| Top: Courtesy of Michael Kass, Pixar and Andrew Witkin, © 1991 ACM, Inc. Reprinted by permission. Bottom: Courtesy of Greg Turk, © 1991 ACM, Inc. Reprinted by permission. |
| --- |

| Courtesy of Daniel Keefe, University of Minnesota. |
| --- |

| (0, 0, 0)  (0, *y*9, 0)  (0, *y*9, 1)  (0, *y*, 0)  (0, *y*, *z*) |
| --- |

| Courtesy of Steve Strassmann. © 1986 ACM, Inc. Reprinted by permission. |
| --- |

| Courtesy of Ken Perlin, © 1985 ACM, Inc. Reprinted by permission. |
| --- |

|  |
| --- |

| Courtesy of Ramesh Raskar; © 2004 ACM, Inc. Reprinted by permission. |
| --- |

|  |
| --- |

| Courtesy of Stephen Marschner, © 2002 ACM, Inc. Reprinted by permission. |
| --- |

|  |
| --- |

| Courtesy of Seungyong Lee, © 2007 ACM, Inc. Reprinted by permission. |
| --- |

**Computer Graphics** Third Edition

*This page intentionally left blank*

**Computer Graphics Principles and Practice**

Third Edition

JOHN F. HUGHES

ANDRIES VAN DAM

MORGAN MCGUIRE

DAVID F. SKLAR

JAMES D. FOLEY

STEVEN K. FEINER

KURT AKELEY

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco New York • Toronto • Montreal • London • Munich • Paris • Madrid Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales

(800) 382-3419

corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales

international@pearsoned.com

Visit us on the Web: informit.com/aw

*Library of Congress Cataloging-in-Publication Data*

Hughes, John F., 1955–

Computer graphics : principles and practice / John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, Kurt Akeley.—Third edition. pages cm

Revised ed. of: Computer graphics / James D. Foley. . . [et al.].—2nd ed. – Reading, Mass. : Addison-Wesley, 1995.

Includes bibliographical references and index.

ISBN 978-0-321-39952-6 (hardcover : alk. paper)—ISBN 0-321-39952-8 (hardcover : alk. paper) 1. Computer graphics. I. Title.

T385.C5735 2014

006.6–dc23

2012045569

Copyright © 2014 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-39952-6

ISBN-10: 0-321-39952-8

Text printed in the United States on recycled paper at RR Donnelley in Willard, Ohio. First printing, July 2013

*To my family, my teacher Rob Kirby, and my parents and Jim Arvo in memoriam.*

*—John F. Hughes*

*To my long-suffering wife, Debbie, who once again put up with never-ending work on “the book,” and to my father, who was the real scientist in the family.*

*—Andries Van Dam*

*To Sarah, Sonya, Levi, and my parents for their constant support; and to my mentor Harold Stone for two decades of guidance through life in science.*

*—Morgan McGuire*

*To my parents in memoriam for their limitless sacrifices to give me the educational opportunities they never enjoyed; and to my dear wife Siew May for her unflinching forbearance with the hundreds of times I retreated to my “man cave” for Skype sessions with Andy. —David Sklar*

*To Marylou, Heather, Jenn, my parents in memoriam, and all my teachers—especially Bert Herzog, who introduced me to the wonderful world of Computer Graphics!*

*—Jim Foley*

*To Michele, Maxwell, and Alex, and to my parents and teachers. —Steve Feiner*

*To Pat Hanrahan, for his guidance and friendship.*

*—Kurt Akeley*

*This page intentionally left blank*

**Contents at a Glance**

*Contents*........................................................................................ ix *Preface* ......................................................................................... xxxv *About the Authors*.......................................................................... xlv

**1 Introduction** ......................................................................... 1 **2 Introduction to 2D Graphics Using WPF**.............................. 35 **3 An Ancient Renderer Made Modern** .................................... 61 **4 A 2D Graphics Test Bed**........................................................ 81 **5 An Introduction to Human Visual Perception**....................... 101

**6 Introduction to Fixed-Function 3D Graphics and Hierarchical Modeling**.......................................................... 117

**7 Essential Mathematics and the Geometry of 2-Space and 3-Space** ................................................................................. 149

**8 A Simple Way to Describe Shape in 2D and 3D**.................... 187 **9 Functions on Meshes**............................................................. 201 **10 Transformations in Two Dimensions**..................................... 221 **11 Transformations in Three Dimensions**.................................. 263 **12 A 2D and 3D Transformation Library for Graphics**............. 287 **13 Camera Specifications and Transformations**......................... 299 **14 Standard Approximations and Representations**.................... 321 **15 Ray Casting and Rasterization** ............................................. 387 **16 Survey of Real-Time 3D Graphics Platforms**........................ 451 **17 Image Representation and Manipulation**.............................. 481 **18 Images and Signal Processing**............................................... 495 **19 Enlarging and Shrinking Images**.......................................... 533

vii

viii Contents at a Glance

**20 Textures and Texture Mapping**............................................. 547 **21 Interaction Techniques**......................................................... 567 **22 Splines and Subdivision Curves**............................................ 595 **23 Splines and Subdivision Surfaces**.......................................... 607 **24 Implicit Representations of Shape** ........................................ 615 **25 Meshes**.................................................................................. 635 **26 Light**..................................................................................... 669 **27 Materials and Scattering** ...................................................... 711 **28 Color** .................................................................................... 745 **29 Light Transport**.................................................................... 783 **30 Probability and Monte Carlo Integration**............................. 801

**31 Computing Solutions to the Rendering Equation: Theoretical Approaches**........................................................ 825

**32 Rendering in Practice** ........................................................... 881 **33 Shaders**................................................................................. 927 **34 Expressive Rendering**........................................................... 945 **35 Motion**.................................................................................. 963 **36 Visibility Determination**........................................................1023 **37 Spatial Data Structures**.........................................................1065 **38 Modern Graphics Hardware**.................................................1103

*List of Principles*........................................................................... 1145 *Bibliography* ................................................................................. 1149 *Index*.............................................................................................1183

**Contents**

*Preface*........................................................................................................................................xxxv *About the Authors*...................................................................................................................... xlv

**1 Introduction**........................................................................................................................ 1

Graphics is a broad field; to understand it, you need information from perception, physics, mathe matics, and engineering. Building a graphics application entails user-interface work, some amount of modeling (i.e., making a representation of a shape), and rendering (the making of pictures of shapes). Rendering is often done via a “pipeline” of operations; one can use this pipeline without understand ing every detail to make many useful programs. But if we want to render things accurately, we need to start from a physical understanding of light. Knowing just a few properties of light prepares us to make a first approximate renderer.

**1.1 An Introduction to Computer Graphics** ..................................................................... 1 1.1.1 The World of Computer Graphics...................................................................... 4 1.1.2 Current and Future Application Areas............................................................... 4 1.1.3 User-Interface Considerations........................................................................... 6

**1.2 A Brief History** ............................................................................................................ 7 **1.3 An Illuminating Example** ............................................................................................ 9 **1.4 Goals, Resources, and Appropriate Abstractions**....................................................... 10

1.4.1 Deep Understanding versus Common Practice .................................................. 12 **1.5 Some Numbers and Orders of Magnitude in Graphics**.............................................. 12 1.5.1 Light Energy and Photon Arrival Rates ............................................................. 12 1.5.2 Display Characteristics and Resolution of the Eye............................................. 13 1.5.3 Digital Camera Characteristics.......................................................................... 13 1.5.4 Processing Demands of Complex Applications.................................................. 14 **1.6 The Graphics Pipeline** ................................................................................................. 14 1.6.1 Texture Mapping and Approximation................................................................ 15 1.6.2 The More Detailed Graphics Pipeline................................................................ 16 **1.7 Relationship of Graphics to Art, Design, and Perception**........................................... 19 **1.8 Basic Graphics Systems**............................................................................................... 20 1.8.1 Graphics Data................................................................................................... 21 **1.9 Polygon Drawing As a Black Box**................................................................................ 23 **1.10 Interaction in Graphics Systems** ................................................................................. 23

ix

x Contents

**1.11 Different Kinds of Graphics Applications**................................................................... 24 **1.12 Different Kinds of Graphics Packages**........................................................................ 25 **1.13 Building Blocks for Realistic Rendering: A Brief Overview**....................................... 26

1.13.1 Light ................................................................................................................ 26 1.13.2 Objects and Materials ....................................................................................... 27 1.13.3 Light Capture ................................................................................................... 29 1.13.4 Image Display .................................................................................................. 29 1.13.5 The Human Visual System................................................................................ 29 1.13.6 Mathematics..................................................................................................... 30 1.13.7 Integration and Sampling .................................................................................. 31 **1.14 Learning Computer Graphics**..................................................................................... 31

**2 Introduction to 2D Graphics Using WPF** ................................................................... 35

A graphics platform acts as the intermediary between the application and the underlying graphics hardware, providing a layer of abstraction to shield the programmer from the details of driving the graphics processor. As CPUs and graphics peripherals have increased in speed and memory capa bilities, the feature sets of graphics platforms have evolved to harness new hardware features and to shoulder more of the application development burden. After a brief overview of the evolution of 2D platforms, we explore a modern package (Windows Presentation Foundation), showing how to construct an animated 2D scene by creating and manipulating a simple hierarchical model. WPF’s declarative XML-based syntax, and the basic techniques of scene specification, will carry over to the presentation of WPF’s 3D support in Chapter 6.

**2.1 Introduction**................................................................................................................. 35 **2.2 Overview of the 2D Graphics Pipeline** ........................................................................ 36 **2.3 The Evolution of 2D Graphics Platforms**.................................................................... 37

2.3.1 From Integer to Floating-Point Coordinates....................................................... 38 2.3.2 Immediate-Mode versus Retained-Mode Platforms ........................................... 39 2.3.3 Procedural versus Declarative Specification....................................................... 40

**2.4 Specifying a 2D Scene Using WPF** .............................................................................. 41 2.4.1 The Structure of an XAML Application ............................................................ 41 2.4.2 Specifying the Scene via an Abstract Coordinate System................................... 42 2.4.3 The Spectrum of Coordinate-System Choices.................................................... 44 2.4.4 The WPF Canvas Coordinate System ................................................................ 45 2.4.5 Using Display Transformations......................................................................... 46 2.4.6 Creating and Using Modular Templates............................................................. 49

**2.5 Dynamics in 2D Graphics Using WPF** ........................................................................ 55 2.5.1 Dynamics via Declarative Animation ................................................................ 55 2.5.2 Dynamics via Procedural Code ......................................................................... 58

**2.6 Supporting a Variety of Form Factors** ........................................................................ 58 **2.7 Discussion and Further Reading** ................................................................................. 59

**3 An Ancient Renderer Made Modern** ........................................................................... 61

We describe a software implementation of an idea shown by Dürer. Doing so lets us create a per spective rendering of a cube, and introduces the notions of transforming meshes by transforming vertices, clipping, and multiple coordinate systems. We also encounter the need for visible surface determination and for lighting computations.

Contents xi

**3.1 A Dürer Woodcut**......................................................................................................... 61 **3.2 Visibility**....................................................................................................................... 65 **3.3 Implementation**............................................................................................................ 65

3.3.1 Drawing ........................................................................................................... 68 **3.4 The Program**................................................................................................................ 72 **3.5 Limitations**................................................................................................................... 75 **3.6 Discussion and Further Reading** ................................................................................. 76 **3.7 Exercises** ...................................................................................................................... 78

**4 A 2D Graphics Test Bed** .................................................................................................. 81

We want you to rapidly test new ideas as you learn them. For most ideas in graphics, even 3D graph ics, a simple 2D program suffices. We describe a test bed, a simple program that’s easy to modify to experiment with new ideas, and show how it can be used to study corner cutting on polygons. A similar 3D program is available on the book’s website.

**4.1 Introduction**................................................................................................................. 81 **4.2 Details of the Test Bed** ................................................................................................. 82 4.2.1 Using the 2D Test Bed ...................................................................................... 82 4.2.2 Corner Cutting.................................................................................................. 83 4.2.3 The Structure of a Test-Bed-Based Program...................................................... 83 **4.3 The C# Code** ................................................................................................................ 88 4.3.1 Coordinate Systems.......................................................................................... 90 4.3.2 WPF Data Dependencies................................................................................... 91 4.3.3 Event Handling................................................................................................. 92 4.3.4 Other Geometric Objects................................................................................... 93 **4.4 Animation** .................................................................................................................... 94 **4.5 Interaction** ................................................................................................................... 95 **4.6 An Application of the Test Bed**.................................................................................... 95 **4.7 Discussion** .................................................................................................................... 98 **4.8 Exercises** ...................................................................................................................... 98

**5 An Introduction to Human Visual Perception**........................................................... 101

The human visual system is the ultimate “consumer” of most imagery produced by graphics. As such, it provides design constraints and goals for graphics systems. We introduce the visual system and some of its characteristics, and relate them to engineering decisions in graphics.

The visual system is both tolerant of bad data (which is why the visual system can make sense of a child’s stick-figure drawing), and at the same time remarkably sensitive. Understanding both aspects helps us better design graphics algorithms and systems. We discuss basic visual processing, constancy, and continuation, and how different kinds of visual cues help our brains form hypotheses about the world. We discuss primarily static perception of shape, leaving discussion of the perception of motion to Chapter 35, and of the perception of color to Chapter 28.

**5.1 Introduction**................................................................................................................. 101 **5.2 The Visual System** ....................................................................................................... 103 **5.3 The Eye** ........................................................................................................................ 106

5.3.1 Gross Physiology of the Eye ............................................................................. 106 5.3.2 Receptors in the Eye ......................................................................................... 107

xii Contents

**5.4 Constancy and Its Influences**....................................................................................... 110 **5.5 Continuation**................................................................................................................ 111 **5.6 Shadows**....................................................................................................................... 112 **5.7 Discussion and Further Reading** ................................................................................. 113 **5.8 Exercises** ...................................................................................................................... 115

**6 Introduction to Fixed-Function 3D Graphics and Hierarchical Modeling** ........ 117

The process of constructing a 3D scene to be rendered using the classic fixed-function graphics pipeline is composed of distinct steps such as specifying the geometry of components, applying surface materials to components, combining components to form complex objects, and placing lights and cameras. WPF provides an environment suitable for learning about and experimenting with this classic pipeline. We first present the essentials of 3D scene construction, and then further extend the discussion to introduce hierarchical modeling.

