤波。AcmTransactions on Graphics, 21 (3) , 257–266。
- Dutre, P., Bala, K., & Bekaert, P. (2002)。高级全局照明。Wellesley, MA: AKPeters. Dwyer, T., Koren, Y., & Marriott, K. (2006)。IP Sep-CoLa: 用于图的分离约束布局的增量Procedure。IEEETrans.可视化和计算机图形学(Proc.[医]资讯06) 12(5) 821–828.Eberly, D. (2000)。3D 游戏引擎设计: 实时计算机图形学的实用方法。旧金山, CA: 摩根考夫曼。
- Eckman, P., & Friesen, W.V. (1978)。面部动作编码系统。帕洛阿尔托 CA: 咨询心理学家出版社。
- Eick, S.G., Steffen, J.L., & Sumner, E.E., Jr. (1992)。Seesoft-用于可视化面向行的软件统计的工具。IEEETrans.软件工程。18(11) 957 968.Erov, S., Kolchin, K., & Myszkowski, K. (2001)。基于油漆成分建模的珠光Ap珠光渲染.计算机图形Fo朗姆酒 20(3) 227–238.Fairchild, 医学博士 (2005)。颜色外观模型 (第二版)。).
- Fairchild, M.D., & Johnson, G.M. (2002)。满足iCAM: 图像颜色Appearance 模型。在IS & TSID第10届彩色成像会议 (第33–38页) 中。弗吉尼亚州斯普林菲尔德: 成像科学与技术学会。
- Fairchild, M.D., & Johnson, G.M. (2004)。图像的iCAM框架 外观、图像差异和图像质量。电子杂志 Imaging, 13, 126–138.
- Farin, G. (2002).CAGD的曲线和曲面: 实用指南。 San 弗朗西斯科, CA: 摩根考夫曼。
- Farin, G., & Hansford, D. (2004)。实用线性代数: 一个几何工具箱。韦尔斯利, MA: AK彼得斯。Farin, G., Hoschek, J., & Kim, M.-S. (Ed s.) (2002).计算机辅助几何设计手册.阿姆斯特丹: 爱思唯尔。
- Fattal, R., Lischinski, D., & Werman, M. (2002)。梯度域高动态范围压缩。A CMTransactions on Graphics, 21 (3) , 249–256。
- Fekete, J.-D., & Plaisant, C. (2002)。交互式信息可视化 万项。在Proc. IEEE信息可视化研讨会 (InfoVis02) (第117–124页)。华盛顿特区: IEEE计算机学会出版社。Ferwerda, J.A., Pattanaik, S., Shirley, P., & Greenberg, D.P. (1996)。逼真的图像合成的视觉适应的模型。在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*. Englewood Cliffs, NJ: Prentice Hall.
- Francis S. Hill, J. (2000). *Computer Graphics Using OpenGL* (Second ed.). Englewood Cliffs, NJ: Prentice Hall.
- Frick, A., Ludwig, A., & Mehldau, H. (1994). A Fast Adaptive Layout Algorithm for Undirected Graphs. In *GD '94: Proceedings of the DIMACS International Workshop on Graph Drawing* (pp. 388–403). London: Springer-Verlag.
- Friendly, M. (2008). A Brief History of Data Visualization. In *Handbook of Data Visualization* (pp. 15–56). (Web document, <http://www.math.yorku.ca/SCS/Gallery/milestone/>)
- Friskin, S., Perry, R., Rockwood, A., & Jones, T. (2000). Adaptively Sampled Distance Fields. In *Siggraph '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 249–254). New York: ACM Press/Addison-Wesley Publishing Co.
- Fua, Y.-H., Ward, M. O., & Rundensteiner, E. A. (1999). Hierarchical Parallel Coordinates for Exploration of Large Datasets. In *Proc. IEEE Visualization Conference (Vis '99)* (pp. 43–50). Washington, DC: IEEE Computer Society Press.
- Fujimoto, A., Tanaka, T., & Iwata, K. (1986). ARTScelerated Ray-Tracing System. *IEEE Computer Graphics & Applications*, 6(4), 16–26.
- Galin, E., & Akkouche, S. (1999). Incremental Polygonization of Implicit Surfaces. *Graphical Models*, 62(1), 19–39.
- Gansner, E. R., Koutsofios, E., North, S. C., & Vo, K.-P. (1993, March). A Technique for Drawing Directed Graphs. *IEEE Transactions on Software Engineering*, 19(3), 214–229.
- Gascuel, M.-P. (1993, Aug). An Implicit Formulation for Precise Contact Modeling Between Flexible Solids. In *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 313–320). 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., ... Economou, E. (1999). An Anchoring Theory of Lightness Perception.

- 第23届计算机图形学及
互动技术（第249–258页）。呐
Ferwerda, J.A., Shirley, P., Pattanaik, S.N., & Greenberg, D.P. (1997)。A
计算机图形视觉掩蔽模型.
第24届计算机图形学及
互动技术（第143–152页）。阅读 马·艾迪生-韦斯利。
Foley, J.D., VanDam, A., Feiner, S.K., & Hughes, J.F. (1990)。电脑
图形：原则和实践（第二版。).阅读，马： 艾迪生
Wesley.
Forsyth, D.A., & Ponce, J. (2002)。计算机视觉：现代方法。Englewo
odsCliffs, NJ: PrenticeHall。FrancisS.Hill, J. (2000)。使用OpenGL的
计算机图形学（第二版）。).恩格尔伍悬崖 NJ:普伦蒂斯霍尔.
Frick, A., Ludwig, A., & Mehldau, H. (1994)。无向图的快速自适应
布局算法。在GD'94:DIMACS国际绘图讲习班会议记录(第388–403页)。伦
敦：SpringerVerlag.
Friendly, M. (2008). 数据可视化简史。
In
数据可视化手册 (第15–56页)。
(Web document,
<http://www.math.yorku.ca/SCS/Gallery/milestone/>) Friskin, S., Perry, R., Rockwood, A.,
& Jones, T. (2000)。自适应采样
距离场。在Siggraph'00: Proceedingsofthe27thAnnualConferenceonComp
uterGraphicsandInteractiveTechniques (第249–254页) 中。
雁 扈 医 帅 九 & 缪
Fua, Y.-H., Ward, M.O., & Rundensteiner, E.A. (1999)。分层并行
用于大型数据集探索的坐标。在Proc. IEEEVisualizationConference(Vis'99)(第43–50页)。华盛顿特区：IEEE计算机学会出版社。
Fujimoto, A., Tanaka, T., & Iwata, K. (1986)。自动射线追踪系统。IEEE计算
机图形与应用, 6 (4) , 16–26。
Galin, E., & Akkouche, S. (1999)。隐式Sur面的增量多边形化。图形模型, 62
(1) , 19–39。Gansner, E.R., Koutsofios, E., North, S.C., & Vo, K.-P. (199
3, 3月)。A
绘制有向图的技术。IEEE软件交
Engineering, 19(3), 214–229.
Gascuel, M.-P. (1993年8月)。柔性固体之间精确接触模型的隐式公式。
在SIGGRAPH'93:ProceedingsOfthe20thAnnualConferenceonComputerGr
aphicsandInteractiveTechniques(pp.313–320)中。 呐吉布森
, J.J. (1950)。视觉世界的感知。剑桥, MA: 河滨出版社。
Gilchrist, A.L., Kossyfidis, C., Bonato, F., Agostini, T., Cataliotti, J., Li, X.,
... Economou, E. (1999)。明度感知的锚定理论。

- 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.
- 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. *Proc. SIGGRAPH '84, Computer Graphics*, 18(3), 213–222.
- Gouraud, H. (1971). Continuous Shading of Curved Surfaces. *Communications of the ACM*, 18(6), 623–629.
- Grassmann, H. (1853). Zur Theorie der Farbenmischung. *Annalen der Physik und Chemie*, 89, 69–84.
- Gregory, R. L. (1997). *Eye and Brain: The Psychology of Seeing* (Fifth ed.). Princeton, NJ: Princeton University Press.
- Grosjean, J., Plaisant, C., & Bederson, B. (2002). SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation. In *Proc. IEEE Symposium on Information Visualization (InfoVis)* (pp. 57–64). Washington, DC: IEEE Computer Society Press.
- Grossman, T., Wigdor, D., & Balakrishnan, R. (2007). Exploring and Reducing the Effects of Orientation on Text Readability in Volumetric Displays. In *Proc. ACM Conf. Human Factors in Computing Systems (CHI)* (pp. 483–492). New York: ACM 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.
- Hanson, A. J. (2005). *Visualizing Quaternions*. San Francisco, CA: Morgan Kaufmann.

- 心理评论, 106 (4) , 795–834。格拉斯纳, A. (1984)。用于快速光线追踪的空间细分。IEEE计算机图形与应用, 4 (10) , 15–22。格拉斯纳, A. (1988)。用于动画的时空光线追踪。IEEE计算机图形与应用, 8 (2) , 60–70。
- 格拉斯纳, A. (Ed.) (1989). 光线追踪简介。伦敦: 学术出版社。
- Glassner, A. (1995)。数字图像合成的原理。旧金山, CA: 摩根考夫曼。
- Goldman, R. (1985)。矢量代数中的非法表达式。AcmTransactions on Graphics , 4 (3) , 223–243. Goldsmith, J., & Salmon, J. (1987)。自动创建用于光线追踪的对象层次结构。IEEE计算机图形与应用, 7 (5) , 14–20。Gooch, A., Gooch, B., Shirley, P., & Cohen, E. (1998)。非真实的
用于自动技术图示的照明模型。在SIGGRAPH'98: 第25届计算机图形学年会论文集和
互动技术 (第447–452页)。
- Goral, C.M., Torrance, K.E., Greenberg, D.P., & Battaile, B. (1984)。模拟光在漫射表面之间的相互作用。Proc.SIGGRAPH'84 ComputerGraphics 18(3) 213–222.
- Gouraud, H. (1971)。曲面的连续遮光。Acm的通信, 18 (6) , 623–629。Grassmann, H. (1853)。ZurTheoriedeFarbenmischung. Annalen derPhysikundChemie, 89, 69–84。格雷戈里, R.L. (1997)。眼睛和大脑: 看到的心理学 (第五版)。普林斯顿, 新泽西州: 普林斯顿大学出版社。
- Grosjean, J., Plaisant, C., & Bederson, B. (2002)。空间树: 支持大节点链接树的解释配给设计进化和经验评估。在Proc. IEEE信息可视化研讨会 (InfoVis) (第57–64页)。华盛顿特区: IEEE计算机学会出版社。Grossman, T., Wigdor, D., & Balakrishnan, R. (2007)。探索和减少方向对体积显示中的文本可读性的影响。在Proc. ACMConf.计算系统中的人为因素 (CHI) (第483–492页)。
- Hammersley, J., & Handscomb, D. (1964)。蒙特卡罗方法。伦敦: Methuen。
- Hanrahan, P., & Lawson, J. (1990)。阴影和照明Calculations的语言。在SIGGRAPH'90: Proceedings Of The 17th Annual Conference on Computer Graphics and Interactive Techniques (pp. 289–298) 中。纽约:
ACM出版社。Hanson, A.J. (2005)。可视化四元数。旧金山, CA: 摩根
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.
- Hart, J. C., & Baker, B. (1996, Oct). Implicit Modeling of Tree Surfaces. In *Proceedings of Implicit Surfaces '96* (pp. 143–152). Aire-la-Ville, Switzerland: Eurographics Association.
- Hartmann, E. (1998). A Marching Method for the Triangulation of Surfaces. *The Visual Computer*, 14(3), 95–108.
- 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. *Proc. SIGGRAPH '92, Computer Graphics*, 26(2), 253–254.
- Hearn, D., & Baker, M. P. (1986). *Computer Graphics*. Englewood Cliffs, N.J.: Prentice Hall.
- Heer, J., & Robertson, G. (2007). Animated Transitions in Statistical Data Graphics. *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis07)*, 13(6), 1240–1247.
- 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.
- Henry, N., & Fekete, J.-D. (2006). MatrixExplorer: a Dual-Representation System to Explore Social Networks. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 06)*, 12(5), 677–684.
- Hoffmann, B. (1975). *About Vectors*. Mineola, NY: Dover.
- Hofstadter, D. (1979). *Gödel, Escher, Bach: an Eternal Golden Braid*. New York: Basic Books.
- 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). University of Washington.
- 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* (6th ed.). Chichester, UK: John Wiley and Sons.
- IEEE Standards Association. (1985). *IEEE Standard for Binary Floating-Point Arithmetic* (Tech. Rep.). New York: IEEE Report. (ANSI/IEEE Std 754-1985)

- 哈里斯, M.J. (2004)。GPU上的快速流体动力学仿真。在GPU中Gems:实时图形的编程技术、技巧和技巧(第三章).38).阅读 马:艾迪生-韦斯利.Hart, J.C., & Baker, B. (1996, Oct)。树表面的隐式建模。InProceedingsOfImplicitSurfaces'96 (第143–152页)。Aire-la-Ville, 瑞士: EurographicsAssociation。
- Hartmann, E. (1998)。面三角测量的行进方法。视觉计算机, 14 (3), 95–108。
- Hausner, M. (1998)。几何的矢量空间方法.米尼奥拉, 纽约: 多佛。
- Havran, V. (2000)。启发式射线拍摄算法 (未发表的博士论文)。布拉格捷克技术大学。他, X.D., Heynen, P.O., Phillips, R.L., Torrance, K.E., Salesin, D.H. & 格林伯格, D.P. (1992)。快速准确的光反射模型。 Proc.SIGGRAPH'92 ComputerGraphics 26(2) 253–254.
- Hearn, D., & Baker, M.P. (1986)。计算机图形学。EnglewoodCliffs, 新泽西州: PrenticeHall。
- Heer, J., & Robertson, G. (2007)。统计数据图ics中的动画转换。 IEEETrans.关于可视化和计算机图形学(Proc.资料 Vis07), 13(6), 1240–1247.
- Heidrich, W., & Seidel, H.-P. (1998)。光线追踪程序位移着色器。在ProceedingsOfGraphicsInterface (第8–16页) 中。韦尔斯利, 妈: 一个K彼得斯和加拿大人机通信协会。
- Henry, N., & Fekete, J.-D. (2006)。MatrixExplorer: 探索社交网络的双重表示系统。IEEETrans.可视化和计算机图形学(Proc.[医]资讯06) 12(5) 677–684.
- 霍夫曼, B. (1975)。关于向量。米尼奥拉, 纽约: 多佛。
- Hofstadter, D. (1979)。戈德尔, 埃舍尔, 巴赫: 永恒的金色辫子。纽约: 基本书籍。
- Hood, D.C., Finkelstein, M.A., & Buckingham, E. (1979)。响应函数模型的心理物理测试。视觉研究, 19, 401–406. Hoppe, H. (1994)。从无组织的点进行表面重建 (Unpublished博士论文)。华盛顿大学。霍恩, B.K.P. (1974)。从图像确定亮度。CVGIP, 3, 277–299. Hughes, J.F., & Möller, T. (1999)。从单位向量构建正交基。图形工具杂志, 4 (4) , 33–35。
- 亨特, R.W.G. (2004)。色彩的再现 (第6版。).英国奇切斯特: 约翰威利和儿子。
- IEEE标准协会。 (1985).二进制浮点IEEE标准 算术 (技术。众议员)。纽约: IEEE报告。(ANSIIEEE标准7541985)

Igarashi, T., Matsuoka, S., & Tanaka, H. (1999). Teddy: A Sketching Interface for 3D Freeform Design. In *Siggraph'99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 409–416). New York: ACM Press/Addison-Wesley Publishing Co.

Immel, D. S., Cohen, M. F., & Greenberg, D. P. (1986). A Radiosity Method for Non-Diffuse Environments. *Proc. SIGGRAPH '86, Computer Graphics*, 20(4), 133–142.

Ingram, S., Munzner, T., & Olano, M. (2009). Glimmer: Multilevel MDS on the GPU. *IEEE Trans. Visualization and Computer Graphics*, 15(2), 249–261.

Inselberg, A., & Dimsdale, B. (1990). Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry. In *Vis '90: Proceedings of the 1st Conference on Visualization '90*. Los Alamitos, CA: IEEE Computer Society Press.

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.

Johansson, G. (1973). Visual Perception of Biological Motion and a Model for Its Analysis. *Perception & Psychophysics*, 14, 201–211.

Johnson, B., & Shneiderman, B. (1991). Treemaps: A Space-filling Approach to the Visualization of Hierarchical Information. In *VIS '91: Proceedings of the 2nd Conference on Visualization '91* (pp. 284–291). Los Alamitos, CA: IEEE Computer Society Press.

Johnson, G. M., & Fairchild, M. D. (2003). Rendering HDR Images. In *IS&T/SID 11th Color Imaging Conference* (pp. 36–41). Springfield, VA: Society for Imaging Science & Technology.

Jones, J. A., Harrold, M. J., & Stasko, J. (2002). Visualization of Test Information to Assist Fault Localization. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering* (pp. 467–477). New York: ACM Press.

Judd, D. B. (1932). Chromaticity Sensibility to Stimulus Differences. *Journal of the Optical Society of America*, 22, 72–108.

Kainz, F., Bogart, R., & Hess, D. (2003). The OpenEXR Image File Format. In *SIGGRAPH Technical Sketches*. New York: ACM Press. (see also: <http://www.openexr.com/>)

Kajiya, J. T. (1986). The Rendering Equation. *Proc SIGGRAPH '86 Computer Graphics*, 20(4), 143–150.

Kalos, M., & Whitlock, P. (1986). *Monte Carlo Methods, Basics*. New York: Wiley-Interscience.

Kalra, D., & Barr, A. (1989, July). Guaranteed Ray Intersections with Implicit Functions. *Computer Graphics (Proc. SIGGRAPH 89)*, 23(3), 297–306.