**6.1 Introduction**................................................................................................................. 117 6.1.1 The Design of WPF 3D..................................................................................... 118 6.1.2 Approximating the Physics of the Interaction of Light with Objects................... 118 6.1.3 High-Level Overview of WPF 3D ..................................................................... 119

**6.2 Introducing Mesh and Lighting Specification**............................................................. 120 6.2.1 Planning the Scene............................................................................................ 120 6.2.2 Producing More Realistic Lighting.................................................................... 124 6.2.3 “Lighting” versus “Shading” in Fixed-Function Rendering................................ 127

**6.3 Curved-Surface Representation and Rendering**......................................................... 128 6.3.1 Interpolated Shading (Gouraud) ........................................................................ 128 6.3.2 Specifying Surfaces to Achieve Faceted and Smooth Effects ............................. 130

**6.4 Surface Texture in WPF** .............................................................................................. 130 6.4.1 Texturing via Tiling .......................................................................................... 132 6.4.2 Texturing via Stretching.................................................................................... 132

**6.5 The WPF Reflectance Model**....................................................................................... 133 6.5.1 Color Specification ........................................................................................... 133 6.5.2 Light Geometry ................................................................................................ 133 6.5.3 Reflectance ....................................................................................................... 133

**6.6 Hierarchical Modeling Using a Scene Graph**.............................................................. 138 6.6.1 Motivation for Modular Modeling..................................................................... 138 6.6.2 Top-Down Design of Component Hierarchy...................................................... 139 6.6.3 Bottom-Up Construction and Composition........................................................ 140 6.6.4 Reuse of Components....................................................................................... 144 **6.7 Discussion** .................................................................................................................... 147

**7 Essential Mathematics and the Geometry of 2-Space and 3-Space**...................... 149

We review basic facts about equations of lines and planes, areas, convexity, and parameterization. We discuss inside-outside testing for points in polygons. We describe barycentric coordinates, and present the notational conventions that are used throughout the book, including the notation for functions. We present a graphics-centric view of vectors, and introduce the notion of covectors.

Contents xiii

**7.1 Introduction**................................................................................................................. 149 **7.2 Notation** ....................................................................................................................... 150 **7.3 Sets**............................................................................................................................... 150 **7.4 Functions** ..................................................................................................................... 151

7.4.1 Inverse Tangent Functions................................................................................. 152 **7.5 Coordinates**.................................................................................................................. 153 **7.6 Operations on Coordinates**.......................................................................................... 153

7.6.1 Vectors ............................................................................................................. 155 7.6.2 How to Think About Vectors............................................................................. 156 7.6.3 Length of a Vector ............................................................................................ 157 7.6.4 Vector Operations............................................................................................. 157 7.6.5 Matrix Multiplication........................................................................................ 161 7.6.6 Other Kinds of Vectors...................................................................................... 162 7.6.7 Implicit Lines ................................................................................................... 164 7.6.8 An Implicit Description of a Line in a Plane...................................................... 164 7.6.9 What About *y* = *mx* + *b*?.................................................................................. 165

**7.7 Intersections of Lines**................................................................................................... 165 7.7.1 Parametric-Parametric Line Intersection............................................................ 166 7.7.2 Parametric-Implicit Line Intersection ................................................................ 167

**7.8 Intersections, More Generally**..................................................................................... 167 7.8.1 Ray-Plane Intersection ...................................................................................... 168 7.8.2 Ray-Sphere Intersection.................................................................................... 170

**7.9 Triangles** ...................................................................................................................... 171 7.9.1 Barycentric Coordinates.................................................................................... 172 7.9.2 Triangles in Space............................................................................................. 173 7.9.3 Half-Planes and Triangles................................................................................. 174

**7.10 Polygons**....................................................................................................................... 175 7.10.1 Inside/Outside Testing ...................................................................................... 175 7.10.2 Interiors of Nonsimple Polygons....................................................................... 177 7.10.3 The Signed Area of a Plane Polygon: Divide and Conquer ................................ 177 7.10.4 Normal to a Polygon in Space ........................................................................... 178 7.10.5 Signed Areas for More General Polygons.......................................................... 179 7.10.6 The Tilting Principle ......................................................................................... 180 7.10.7 Analogs of Barycentric Coordinates.................................................................. 182

**7.11 Discussion** .................................................................................................................... 182 **7.12 Exercises** ...................................................................................................................... 182

**8 A Simple Way to Describe Shape in 2D and 3D**........................................................ 187

The triangle mesh is a fundamental structure in graphics, widely used for representing shape. We describe 1D meshes (polylines) in 2D and generalize to 2D meshes in 3D. We discuss several rep resentations for triangle meshes, simple operations on meshes such as computing the boundary, and determining whether a mesh is oriented.

**8.1 Introduction**................................................................................................................. 187 **8.2 “Meshes” in 2D: Polylines** ........................................................................................... 189 8.2.1 Boundaries ....................................................................................................... 190 8.2.2 A Data Structure for 1D Meshes ....................................................................... 191 **8.3 Meshes in 3D**................................................................................................................ 192

xiv Contents

8.3.1 Manifold Meshes.............................................................................................. 193 8.3.2 Nonmanifold Meshes........................................................................................ 195 8.3.3 Memory Requirements for Mesh Structures ...................................................... 196 8.3.4 A Few Mesh Operations.................................................................................... 197 8.3.5 Edge Collapse................................................................................................... 197 8.3.6 Edge Swap........................................................................................................ 197

**8.4 Discussion and Further Reading** ................................................................................. 198 **8.5 Exercises** ...................................................................................................................... 198

**9 Functions on Meshes**........................................................................................................ 201

A real-valued function defined at the vertices of a mesh can be extended linearly across each face by barycentric interpolation to define a function on the entire mesh. Such extensions are used in texture mapping, for instance. By considering what happens when a single vertex value is 1, and all others are 0, we see that all our piecewise-linear extensions are combinations of certain basic piecewise linear mesh functions; replacing these basis functions with other, smoother functions can lead to smoother interpolation of values.

**9.1 Introduction**................................................................................................................. 201 **9.2 Code for Barycentric Interpolation**............................................................................. 203 9.2.1 A Different View of Linear Interpolation........................................................... 207 9.2.2 Scanline Interpolation ....................................................................................... 208 **9.3 Limitations of Piecewise Linear Extension** ................................................................. 210 9.3.1 Dependence on Mesh Structure......................................................................... 211 **9.4 Smoother Extensions**................................................................................................... 211 9.4.1 Nonconvex Spaces............................................................................................ 211 9.4.2 Which Interpolation Method Should I Really Use?............................................ 213 **9.5 Functions Multiply Defined at Vertices**....................................................................... 213 **9.6 Application: Texture Mapping** .................................................................................... 214 9.6.1 Assignment of Texture Coordinates................................................................... 215 9.6.2 Details of Texture Mapping............................................................................... 216 9.6.3 Texture-Mapping Problems............................................................................... 216 **9.7 Discussion** .................................................................................................................... 217 **9.8 Exercises** ...................................................................................................................... 217

**10 Transformations in Two Dimensions** ........................................................................... 221

Linear and affine transformations are the building blocks of graphics. They occur in modeling, in rendering, in animation, and in just about every other context imaginable. They are the natural tools for transforming objects represented as meshes, because they preserve the mesh structure perfectly. We introduce linear and affine transformations in the plane, because most of the interesting phenom ena are present there, the exception being the behavior of rotations in three dimensions, which we discuss in Chapter 11. We also discuss the relationship of transformations to matrices, the use of homogeneous coordinates, the uses of hierarchies of transformations in modeling, and the idea of coordinate “frames.”

**10.1 Introduction**................................................................................................................. 221 **10.2 Five Examples**.............................................................................................................. 222

Contents xv

**10.3 Important Facts about Transformations**..................................................................... 224 10.3.1 Multiplication by a Matrix Is a Linear Transformation....................................... 224 10.3.2 Multiplication by a Matrix Is the *Only* Linear Transformation ........................... 224 10.3.3 Function Composition and Matrix Multiplication Are Related........................... 225 10.3.4 Matrix Inverse and Inverse Functions Are Related............................................. 225 10.3.5 Finding the Matrix for a Transformation............................................................ 226 10.3.6 Transformations and Coordinate Systems.......................................................... 229 10.3.7 Matrix Properties and the Singular Value Decomposition .................................. 230 10.3.8 Computing the SVD ......................................................................................... 231 10.3.9 The SVD and Pseudoinverses............................................................................ 231

**10.4 Translation**................................................................................................................... 233 **10.5 Points and Vectors Again**............................................................................................. 234 **10.6 Why Use 3** *×* **3 Matrices Instead of a Matrix and a Vector?**...................................... 235 **10.7 Windowing Transformations**....................................................................................... 236 **10.8 Building 3D Transformations**...................................................................................... 237 **10.9 Another Example of Building a 2D Transformation**................................................... 238 **10.10 Coordinate Frames**...................................................................................................... 240 **10.11 Application: Rendering from a Scene Graph**.............................................................. 241

10.11.1 Coordinate Changes in Scene Graphs................................................................ 248 **10.12 Transforming Vectors and Covectors**.......................................................................... 250 10.12.1 Transforming Parametric Lines......................................................................... 254 **10.13 More General Transformations**................................................................................... 254 **10.14 Transformations versus Interpolation**......................................................................... 259 **10.15 Discussion and Further Reading** ................................................................................. 259 **10.16 Exercises** ...................................................................................................................... 260

**11 Transformations in Three Dimensions**........................................................................ 263

Transformations in 3-space are analogous to those in the plane, except for rotations: In the plane, we can swap the order in which we perform two rotations about the origin without altering the result; in 3-space, we generally cannot. We discuss the group of rotations in 3-space, the use of quaternions to represent rotations, interpolating between quaternions, and a more general technique for interpolating among any sequence of transformations, provided they are “close enough” to one another. Some of these techniques are applied to user-interface designs in Chapter 21.

**11.1 Introduction**................................................................................................................. 263 11.1.1 Projective Transformation Theorems................................................................. 265 **11.2 Rotations**...................................................................................................................... 266 11.2.1 Analogies between Two and Three Dimensions................................................. 266 11.2.2 Euler Angles..................................................................................................... 267 11.2.3 Axis-Angle Description of a Rotation ............................................................... 269 11.2.4 Finding an Axis and Angle from a Rotation Matrix ........................................... 270 11.2.5 Body-Centered Euler Angles............................................................................. 272 11.2.6 Rotations and the 3-Sphere ............................................................................... 273 11.2.7 Stability of Computations ................................................................................. 278 **11.3 Comparing Representations**........................................................................................ 278 **11.4 Rotations versus Rotation Specifications** .................................................................... 279 **11.5 Interpolating Matrix Transformations**........................................................................ 280 **11.6 Virtual Trackball and Arcball**..................................................................................... 280

xvi Contents

**11.7 Discussion and Further Reading** ................................................................................. 283 **11.8 Exercises** ...................................................................................................................... 284

**12 A 2D and 3D Transformation Library for Graphics** ............................................... 287

Because we represent so many things in graphics with arrays of three floating-point numbers (RGB colors, locations in 3-space, vectors in 3-space, covectors in 3-space, etc.) it’s very easy to make conceptual mistakes in code, performing operations (like adding the coordinates of two points) that don’t make sense. We present a sample mathematics library that you can use to avoid such problems. While such a library may have no place in high-performance graphics, where the overhead of type checking would be unreasonable, it can be very useful in the development of programs in their early stages.

**12.1 Introduction**................................................................................................................. 287 **12.2 Points and Vectors** ....................................................................................................... 288 **12.3 Transformations** .......................................................................................................... 288

12.3.1 Efficiency ......................................................................................................... 289 **12.4 Specification of Transformations**................................................................................. 290 **12.5 Implementation**............................................................................................................ 290

12.5.1 Projective Transformations................................................................................ 291 **12.6 Three Dimensions**........................................................................................................ 293 **12.7 Associated Transformations**........................................................................................ 294 **12.8 Other Structures**.......................................................................................................... 294 **12.9 Other Approaches**........................................................................................................ 295 **12.10 Discussion** .................................................................................................................... 297 **12.11 Exercises** ...................................................................................................................... 297

**13 Camera Specifications and Transformations**............................................................. 299

To convert a model of a 3D scene to a 2D image seen from a particular point of view, we have to specify the view precisely. The rendering process turns out to be particularly simple if the camera is at the origin, looking along a coordinate axis, and if the field of view is 90*◦* in each direction. We therefore transform the general problem to the more specific one. We discuss how the virtual camera is specified, and how we transform any rendering problem to one in which the camera is in a standard position with standard characteristics. We also discuss the specification of parallel (as opposed to perspective) views.

**13.1 Introduction**................................................................................................................. 299 **13.2 A 2D Example**.............................................................................................................. 300 **13.3 Perspective Camera Specification** ............................................................................... 301 **13.4 Building Transformations from a View Specification** ................................................. 303 **13.5 Camera Transformations and the Rasterizing Renderer Pipeline** ............................. 310 **13.6 Perspective and *z*-values**.............................................................................................. 313 **13.7 Camera Transformations and the Modeling Hierarchy** ............................................. 313 **13.8 Orthographic Cameras**................................................................................................ 315

13.8.1 Aspect Ratio and Field of View......................................................................... 316 **13.9 Discussion and Further Reading** ................................................................................. 317 **13.10 Exercises** ...................................................................................................................... 318

Contents xvii

**14 Standard Approximations and Representations**....................................................... 321

The real world contains too much detail to simulate efficiently from first principles of physics and geometry. Models make graphics computationally tractable but introduce restrictions and errors. We explore some pervasive approximations and their limitations. In many cases, we have a choice between competing models with different properties.