Igarashi, T., Matsuoka, S., & Tanaka, H. (1999)。Teddy: 用于3d自由曲面设计的草图绘制界面。在Siggraph'99: 第26届计算机图形学和交互技术年会论文集（第409–416页）中。
Immel, D.S., Cohen, M.F., & Greenberg, D.P. (1986)。一种辐射度方法
非漫射环境。Proc.SIGGRAPH'86 ComputerGraphics 20(4) 133–142.
Ingram, S., Munzner, T., & Olano, M. (2009)。微光：GPU上的多级MDS。
IEEETrans.可视化与计算机图形学, 15 (2) , 249–261.
Inselberg, A., & Dimsdale, B. (1990)。平行坐标：用于可视化多维几何的工具。在Vis'90：第一届可视化会议记录'90。LosAlamitos, CA: IEEEComputerSocietyPress.
Jansen, F.W. (1986)。光线追踪的数据结构。在法律程序中
讲习班Eurographics研讨会上的数据结构的光栅图形(pp.57–73)。
Jensen, H.W. (2001)。使用光子映射的逼真图像合成。Wellesley, MA: AKPeters.
Jensen, G. (1973)。生物运动的视觉感知及其分析模型.感知与心理物理学, 14, 201–211。
Johnson, B., & Shneiderman, B. (1991)。Treemaps: 分层信息可视化的空间填充方法。在VIS'91：第二次可视化会议记录'91（第284–291页）中。洛斯阿拉米托斯, CA: IEEE计算机学会出版社。
Johnson, G.M., & Fairchild, M.D. (2003)。渲染HDR图像。在IS & SID第11届彩色成像会议（第36–41页）中。弗吉尼亚州斯普林菲尔德：成像科学与技术学会。
Jones, J.A., Harrold, M.J., & Stasko, J. (2002)。测试信息的可视化，以协助故障定位。在ICSE'02：第24届国际软件工程会议记录（第467–477页）。
贾德, D.B. (1932)。色度对刺激差异的敏感性。美国光学学会杂志, 22, 72–108。Kainz, F., Bogart, R., & Hess, D. (2003)。的OpenEXR图像文件格式。
在SIGGRAPH技术草图。 (另请参阅：<http://www.openexr.com/>) Kajiya, J.T. (1986)。的渲染方程。ProcSIGGRAPH'86ComputerGraphics 20(4) 143–150.
Kalos, M., & Whitlock, P. (1986)。蒙特卡罗方法，基础。
Kalra, D., & Barr, A. (1989, 7月)。具有隐式的保证射线相交函数。计算机图形学 (Proc.[医]SIGGRAPH89) 23(3) 297–306.

- Kay, D. S., & Greenberg, D. (1979). Transparency for Computer Synthesized Images. *Proc. SIGGRAPH '79 Computer Graphics*, 13(2), 158–164.
- 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., 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*. Melbourne, Australia: Australian Computer Graphics Association.
- 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 Interdisciplinary 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.
- Kovitz, B. L. (1999). *Practical Software Requirements: A Manual of Content & Style*. New York: Manning.
- 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.
- Laramee, R. S., Hauser, H., Doleisch, H., Vrolijk, B., Post, F. H., & Weiskopf, D. (2004). The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. *Computer Graphics Forum*, 23(2), 203–221.
- Lasseter, J. (1987). Principles of Traditional Animation Applied to 3D Computer Animation. *Proc. SIGGRAPH '87, Computer Graphics*, 21(4), 35–44.
- Lawrence, J., Rusinkiewicz, S., & Ramamoorthi, R. (2004). Efficient BRDF Importance Sampling Using a Factored Representation. *ACM Transactions on Graphics (Proc. SIGGRAPH '04)*, 23(3), 496–505.
- Lee, D. N., & Reddish, P. (1981). Plummeting Gannets: A Paradigm of Ecological Optics. *Nature*, 293, 293–294.
- 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.
- Lewis, C., & Rieman, J. (1993). *Task-Centered User Interface Design: A Practical Introduction*. <http://hcibib.org/tcuid/>.
- Lewis, R. R. (1994). Making Shaders More Physically Plausible. *Computer Graphics Forum*, 13(2), 109–120.

- Kay, D.S., & Greenberg, D. (1979)。计算机合成图像的透明度。Proc.SIGGRA PH'79计算机图形学, 13 (2) , 158–164。
- Kernighan, B.W., & Pike, R. (1999)。编程的实践。阅读 马:艾迪生-韦斯利。
- Kersten, D., Mamassian, P., & Knill, D.C. (1997)。移动的投射阴影在深度上引起明显的运动感知, 26 (2) , 171–192。
- Kindlmann, G., Weinstein, D., & Hart, D. (2000)。直接交易量的策略扩散张量场的渲染。IEEE TransactionsonVisualizationandComputerGraphi cs 6(2) 124–138.Kirk, D., & Arvo, J. (1988)。光线追踪内核。在Aus图的程序中.澳大利亚墨尔本：澳大利亚计算机图形协会。Klatzky, R.L. (1998)。以自我为中心和以自我为中心的空间表示:去中心、区别和相互联系。
- K.F.Wender (Eds。) , 空间认知-空间知识表示和处理的跨学科方法 (Vol. 5, 第1-17页)。
- Berlin: Springer-Verlag.
- Knill, D.C. (1998)。纹理表面定向：理想的观察者，通用的观察者和纹理线索的信息内容。视觉研究, 38, 1655–1682。Kollig, T., & Keller, A. (2002)。高效的多维采样。计算机图形论坛, 21 (3) , 557–564。
- Kovitz, B.L. (1999)。实用软件要求：内容手册& 风格。
- Lafort (1997).反射率函数的非线性近似。在 SIGGRAPH '97 (pp. 117–126). Reading, MA: Addison-Wesley.
- Laramee, R.S., Hauser, H., Doleisch, H., Vrolijk, B., Post, F.H., & Weiskopf D. (2004)。流动可视化技术的现状：密集和纹理 基于技术。计算机图形论坛, 23 (2) , 203–221。
- Lasseter, J. (1987)。传统动画应用于3D计算机动画的原理。Proc.SIGGRAPH'87 , 计算机图形学, 21 (4) , 35–44。
- Lawrence, J., Rusinkiewicz, S., & Ramamoorthi, R. (2004)。高效BRDF 使用因子表示的重要性抽样。图形上的ACM交易(Proc.[医]SIGGRAPH'04) 2 3(3) 496–505.Lee, D.N., & Reddish, P. (1981)。直线下降的塘鹅：生态光学的典范。自然, 293, 293–294。Leung, T., & Malik, J. (1997)。关于垂直纹理：为什么我们看到更多
- 远处的花？在Proc. IEEE计算机视觉和模式识别会议（第807–813页）。洛斯阿拉米托斯, CA: IEEE出版社。Lewis, C., & Rieman, J. (1993)。以任务为中心的用户界面设计：一个实用的cal介绍。<http://hcibib.org/tcuid/>。刘易斯, R.R. (1994)。使着色器在物理上更合理。电脑图形论坛, 13 (2) , 109–120。

- Loop, C. (2000). *Managing Adjacency in Triangular Meshes* (Tech. Rep. No. MSR-TR-2000-24). Bellingham, WA: Microsoft Research.
- Lorensen, W., & Cline, H. (1987). Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4), 163–169.
- Luboschik, M., Schumann, H., & Cords, H. (2008). Particle-Based Labeling: Fast Point-Feature Labeling without Obscuring Other Visual Features. *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis08)*, 14(6), 1237–1244.
- MacEachren, A., Dai, X., Hardisty, F., Guo, D., & Lengerich, G. (2003). Exploring High-D Spaces with Multiform Matrices and Small Multiples. In *Proc. IEEE Symposium on Information Visualization (InfoVis)* (pp. 31–38). Washington, DC: IEEE Computer Society Press.
- Mackinlay, J. (1986). Automating the Design of Graphical Presentations of Relational Information. *ACM Trans. on Graphics (TOG)*, 5(2), 110–141.
- Malley, T. (1988). *A Shading Method for Computer Generated Images* (Unpublished master's thesis). Computer Science Department, University of Utah.
- Marschner, S. R., & Lobb, R. J. (1994, Oct). An Evaluation of Reconstruction Filters for Volume Rendering. In *VIS '94: Proceedings of the Conference on Visualization '94* (pp. 100–107). Washington, DC: IEEE Computer Society Press.
- 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 (Proc. SIGGRAPH '03)*, 22(3), 759–769.
- McLoughlin, T., Laramee, R. S., Peikert, R., Post, F. H., & Chen, M. (2009). Over Two Decades of Integration-Based Geometric Flow Visualization. In *Proc. Eurographics 2009, State of the Art Reports*. Aire-la-Ville, Switzerland: Eurographics Association.
- 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. (1990, May). Robust Ray Intersection with Interval Arithmetic. In *Graphics interface '90* (pp. 68–74). Wellesley, MA: Canadian Human-Computer Communications Society & A K Peters.
- 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.

循环, C. (2000)。管理三角形网格中的邻接 (技术)。代表号MSR-TR-2000-24)。华盛顿州贝灵汉:微软研究院。

Lorensen, W., & Cline, H. (1987)。行进立方体:高分辨率3D面构建算法。计算机图形学 (Proc.[医]SIGGRAPH87) 21(4) 163–169. Luboschik, M., Schumann, H., & Cords, H. (2008)。基于粒子的标记:快速

点特征标注而不遮挡其他视觉特征。IEEETrans.关于可视化和计算机图形学 (Proc.InfoVis08), 14(6), 1237–1244。 MacEachren, A., Dai, X., Hardisty, F., Guo, D., & Lengerich, G. (2003)。用多格式矩阵和小倍数绘制高D空间。在Proc. IEEE信息可视化研讨会 (InfoVis) (第31–38页)。

华盛顿特区: IEEE计算机学会出版社。

Mackinlay, J. (1986)。自动设计重新信息的图形演示.ACM反式。关于图形 (TOG) , 5 (2) , 110–141。 Malley, T. (1988)。计算机生成图像的遮光方法 (Unpublished硕士论文)。犹他大学计算机科学系。 Marschner, S.R., & Lobb, R.J. (1994, Oct)。对体积渲染的重建Filters的评估.在VIS'94: 可视化会议记录'94 (第100–107页)。华盛顿特区: IEEE计算机协会

Press.

Marshall, J.A., Burbeck, C.A., Arely, D., Rolland, J.P., & Martin, K.E. (1999)。遮挡边缘模糊:相对视觉深度的提示。香港邮政美国光学学会A, 13, 681–688。

Matusik, W., Pfister, H., Brand, M., & McMillan, L. (2003)。数据驱动反射率模型。图形上的ACM交易(Proc.[医]SIGGRAPH'03) 22(3) 759–769. McLoughlin, T., Laramee, R.S., Peikert, R., Post, F.H., & Chen, M. (2009)。结束

二十年的基于集成的几何流可视化.在Proc. Eurographics2009, 国家的艺术报告。Aire-la-Ville 瑞士: Eurographics Association.

Meyers, S. (1995)。更有效的C++: 35种改进程序和设计的新方法。阅读马:艾迪生-韦斯利.Meyers, S. (1997)。有效的C++:50种改进程序和设计的具体方法(第二版).阅读马:艾迪生-韦斯利.Mitchell, D.P. (1990年5月)。具有间隔算术的鲁棒射线相交。

在Graphicsinterface'90(第68–74页) 中。韦尔斯利, MA: 加拿大计算机通信协会&一个K彼得斯。

米切尔, D.P. (1996)。图形分层采样的后果。在 SIGGRAPH'96: 第23届计算机图形学和交互技术年会论文集 (第27 7–280页)。纽约: ACM Press.

- Mitchell, D. P., & Netravali, A. N. (1988). Reconstruction Filters in Computer Graphics. *Computer Graphics*, 22(4), 221–228.
- Morse, B., Yoo, T., Rheingans, P., Chen, D., & Subramanian, K. (2001). Interpolating Implicit Surfaces from Scattered Surface Data Using Compactly Supported Radial Basis Functions. In *Proceedings of Shape Modeling International* (pp. 89–98). Washington, DC: IEEE Computer Society Press.
- Mortenson, M. (1985). *Geometric Modeling*. New York: John Wiley & Sons.
- Munkres, J. (2000). *Topology* (Second ed.). Englewood Cliffs, NJ: Prentice Hall.
- Munzner, T. (2000). *Interactive Visualization of Large Graphs and Networks* (Unpublished doctoral dissertation). Stanford University Department of Computer Science.
- Munzner, T., Guimbretière, F., Tasiran, S., Zhang, L., & Zhou, Y. (2003). Tree-Juxtaposer: Scalable Tree Comparison Using Focus+Context with Guaranteed Visibility. *ACM Transactions on Graphics (Proc. SIGGRAPH '03)*, 22(3), 453–462.
- Museth, K., Breen, D., Whitaker, R., & Barr, A. (2002). Level Set Surface Editing Operators. *ACM Transactions on Graphics*, 21(3), 330–338.
- 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.
- Nishimura, H., Hirai, A., Kawai, T., Kawata, T., Shirikawa, I., & Omura, K. (1985). Object Modeling by Distribution Function and a Method of Image Generation. In *J. Electronics Comm. Conf. '85* (Vol. J69-D, pp. 718–725).
- Ohtake, Y., Belyaev, A., & Pasko, A. (2003). Dynamic Mesh Optimization for Polygonized Implicit Surfaces with Sharp Features. *The Visual Computer*, 19(2), 115–126.
- 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 *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (pp. 239–246). New York: ACM Press.
- Osher, S., & Sethian, J. A. (1988). Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton–Jacobi Formulations. *Journal of Computational Physics*, 79(1), 12–49.
- 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.

- Mitchell, D.P., & Netravali, A.N. (1988)。计算机图形学中的重建滤波器。计算机图形学, 22 (4) , 221–228。
- Morse, B., Yoo, T., Rheingans, P., Chen, D., & Subramanian, K. (2001)。使用紧凑支持的径向基函数从分散的表面数据中对隐式表面进行间极化.InProceedingsOfShapeModelingInternational (第89–98页)。华盛顿特区: IEEE计算机学会出版社。Mortenson, M. (1985)。几何建模。呐
- Munkres, J. (2000)。拓扑 (第二版。)恩格尔伍悬崖 NJ:普伦蒂斯霍尔。
- Munzner, T. (2000)。大型图形和网络的交互式可视化 (未发表的博士论文)。斯坦福大学系
计算机科学。
- Munzner, T., Guimbretière, F., Tasiran, S., Zhang, L., & Zhou, Y. (2003)。树Juxtaposer: 使用焦点+上下文与Guaranteed可见性的可扩展树比较。图形上的ACM交易(Proc.[医]SIGGRAPH'03) 22(3) 453–462.Museth, K., Breen, D., Whitaker, R., & Barr, A. (2002)。关卡集曲面编辑运算符。A cmTransactionsOnGraphics, 21 (3) , 330–338。Muuss, M.J. (1995)。实现组合实体几何模型的实时光线追踪.在BRL-CAD研讨会的记录中。
- Nicodemus, F.E., Richmond, J.C., Hsia, J.J., Ginsberg, I., & Limperis, T. (19 反射率的几何考虑和命名 (技术)。众议员160号)。华盛顿特区: 国家标准局。Nishimura, H., Hirai, A., Kawai, T., Kawata, T., Shirikawa, I., & Omura, K. (1985)。通过分布函数进行对象建模以及图像生成的方法。在J.ElectronicsComm. Conf. 85年 (卷) J69-D, 第718–725页)。Ohtake, Y., Belyaev, A., & Pasko, A. (2003)。动态网格优化 具有尖锐特征的多边形隐式曲面。视觉计算机, 19 (2) , 115–126。Oppenheim, A.V., Schafer, R., & Stockham, T. (1968)。非线性滤波 相乘和卷积的信号。IEEE会议记录 56(8) 1264 1291.Oren, M., & Nayar, S.K. 兰伯特反射率的泛化 模特。在SIGGRAPH'94:ProceedingsOfthe21stAnnualConferenceonComputerGraphicsandInteractiveTechniques(pp.239–246)中。呐
- Osher, S., & Sethian, J.A. (1988)。与曲率相关的前沿传播速度: 基于Hamilton Jacobi公式的算法。香港邮政 计算物理学, 79 (1) , 12–49。
- Osterberg, G. (1935)。人视网膜中棒和锥体层的地形。ActaOphthalmologica , 6 (1) , 11–97. (补充)
- Paeth, A.W. (1990)。一般栅格旋转的快速算法。在图形 宝石(第179–195页)。波士顿, MA: 学术出版社。

- Palmer, S. E. (1999). *Vision Science—Photons 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. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., ... Stich, M. (2010, July). Optix: A General Purpose Ray Tracing Engine. *ACM Trans. Graph.*, 29(4), 66:1–66:13. doi: 10.1145/1778765.1778803
- Pascale, D. (2003). *A review of RGB color spaces* (Tech. Rep.). The BabelColor Company. (www.babelcolor.com)
- Pashler, H. E. (1998). *The Psychology of Attention*. Cambridge, MA: MIT Press.
- Pasko, A., Adzhiev, V., Sourin, A., & Savchenko, V. (1995). Function Representation in Geometric Modeling: Concepts, Implementation and Applications. *The Visual Computer*, 11(8), 419–428.
- Pasko, G., Pasko, A., Ikeda, M., & Kunii, T. (2002, May). Bounded Blending Operations. In *Proceedings of the International Conference on Shape Modeling and Applications (SMI 2002)* (pp. 95–103). Washington, DC: IEEE Computer Society 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. *Proc. SIGGRAPH '85, Computer Graphics*, 19(3), 279–286.
- 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)
- 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: Proceed-*