**14.1 Introduction**................................................................................................................. 321 **14.2 Evaluating Representations**......................................................................................... 322 14.2.1 The Value of Measurement ............................................................................... 323 14.2.2 Legacy Models ................................................................................................. 324 **14.3 Real Numbers**.............................................................................................................. 324 14.3.1 Fixed Point ....................................................................................................... 325 14.3.2 Floating Point ................................................................................................... 326 14.3.3 Buffers ............................................................................................................. 327 **14.4 Building Blocks of Ray Optics**..................................................................................... 330 14.4.1 Light ................................................................................................................ 330 14.4.2 Emitters............................................................................................................ 334 14.4.3 Light Transport................................................................................................. 335 14.4.4 Matter............................................................................................................... 336 14.4.5 Cameras ........................................................................................................... 336 **14.5 Large-Scale Object Geometry** ..................................................................................... 337 14.5.1 Meshes............................................................................................................. 338 14.5.2 Implicit Surfaces............................................................................................... 341 14.5.3 Spline Patches and Subdivision Surfaces........................................................... 343 14.5.4 Heightfields...................................................................................................... 344 14.5.5 Point Sets ......................................................................................................... 345 **14.6 Distant Objects** ............................................................................................................ 346 14.6.1 Level of Detail.................................................................................................. 347 14.6.2 Billboards and Impostors .................................................................................. 347 14.6.3 Skyboxes.......................................................................................................... 348 **14.7 Volumetric Models**....................................................................................................... 349 14.7.1 Finite Element Models...................................................................................... 349 14.7.2 Voxels .............................................................................................................. 349 14.7.3 Particle Systems................................................................................................ 350 14.7.4 Fog................................................................................................................... 351 **14.8 Scene Graphs**............................................................................................................... 351 **14.9 Material Models**........................................................................................................... 353 14.9.1 Scattering Functions (BSDFs)........................................................................... 354 14.9.2 Lambertian ....................................................................................................... 358 14.9.3 Normalized Blinn-Phong .................................................................................. 359 **14.10 Translucency and Blending** ......................................................................................... 361 14.10.1 Blending........................................................................................................... 362 14.10.2 Partial Coverage (*α*).......................................................................................... 364 14.10.3 Transmission .................................................................................................... 367 14.10.4 Emission........................................................................................................... 369 14.10.5 Bloom and Lens Flare....................................................................................... 369 **14.11 Luminaire Models** ....................................................................................................... 369 14.11.1 The Radiance Function ..................................................................................... 370 14.11.2 Direct and Indirect Light................................................................................... 370

xviii Contents

14.11.3 Practical and Artistic Considerations................................................................. 370 14.11.4 Rectangular Area Light..................................................................................... 377 14.11.5 Hemisphere Area Light..................................................................................... 378 14.11.6 Omni-Light....................................................................................................... 379 14.11.7 Directional Light .............................................................................................. 380 14.11.8 Spot Light......................................................................................................... 381 14.11.9 A Unified Point-Light Model ............................................................................ 382

**14.12 Discussion** .................................................................................................................... 384 **14.13 Exercises** ...................................................................................................................... 385

**15 Ray Casting and Rasterization**...................................................................................... 387

A 3D renderer identifies the surface that covers each pixel of an image, and then executes some shading routine to compute the value of the pixel. We introduce a set of coverage algorithms and some straw-man shading routines, and revisit the graphics pipeline abstraction. These are practical design points arising from general principles of geometry and processor architectures.

For coverage, we derive the ray-casting and rasterization algorithms and then build the complete source code for a render on top of it. This requires graphics-specific debugging techniques such as visualizing intermediate results. Architecture-aware optimizations dramatically increase the per formance of these programs, albeit by limiting abstraction. Alternatively, we can move abstractions above the pipeline to enable dedicated graphics hardware. APIs abstracting graphics processing units (GPUs) enable efficient rasterization implementations. We port our render to the programmable shad ing framework common to such APIs.

**15.1 Introduction**................................................................................................................. 387 **15.2 High-Level Design Overview**....................................................................................... 388 15.2.1 Scattering ......................................................................................................... 388 15.2.2 Visible Points ................................................................................................... 390 15.2.3 Ray Casting: Pixels First................................................................................... 391 15.2.4 Rasterization: Triangles First............................................................................. 391 **15.3 Implementation Platform** ............................................................................................ 393 15.3.1 Selection Criteria .............................................................................................. 393 15.3.2 Utility Classes .................................................................................................. 395 15.3.3 Scene Representation........................................................................................ 400 15.3.4 A Test Scene..................................................................................................... 402 **15.4 A Ray-Casting Renderer** ............................................................................................. 403 15.4.1 Generating an Eye Ray ..................................................................................... 404 15.4.2 Sampling Framework: Intersect and Shade ........................................................ 407 15.4.3 Ray-Triangle Intersection.................................................................................. 408 15.4.4 Debugging........................................................................................................ 411 15.4.5 Shading ............................................................................................................ 412 15.4.6 Lambertian Scattering....................................................................................... 413 15.4.7 Glossy Scattering.............................................................................................. 414 15.4.8 Shadows........................................................................................................... 414 15.4.9 A More Complex Scene .................................................................................... 417 **15.5 Intermezzo** ................................................................................................................... 417 **15.6 Rasterization**................................................................................................................ 418 15.6.1 Swapping the Loops ......................................................................................... 418 15.6.2 Bounding-Box Optimization ............................................................................. 420 15.6.3 Clipping to the Near Plane ................................................................................ 422 15.6.4 Increasing Efficiency ........................................................................................ 422

Contents xix

15.6.5 Rasterizing Shadows......................................................................................... 428 15.6.6 Beyond the Bounding Box ................................................................................ 429 **15.7 Rendering with a Rasterization API** ........................................................................... 432 15.7.1 The Graphics Pipeline....................................................................................... 432 15.7.2 Interface ........................................................................................................... 434 **15.8 Performance and Optimization**................................................................................... 444 15.8.1 Abstraction Considerations............................................................................... 444 15.8.2 Architectural Considerations............................................................................. 444 15.8.3 Early-Depth-Test Example ................................................................................ 445 15.8.4 When Early Optimization Is Good .................................................................... 446 15.8.5 Improving the Asymptotic Bound ..................................................................... 447 **15.9 Discussion** .................................................................................................................... 447 **15.10 Exercises** ...................................................................................................................... 449

**16 Survey of Real-Time 3D Graphics Platforms**............................................................ 451

There is great diversity in the feature sets and design goals among 3D graphics platforms. Some are thin layers that bring the application as close to the hardware as possible for optimum perfor mance and control; others provide a thick layer of data structures for the storage and manipulation of complex scenes; and at the top of the power scale are the game-development environments that additionally provide advanced features like physics and joint/skin simulation. Platforms supporting games render with the highest possible speed to ensure interactivity, while those used by the spe cial effects industry sacrifice speed for the utmost in image quality. We present a broad overview of modern 3D platforms with an emphasis on the design goals behind the variations.

**16.1 Introduction**................................................................................................................. 451 16.1.1 Evolution from Fixed-Function to Programmable Rendering Pipeline................ 452 **16.2 The Programmer’s Model: OpenGL Compatibility (Fixed-Function) Profile** ........... 454 16.2.1 OpenGL Program Structure .............................................................................. 455 16.2.2 Initialization and the Main Loop ....................................................................... 456 16.2.3 Lighting and Materials...................................................................................... 458 16.2.4 Geometry Processing ........................................................................................ 458 16.2.5 Camera Setup ................................................................................................... 460 16.2.6 Drawing Primitives........................................................................................... 461 16.2.7 Putting It All Together—Part 1: Static Frame .................................................... 462 16.2.8 Putting It All Together—Part 2: Dynamics........................................................ 463 16.2.9 Hierarchical Modeling ...................................................................................... 463 16.2.10 Pick Correlation................................................................................................ 464 **16.3 The Programmer’s Model: OpenGL Programmable Pipeline** ................................... 464 16.3.1 Abstract View of a Programmable Pipeline ....................................................... 464 16.3.2 The Nature of the Core API .............................................................................. 466 **16.4 Architectures of Graphics Applications**...................................................................... 466 16.4.1 The Application Model..................................................................................... 466 16.4.2 The Application-Model-to-IM-Platform Pipeline (AMIP).................................. 468 16.4.3 Scene-Graph Middleware.................................................................................. 474 16.4.4 Graphics Application Platforms ........................................................................ 477 **16.5 3D on Other Platforms** ................................................................................................ 478 16.5.1 3D on Mobile Devices ...................................................................................... 479 16.5.2 3D in Browsers................................................................................................. 479 **16.6 Discussion** .................................................................................................................... 479

xx Contents

**17 Image Representation and Manipulation** ................................................................... 481

Much of graphics produces *images* as output. We describe how images are stored, what information they can contain, and what they can represent, along with the importance of knowing the precise meaning of the pixels in an image file. We show how to composite images (i.e., blend, overlay, and otherwise merge them) using coverage maps, and how to simply represent images at multiple scales with MIP mapping.

**17.1 Introduction**................................................................................................................. 481 **17.2 What *Is* an Image?**....................................................................................................... 482 17.2.1 The Information Stored in an Image .................................................................. 482 **17.3 Image File Formats**...................................................................................................... 483 17.3.1 Choosing an Image Format ............................................................................... 484 **17.4 Image Compositing**...................................................................................................... 485 17.4.1 The Meaning of a Pixel During Image Compositing .......................................... 486 17.4.2 Computing ***U*** over ***V***......................................................................................... 486 17.4.3 Simplifying Compositing.................................................................................. 487 17.4.4 Other Compositing Operations.......................................................................... 488 17.4.5 Physical Units and Compositing........................................................................ 489 **17.5 Other Image Types** ...................................................................................................... 490 17.5.1 Nomenclature ................................................................................................... 491 **17.6 MIP Maps**.................................................................................................................... 491 **17.7 Discussion and Further Reading** ................................................................................. 492 **17.8 Exercises** ...................................................................................................................... 493

**18 Images and Signal Processing**........................................................................................ 495

The pattern of light arriving at a camera sensor can be thought of as a function defined on a 2D rectangle, the value at each point being the light energy density arriving there. The resultant image is an array of values, each one arrived at by some sort of averaging of the input function. The rela tionship between these two functions—one defined on a continuous 2D rectangle, the other defined on a rectangular grid of points—is a deep one. We study the relationship with the tools of Fourier analysis, which lets us understand what parts of the incoming signal can be accurately captured by the discrete signal. This understanding helps us avoid a wide range of image problems, including “jaggies” (ragged edges). It’s also the basis for understanding other phenomena in graphics, such as moiré patterns in textures.

**18.1 Introduction**................................................................................................................. 495 18.1.1 A Broad Overview............................................................................................ 495 18.1.2 Important Terms, Assumptions, and Notation.................................................... 497

**18.2 Historical Motivation**................................................................................................... 498 **18.3 Convolution**.................................................................................................................. 500 **18.4 Properties of Convolution**............................................................................................ 503 **18.5 Convolution-like Computations**................................................................................... 504 **18.6 Reconstruction**............................................................................................................. 505 **18.7 Function Classes**.......................................................................................................... 505 **18.8 Sampling** ...................................................................................................................... 507 **18.9 Mathematical Considerations** ..................................................................................... 508

18.9.1 Frequency-Based Synthesis and Analysis.......................................................... 509 **18.10 The Fourier Transform: Definitions**............................................................................ 511

Contents xxi

**18.11 The Fourier Transform of a Function on an Interval** ................................................. 511 18.11.1 Sampling and Band Limiting in an Interval ....................................................... 514 **18.12 Generalizations to Larger Intervals and All of R** ....................................................... 516 **18.13 Examples of Fourier Transforms** ................................................................................ 516 18.13.1 Basic Examples ................................................................................................ 516 18.13.2 The Transform of a Box Is a Sinc...................................................................... 517 18.13.3 An Example on an Interval................................................................................ 518 **18.14 An Approximation of Sampling** .................................................................................. 519 **18.15 Examples Involving Limits**.......................................................................................... 519 18.15.1 Narrow Boxes and the Delta Function ............................................................... 519 18.15.2 The Comb Function and Its Transform.............................................................. 520 **18.16 The Inverse Fourier Transform**................................................................................... 520 **18.17 Properties of the Fourier Transform**........................................................................... 521 **18.18 Applications**................................................................................................................. 522 18.18.1 Band Limiting .................................................................................................. 522 18.18.2 Explaining Replication in the Spectrum............................................................. 523 **18.19 Reconstruction and Band Limiting** ............................................................................. 524 **18.20 Aliasing Revisited** ........................................................................................................ 527 **18.21 Discussion and Further Reading** ................................................................................. 529 **18.22 Exercises** ...................................................................................................................... 532

**19 Enlarging and Shrinking Images**.................................................................................. 533

We apply the ideas of the previous two chapters to a concrete example—enlarging and shrinking of images—to illustrate their use in practice. We see that when an image, conventionally represented, is shrunk, problems will arise unless certain high-frequency information is removed before the shrink ing process.

**19.1 Introduction**................................................................................................................. 533 **19.2 Enlarging an Image** ..................................................................................................... 534 **19.3 Scaling Down an Image** ............................................................................................... 537 **19.4 Making the Algorithms Practical**................................................................................ 538 **19.5 Finite-Support Approximations** .................................................................................. 540

19.5.1 Practical Band Limiting .................................................................................... 541 **19.6 Other Image Operations and Efficiency**...................................................................... 541 **19.7 Discussion and Further Reading** ................................................................................. 544 **19.8 Exercises** ...................................................................................................................... 545

**20 Textures and Texture Mapping** ..................................................................................... 547

Texturing, and its variants, add visual richness to models without introducing geometric complexity. We discuss basic texturing and its implementation in software, and some of its variants, like bump mapping and displacement mapping, and the use of 1D and 3D textures. We also discuss the creation of texture correspondences (assigning texture coordinates to points on a mesh) and of the texture images themselves, through techniques as varied as “painting the model” and probabilistic texture synthesis algorithms.

**20.1 Introduction**................................................................................................................. 547 **20.2 Variations of Texturing**................................................................................................ 549

xxii Contents

20.2.1 Environment Mapping ...................................................................................... 549 20.2.2 Bump Mapping................................................................................................. 550 20.2.3 Contour Drawing .............................................................................................. 551

**20.3 Building Tangent Vectors from a Parameterization** ................................................... 552 **20.4 Codomains for Texture Maps**...................................................................................... 553 **20.5 Assigning Texture Coordinates** ................................................................................... 555 **20.6 Application Examples**.................................................................................................. 557 **20.7 Sampling, Aliasing, Filtering, and Reconstruction** ..................................................... 557 **20.8 Texture Synthesis**......................................................................................................... 559

20.8.1 Fourier-like Synthesis....................................................................................... 559 20.8.2 Perlin Noise...................................................................................................... 560 20.8.3 Reaction-Diffusion Textures.............................................................................. 561

**20.9 Data-Driven Texture Synthesis**.................................................................................... 562 **20.10 Discussion and Further Reading** ................................................................................. 564 **20.11 Exercises** ...................................................................................................................... 565

**21 Interaction Techniques**.................................................................................................... 567

Certain interaction techniques use a substantial amount of the mathematics of transformations, and therefore are more suitable for a book like ours than one that concentrates on the design of the interaction itself, and the human factors associated with that design. We illustrate these ideas with three 3D manipulators—the arcball, trackball, and Unicam—and with a a multitouch interface for manipulating images.