- Palmer, S.E. (1999)。视觉科学-光子到现象学.剑桥, MA: 麻省理工学院出版社。
- Parker, S., Martin, W., Sloan, P., Shirley, P., Smits, B., & Hansen, C. (1999)。交互式光线追踪。在ACMSymposiumonInteractive3DGraphics (第119–126页) 中。 周呐Parker, S.G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., ..
- Stich, M. (2010, 7月)。Optix: 通用光线追踪引擎。 ACM反式。图。 29(4) 66:1 66:13.doi: 10.1145/1778765.1778803
- Pascale, D. (2003)。RGB颜色空间的回顾(技术.众议员)。巴贝尔科洛公司。 (www.babelcolor.com)
- Pashler, H.E. (1998)。关注的心理。剑桥, MA: 麻省理工学院出版社。 Pasko , A., Adzhiev, V., Sourin, A., & Savchenko, V. (1995)。几何建模中的函数代表.概念、实现和应用.视觉计算机, 11 (8) , 419–428。
- Pasko, G., Pasko, A., Ikeda, M., & Kunii, T. (2002, 5月)。有界混合行动。InProceedingsOfTheInternationalConferenceonShapeModelingandA pplications(SMI2002)(pp.95–103).华盛顿特区: IEEE计算机学会出版社。
- Pattanaik, S. N., Ferwerda, J. A., Fairchild, M. D., & Greenberg, D. P. (1998). 适应和空间视觉的多尺度模型, 用于逼真的图像显示。在SIGGRAPH' 98:ProceedingsOfthe25thAnnualConferenceonComputerGraphicsa ndInteractiveTechniques(pp.287–298)中。 周呐
- Pattanaik, S.N., & Yee, H. (2002)。高动态自适应增益控制范围图像显示。在SCCG'02: Proceedingsofthe18thSpringConferenceonCompute rGraphics (第83–87页) 中。 周呐Patterson, J., Hoggar, S., & Logie , J. (1991)。逆位移映射。计算机图形论坛, 10 (2) , 129–139。
- Peachey, D.R. (1985)。复杂表面的实体纹理。Proc.SIGGRAPH'85, 计算机图形学, 19 (3) , 279–286。 Penna, M., & Patterson, R. (1986)。射影几何及其在计算机图形学中的应用.恩格尔伍悬崖 NJ:普伦蒂斯霍尔。
- Perlin, K. (1985)。的图像合成器。计算机图形学, 19 (3) , 287–296 。 (SIGGRAPH'85) Perlin, K., & Hoffert, E.M. (1989)。Hypertexture 。计算机图形学, 23 (3) , 253–262。 (SIGGRAPH'89) Pharr, M., & Hanrahan, P. (1996)。用于光线追踪置换贴图的几何体缓存。在欧洲制图学讲习班的记录'96(第31–40页)。英国伦敦: Springer-Verlag。
- Pharr, M., & Humphreys, G. (2004)。基于物理的渲染。旧金山, CA : 摩根考夫曼。
- Pharr, M., Kolb, C., Gershbein, R., & Hanrahan, P. (1997)。渲染复杂具有记忆相干光线追踪的场景。在SIGGRAPH'97: 继续

- ings 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. *Proc. SIGGRAPH '88, Computer Graphics*, 22(4), 17–20.
- Pitteway, M. L. V. (1967). Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotter. *Computer Journal*, 10(3), 282–289.
- Plauger, P. J. (1991). *The Standard C Library*. Englewood Cliffs, NJ: Prentice Hall.
- Plumlee, M., & Ware, C. (2006). Zooming Versus Multiple Window Interfaces: Cognitive Costs of Visual Comparisons. *Proc. ACM Trans. on Computer-Human Interaction (ToCHI)*, 13(2), 179–209.
- Porter, T., & Duff, T. (1984). Compositing Digital Images. In *SIGGRAPH '84: Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 253–259). New York: ACM Press.
- Poynton, C. (2003). *Digital Video and HDTV: Algorithms and Interfaces*. San Francisco: Morgan Kaufmann Publishers.
- 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.
- Purcell, T. J., Buck, I., Mark, W. R., & Hanrahan, P. (2002). Ray Tracing on Programmable Graphics Hardware. *ACM Transactions on Graphics (Proc. SIGGRAPH '02)*, 21(3), 703–712.
- 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.
- Rao, R., & Card, S. K. (1994). The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information. In *Proc. ACM Human Factors in Computing Systems (CHI)* (pp. 318–322). New York: ACM Press.
- Reeves, W. T. (1983). Particle Systems—A Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics*, 2(2), 91–108.
- Reingold, E. M., & Tilford, J. S. (1981). Tidier Drawings of Trees. *IEEE Trans. Software Engineering*, 7(2), 223–228.
- 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.

- 第24届计算机图形学与交互技术年会ings (第101–108页)。阅读马·艾迪生-韦斯利。
- Phong, B.-T. (1975)。计算机生成图像的照明。Acm 18(6) 311–317。
- Pineda, J. (1988)。多边形光栅化的并行算法。Proc.SIGGRAPH'88 ComputerGraphics 22(4) 17–20。
- Pitteway, M.L.V. (1967)。用数字绘图仪绘制椭圆或双曲线的算法。计算机期刊, 10 (3) , 282–289。
- Plauger, P.J. (1991)。的标准C库。恩格尔伍悬崖 NJ:普伦蒂斯霍尔。
- Plumlee, M., & Ware, C. (2006)。缩放与多个窗口接口: 视觉比较的认知成本。Proc.ACM反式。在电脑上人类互动 (ToCHI) , 13 (2) , 179–209。
- Porter, T., & Duff, T. (1984)。合成数字图像。在SIGGRAPH'84: 第十一届计算机图形学及互动技术 (第253–259页)。呐
- Poynton, C. (2003)。数字视频和高清电视: 算法和接口。旧金山: 摩根考夫曼出版社。
- Press, W.H., Teukolsky, S.A., Vetterling, W.T., & Flannery, B.P. (1992)。C 中的Numerical食谱: 科学计算的艺术 (第二版。)英国Cambridge: 剑桥大学出版社。Purcell, T.J., Buck, I., Mark, W.R., & Hanrahan, P. (2002)。光线追踪可编程图形硬件。图形上的ACM交易(Proc. SIGGRAPH '02), 21(3), 703–712。
- Rahman, Z., Jobson, D. J., & Woodell, G. A. (1996). A Multiscale Retinex for 色彩再现和动态范围压缩。在SPIEProceedings:ApplicationsOfDigital ImageProcessingXIX(Vol.2847).贝灵汉 WA: SPIE.
- Rao, R., & Card, S.K. (1994)。表格镜头: 合并图形和符号交互式焦点中的表示+表格信息的上下文可视化。在Proc. ACM计算系统中的人为因素 (CHI) (第318–322页)。呐里夫斯, W.T. (1983)。粒子系统—一类模糊对象建模的技术。AcmTransactions on Graphics, 2 (2) , 91–108。
- Reingold, E.M., & Tilford, J.S. (1981)。树木的整洁图纸。IEEETrans.软件工程, 7 (2) , 223–228。Reinhard, E. (2003)。摄影色调再现的参数估计。图形工具杂志, 7 (1) , 45–51。Reinhard, E., Ashikhmin, M., Gooch, B., & Shirley, P. (2001)。图像之间的颜色转移。IEEEComputerGraphicsandApplications 21 34–41。Reinhard, E., & Devlin, K. (2005)。受Photoreceptor生理学启发的动态范围缩小。IEEETransactions on VisualizationandComputerGraphics 11(1) 13–24。

- Reinhard, E., Khan, E. A., Akyüz, A. O., & Johnson, G. M. (2008). *Color Imaging: Fundamentals and Applications*. Wellesley: A K Peters.
- Reinhard, E., Stark, M., Shirley, P., & Ferwerda, J. (2002). Photographic Tone Reproduction for Digital Images. *ACM Transactions on Graphics (Proc. SIGGRAPH '02)*, 21(3), 267–276.
- Reinhard, E., Ward, G., Debevec, P., & Pattanaik, S. (2005). *High Dynamic Range Imaging*. San Francisco: Morgan Kaufmann.
- Requicha, A. A. G. (1980). Representations for Rigid Solids: Theory, Methods and Systems. *Computing Surveys*, 12(4), 437–464.
- Reuter, P. (2003). *Reconstruction and Rendering of Implicit Surfaces from Large Unorganized Point Sets* (Unpublished doctoral dissertation). LABRI - Université Bordeaux.
- Reynolds, C. W. (1987). Flocks, Herds and Schools: A Distributed Behavioral Model. *Proc. SIGGRAPH '87, Computer Graphics*, 21(4), 25–34.
- Ricci, A. (1973, May). Constructive Geometry for Computer Graphics. *Computer Journal*, 16(2), 157–160.
- 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.
- Robertson, G., Fernandez, R., Fisher, D., Lee, B., & Stasko, J. (2008). Effectiveness of Animation in Trend Visualization. *IEEE Trans. on Visualization and Computer Graphics (Proc. InfoVis08)*, 14(6), 1325–1332.
- 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: Academic Press.
- Roth, S. (1982). Ray Casting for Modeling 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. *Proc. SIGGRAPH '80, Computer Graphics*, 14(3), 110–116.
- 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.

- Reinh (2008). 彩色成像:基础和应用. 韦尔斯利:一个K彼得斯.
- Reinhard, E., Stark, M., Shirley, P., & Ferwerda, J. (2002)。摄影色调 数字图像的再现。图形上的ACM交易(Proc. SIGGRAPH '02), 21(3), 267–276.
- Reinhard, E., Ward, G., Debevec, P., & Pattanaik, S. (2005)。高动态范围成像。旧金山:摩根考夫曼。Requicha, A.A.G. (1980)。刚性固体的表示:理论、方法和系统.计算调查, 12 (4) , 437–464。Reuter, P. (2003)。从大的隐式表面的重建和渲染 无组织点集 (未发表的博士论文)。LABRIuniversite波尔多.雷诺兹, C.W. (1987)。羊群、牛群和学校:分布式行为模型。Proc.SIGGRAPH'87, 计算机图形学, 21 (4) , 25–34。
- Ricci, A. (1973, 5月)。计算机图形学的建设性几何.计算机期刊, 16 (2) , 157–160。
- Riesenfeld, R.F. (1981, 1月)。齐次坐标和射影 计算机图形学中的平面。IEEE计算机图形与应用, 1 (1) , 50–55。Roberts, L. (1965年5月)。N构造的同质矩阵表示和操作 (技术)。代表号 MS-1505)。列克星敦, MA: 麻省理工学院林肯实验室。
- Robertson, G., Fernandez, R., Fisher, D., Lee, B., & Stasko, J. (2008)。趋势可视化中动画的有效ness。IEEETrans.关于可视化和计算机图形学(Proc.InfoVis08), 14(6), 1325–1332。罗杰斯, D.F. (1985)。计算机图形学的程序元素。 呐 罗杰斯, D.F. (1989)。计算机图形学的数学元素。 呐
- 罗杰斯, D.F. (2000)。NURBS简介:具有历史视角。旧金山, CA: 摩根考夫曼。
- 罗杰斯, S. (1995)。感知图像空间。在W.Epstein & S.Rogers (Eds.) 对空间和运动的感知 (Vol. 5, 第119–163页)。圣地亚哥: Academic出版社。罗斯 小号.(1982).用于实体建模的射线铸造。计算机图形和图像处理, 18 (2) , 109–144。
- Rubin, S.M., & Whitted, T. (1980)。快速的三维表示 复杂场景的渲染。Proc.SIGGRAPH'80, 计算机图形学, 14 (3) , 110–116。Rusinkiewicz, S., & Levoy, M. (2000)。QSplat:一个用于大型网格的多分辨率点渲染系统.在SIGGRAPH'00:ProceedingsOfthe27thAnnualConferenceonComputerGraphicsandInteractiveTechniques(pp.343–352)中。阅读 马:艾迪生-韦斯利.