**21.1 Introduction**................................................................................................................. 567 **21.2 User Interfaces and Computer Graphics**.................................................................... 567 21.2.1 Prescriptions..................................................................................................... 571 21.2.2 Interaction Event Handling ............................................................................... 573 **21.3 Multitouch Interaction for 2D Manipulation**.............................................................. 574 21.3.1 Defining the Problem........................................................................................ 575 21.3.2 Building the Program........................................................................................ 576 21.3.3 The Interactor................................................................................................... 576 **21.4 Mouse-Based Object Manipulation in 3D**................................................................... 580 21.4.1 The Trackball Interface ..................................................................................... 580 21.4.2 The Arcball Interface ........................................................................................ 584 **21.5 Mouse-Based Camera Manipulation: Unicam**............................................................ 584 21.5.1 Translation........................................................................................................ 585 21.5.2 Rotation............................................................................................................ 586 21.5.3 Additional Operations....................................................................................... 587 21.5.4 Evaluation ........................................................................................................ 587 **21.6 Choosing the Best Interface**......................................................................................... 587 **21.7 Some Interface Examples**............................................................................................ 588 21.7.1 First-Person-Shooter Controls........................................................................... 588 21.7.2 3ds Max Transformation Widget ....................................................................... 588 21.7.3 Photoshop’s Free-Transform Mode ................................................................... 589 21.7.4 Chateau ............................................................................................................ 589 21.7.5 Teddy ............................................................................................................... 590 21.7.6 Grabcut and Selection by Strokes...................................................................... 590 **21.8 Discussion and Further Reading** ................................................................................. 591 **21.9 Exercises** ...................................................................................................................... 593

Contents xxiii

**22 Splines and Subdivision Curves**.................................................................................... 595

Splines are, informally, curves that pass through or near a sequence of “control points.” They’re used to describe shapes, and to control the motion of objects in animations, among other things. Splines make sense not only in the plane, but also in 3-space and in 1-space, where they provide a means of interpolating a sequence of values with various degrees of continuity. Splines, as a modeling tool in graphics, have been in part supplanted by subdivision curves (which we saw in the form of corner cutting curves in Chapter 4) and subdivision surfaces. The two classes—splines and subdivision—are closely related. We demonstrate this for curves in this chapter; a similar approach works for surfaces.

**22.1 Introduction**................................................................................................................. 595 **22.2 Basic Polynomial Curves**............................................................................................. 595 **22.3 Fitting a Curve Segment between Two Curves: The Hermite Curve** ......................... 595

22.3.1 Bézier Curves................................................................................................... 598 **22.4 Gluing Together Curves and the Catmull-Rom Spline** ............................................... 598 22.4.1 Generalization of Catmull-Rom Splines............................................................ 601 22.4.2 Applications of Catmull-Rom Splines............................................................... 602 **22.5 Cubic B-splines**............................................................................................................ 602 22.5.1 Other B-splines................................................................................................. 604 **22.6 Subdivision Curves**...................................................................................................... 604 **22.7 Discussion and Further Reading** ................................................................................. 605 **22.8 Exercises** ...................................................................................................................... 605

**23 Splines and Subdivision Surfaces**.................................................................................. 607

Spline surfaces and subdivision surfaces are natural generalizations of spline and subdivision curves. Surfaces are built from rectangular patches, and when these meet four at a vertex, the generalization is reasonably straightforward. At vertices where the degree is not four, certain challenges arise, and dealing with these “exceptional vertices” requires care. Just as in the case of curves, subdivision surfaces, away from exceptional vertices, turn out to be identical to spline surfaces. We discuss spline patches, Catmull-Clark subdivision, other subdivision approaches, and the problems of exceptional points.

**23.1 Introduction**................................................................................................................. 607 **23.2 Bézier Patches**.............................................................................................................. 608 **23.3 Catmull-Clark Subdivision Surfaces**........................................................................... 610 **23.4 Modeling with Subdivision Surfaces**........................................................................... 613 **23.5 Discussion and Further Reading** ................................................................................. 614

**24 Implicit Representations of Shape**................................................................................ 615

Implicit curves are defined as the level set of some function on the plane; on a weather map, the isotherm lines constitute implicit curves. By choosing particular functions, we can make the shapes of these curves controllable. The same idea applies in space to define implicit surfaces. In each case, it’s not too difficult to convert an implicit representation to a mesh representation that approximates the surface. But the implicit representation itself has many advantages. Finding a ray-shape intersection with an implicit surface reduces to root finding, for instance, and it’s easy to combine implicit shapes with operators that result in new shapes without sharp corners.

xxiv Contents

**24.1 Introduction**................................................................................................................. 615 **24.2 Implicit Curves**............................................................................................................ 616 **24.3 Implicit Surfaces**.......................................................................................................... 619 **24.4 Representing Implicit Functions**................................................................................. 621

24.4.1 Interpolation Schemes....................................................................................... 621 24.4.2 Splines.............................................................................................................. 623 24.4.3 Mathematical Models and Sampled Implicit Representations............................. 623

**24.5 Other Representations of Implicit Functions**.............................................................. 624 **24.6 Conversion to Polyhedral Meshes** ............................................................................... 625 24.6.1 Marching Cubes ............................................................................................... 628 **24.7 Conversion from Polyhedral Meshes to Implicits**....................................................... 629 **24.8 Texturing Implicit Models**........................................................................................... 629 24.8.1 Modeling Transformations and Textures............................................................ 630 **24.9 Ray Tracing Implicit Surfaces**..................................................................................... 631 **24.10 Implicit Shapes in Animation**...................................................................................... 631 **24.11 Discussion and Further Reading** ................................................................................. 632 **24.12 Exercises** ...................................................................................................................... 633

**25 Meshes**.................................................................................................................................. 635

Meshes are a dominant structure in today’s graphics. They serve as approximations to smooth curves and surfaces, and much mathematics from the smooth category can be transferred to work with meshes. Certain special classes of meshes—heightfield meshes, and very regular meshes—support fast algorithms particularly well. We discuss level of detail in the context of meshes, where practical algorithms abound, but also in a larger context. We conclude with some applications.

**25.1 Introduction**................................................................................................................. 635 **25.2 Mesh Topology**............................................................................................................. 637 25.2.1 Triangulated Surfaces and Surfaces with Boundary ........................................... 637 25.2.2 Computing and Storing Adjacency.................................................................... 638 25.2.3 More Mesh Terminology................................................................................... 641 25.2.4 Embedding and Topology ................................................................................. 642 **25.3 Mesh Geometry** ........................................................................................................... 643 25.3.1 Mesh Meaning.................................................................................................. 644 **25.4 Level of Detail**.............................................................................................................. 645 25.4.1 Progressive Meshes........................................................................................... 649 25.4.2 Other Mesh Simplification Approaches............................................................. 652 **25.5 Mesh Applications 1: Marching Cubes, Mesh Repair, and Mesh Improvement**........ 652 25.5.1 Marching Cubes Variants.................................................................................. 652 25.5.2 Mesh Repair ..................................................................................................... 654 25.5.3 Differential or Laplacian Coordinates................................................................ 655 25.5.4 An Application of Laplacian Coordinates.......................................................... 657 **25.6 Mesh Applications 2: Deformation Transfer and Triangle-Order Optimization** ....... 660 25.6.1 Deformation Transfer........................................................................................ 660 25.6.2 Triangle Reordering for Hardware Efficiency .................................................... 664 **25.7 Discussion and Further Reading** ................................................................................. 667 **25.8 Exercises** ...................................................................................................................... 668

Contents xxv

**26 Light** ..................................................................................................................................... 669

We discuss the basic physics of light, starting from blackbody radiation, and the relevance of this physics to computer graphics. In particular, we discuss both the wave and particle descriptions of light, polarization effects, and diffraction. We then discuss the measurement of light, including the various units of measure, and the continuum assumption implicit in these measurements. We focus on the radiance, from which all other radiometric terms can be derived through integration, and which is constant along rays in empty space. Because of the dependence on integration, we discuss solid angles and integration over these. Because the radiance field in most scenes is too complex to express in simple algebraic terms, integrals of radiance are almost always computed stochastically, and so we introduce stochastic integration. Finally, we discuss reflectance and transmission, their measurement, and the challenges of computing integrals in which the integrands have substantial variation (like the specular and nonspecular parts of the reflection from a glossy surface).

**26.1 Introduction**................................................................................................................. 669 **26.2 The Physics of Light** .................................................................................................... 669 **26.3 The Microscopic View** ................................................................................................. 670 **26.4 The Wave Nature of Light**........................................................................................... 674

26.4.1 Diffraction........................................................................................................ 677 26.4.2 Polarization ...................................................................................................... 677 26.4.3 Bending of Light at an Interface........................................................................ 679

**26.5 Fresnel’s Law and Polarization**................................................................................... 681 26.5.1 Radiance Computations and an “Unpolarized” Form of Fresnel’s Equations...... 683 **26.6 Modeling Light as a Continuous Flow** ........................................................................ 683 26.6.1 A Brief Introduction to Probability Densities..................................................... 684 26.6.2 Further Light Modeling..................................................................................... 686 26.6.3 Angles and Solid Angles................................................................................... 686 26.6.4 Computations with Solid Angles....................................................................... 688 26.6.5 An Important Change of Variables .................................................................... 690 **26.7 Measuring Light** .......................................................................................................... 692 26.7.1 Radiometric Terms............................................................................................ 694 26.7.2 Radiance........................................................................................................... 694 26.7.3 Two Radiance Computations............................................................................. 695 26.7.4 Irradiance ......................................................................................................... 697 26.7.5 Radiant Exitance............................................................................................... 699 26.7.6 Radiant Power or Radiant Flux.......................................................................... 699 **26.8 Other Measurements**................................................................................................... 700 **26.9 The Derivative Approach** ............................................................................................ 700 **26.10 Reflectance** ................................................................................................................... 702 26.10.1 Related Terms................................................................................................... 704 26.10.2 Mirrors, Glass, Reciprocity, and the BRDF........................................................ 705 26.10.3 Writing *L* in Different Ways.............................................................................. 706 **26.11 Discussion and Further Reading** ................................................................................. 707 **26.12 Exercises** ...................................................................................................................... 707

**27 Materials and Scattering**................................................................................................. 711

The appearance of an object made of some material is determined by the interaction of that material with the light in the scene. The interaction (for fairly homogeneous materials) is described by the

xxvi Contents

reflection and transmission distribution functions, at least for at-the-surface scattering. We present several different models for these, ranging from the purely empirical to those incorporating various degrees of physical realism, and observe their limitations as well. We briefly discuss scattering from volumetric media like smoke and fog, and the kind of subsurface scattering that takes place in media like skin and milk. Anticipating our use of these material models in rendering, we also discuss the software interface a material model must support to be used effectively.

**27.1 Introduction**................................................................................................................. 711 **27.2 Object-Level Scattering**............................................................................................... 711 **27.3 Surface Scattering** ....................................................................................................... 712

27.3.1 Impulses........................................................................................................... 713 27.3.2 Types of Scattering Models............................................................................... 713 27.3.3 Physical Constraints on Scattering .................................................................... 713

**27.4 Kinds of Scattering** ...................................................................................................... 714 **27.5 Empirical and Phenomenological Models for Scattering**............................................ 717 27.5.1 Mirror “Scattering”........................................................................................... 717 27.5.2 Lambertian Reflectors....................................................................................... 719 27.5.3 The Phong and Blinn-Phong Models................................................................. 721 27.5.4 The Lafortune Model........................................................................................ 723 27.5.5 Sampling .......................................................................................................... 724 **27.6 Measured Models**......................................................................................................... 725 **27.7 Physical Models for Specular and Diffuse Reflection** ................................................. 726 **27.8 Physically Based Scattering Models**............................................................................ 727 27.8.1 The Fresnel Equations, Revisited ...................................................................... 727 27.8.2 The Torrance-Sparrow Model............................................................................ 729 27.8.3 The Cook-Torrance Model ................................................................................ 731 27.8.4 The Oren-Nayar Model..................................................................................... 732 27.8.5 Wave Theory Models........................................................................................ 734 **27.9 Representation Choices** ............................................................................................... 734 **27.10 Criteria for Evaluation** ................................................................................................ 734 **27.11 Variations across Surfaces**........................................................................................... 735 **27.12 Suitability for Human Use**........................................................................................... 736 **27.13 More Complex Scattering**............................................................................................ 737 27.13.1 Participating Media........................................................................................... 737 27.13.2 Subsurface Scattering ....................................................................................... 738 **27.14 Software Interface to Material Models** ....................................................................... 740 **27.15 Discussion and Further Reading** ................................................................................. 741 **27.16 Exercises** ...................................................................................................................... 743

**28 Color**..................................................................................................................................... 745

While color appears to be a physical property—that book is blue, that sun is yellow—it is, in fact, a perceptual phenomenon, one that’s closely related to the spectral distribution of light, but by no means completely determined by it. We describe the perception of color and its relationship to the physiology of the eye. We introduce various systems for naming, representing, and selecting colors. We also discuss the perception of brightness, which is nonlinear as a function of light energy, and the consequences of this for the efficient representation of varying brightness levels, leading to the notion

Contents xxvii

of *gamma,* an exponent used in compressing brightness data. We also discuss the gamuts (range of colors) of various devices, and the problems of color interpolation.

**28.1 Introduction**................................................................................................................. 745 28.1.1 Implications of Color........................................................................................ 746 **28.2 Spectral Distribution of Light** ..................................................................................... 746 **28.3 The Phenomenon of Color Perception and the Physiology of the Eye**........................ 748 **28.4 The Perception of Color** .............................................................................................. 750 28.4.1 The Perception of Brightness ............................................................................ 750 **28.5 Color Description** ........................................................................................................ 756 **28.6 Conventional Color Wisdom** ....................................................................................... 758 28.6.1 Primary Colors ................................................................................................. 758 28.6.2 Purple Isn’t a Real Color................................................................................... 759 28.6.3 Objects Have Colors; You Can Tell by Looking at Them in White Light............ 759 28.6.4 Blue and Green Make Cyan .............................................................................. 760 28.6.5 Color Is RGB.................................................................................................... 761 **28.7 Color Perception Strengths and Weaknesses**.............................................................. 761 **28.8 Standard Description of Colors**................................................................................... 761 28.8.1 The CIE Description of Color ........................................................................... 762 28.8.2 Applications of the Chromaticity Diagram ........................................................ 766 **28.9 Perceptual Color Spaces**.............................................................................................. 767 28.9.1 Variations and Miscellany ................................................................................. 767 **28.10 Intermezzo** ................................................................................................................... 768 **28.11 White**............................................................................................................................ 769 **28.12 Encoding of Intensity, Exponents, and Gamma Correction** ....................................... 769 **28.13 Describing Color**.......................................................................................................... 771 28.13.1 The RGB Color Model...................................................................................... 772 **28.14 CMY and CMYK Color** .............................................................................................. 774 **28.15 The YIQ Color Model**.................................................................................................. 775 **28.16 Video Standards**........................................................................................................... 775 **28.17 HSV and HLS** .............................................................................................................. 776 28.17.1 Color Choice .................................................................................................... 777 28.17.2 Color Palettes ................................................................................................... 777 **28.18 Interpolating Color**...................................................................................................... 777 **28.19 Using Color in Computer Graphics** ............................................................................ 779 **28.20 Discussion and Further Reading** ................................................................................. 780 **28.21 Exercises** ...................................................................................................................... 780

**29 Light Transport**................................................................................................................. 783

Using the formal descriptions of radiance and scattering, we derive the *rendering equation,* an inte gral equation characterizing the radiance field, given a description of the illumination, geometry, and materials in the scene.

**29.1 Introduction**................................................................................................................. 783 **29.2 Light Transport** ........................................................................................................... 783 29.2.1 The Rendering Equation, First Version.............................................................. 786 **29.3 A Peek Ahead**............................................................................................................... 787 **29.4 The Rendering Equation for General Scattering**........................................................ 789 29.4.1 The Measurement Equation .............................................................................. 791

xxviii Contents

**29.5 Scattering, Revisited**.................................................................................................... 792 **29.6 A Worked Example**...................................................................................................... 793 **29.7 Solving the Rendering Equation** ................................................................................. 796 **29.8 The Classification of Light-Transport Paths**............................................................... 796

29.8.1 Perceptually Significant Phenomena and Light Transport .................................. 797 **29.9 Discussion** .................................................................................................................... 799 **29.10 Exercise**........................................................................................................................ 799

**30 Probability and Monte Carlo Integration**................................................................... 801

Probabilistic methods are at the heart of modern rendering techniques, especially methods for esti mating integrals, because solving the rendering equation involves computing an integral that’s impos sible to evaluate exactly in any but the simplest scenes. We review basic discrete probability, gener alize to continuum probability, and use this to derive the single-sample estimate for an integral and the importance-weighted single-sample estimate, which we’ll use in the next two chapters.

**30.1 Introduction**................................................................................................................. 801 **30.2 Numerical Integration** ................................................................................................. 801 **30.3 Random Variables and Randomized Algorithms**........................................................ 802

30.3.1 Discrete Probability and Its Relationship to Programs ....................................... 803 30.3.2 Expected Value ................................................................................................. 804 30.3.3 Properties of Expected Value, and Related Terms.............................................. 806 30.3.4 Continuum Probability...................................................................................... 808 30.3.5 Probability Density Functions........................................................................... 810 30.3.6 Application to the Sphere.................................................................................. 813 30.3.7 A Simple Example............................................................................................ 813 30.3.8 Application to Scattering .................................................................................. 814

**30.4 Continuum Probability, Continued**............................................................................. 815 **30.5 Importance Sampling and Integration**........................................................................ 818 **30.6 Mixed Probabilities**...................................................................................................... 820 **30.7 Discussion and Further Reading** ................................................................................. 821 **30.8 Exercises** ...................................................................................................................... 821

**31 Computing Solutions to the Rendering Equation: Theoretical Approaches**..... 825

The rendering equation can be approximately solved by many methods, including ray tracing (an approximation to the series solution), radiosity (an approximation arising from a finite-element approach), Metropolis light transport, and photon mapping, not to mention basic polygonal render ers using direct-lighting-plus-ambient approximations. Each method has strengths and weaknesses that can be analyzed by considering the nature of the materials in the scene, by examining different classes of light paths from luminaires to detectors, and by uncovering various kinds of approximation errors implicit in the methods.

**31.1 Introduction**................................................................................................................. 825 **31.2 Approximate Solutions of Equations**........................................................................... 825 **31.3 Method 1: Approximating the Equation** ..................................................................... 826 **31.4 Method 2: Restricting the Domain**.............................................................................. 827 **31.5 Method 3: Using Statistical Estimators**....................................................................... 827 31.5.1 Summing a Series by Sampling and Estimation................................................. 828

Contents xxix

**31.6 Method 4: Bisection**..................................................................................................... 830 **31.7 Other Approaches**........................................................................................................ 831 **31.8 The Rendering Equation, Revisited** ............................................................................ 831

31.8.1 A Note on Notation........................................................................................... 835 **31.9 What Do We Need to Compute?**.................................................................................. 836 **31.10 The Discretization Approach: Radiosity** ..................................................................... 838 **31.11 Separation of Transport Paths** .................................................................................... 844 **31.12 Series Solution of the Rendering Equation** ................................................................. 844 **31.13 Alternative Formulations of Light Transport** ............................................................. 846 **31.14 Approximations of the Series Solution**........................................................................ 847 **31.15 Approximating Scattering: Spherical Harmonics**....................................................... 848 **31.16 Introduction to Monte Carlo Approaches**................................................................... 851 **31.17 Tracing Paths**............................................................................................................... 855 **31.18 Path Tracing and Markov Chains**............................................................................... 856

31.18.1 The Markov Chain Approach............................................................................ 857 31.18.2 The Recursive Approach................................................................................... 861 31.18.3 Building a Path Tracer ...................................................................................... 864 31.18.4 Multiple Importance Sampling.......................................................................... 868 31.18.5 Bidirectional Path Tracing................................................................................. 870 31.18.6 Metropolis Light Transport ............................................................................... 871

**31.19 Photon Mapping** .......................................................................................................... 872 31.19.1 Image-Space Photon Mapping .......................................................................... 876 **31.20 Discussion and Further Reading** ................................................................................. 876 **31.21 Exercises** ...................................................................................................................... 879

**32 Rendering in Practice** ...................................................................................................... 881

We describe the implementation of a path tracer, which exhibits many of the complexities associated with ray-tracing-like renderers that attempt to estimate radiance by estimating integrals associated to the rendering equations, and a photon mapper, which quickly converges to a biased but consistent and plausible result.

**32.1 Introduction**................................................................................................................. 881 **32.2 Representations** ........................................................................................................... 881 **32.3 Surface Representations and Representing BSDFs Locally** ....................................... 882

32.3.1 Mirrors and Point Lights................................................................................... 886 **32.4 Representation of Light**............................................................................................... 887 32.4.1 Representation of Luminaires............................................................................ 888 **32.5 A Basic Path Tracer**..................................................................................................... 889 32.5.1 Preliminaries .................................................................................................... 889 32.5.2 Path-Tracer Code .............................................................................................. 893 32.5.3 Results and Discussion ..................................................................................... 901 **32.6 Photon Mapping** .......................................................................................................... 904 32.6.1 Results and Discussion ..................................................................................... 910 32.6.2 Further Photon Mapping ................................................................................... 913 **32.7 Generalizations**............................................................................................................ 914 **32.8 Rendering and Debugging**........................................................................................... 915 **32.9 Discussion and Further Reading** ................................................................................. 919 **32.10 Exercises** ...................................................................................................................... 923

xxx Contents

**33 Shaders**................................................................................................................................ 927

On modern graphics cards, we can execute small (and not-so-small) programs that operate on model data to produce pictures. In the simplest form, these are vertex shaders and fragment shaders, the first of which can do processing based on the geometry of the scene (typically the vertex coordinates), and the second of which can process fragments, which correspond to pieces of polygons that will appear in a single pixel. To illustrate the more basic use of shaders we describe how to implement basic Phong shading, environment mapping, and a simple nonphotorealistic renderer.

**33.1 Introduction**................................................................................................................. 927 **33.2 The Graphics Pipeline in Several Forms**..................................................................... 927 **33.3 Historical Development** ............................................................................................... 929 **33.4 A Simple Graphics Program with Shaders**................................................................. 932 **33.5 A Phong Shader**........................................................................................................... 937 **33.6 Environment Mapping** ................................................................................................ 939 **33.7 Two Versions of Toon Shading**..................................................................................... 940 **33.8 Basic XToon Shading**................................................................................................... 942 **33.9 Discussion and Further Reading** ................................................................................. 943 **33.10 Exercises** ...................................................................................................................... 943

**34 Expressive Rendering** ...................................................................................................... 945

Expressive rendering is the name we give to renderings that do not aim for photorealism, but rather aim to produce imagery that communicates with the viewer, conveying what the creator finds impor tant, and suppressing what’s unimportant. We summarize the theoretical foundations of expressive rendering, particularly various kinds of abstraction, and discuss the relationship of the “message” of a rendering and its style. We illustrate with a few expressive rendering techniques.

**34.1 Introduction**................................................................................................................. 945 34.1.1 Examples of Expressive Rendering ................................................................... 948 34.1.2 Organization of This Chapter ............................................................................ 948

**34.2 The Challenges of Expressive Rendering** .................................................................... 949 **34.3 Marks and Strokes**....................................................................................................... 950 **34.4 Perception and Salient Features**.................................................................................. 951 **34.5 Geometric Curve Extraction** ....................................................................................... 952

34.5.1 Ridges and Valleys............................................................................................ 956 34.5.2 Suggestive Contours ......................................................................................... 957 34.5.3 Apparent Ridges............................................................................................... 958 34.5.4 Beyond Geometry............................................................................................. 959

**34.6 Abstraction** .................................................................................................................. 959 **34.7 Discussion and Further Reading** ................................................................................. 961

**35 Motion**.................................................................................................................................. 963

An animation is a sequence of rendered frames that gives the perception of smooth motion when displayed quickly. The algorithms to control the underlying 3D object motion generally interpolate between key poses using splines, or simulate the laws of physics by numerically integrating veloc ity and acceleration. Whereas rendering primarily is concerned with surfaces, animation algorithms require a model with additional properties like articulation and mass. Yet these models still simplify

Contents xxxi

the real world, accepting limitations to achieve computational efficiency. The hardest problems in animation involve artificial intelligence for planning realistic character motion, which is beyond the scope of this chapter.

**35.1 Introduction**................................................................................................................. 963 **35.2 Motivating Examples**................................................................................................... 966 35.2.1 A Walking Character (Key Poses) ..................................................................... 966 35.2.2 Firing a Cannon (Simulation)............................................................................ 969 35.2.3 Navigating Corridors (Motion Planning) ........................................................... 972 35.2.4 Notation ........................................................................................................... 973 **35.3 Considerations for Rendering** ..................................................................................... 975 35.3.1 Double Buffering.............................................................................................. 975 35.3.2 Motion Perception ............................................................................................ 976 35.3.3 Interlacing ........................................................................................................ 978 35.3.4 Temporal Aliasing and Motion Blur.................................................................. 980 35.3.5 Exploiting Temporal Coherence ........................................................................ 983 35.3.6 The Problem of the First Frame......................................................................... 984 35.3.7 The Burden of Temporal Coherence.................................................................. 985 **35.4 Representations** ........................................................................................................... 987 35.4.1 Objects............................................................................................................. 987 35.4.2 Limiting Degrees of Freedom ........................................................................... 988 35.4.3 Key Poses......................................................................................................... 989 35.4.4 Dynamics ......................................................................................................... 989 35.4.5 Procedural Animation ....................................................................................... 990 35.4.6 Hybrid Control Schemes................................................................................... 990 **35.5 Pose Interpolation**........................................................................................................ 992 35.5.1 Vertex Animation.............................................................................................. 992 35.5.2 Root Frame Motion........................................................................................... 993 35.5.3 Articulated Body .............................................................................................. 994 35.5.4 Skeletal Animation ........................................................................................... 995 **35.6 Dynamics** ..................................................................................................................... 996 35.6.1 Particle ............................................................................................................. 996 35.6.2 Differential Equation Formulation..................................................................... 997 35.6.3 Piecewise-Constant Approximation................................................................... 999 35.6.4 Models of Common Forces............................................................................... 1000 35.6.5 Particle Collisions............................................................................................. 1008 35.6.6 Dynamics as a Differential Equation ................................................................. 1012 35.6.7 Numerical Methods for ODEs........................................................................... 1017 **35.7 Remarks on Stability in Dynamics**.............................................................................. 1020 **35.8 Discussion** .................................................................................................................... 1022

**36 Visibility Determination**.................................................................................................. 1023

Efficient determination of the subset of a scene that affects the final image is critical to the per formance of a renderer. The first approximation of this process is conservative determination of surfaces visible to the eye. This problem has been addressed by algorithms with radically different space, quality, and time bounds. The preferred algorithms vary over time with the cost and perfor mance of hardware architectures. Because analogous problems arise in collision detection, selection,

xxxii Contents

global illumination, and document layout, even visibility algorithms that are currently out of favor for primary rays may be preferred in other applications.

**36.1 Introduction**................................................................................................................. 1023 36.1.1 The Visibility Function ..................................................................................... 1025 36.1.2 Primary Visibility ............................................................................................. 1027 36.1.3 (Binary) Coverage ............................................................................................ 1027 36.1.4 Current Practice and Motivation........................................................................ 1028

**36.2 Ray Casting**.................................................................................................................. 1029 36.2.1 BSP Ray-Primitive Intersection......................................................................... 1030 36.2.2 Parallel Evaluation of Ray Tests........................................................................ 1032

**36.3 The Depth Buffer**......................................................................................................... 1034 36.3.1 Common Depth Buffer Encodings..................................................................... 1037 **36.4 List-Priority Algorithms**.............................................................................................. 1040 36.4.1 The Painter’s Algorithm.................................................................................... 1041 36.4.2 The Depth-Sort Algorithm ................................................................................ 1042 36.4.3 Clusters and BSP Sort....................................................................................... 1043 **36.5 Frustum Culling and Clipping** .................................................................................... 1044 36.5.1 Frustum Culling................................................................................................ 1044 36.5.2 Clipping ........................................................................................................... 1045 36.5.3 Clipping to the Whole Frustum ......................................................................... 1047 **36.6 Backface Culling**.......................................................................................................... 1047 **36.7 Hierarchical Occlusion Culling** ................................................................................... 1049 **36.8 Sector-based Conservative Visibility** ........................................................................... 1050 36.8.1 Stabbing Trees.................................................................................................. 1051 36.8.2 Portals and Mirrors........................................................................................... 1052 **36.9 Partial Coverage** .......................................................................................................... 1054 36.9.1 Spatial Antialiasing (*xy*).................................................................................... 1055 36.9.2 Defocus (*uv*)..................................................................................................... 1060 36.9.3 Motion Blur (*t*)................................................................................................. 1061 36.9.4 Coverage as a Material Property (*α*).................................................................. 1062 **36.10 Discussion and Further Reading** ................................................................................. 1063 **36.11 Exercise**........................................................................................................................ 1063

**37 Spatial Data Structures**................................................................................................... 1065

Spatial data structures like bounding volume hierarchies provide intersection queries and set opera tions on geometry embedded in a metric space. Intersection queries are necessary for light transport, interaction, and dynamics simulation. These structures are classic data structures like hash tables, trees, and graphs extended with the constraints of 3D geometry.