- Rvachev, V. L. (1963). On the Analytic Description of Some Geometric Objects. *Reports of the Ukrainian Academy of Sciences*, 153, 765–767.
- Saito, T., & Takahashi, T. (1990). Comprehensible Rendering of 3-D Shapes. *Proc. SIGGRAPH '90, Computer Graphics*, 24(4), 197–206.
- Salomon, D. (1999). *Computer Graphics and Geometric Modeling*. New York: Springer-Verlag.
- Savchenko, V., Pasko, A., Okunev, O., & Kunii, T. (1995). Function Representation of Solids Reconstructed from Scattered Surface Points and Contours. *Computer Graphics Forum*, 14(4), 181–188.
- Savchenko, V. V., Pasko, A. A., Sourin, A. I., & Kunii, T. L. (1998). Volume Modelling: Representations and Advanced Operations. In *CGI '98: Proceedings of the Computer Graphics International 1* (pp. 4–13). Washington, DC: IEEE Computer Society Press.
- Sbert, M. (1997). *The Use of Global Random Directions to Compute Radiosity. Global Monte Carlo Techniques* (PhD. Thesis). Universitat Politècnica 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.
- Schmidt, R., Grimm, C., & Wyvill, B. (2006, July). Interactive Decal Compositing with Discrete Exponential Maps. *ACM Transactions on Graphics*, 25(3), 605–613.
- Schmidt, R., Wyvill, B., & Galin, E. (2005). Interactive Implicit Modeling with Hierarchical Spatial Caching. In *SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005* (pp. 104–113). Washington, DC: IEEE Computer Society Press.
- Schmidt, R., Wyvill, B., Sousa, M. C., & Jorge, J. A. (2005). Shapeshop: Sketch-Based Solid Modeling with BlobTrees. In *Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*. Aire-la-ville, Switzerland: Eurographics Association.
- Sederberg, T. W., & Parry, S. R. (1986). Free-Form Deformation of Solid Geometric Models. *Proc. SIGGRAPH '86, Computer Graphics*, 20(4), 151–160.
- Seetzen, H., Heidrich, W., Stuerzlinger, W., Ward, G., Whitehead, L., Trentacoste, M., ... Vorozcovs, A. (2004). High Dynamic Range Display Systems. *ACM Transactions on Graphics (Proc. SIGGRAPH '04)*, 23(3), 760–768.
- 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.

- Rvachev, V.L. (1963)。关于一些几何对象的分析描述。乌克兰科学院的报告, 153, 765–767。
- Saito, T., & Takahashi, T. (1990)。3-D形状的可理解渲染。Proc.SIGGRAPH'90, 计算机图形学, 24 (4) , 197–206。
- 呐 Salomon, D. (1999)。计算机图形学和几何建模。
- Savchenko, V., Pasko, A., Okunev, O., & Kunii, T. (1995)。函数表示从散乱的表面点和轮廓重建的固体.计算机图形论坛, 14 (4) , 181–188。
- Savchenko, V.V., Pasko, A.A., Sourin, A.I., & Kunii, T.L. (1998)。V olò建模: 表示和高级操作。在CGI'98: ProceedingsOfTheComputerGrap hicsInternational1 (第4–13页) 中。Washington, DC: IEEE计算机学会出版社。斯伯特, M. (1997)。使用全局随机方向来计算辐射度。
- 全球蒙特卡罗技术 (博士。论文)。政治大学 Catalunya.
- Schlick, C. (1994a)。用于基于物理的渲染ing的廉价BRDF模型。计算机图形论坛, 13 (3) , 233–246。Schlick, C. (1994b)。用于高动态范围图片可视化的量化技术。在P.Shirley, G.Sakas, & S.Müller (Eds.) , 照片逼真的渲染技术 (第7–20页)。柏林: Springer-Verlag。Schmidt, R., Grimm, C., & Wyvill, B. (2006, 7月)。交互式贴花compositing与离散指数地图.AcmTransactions on Graphics, 25 (3) , 605–613。Schmidt, R., Wyvill, B., & Galin, E. (2005)。交互式隐式建模
- 分层空间缓存。在SMI'05:2005年国际形状建模和应用会议记录(第104–13页).华盛顿特区: IEEE计算机学会出版社。Schmidt, R., Wyvill, B., So usa, M.C., & Jorge, J.A.
- 基于草图的实体建模与BlobTrees。在第二届基于草图的界面和建模欧洲摄影研讨会的会议记录中。Airela-Ville, 瑞士: EurographicsAssociation. S ederberg, T.W., & Parry, S.R. (1986)。实体几何模型的自由变形。Pro c.SIGGRAPH'86, 计算机图形学 20(4) 151 160.Seetzen, H., Heidrich, W., Stuerzlinger, W., Ward, G., Whitehead, L., Trentacoste M., 。Vorozcovs, A. (2004)。高动态范围显示系统。 图形上的ACM交易(Proc.[医]SIGGRAPH'04) 23(3) 760–768.
- Seetzen, H., Whitehead, L.A., & Ward, G. (2003)。高动态范围 使用低分辨率和高分辨率调制器显示。在信息显示学会国际研讨会上.加州圣何塞: 信息显示。

- Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., & Haeberli, P. (1992). Fast Shadows and Lighting Effects Using Texture Mapping. *Proc. SIGGRAPH '92, Computer Graphics*, 26(2), 249–252.
- Shannon, C. E., & Weaver, W. (1964). *The Mathematical Theory of Communication*. Urbana, IL: University of Illinois Press.
- Shapiro, V. (1988). *Theory of R-Functions and Applications: A Primer* (Tech. Rep. No. CPA88-3). Ithaca, NY: Cornell University.
- Shapiro, V. (1994). Real Functions for Representation of Rigid Solids. *Computer Aided Geometric Design*, 11, 153–175.
- 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>)
- Sherstyuk, A. (1999). Interactive Shape Design with Convolution Surfaces. In *SMI '99: Proceedings of the International Conference on Shape Modeling and Applications* (pp. 56–65). Washington, DC: IEEE Computer Society Press.
- Shirley, P. (1991). *Physically Based Lighting Calculations for Computer Graphics* (Unpublished doctoral dissertation). University of Illinois, Urbana-Champaign.
- Shirley, P., Smits, B., Hu, H., & Lafourte, 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 Press.
- Shirley, P., Wang, C., & Zimmerman, K. (1996). Monte Carlo Techniques for Direct Lighting Calculations. *ACM Transactions on Graphics*, 15(1), 1–36.
- Shneiderman, B. (1996). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proc. IEEE Visual Languages* (pp. 336–343). Washington, DC: IEEE Computer Society Press.
- 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.
- Simons, D. J. (2000). Current Approaches to Change Blindness. *Visual Cognition*, 7(1/2/3), 1–15.
- Slocum, T. A., McMaster, R. B., Kessler, F. C., & Howard, H. H. (2008). *Thematic Cartography and Geovisualization* (3rd ed.). Englewood Cliffs, NJ: Prentice Hall.
- Smith, A. R. (1995). *A Pixel is Not a Little Square!* (Technical Memo No. 6). Bellingham, WA: Microsoft Research.
- Smith, S., Grinstein, G., & Bergeron, R. D. (1991). Interactive Data Exploration with a Supercomputer. In *VIS '91: Proceedings of the 2nd Conference on Visualization '91* (pp. 248–254). Los Alamitos, CA: IEEE Computer Society Press.

- Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., & Haeberli, P. (1992). 快使用纹理映射的阴影和光照效果。 *Proc.SIGGRAPH'92 ComputerGraphics* 26(2) 249–252. Shannon, C.E., & Weaver, W. (1964). 共融的数学理论。 厄巴纳, IL: 伊利诺伊大学出版社。 Shapiro, V. (1988). R函数理论与应用: 入门 (技术)。 代表号CPA88-3)。 伊萨卡, 纽约州: 康奈尔大学。
- Shapiro, V. (1994)。 用于表示刚性固体的实函数。 计算机辅助几何设计, 11, 153–175。
- Shene, C.-K. (2003)。 CS3621介绍计算与几何笔记。 可从万维网获得。 (<http://www.cs.mtu.edu/shene/COURSES/cs3621/NOTES/notes.html>)
- Sherstyuk, A. (1999)。 具有卷积曲面的交互式形状设计。 在 *SMI'99: 形状建模与应用国际会议记录* (第56–65页)。 华盛顿特区: IEEE计算机学会出版社。
- Shirley, P. (1991)。 计算机图形ic的基于物理的照明计算 (未发表的博士论文)。 伊利诺伊大学厄巴纳分校 Champaign.
- Shirley, P., Smits, B., Hu, H., & Lafourte, E. (1997)。 实践者对光反射模型的分析.在PG'97: 第五届太平洋计算机图形与应用会议记录 (第40–49页)。 洛斯阿拉米托斯, CA: IEEE计算机学会出版社。
- Shirley, P., Wang, C., & Zimmerman, K. (1996)。 直接照明计算的蒙特卡罗技术。 *AcMTransactions on Graphics*, 15 (1), 1–36. Shneiderman, B. (1996)。 眼睛有它: 信息可视化的数据类型分类法的任务。 在 *Proc. IEEE视觉语言* (第336–343页). 华盛顿特区: IEEE计算机学会出版社。 Shreiner, D., Neider, J., Woo, M., & Davis, T. (2004)。 OpenGL编程指南 (第四版)。 阅读马:艾迪生-韦斯利。
- Sillion, F.X., & Puech, C. (1994)。 辐射度和全局照明。 隆戮鹿芦赂忙隆驴卤戮 跋路垄虏录 禄煤 卢虏陆 酶 戮拢隆 西蒙斯, D.J. (2000)。 目前改变失明的方法。 视觉认知 7(123) 1–15. Slocum, T.A., McMaster, R.B., Kessler, F.C., & Howard, H.H. (2008)。 马蒂克制图和地理化 (第3版)。).恩格尔伍悬崖 NJ:普伦蒂斯霍尔。
- 史密斯, A.R. (1995)。 一个像素不是一个正方形! (技术备忘录第6号)。 华盛顿州贝灵汉: 微软研究院。
- Smith, S., Grinstein, G., & Bergeron, R.D. (1991)。 用超级计算机进行交互式数据探索.在VIS'91: 第二次可视化会议记录'91 (第248–254页) 中。 洛斯阿拉米托斯, CA: IEEE计算机学会出版社。

- 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. *Proc. SIGGRAPH '87, Computer Graphics*, 21(4), 119–128.
- 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.
- Stolte, C., Tang, D., & Hanrahan, P. (2008). Polaris: A System for Query, Analysis, and Visualization of Multidimensional Databases. *Commun. ACM*, 51(11), 75–84.
- Stone, M. C. (2003). *A Field Guide to Digital Color*. Natick, MA: A K Peters.
- 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 Computer Society Press.
- Tory, M., Kirkpatrick, A. E., Atkins, M. S., & Möller, T. (2006). Visualization Task Performance with 2D, 3D, and Combination Displays. *IEEE Trans. Visualization and Computer Graphics (TVCG)*, 12(1), 2–13.
- Tumblin, J., & Turk, G. (1999). LCIS: A Boundary Hierarchy for Detail-Preserving 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., & O'Brien, J. (1999). Shape Transformation Using Variational Implicit Functions. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference*

- Smits, B.E., Shirley, P., & Stark, M.M. (2000)。置换映射三角形的直接光线追踪。《2000年欧洲制图技术讲习班会议记录》(第307–318页)。英国伦敦: Springer-Verlag. Snyder, J.M., & Barr, A.H. (1987)。包含以下内容的光线追踪复杂模型
表面镶嵌。Proc.SIGGRAPH'87, 计算机图形学, 21 (4) , 119–128。Sobel, I., Stone, J., & Messer, R. (1975)。的蒙特卡洛方法。芝加哥 IL: 芝加哥大学出版社。
所罗门, H. (1978)。几何概率。费城 PA:暹罗出版社。
Stam, J. (1999)。衍射着色器。在SIGGRAPH'99: 第26届会议记录
计算机图形学和交互技术年度会议(第101–110页)。阅读马:艾迪生-韦斯利.Stark, M.M., Arvo, J., & Smits, B.
(2005).各向同性BRDFs的重心参数化。IEEE TransactionsonVisualizati
onandComputerGraphics 11(2) 126–138.
Stockham, T. (1972)。视觉模型上下文中的图像处理。PROceedingsOF
THEIEEE 60(7) 828–842.Stolte, C., Tang, D., & Hanrahan, P. (2008)
。Polaris: 用于多维数据库的查询, 分析和可视化的系统。康姆。ACM,
51 (11) , 75–84。Stone, M.C. (2003)。数字色彩的现场指南.Natick
, MA: AKPeters.
Strang, G. (1988)。线性代数及其应用 (第三版。).佛罗伦萨 KY:布鲁克
斯*科尔。
Sutherland, I.E., Sproutll, R.F., & Schumacker, R.A. (1974)。十种隐面算法的
表征.ACM计算调查, 6 (1) , 1–55. Thompson, W.B., & Pong, T.C. (1990)
。检测运动物体。国际计算机视觉杂志, 4 (1) , 39–57. Thompson, W.B., Shi
rley, P., & Ferwerda, J. (2002)。一种空间后处理
夜景图像的算法。图形工具杂志, 7 (1) , 1–12。Tomasi, C., & Mand
uchi, R. (1998)。灰色和彩色Im年龄的双边滤波。在Proc. IEEE计算
机视觉国际会议 (第836–846页)。华盛顿特区: IEEE计算机学会出版社。T
ory, M., Kirkpatrick, A.E., Atkins, M.S., & Möller, T. (2006)。可视化
任务性能与2D, 3D和组合显示。IEEETrans.
可视化与计算机图形学 (TVCG) , 12 (1) , 2–13。
Tumblin, J., & Turk, G. (1999). LCIS: 细节的边界层次结构
保持对比度降低。在A.Rockwood (Ed.) , SIGGRAPH'99: 第26届计算
机图形学和交互技术年会论文集 (第83–90页)。阅读马:艾迪生韦斯利长
人.Turk, G., & O'Brien, J. (1999)。使用变分隐式的形状变换
函数。在SIGGRAPH'99: 第26届年会记录

- on Computer Graphics and Interactive Techniques* (pp. 335–342). New York: ACM Press/Addison-Wesley Publishing Co.
- Turk, G., & O'Brien, J. F. (2002). Modelling with Implicit Surfaces that Interpolate. *ACM Transactions on Graphics*, 21(4), 855–873.
- Turkowski, K. (1990). Properties of Surface-Normal Transformations. In *Graphics Gems* (pp. 539–547). Boston: Academic Press.
- Tversky, B., Morrison, J., & Betrancourt, M. (2002). Animation: Can It Facilitate? *International Journal of Human Computer Studies*, 57(4), 247–262.
- 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.
- van Ham, F., & van Wijk, J. J. (2004). Interactive Visualization of Small World Graphs. In *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization* (pp. 199–206). Washington, DC: IEEE Computer Society Press.
- van Wijk, J. J., & van Selow, E. R. (1999). Cluster and Calendar-Based Visualization of Time Series Data. In *INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization* (pp. 4–9). Washington, DC: IEEE Computer Society Press.
- van Overveld, K., & Wyvill, B. (2004). Shrinkwrap: An Efficient Adaptive Algorithm for Triangulating an Iso-Surface. *The Visual Computer*, 20(6), 362–369.
- 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.

- 论计算机图形学和交互技术(第335–342页)。 雁扶
蟋医 帮九& 碧
- Turk, G., & O'Brien, J.F. (2002)。用迟交的隐式表面建模.ACMTransactions on Graphics, 21 (4) , 855–873。Turkowski, K. (1990)。表面法线变换的属性。在GraphicsGems (第539–547页) 中。波士顿: 学术出版社。Tversky, B., Morrison, J., & Betrancourt, M. (2002)。动画: 它可以Facili泰特?国际人类计算机研究杂志, 57 (4) , 247–262。Upstill, S. (1985)。合成图像的真实呈现: 计算机图形学中的图像处理(未发表的博士论文).加州大学伯克利分校的大学。vanAken, J., & Novak, M. (1985)。光栅Dis播放的曲线绘制算法.AcmTransactions on Graphics, 4 (2) , 147–169。vanHam, F., & vanWijk, J.J. (2004)。小世界图的交互式可视化。InINFOVIS'04:Proceedings OF THE IEEE Symposium on Information Visualization(pp.199–206)。华盛顿特区: IEEE计算机学会出版社。vanWijk, J.J., & vanSelow, E.R. (1999)。基于集群和日历的时间序列数据可视化。InINFOVIS'99:Proceedings Of THE 1999 IEEE Symposium on Information Visualization(pp.4–9)。华盛顿特区:
- IEEE计算机学会出版社。vanOver
(2004).Shrinkwrap: 用于三角测量Iso曲面的高效自适应算法。视觉计算机, 20 (6) , 362–369。Veach, E., & Guibas, L.J. (1997)。大都市轻型运输。在SIGGRAPH'97:Proceedings Of the 24th Annual Conference on Computer Graphics and Interactive Techniques(pp.65–76)中。阅读马:艾迪生-韦斯利.Wald, I., Slusallek, P., Benthin, C., & Wagner, M. (2001)。交互式分布式
高度复杂模型的光线追踪。在第12届欧洲图形渲染技术研讨会会议记录 (第277–288页)。英国伦敦: Springer-Verlag。
- Walter, B., Hubbard, P.M., Shirley, P., & Greenberg, D.F. (1997)
。使用局部线性密度估计的全局照明。ACMTransactions on Graphics, 16 (3) , 217–259。Wandell, B.A. (1995)。愿景的基础。桑德兰, MA : SinauerAssociates。Wann, J.P., Rushton, S., & Mon-Williams, M. (1995)。自然问题
虚拟环境中的立体深度感知。视觉研究, 35 (19) , 2731–2736。Ward, G., & Simmons, M. (2004)。高动态范围的子带编码
意象。首届ACM图形应用感知研讨会
可视化(APGV) (第83–90页)。纽约: ACM出版社。