**37.1 Introduction**................................................................................................................. 1065 37.1.1 Motivating Examples........................................................................................ 1066 **37.2 Programmatic Interfaces**............................................................................................. 1068 37.2.1 Intersection Methods......................................................................................... 1069 37.2.2 Extracting Keys and Bounds ............................................................................. 1073 **37.3 Characterizing Data Structures** .................................................................................. 1077 37.3.1 1D Linked List Example ................................................................................... 1078 37.3.2 1D Tree Example.............................................................................................. 1079

Contents xxxiii

**37.4 Overview of *k*d Structures**........................................................................................... 1080 **37.5 List** ............................................................................................................................... 1081 **37.6 Trees**............................................................................................................................. 1083

37.6.1 Binary Space Partition (BSP) Trees................................................................... 1084 37.6.2 Building BSP Trees: oct tree, quad tree, BSP tree, *k*d tree.................................. 1089 37.6.3 Bounding Volume Hierarchy............................................................................. 1092

**37.7 Grid**.............................................................................................................................. 1093 37.7.1 Construction ..................................................................................................... 1093 37.7.2 Ray Intersection................................................................................................ 1095 37.7.3 Selecting Grid Resolution ................................................................................. 1099 **37.8 Discussion and Further Reading** ................................................................................. 1101

**38 Modern Graphics Hardware**.......................................................................................... 1103

We describe the structure of modern graphics cards, their design, and some of the engineering trade offs that influence this design.

**38.1 Introduction**................................................................................................................. 1103 **38.2 NVIDIA GeForce 9800 GTX**...................................................................................... 1105 **38.3 Architecture and Implementation**............................................................................... 1107

38.3.1 GPU Architecture ............................................................................................. 1108 38.3.2 GPU Implementation ........................................................................................ 1111 **38.4 Parallelism** ................................................................................................................... 1111 **38.5 Programmability** ......................................................................................................... 1114 **38.6 Texture, Memory, and Latency** ................................................................................... 1117 38.6.1 Texture Mapping............................................................................................... 1118 38.6.2 Memory Basics................................................................................................. 1121 38.6.3 Coping with Latency......................................................................................... 1124 **38.7 Locality** ........................................................................................................................ 1127 38.7.1 Locality of Reference........................................................................................ 1127 38.7.2 Cache Memory ................................................................................................. 1129 38.7.3 Divergence ....................................................................................................... 1132 **38.8 Organizational Alternatives** ........................................................................................ 1135 38.8.1 Deferred Shading.............................................................................................. 1135 38.8.2 Binned Rendering............................................................................................. 1137 38.8.3 Larrabee: A CPU/GPU Hybrid.......................................................................... 1138 **38.9 GPUs as Compute Engines**.......................................................................................... 1142 **38.10 Discussion and Further Reading** ................................................................................. 1143 **38.11 Exercises** ...................................................................................................................... 1143

*List of Principles*.......................................................................................................................1145 *Bibliography* ..............................................................................................................................1149 *Index*............................................................................................................................................1183

*This page intentionally left blank*

**Preface**

This book presents many of the important ideas of computer graphics to stu dents, researchers, and practitioners. Several of these ideas are not new: They have already appeared in widely available scholarly publications, technical reports, textbooks, and lay-press articles. The advantage of writing a textbook sometime after the appearance of an idea is that its long-term impact can be understood bet ter and placed in a larger context. Our aim has been to treat ideas with as much sophistication as possible (which includes omitting ideas that are no longer as important as they once were), while still introducing beginning students to the subject lucidly and gracefully.

This is a second-generation graphics book: Rather than treating all prior work as implicitly valid, we evaluate it in the context of today’s understanding, and update the presentation as appropriate.

Even the most elementary issues can turn out to be remarkably subtle. Sup pose, for instance, that you’re designing a program that must run in a low-light environment—a darkened movie theatre, for instance. Obviously you cannot use a bright display, and so using brightness contrast to distinguish among different items in your program display would be inappropriate. You decide to use color instead. Unfortunately, color perception in low-light environments is not nearly as good as in high-light environments, and some text colors are easier to read than others in low light. Is your cursor still easy to see? Maybe to make that simpler, you should make the cursor constantly jitter, exploiting the motion sensitivity of the eye. So what seemed like a simple question turns out to involve issues of inter face design, color theory, and human perception.

This example, simple as it is, also makes some unspoken assumptions: that the application uses graphics (rather than, say, tactile output or a well-isolated audio earpiece), that it does not use the regular theatre screen, and that it does not use a head-worn display. It makes explicit assumptions as well—for instance, that a cursor will be used (some UIs intentionally don’t use a cursor). Each of these assumptions reflects a user-interface choice as well.

Unfortunately, this interrelatedness of things makes it impossible to present topics in a completely ordered fashion and still motivate them well; the subject is simply no longer linearizable. We *could* have covered all the mathematics, the ory of perception, and other, more abstract, topics first, and only then moved on to their graphics applications. Although this might produce a better reference work (you know just where to look to learn about generalized cross products,

xxxv

xxxvi Preface

for instance), it doesn’t work well for a textbook, since the motivating applica tions would all come at the end. Alternatively, we could have taken a case-study approach, in which we try to complete various increasingly difficult tasks, and introduce the necessary material as the need arises. This makes for a natural pro gression in some cases, but makes it difficult to give a broad organizational view of the subject. Our approach is a compromise: We start with some widely used math ematics and notational conventions, and then work from topic to topic, introducing supporting mathematics as needed. Readers already familiar with the mathemat ics can safely skip this material without missing any computer graphics; others may learn a good deal by reading these sections. Teachers may choose to include or omit them as needed. The topic-based organization of the book entails some redundancy. We discuss the graphics pipeline multiple times at varying levels of detail, for instance. Rather than referring the reader back to a previous chapter, sometimes we redescribe things, believing that this introduces a more natural flow. Flipping back 500 pages to review a figure can be a substantial distraction.

The other challenge for a textbook author is to decide how encyclopedic to make the text. The first edition of this book really did cover a very large fraction of the published work in computer graphics; the second edition at least made pass ing references to much of the work. This edition abandons any pretense of being encyclopedic, for a very good reason: When the second edition was written, a sin gle person could carry, under one arm, all of the proceedings of SIGGRAPH, the largest annual computer graphics conference, and these constituted a fair represen tation of all technical writings on the subject. Now the SIGGRAPH proceedings (which are just one of many publication venues) occupy several cubic feet. Even a telegraphic textbook cannot cram all that information into a thousand pages. Our goal in this book is therefore to lead the reader to the point where he or she can read and reproduce many of today’s SIGGRAPH papers, albeit with some caveats:

• First, computer graphics and computer vision are overlapping more and more, but there is no excuse for us to write a computer vision textbook; others with far greater knowledge have already done so.

• Second, computer graphics involves programming; many graphics applica tions are quite large, but we do not attempt to teach either programming or software engineering in this book. We do briefly discuss programming (and especially debugging) approaches that are unique to graphics, however.

• Third, most graphics applications have a user interface. At the time of this writing, most of these interfaces are based on windows with menus, and mouse interaction, although touch-based interfaces are becoming common place as well. There was a time when user-interface research was a part of graphics, but it’s now an independent community—albeit with substantial overlap with graphics—and we therefore assume that the student has some experience in creating programs with user interfaces, and don’t discuss these in any depth, except for some 3D interfaces whose implementations are more closely related to graphics.

Of course, research papers in graphics differ. Some are highly mathematical, others describe large-scale systems with complex engineering tradeoffs, and still others involve a knowledge of physics, color theory, typography, photography, chemistry, zoology. . . the list goes on and on. Our goal is to prepare the reader to understand the computer graphics in these papers; the other material may require considerable external study as well.

Preface xxxvii

**Historical Approaches**

The history of computer graphics is largely one of ad hoc approaches to the imme diate problems at hand. Saying this is in no way an indictment of the people who took those approaches: They had jobs to do, and found ways to do them. Sometimes their solutions had important ideas wrapped up within them; at other times they were merely ways to get things done, and their adoption has inter fered with progress in later years. For instance, the image-compositing model used in most graphics systems assumes that color values stored in images can be blended linearly. In actual practice, the color values stored in images are non linearly related to light intensity; taking linear combinations of these does not correspond to taking linear combinations of intensity. The difference between the two approaches began to be noticed when studios tried to combine real-world and computer-generated imagery; this compositing technology produced unacceptable results. In addition, some early approaches were deeply principled, but the associ ated programs made assumptions about hardware that were no longer valid a few years later; readers, looking first at the details of implementation, said, “Oh, this is old stuff—it’s not relevant to us at all,” and missed the still important ideas of the research. All too frequently, too, researchers have simply reinvented things known in other disciplines for years.

We therefore do *not* follow the chronological development of computer graph ics. Just as physics courses do not begin with Aristotle’s description of dynamics, but instead work directly with Newton’s (and the better ones describe the limita tions of even *that* system, setting the stage for quantum approaches, etc.), we try to start directly from the best current understanding of issues, while still presenting various older ideas when relevant. We also try to point out sources for ideas that may not be familiar ones: Newell’s formula for the normal vector to a polygon in 3-space was known to Grassmann in the 1800s, for instance. Our hope in refer encing these sources is to increase the reader’s awareness of the variety of already developed ideas that are waiting to be applied to graphics.

**Pedagogy**

The most striking aspect of graphics in our everyday lives is the 3D imagery being used in video games and special effects in the entertainment industry and adver tisements. But our day-to-day interactions with home computers, cell phones, etc., are also based on computer graphics. Perhaps they are less visible in part because they are more successful: The best interfaces are the ones you don’t notice. It’s tempting to say that “2D graphics” is simpler—that 3D graphics is just a more complicated version of the same thing. But many of the issues in 2D graphics— how best to display images on a screen made of a rectangular grid of light-emitting elements, for instance, or how to construct effective and powerful interfaces—are just as difficult as those found in making pictures of three-dimensional scenes. And the simple models conventionally used in 2D graphics can lead the student into false assumptions about how best to represent things like color or shape. We therefore have largely integrated the presentation of 2D and 3D graphics so as to address simultaneously the subtle issues common to both.

This book is unusual in the level at which we put the “black box.” Almost every computer science book has to decide at what level to abstract something about the computers that the reader will be familiar with. In a graphics book, we have to

xxxviii Preface

decide what graphics system the reader will be encountering as well. It’s not hard (after writing a first program or two) to believe that some combination of hardware and software inside your computer can make a colored triangle appear on your display when you issue certain instructions. The details of how this happens are not relevant to a surprisingly large part of graphics. For instance, what happens if you ask the graphics system to draw a red triangle that’s below the displayable area of your screen? Are the pixel locations that need to be made red computed and then ignored because they’re off-screen? Or does the graphics system realize, before computing any pixel values, that the triangle is off-screen and just quit? In some sense, unless you’re designing a graphics card, it just doesn’t matter all that much; indeed, it’s something you, as a user of a graphics system, can’t really control. In much of the book, therefore, we treat the graphics system as something that can display certain pixel values, or draw triangles and lines, without worrying too much about the “how” of this part. The details *are* included in the chapters on rasterization and on graphics hardware. But because they are mostly beyond our control, topics like clipping, antialiasing of lines, and rasterization algorithms are all postponed to later chapters.

Another aspect of the book’s pedagogy is that we generally try to show *how* ideas or techniques arise. This can lead to long explanations, but helps, we hope, when students need to derive something for themselves: The approaches they’ve encountered may suggest an approach to their current problem.

We believe that the best way to learn graphics is to first learn the mathematics behind it. The drawback of this approach compared to jumping to applications is that learning the abstract math increases the amount of time it takes to learn your first few techniques. But you only pay that overhead once. By the time you’re learning the tenth related technique, your investment will pay off because you’ll recognize that the new method combines elements you’ve already studied.

Of course, you’re reading this book because you are motivated to write pro grams that make pictures. So we try to start many topics by diving straight into a solution before stepping back to deeply consider the broader mathematical issues. Most of this book is concerned with that stepping-back process. Having inves tigated the mathematics, we’ll then close out topics by sketching other related problems and some solutions to them. Because we’ve focused on the underlying principles, you won’t need us to tell you the details for these sketches. From your understanding of the principles, the approach of each solution should be clear, and you’ll have enough knowledge to be able to read and understand the original cited publication in its author’s own words, rather than relying on us to translate it for you. What we *can* do is present some older ideas in a slightly more modern form so that when you go back to read the original paper, you’ll have some idea how its vocabulary matches your own.

**Current Practice**

Graphics is a hands-on discipline. And since the business of graphics is the pre sentation of visual information to a viewer, and the subsequent interaction with it, graphical tools can often be used effectively to debug new graphical algo rithms. But doing this requires the ability to write graphics programs. There are many alternative ways to produce graphical imagery on today’s computers, and for much of the material in this book, one method is as good as another. The conversion between one programming language and its libraries and another is

Preface xxxix

routine. But for teaching the subject, it seems best to work in a single language so that the student can concentrate on the deeper ideas. Throughout this book, we’ll suggest exercises to be written using Windows Presentation Foundation (WPF), a widely available graphics system, for which we’ve written a basic and easily mod ified program we call a “test bed” in which the student can work. For situations where WPF is not appropriate, we’ve often used G3D, a publicly available graph ics library maintained by one of the authors. And in many situations, we’ve written pseudocode. It provides a compact way to express ideas, and for most algorithms, actual code (in the language of your choice) can be downloaded from the Web; it seldom makes sense to include it in the text. The formatting of code varies; in cases where programs are developed from an informal sketch to a nearly com plete program in some language, things like syntax highlighting make no sense until quite late versions, and may be omitted entirely. Sometimes it’s nice to have the code match the mathematics, leading us to use variables with names like *x*R, which get typeset as math rather than code. In general, we italicize pseudocode, and use indentation rather than braces in pseudocode to indicate code blocks. In general, our pseudocode is very informal; we use it to convey the broad ideas rather than the details.

This is *not* a book about writing graphics programs, nor is it about *using* them. Readers will find no hints about the best ways to store images in Adobe’s latest image-editing program, for instance. But we hope that, having understood the concepts in this book and being competent programmers already, they will both be able to write graphics programs and understand how to use those that are already written.

**Principles**

Throughout the book we have identified certain computer graphics principles that will help the reader in future work; we’ve also included sections on cur rent *practice*—sections that discuss, for example, how to approximate your ideal solution on today’s hardware, or how to compute your actual ideal solution more rapidly. Even practices that are tuned to today’s hardware can prove useful tomor row, so although in a decade the practices described may no longer be directly applicable, they show approaches that we believe will still be valuable for years.

**Prerequisites**

Much of this book assumes little more preparation than what a technically savvy undergraduate student may have: the ability to write object-oriented programs; a working knowledge of calculus; some familiarity with vectors, perhaps from a math class or physics class or even a computer science class; and at least some encounter with linear transformations. We also expect that the typical student has written a program or two containing 2D graphical objects like buttons or check boxes or icons.

Some parts of this book, however, depend on far more mathematics, and attempting to teach that mathematics within the limits of this text is impossible. Generally, however, this sophisticated mathematics is carefully limited to a few sections, and these sections are more appropriate for a graduate course than an introductory one. Both they and certain mathematically sophisticated exercises are marked with a “math road-sign” symbol thus: . Correspondingly, topics that

xl Preface

use deeper notions from computer science are marked with a “computer science road-sign,” .

Some mathematical aspects of the text may seem strange to those who have met vectors in other contexts; the first author, whose Ph.D. is in mathematics, cer tainly was baffled by some of his first encounters with how graphics researchers do things. We attempt to explain these variations from standard mathematical approaches clearly and thoroughly.

**Paths through This Book**

This book can be used for a semester-long or yearlong undergraduate course, or as a general reference in a graduate course. In an undergraduate course, the advanced mathematical topics can safely be omitted (e.g., the discussions of analogs to barycentric coordinates, manifold meshes, spherical harmonics, etc.) while con centrating on the basic ideas of creating and displaying geometric models, under standing the mathematics of transformations, camera specifications, and the stan dard models used in representing light, color, reflectance, etc., along with some hints of the limitations of these models. It should also cover basic graphics appli cations and the user-interface concerns, design tradeoffs, and compromises neces sary to make them efficient, possibly ending with some special topic like creating simple animations, or writing a basic ray tracer. Even this is too much for a sin gle semester, and even a yearlong course will leave many sections of the book untouched, as future reading for interested students.

An aggressive semester-long (14-week) course could cover the following.

1. Introduction and a simple 2D program: Chapters 1, 2, and 3. 2. Introduction to the geometry of rendering, and further 2D and 3D pro grams: Chapters 3 and 4. Visual perception and the human visual system: Chapter 5.

3. Modeling of geometry in 2D and 3D: meshes, splines, and implicit models. Sections 7.1–7.9, Chapters 8 and 9, Sections 22.1–22.4, 23.1–23.3, and 24.1–24.5.

4. Images, part 1: Chapter 17, Sections 18.1–18.11.

5. Images, part 2: Sections 18.12–18.20, Chapter 19.

6. 2D and 3D transformations: Sections 10.1–10.12, Sections 11.1–11.3, Chapter 12.

7. Viewing, cameras, and post-homogeneous interpolation. Sections 13.1– 13.7, 15.6.4.

8. Standard approximations in graphics: Chapter 14, selected sections. 9. Rasterization and ray casting: Chapter 15.

10. Light and reflection: Sections 26.1–26.7 (Section 26.5 optional); Section 26.10.

11. Color: Sections 28.1–28.12.

12. Basic reflectance models, light transport: Sections 27.1–27.5, 29.1–29.2, 29.6, 29.8.

13. Recursive ray-tracing details, texture: Sections 24.9, 31.16, 20.1–20.6.

Preface xli

14. Visible surface determination and acceleration data structures; overview of more advanced rendering techniques: selections from Chapters 31, 36, and 37.

However, not all the material in every section would be appropriate for a first course.

Alternatively, consider the syllabus for a 12-week undergraduate course on physically based rendering that takes first principles from offline to real-time ren dering. It could dive into the core mathematics and radiometry behind ray tracing, and then cycle back to pick up the computer science ideas needed for scalability and performance.

1. Introduction: Chapter 1

2. Light: Chapter 26

3. Perception; light transport: Chapters 5 and 29

4. A brief overview of meshes and scene graphs: Sections 6.6, 14.1–5 5. Transformations: Chapters 10 and 13, briefly.

6. Ray casting: Sections 15.1–4, 7.6–9

7. Acceleration data structures: Chapter 37; Sections 36.1–36.3, 36.5–36.6, 36.9

8. Rendering theory: Chapters 30 and 31

9. Rendering practice: Chapter 32

10. Color and material: Sections 14.6–14.11, 28, and 27

11. Rasterization: Sections 15.5–9

12. Shaders and hardware: Sections 16.3–5, Chapters 33 and 38

Note that these paths touch chapters out of numerical order. We’ve intention ally written this book in a style where most chapters are self-contained, with cross references instead of prerequisites, to support such traversal.

**Differences from the Previous Edition**

This edition is almost completely new, although many of the topics covered in the previous edition appear here. With the advent of the GPU, triangles are converted to pixels (or samples) by radically different approaches than the old scan-conversion algorithms. We no longer discuss those. In discussing light, we strongly favor physical units of measurement, which adds some complexity to discussions of older techniques that did not concern themselves with units. Rather than preparing two graphics packages for 2D and 3D graphics, as we did for the previous editions, we’ve chosen to use widely available systems, and provide tools to help the student get started using them.

**Website**

Often in this book you’ll see references to the book’s website. It’s at http:// cgpp.net and contains not only the testbed software and several examples

xlii Preface

derived from it, but additional material for many chapters, and the interactive experiments in WPF for Chapters 2 and 6.

**Acknowledgments**

A book like this is written by the authors, but it’s enormously enhanced by the contributions of others.

Support and encouragement from Microsoft, especially from Eric Rudder and S. Somasegur, helped to both initiate and complete this project. The 3D test bed evolved from code written by Dan Leventhal; we also thank Mike Hodnick at kindohm.com, who graciously agreed to let us use his code as a starting point for an earlier draft, and Jordan Parker and Anthony Hodsdon for assisting with WPF.

Two students from Williams College worked very hard in supporting the book: Guedis Cardenas on the bibliography, and Michael Mara on the G3D Innovation Engine used in several chapters; Corey Taylor of Electronic Arts also helped with G3D.

Nancy Pollard of CMU and Liz Marai of the University of Pittsburgh both used early drafts of several chapters in their graphics courses, and provided excellent feedback.

Jim Arvo served not only as an oracle on everything relating to rendering, but helped to reframe the first author’s understanding of the field. Many others, in addition to some of those just mentioned, read chapter drafts, prepared images or figures, suggested topics or ways to present them, or helped out in other ways. In alphabetical order, they are John Anderson, Jim Arvo, Tom Banchoff, Pascal Barla, Connelly Barnes, Brian Barsky, Ronen Barzel, Melissa Byun, Marie-Paule Cani, Lauren Clarke, Elaine Cohen, Doug DeCarlo, Patrick Doran, Kayvon Fatahalian, Adam Finkelstein, Travis Fischer, Roger Fong, Mike Fredrickson, Yudi Fu, Andrew Glassner, Bernie Gordon, Don Greenberg, Pat Hanrahan, Ben Herila, Alex Hills, Ken Joy, Olga Karpenko, Donnie Kendall, Justin Kim, Philip Klein, Joe LaViola, Kefei Lei, Nong Li, Lisa Manekofsky, Bill Mark, John Montrym, Henry Moreton, Tomer Moscovich, Jacopo Pantaleoni, Jill Pipher, Charles Poynton, Rich Riesenfeld, Alyn Rockwood, Peter Schroeder, François Sillion, David Simons, Alvy Ray Smith, Stephen Spencer, Erik Sudderth, Joelle Thollot, Ken Torrance, Jim Valles, Daniel Wigdor, Dan Wilk, Brian Wyvill, and Silvia Zuffi. Despite our best efforts, we have probably forgotten some people, and apologize to them.

It’s a sign of the general goodness of the field that we got a lot of support in writing from authors of competing books. Eric Haines, Greg Humphreys, Steve Marschner, Matt Pharr, and Pete Shirley all contributed to making this a better book. It’s wonderful to work in a field with folks like this.

We’d never had managed to produce this book without the support, tolerance, indulgence, and vision of our editor, Peter Gordon. And we all appreciate the enormous support of our families throughout this project.

**For the Student**

Your professor will probably choose some route through this book, selecting top ics that fit well together, perhaps following one of the suggested trails mentioned

Preface xliii

earlier. Don’t let that constrain you. If you want to know about something, use the index and start reading. Sometimes you’ll find yourself lacking background, and you won’t be able to make sense of what you read. When that happens, read the background material. It’ll be easier than reading it at some other time, because right now you have a *reason* to learn it. If you stall out, search the Web for some one’s implementation and download and run it. When you notice it doesn’t look quite right, you can start examining the implementation, and trying to reverse engineer it. Sometimes this is a great way to understand something. Follow the practice-theory-practice model of learning: Try something, see whether you can make it work, and if you can’t, read up on how others did it, and then try again. The first attempt may be frustrating, but it sets you up to better understand the theory when you get to it. If you can’t bring yourself to follow the practice-theory practice model, at the very least you should take the time to do the inline exercises for any chapter you read.

Graphics is a young field, so young that undergraduates are routinely coau thors on SIGGRAPH papers. In a year you can learn enough to start contributing new ideas.

Graphics also uses a lot of mathematics. If mathematics has always seemed abstract and theoretical to you, graphics can be really helpful: The uses of math ematics in graphics are practical, and you can often *see* the consequences of a theorem in the pictures you make. If mathematics has always come easily to you, you can gain some enjoyment from trying to take the ideas we present and extend them further. While this book contains a lot of mathematics, it only scratches the surface of what gets used in modern research papers.

Finally, *doubt everything*. We’ve done our best to tell the truth in this book, as we understand it. We think we’ve done pretty well, and the great bulk of what we’ve said is true. In a few places, we’ve deliberately told partial truths when we introduced a notion, and then amplified these in a later section when we’re discussing details. But aside from that, we’ve surely failed to tell the truth in other places as well. In some cases, we’ve simply made errors, leaving out a minus sign, or making an off-by-one error in a loop. In other cases, the current understanding of the graphics community is just inadequate, and we’ve believed what others have said, and will have to adjust our beliefs later. These errors are opportunities for you. Martin Gardner said that the true sound of scientific discovery is not “Aha!” but “Hey, *that’s* odd. . . .” So if every now and then something seems odd to you, go ahead and doubt it. Look into it more closely. If it turns out to be true, you’ll have cleared some cobwebs from your understanding. If it’s false, it’s a chance for you to advance the field.

**For the Teacher**

If you’re like us, you probably read the “For the Student” section even though it wasn’t for you. (And your students are probably reading this part, too.) You know that we’ve advised them to graze through the book at random, and to doubt everything.

We recommend to you (aside from the suggestions in the remainder of this preface) two things. The first is that you encourage, or even require, that your students answer the inline exercises in the book. To the student who says, “I’ve got too much to do! I can’t waste time stopping to do some exercise,” just say, “We

xliv Preface

don’t have time to stop for gas . . . we’re already late.” The second is that you assign your students projects or homeworks that have both a fixed goal and an open ended component. The steady students will complete the fixed-goal parts and learn the material you want to cover. The others, given the chance to do something fun, may do things with the open-ended exercises that will amaze you. And in doing so, they’ll find that they need to learn things that might seem *just* out of reach, until they suddenly master them, and become empowered. Graphics is a terrific medium for this: Successes are instantly visible and rewarding, and this sets up a feedback loop for advancement. The combination of visible feedback with the ideas of scalability that they’ve encountered elsewhere in computer science can be revelatory.

**Discussion and Further Reading**

Most chapters of this book contain a “Discussion and Further Reading” section like this one, pointing to either background references or advanced applications of the ideas in the chapter. For this preface, the only suitable further reading is very general: We recommend that you immediately begin to look at the proceedings of ACM SIGGRAPH conferences, and of other graphics conferences like Euro graphics and Computer Graphics International, and, depending on your evolving interest, some of the more specialized venues like the Eurographics Symposium on Rendering, I3D, and the Symposium on Computer Animation. While at first the papers in these conferences will seem to rely on a great deal of prior knowl edge, you’ll find that you rapidly get a sense of what things are possible (if only by looking at the pictures), and what sorts of skills are necessary to achieve them. You’ll also rapidly discover ideas that keep reappearing in the areas that most interest you, and this can help guide your further reading as you learn graphics.

**About the Authors**

**John F. Hughes** (B.A., Mathematics, Princeton, 1977; Ph.D., Mathematics, U.C. Berkeley, 1982) is a Professor of Computer Science at Brown University. His primary research is in computer graphics, particularly those aspects of graph ics involving substantial mathematics. As author or co-author of 19 SIGGRAPH papers, he has done research in geometric modeling, user interfaces for modeling, nonphotorealistic rendering, and animation systems. He’s served as an associate editor for *ACM Transaction on Graphics* and the *Journal of Graphics Tools,* and has been on the SIGGRAPH program committee multiple times. He co-organized Implicit Surfaces ’99, the 2001 Symposium in Interactive 3D Graphics, and the first Eurographics Workshop on Sketch-Based Interfaces and Modeling, and was the Papers Chair for SIGGRAPH 2002.

**Andries van Dam** is the Thomas J. Watson, Jr. University Professor of Tech nology and Education, and Professor of Computer Science at Brown Univer sity. He has been a member of Brown’s faculty since 1965, was a co-founder of Brown’s Computer Science Department and its first Chairman from 1979 to 1985, and was also Brown’s first Vice President for Research from 2002–2006. Andy’s research includes work on computer graphics, hypermedia systems, post-WIMP user interfaces, including immersive virtual reality and pen- and touch-computing, and educational software. He has been working for over four decades on systems for creating and reading electronic books with interactive illustrations for use in teaching and research. In 1967 Andy co-founded ACM SICGRAPH, the forerun ner of SIGGRAPH, and from 1985 through 1987 was Chairman of the Computing Research Association. He is a Fellow of ACM, IEEE, and AAAS, a member of the National Academy of Engineering and the American Academy of Arts & Sci ences, and holds four honorary doctorates. He has authored or co-authored over 100 papers and nine books.

**Morgan McGuire** (B.S., MIT, 2000, M.Eng., MIT 2000, Ph.D., Brown Uni versity, 2006) is an Associate Professor of Computer Science at Williams Col lege. He’s contributed as an industry consultant to products including the Mar vel Ultimate Alliance and Titan Quest video game series, the E Ink display used in the Amazon Kindle, and NVIDIA GPUs. Morgan has published papers on high-performance rendering and computational photography in *SIGGRAPH, High Performance Graphics,* the *Eurographics Symposium on Rendering, Interactive*

xlv

xlvi About the Authors

*3D Graphics and Games,* and *Non-Photorealistic Animation and Rendering*. He founded the *Journal of Computer Graphics Techniques,* chaired the Symposium on Interactive 3D Graphics and Games and the Symposium on Non-Photorealistic Animation and Rendering, and is the project manager for the G3D Innovation Engine. He is the co-author of *Creating Games, The Graphics Codex,* and chap ters of several *GPU Gems, ShaderX* and *GPU Pro* volumes.

**David Sklar** (B.S., Southern Methodist University, 1982; M.S., Brown University, 1983) is currently a Visualization Engineer at Vizify.com, working on algorithms for presenting animated infographics on computing devices across a wide range of form factors. Sklar served on the computer science faculty at Brown University in the 1980s, presenting introductory courses and co-authoring several chapters of (and the auxiliary software for) the second edition of this book. Subsequently, Sklar transitioned into the electronic-book industry, with a focus on SGML/XML markup standards, during which time he was a frequent presenter at GCA confer ences. Thereafter, Sklar and his wife Siew May Chin co-founded PortCompass, one of the first online retail shore-excursion marketers, which was the first in a long series of entrepreneurial start-up endeavors in a variety of industries ranging from real-estate management to database consulting.

**James Foley** (B.S.E.E., Lehigh University, 1964; M.S.E.E., University of Michigan 1965; Ph.D., University of Michigan, 1969) holds the Fleming Chair and is Professor of Interactive Computing in the College of Computing at Geor gia Institute of Technology. He previously held faculty positions at UNC-Chapel Hill and The George Washington University and management positions at Mit subishi Electric Research. In 1992 he founded the GVU Center at Georgia Tech and served as director through 1996. During much of that time he also served as editor-in-chief of *ACM Transactions on Graphics*. His research contributions have been to computer graphics, human-computer interaction, and information visual ization. He is a co-author of three editions of this book and of its 1980 predecessor, *Fundamentals of Interactive Computer Graphics*. He is a fellow of the ACM, the American Association for the Advancement of Science and IEEE, recipient of lifetime achievement awards from SIGGRAPH (the Coons award) and SIGCHI, and a member of the National Academy of Engineering.

**Steven Feiner** (A.B., Music, Brown University, 1973; Ph.D., Computer Science, Brown University, 1987) is a Professor of Computer Science at Columbia Uni versity, where he directs the Computer Graphics and User Interfaces Lab and co directs the Columbia Vision and Graphics Center. His research addresses 3D user interfaces, augmented reality, wearable computing, and many topics at the inter section of human-computer interaction and computer graphics. Steve has served as an associate editor of *ACM Transactions on Graphics,* a member of the edito rial board of *IEEE Transactions on Visualization and Computer Graphics,* and a member of the editorial advisory board of *Computers & Graphics*. He was elected to the CHI Academy and, together with his students, has received the ACM UIST Lasting Impact Award, and best paper awards from IEEE ISMAR, ACM VRST, ACM CHI, and ACM UIST. Steve has been program chair or co-chair for many conferences, such as IEEE Virtual Reality, ACM Symposium on User Inter face Software & Technology, Foundations of Digital Games, ACM Symposium

About the Authors xlvii

on Virtual Reality Software & Technology, IEEE International Symposium on Wearable Computers, and ACM Multimedia.

**Kurt Akeley** (B.E.E., University of Delaware, 1980; M.S.E.E., Stanford Uni versity, 1982; Ph.D., Electrical Engineering, Stanford University, 2004) is Vice President of Engineering at Lytro, Inc. Kurt is a co-founder of Silicon Graphics (later SGI), where he led the development of a sequence of high-end graphics sys tems, including RealityEngine, and also led the design and standardization of the OpenGL graphics system. He is a Fellow of the ACM, a recipient of ACM’s SIG GRAPH computer graphics achievement award, and a member of the National Academy of Engineering. Kurt has authored or co-authored papers published in *SIGGRAPH, High Performance Graphics, Journal of Vision,* and *Optics Express*. He has twice chaired the SIGGRAPH technical papers program, first in 2000, and again in 2008 for the inaugural SIGGRAPH Asia conference.

*This page intentionally left blank*

**Chapter 10**

**Transformations in**

**Two Dimensions**

**10.1 Introduction**

As you saw in Chapters 2 and 6, when we think about taking an object for which we have a geometric model and putting it in a scene, we typically need to do three things: *Move* the object to some location, *scale* it up or down so that it fits well with the other objects in the scene, and *rotate* it until it has the right orientation. These operations—translation, scaling, and rotation—are part of every graphics system. Both scaling and rotation are **linear transformations** on the coordinates of the object’s points. Recall that a linear transformation,

*T* : **R2** *→* **R2**, (10.1)

is one for which *T*(**v** + *α***w**) = *T*(**v**) + *αT*(**w**) for any two vectors **v** and **w** in **R2**, and any real number *α*. Intuitively, it’s a transformation that preserves lines and leaves the origin unmoved.

| **Inline Exercise 10.1:** Suppose *T* is linear. Insert *α* = 1 in the definition of linearity. What does it say? Insert **v** = **0** in the definition. What does it say? |
| --- |

| **Inline Exercise 10.2:** When we say that a linear transformation “preserves lines,” we mean that if is a line, then the set of points *T*() must also *lie in* some line. You might expect that we’d require that *T*() actually *be* a line, but that would mean that transformations like “project everything perpendicularly onto the *x*-axis” would not be counted as “linear.” For this particular projection transformation, describe a line such that *T*() is contained in a line, but is not itself a line. |
| --- |

221

222 Transformations in Two Dimensions

The definition of linearity guarantees that for any linear transformation *T*, we have *T*(**0**) = **0**: If we choose **v** = **w** = **0** and *α* = 1, the definition tells us that

*T*(**0**) = *T*(**0** + 1**0**) = *T*(**0**) + 1*T*(**0**) = *T*(**0**) + *T*(**0**). (10.2)

Subtracting *T*(**0**) from the first and last parts of this chain gives us **0** = *T*(**0**). This means that **translation**—moving every point of the plane by the same amount— is, in general, *not* a linear transformation except in the special case of translation by zero, in which all points are left where they are. Shortly we’ll describe a trick for putting the Euclidean plane into **R3** (but *not* as the *z* = 0 plane as is usually done); once we do this, we’ll see that certain linear transformations on **R3** end up performing translations on this embedded plane.

For now, let’s look at only the plane. We assume that you have *some* famil iarity with linear transformations already; indeed, the serious student of computer graphics should, at some point, study linear algebra carefully. But one can learn a great deal about graphics with only a modest amount of knowledge of the subject, which we summarize here briefly.

In the first few sections, we use the convention of most linear-algebra texts:

*u*

The vectors are arrows at the origin, and we think of the vector *v*

as being

identified with the point (*u*, *v*). Later we’ll return to the point-vector distinction. For any 2 *×* 2 matrix **M**, the function **v** *→* **Mv** is a linear transformation from **R2** to **R2**. We refer to this as a **matrix transformation.** In this chapter, we look at five such transformations in detail, study matrix transformations in general, and introduce a method for incorporating translation into the matrix-transformation formulation. We then apply these ideas to transforming objects *and* changing coor dinate systems, returning to the clock example of Chapter 2 to see the ideas in practice.

**10.2 Five Examples**

We begin with five examples of linear transformations in the plane; we’ll refer to these by the names *T*1, *...* , *T*5 throughout the chapter.

*y*

*x*

and

**Example 1: Rotation.** Let **M**1 =

cos 30*◦ −* sin 30*◦* sin 30*◦* cos 30*◦*

**Before** *y*

*T*1 : **R2** *→* **R2** :

*x y*

*→* **M**1

*x y*

=

cos 30*◦ −* sin 30*◦* sin 30*◦* cos 30*◦*

*x y*

. (10.3)

Recall that **e**1 denotes the vector

1 0

and **e**2 =

0 1

; this transformation sends

*x*

**e**1 to the vector

cos 30*◦* sin 30*◦*

and **e**2 to

*−* sin 30*◦* cos 30*◦*

, which are vectors that are 30*◦*

counterclockwise from the *x*- and *y*-axes, respectively (see Figure 10.1). There’s nothing special about the number 30 in this example; by replacing 30*◦* with any angle, you can build a transformation that rotates things counterclock wise by that angle.

**After**

*Figure 10.1: Rotation by* 30*◦.*

10.2 Five Examples 223

| **Inline Exercise 10.3:** Write down the matrix transformation that rotates every thing in the plane by 180*◦* counterclockwise. Actually compute the sines and cosines so that you end up with a matrix filled with numbers in your answer. Apply this transformation to the corners of the unit square, (0, 0),(1, 0),(0, 1), and (1, 1). |
| --- |

*y*

*x*

**Example 2: Nonuniform scaling.** Let **M**2 =

3 0 0 2

and

*T*2 : **R2** *→* **R2** :

*x y*

*→* **M**2

*x y*

=

3 0 0 2

*x y*

=

3*x* 2*y*

. (10.4)

**Before** *y*

This transformation stretches everything by a factor of three in the *x*-direction

and a factor of two in the *y*-direction, as shown in Figure 10.2. If both stretch

factors were three, we’d say that the transformation “scaled things up by three”

and is a **uniform scaling transformation.** *T*2 represents a generalization of this

idea: Rather than scaling uniformly in each direction, it’s called a **nonuniform**

*x*

**scaling transformation** or, less formally, a **nonuniform scale.**

Once again the example generalizes: By placing numbers other than 2 and 3

along the diagonal of the matrix, we can scale each axis by any amount we please.

These scaling amounts can include zero and negative numbers.

**After**

| **Inline Exercise 10.4:** Write down the matrix for a uniform scale by *−*1. How does your answer relate to your answer to inline Exercise 10.3? Can you explain? |
| --- |

*Figure 10.2: T*2 *stretches the x-axis by three and the y-axis*

*by two.*

| **Inline Exercise 10.5:** Write down a transformation matrix that scales in *x* by zero and in *y* by 1. Informally describe what the associated transformation does to the house. |
| --- |

*y*

*x*

**Example 3: Shearing.** Let **M**3 =

1 2 0 1

and

*T*3 : **R2** *→* **R2** :

*x y*

*→* **M**3

*x y*

=

1 2 0 1

*x y*

=

*x* + 2*y y*

. (10.5)

**Before**

*y*

As Figure 10.3 shows, *T*3 preserves height along the *y*-axis but moves points

parallel to the *x*-axis, with the amount of movement determined by the *y*-value.

The *x*-axis itself remains fixed. Such a transformation is called a **shearing trans**

**formation.**

| **Inline Exercise 10.6:** Generalize to build a transformation that keeps the *y*-axis fixed but shears vertically instead of horizontally. |
| --- |

*x*

**Example 4: A general transformation.** Let **M**4 =

1 *−*1 2 2

and

*T*4 : **R2** *→* **R2** :

*x y*

*→* **M**4

*x y*

=

1 *−*1 2 2

*x y*

. (10.6)

**After**

*Figure 10.3: A shearing transfor mation, T*3*.*

224 Transformations in Two Dimensions

Figure 10.4 shows the effects of *T*4. It distorts the house figure, but not by just a rotation or scaling or shearing along the coordinate axes.

**Example 5: A degenerate (or singular) transformation** Let

*y*

*x*

*T*5 : **R2** *→* **R2** :

*x y*

*→*

1 *−*1 2 *−*2

*x y*

=

*x − y* 2*x −* 2*y*

. (10.7)

**Before**

Figure 10.5 shows why we call this transformation **degenerate:** Unlike the others, it collapses the whole two-dimensional plane down to a one-dimensional

*y*

subspace, a line. There’s no longer a nice correspondence between points in the domain and points in the codomain: Certain points in the codomain no longer correspond to *any* point in the domain; others correspond to *many* points in the domain. Such a transformation is also called **singular,** as is the matrix defining it. Those familiar with linear algebra will note that this is equivalent to saying that

the determinant of **M**5 = dependent.

1 *−*1 2 *−*2

is zero, or saying that its columns are linearly

*x*

**10.3 Important Facts about Transformations**

Here we’ll describe several properties of linear transformations from **R2** to **R2**. These properties are important in part because they all generalize: They apply (in some form) to transformations from **R***n* to **R***k* for any *n* and *k*. We’ll mostly be concerned with values of *n* and *k* between 1 and 4; in this section, we’ll concentrate on *n* = *k* = 2.

**10.3.1 Multiplication by a Matrix Is a Linear Transformation**

If **M** is a 2 *×* 2 matrix, then the function *T***M** defined by

*T***M** : **R2** *→* **R2** : **x** *→* **Mx** (10.8)

is linear. All five examples above demonstrate this.

For nondegenerate transformations, lines are sent to lines, as *T*1 through *T*4 show. For degenerate ones, a line may be sent to a single point. For instance, *T*5

**After**

*Figure 10.4: A general transfor mation. The house has been quite distorted, in a way that’s hard to describe simply, as we’ve done for the earlier examples.*

*y*

*x*

**Before**

*b*

sends the line consisting of all vectors of the form *b*

*y*

to the zero vector.

Because multiplication by a matrix **M** is always a linear transformation, we’ll call *T***M** the **transformation associated to the matrix M.**

**10.3.2 Multiplication by a Matrix Is the *Only* Linear Transformation**

In **R***n*, it turns out that for *every* linear transform *T*, there’s a matrix **M** with *T*(**x**) = **Mx**, which means that every linear transformation is a matrix transfor mation. We’ll see in Section 10.3.5 how to find **M**, given *T*, even if *T* is expressed in some other way. This will show that the matrix **M** is completely determined

*x*

**After**

by the transformation *T*, and we can thus call it the **matrix associated to the transformation.**

*Figure 10.5: A degenerate trans formation, T*5*.*

10.3 Important Facts about Transformations 225

As a special example, the matrix **I**, with ones on the diagonal and zeroes off the diagonal, is called the **identity matrix;** the associated transformation

*T*(**x**) = **Ix** (10.9)

is special: It’s the identity transformation that leaves every vector **x** unchanged.

| **Inline Exercise 10.7:** There is an identity matrix of every size: a 1*×*1 identity, a 2 *×* 2 identity, etc. Write out the first three. |
| --- |

**10.3.3 Function Composition and Matrix Multiplication Are Related**

If **M** and **K** are 2*×*2 matrices, then they define transformations *T***M** and *T***K**. When we compose these, we get the transformation

*T***M** *◦ T***K** : **R2** *→* **R2** : **x** *→ T***M**(*T***K**(**x**)) = *T***M**(**Kx**) (10.