- Ward, G. J. (1992). Measuring and Modeling Anisotropic Reflection. *Proc. SIGGRAPH '92, Computer Graphics*, 26(2), 265–272.
- Ward, G. J. (1994). The RADIANCE Lighting Simulation and Rendering System. In A. Glassner (Ed.), *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (pp. 459–472). New York: ACM Press.
- Ward, M. O. (2002, December). A Taxonomy of Glyph Placement Strategies for Multidimensional Data Visualization. *Information Visualization Journal*, 1(3–4), 194–210.
- 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.
- Ware, C. (2000). *Information Visualization: Perception for Design*. Boston, MA: Morgan Kaufmann/Academic Press.
- Ware, C. (2001). Designing With a 2 1/2 D Attitude. *Information Design Journal*, 10(3), 255–262.
- Ware, C., Purchase, H., Colpys, L., & McGill, M. (2002). Cognitive Measures of Graph Aesthetics. *Information Visualization*, 1(2), 103–110.
- Warn, D. R. (1983). Lighting Controls for Synthetic Images. *Proc. SIGGRAPH '83, Computer Graphics*, 17(3), 13–21.
- Watt, A. (1991). *Advanced Animation and Rendering Techniques*. Reading, MA: Addison-Wesley.
- Watt, A. (1993). *3D Computer Graphics*. Reading, MA: Addison-Wesley.
- Wegman, E. J. (1990, Sep). Hyperdimensional Data Analysis Using Parallel Coordinates. *Journ. American Statistical Association*, 85(411), 664–675.
- 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. (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. (2000). *Color Science: Concepts and Methods, Quantitative Data and Formulae* (Second ed.). New York: Wiley.
- Wyvill, B., Galin, E., & Guy, A. (1999). Extending the CSG Tree: Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum*, 18(2), 149–158.
- Wyvill, B., McPheeers, C., & Wyvill, G. (1986). Data Structures for Soft Objects. *The Visual Computer*, 2(4), 227–234.

- Ward, G.J. (1992)。测量和建模各向异性反射。 Proc.SIGGRAPH'92 ComputerGraphics 26(2) 265–272.
- Ward, G.J. (1994)。发光照明模拟和渲染系统。 在A.Glassner (Ed.) , SIGGRAPH'94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (第459–472页)。 呃内
- 沃德 M.O.(2002年12月)。字形放置策略的分类法
多维数据可视化。信息可视化杂志, 1 (3–4) , 194–210。 WardLarson , G., Rushmeier, H., & Piatko, C. (1997)。一种可见性匹配色调
- 高动态范围场景的再现算子。 IEEETransactions on Visualization and Computer Graphics 3(4) 291–306.WardLarson, G., & Shakespeare, R.A. (1998)。渲染与光芒。旧金山, CA: MorganKaufmannPublishers。
- Ware, C. (2000)。信息可视化：设计感知。波士顿，MA：摩根考夫曼学术出版社。
- Ware, C. (2001)。以212d的态度设计.信息设计期刊, 10 (3) , 255–262。 Ware, C., Purchase, H., Colpys, L., & McGill, M. (2002)。图美学的认知度量。信息可视化, 1 (2) , 103–110。
- 警告, D.R. (1983)。合成图像的照明控制。 Proc.SIGGRAPH'83, 计算机图形学, 17 (3) , 13–21。瓦特, A. (1991)。高级动画和渲染技术。 阅读, 妈: Addison-Wesley.
- 瓦特, A. (1993)。3D计算机图形学。阅读 马:艾迪生-韦斯利。
- Wegman, E.J. (1990, Sep)。使用平行坐标进行超多维数据分析。乔恩。美国统计协会, 85 (411) , 664–675。
- Whitted, T. (1980)。阴影显示的改进的照明模型。 Acm的communications, 23 (6) , 343–349。 Williams, A., Barrus, S., Morley, R.K., & Shirley, P. (2005)。一个高效和
鲁棒的Ray-Box交集算法。图形工具杂志, 10 (1) , 49–54。威廉姆斯, L. (1991)。二维阴影。在图形程序中
- 界面 (第143–151页)。韦尔斯利, 马: 一个K彼得斯和加拿大人
计算机通信学会。
- Wyszecki, G., & Stiles, W. (2000). 色彩科学：概念和方法
定量数据和公式 (第二版)。 呃内
- Wyvill, B., Galin, E., & Guy, A. (1999)。扩展CSG树：翘曲
隐式曲面建模系统中的混合和布尔运算。
计算机图形论坛, 18 (2) , 149–158。
- Wyvill, B., McPheeers, C., & Wyvill, G. (1986)。软对象的数据结构。
视觉计算机, 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. *Proc. SIGGRAPH '79, Computer Graphics*, 13(2), 190–198.
- Yonas, A., Goldsmith, L. T., & Hallstrom, J. L. (1978). The Development of Sensitivity to Information from Cast Shadows in Pictures. *Perception*, 7, 333–342.



- Yantis, S. (Ed.).(2000).视觉感知：基本阅读。英国伦敦：泰勒和弗朗西斯集团。
- Yessios, C.I. (1979) 。石材、木材、植物和地面材料的计算机绘图.Proc.SIGGRAPH '79, 计算机图形学, 13 (2) , 190–198。
- Yonas, A., Goldsmith, L.T., & Hallstrom, J.L. (1978) 。的发展对图片中投射阴影信息的敏感度。感知, 7, 333–342。

Computer Science & Engineering

Fundamentals of Computer Graphics

FOURTH EDITION

Steve Marschner
Peter Shirley

with

Michael Ashikhmin
Michael Gleicher
Naty Hoffman
Garrett Johnson
Tamara Munzner
Erik Reinhard
William B. Thompson
Peter Willemsen
Brian Wyvill

Computer Science & Engineering

Fundamentals of Computer Graphics

FOURTH EDITION

Steve Marschner
Peter Shirley

with

Michael Ashikhmin
Michael Gleicher
Naty Hoffman
Garrett Johnson
Tamara Munzner
Erik Reinhard
William B. Thompson
Peter Willemsen
Brian Wyvill

Drawing on an impressive roster of experts in the field, **Fundamentals of Computer Graphics, Fourth Edition** offers an ideal resource for computer course curricula as well as a user-friendly personal or professional reference.

Focusing on geometric intuition, the book gives the necessary information for understanding how images get onto the screen by using the complementary approaches of ray tracing and rasterization. It covers topics common to an introductory course, such as sampling theory, texture mapping, spatial data structure, and splines. It also includes a number of contributed chapters from authors known for their expertise and clear way of explaining concepts.

Highlights of the Fourth Edition Include:

- Updated coverage of existing topics
- Major updates and improvements to several chapters, including texture mapping, graphics hardware, signal processing, and data structures
- A text now printed entirely in four-color to enhance illustrative figures of concepts

The fourth edition of **Fundamentals of Computer Graphics** continues to provide an outstanding and comprehensive introduction to basic computer graphic technology and theory. It retains an informal and intuitive style while improving precision, consistency, and completeness of material, allowing aspiring and experienced graphics programmers to better understand and apply foundational principles to the development of efficient code in creating film, game, or web designs.

K22616

ISBN: 978-1-4822-2939-4



 **CRC Press**
Taylor & Francis Group
an informa business
www.crcpress.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
711 Third Avenue
New York, NY 10017
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK

Drawing on an impressive roster of experts in the field, **Fundamentals of Computer Graphics, Fourth Edition** offers an ideal resource for computer course curricula as well as a user-friendly personal or professional reference.

Focusing on geometric intuition, the book gives the necessary information for understanding how images get onto the screen by using the complementary approaches of ray tracing and rasterization. It covers topics common to an introductory course, such as sampling theory, texture mapping, spatial data structure, and splines. It also includes a number of contributed chapters from authors known for their expertise and clear way of explaining concepts.

Highlights of the Fourth Edition Include:

- Updated coverage of existing topics
- Major updates and improvements to several chapters, including texture mapping, graphics hardware, signal processing, and data structures
- A text now printed entirely in four-color to enhance illustrative figures of concepts

The fourth edition of **Fundamentals of Computer Graphics** continues to provide an outstanding and comprehensive introduction to basic computer graphic technology and theory. It retains an informal and intuitive style while improving precision, consistency, and completeness of material, allowing aspiring and experienced graphics programmers to better understand and apply foundational principles to the development of efficient code in creating film, game, or web designs.

K22616

ISBN: 978-1-4822-2939-4



 **CRC Press**
Taylor & Francis Group
an informa business
www.crcpress.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
711 Third Avenue
New York, NY 10017
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK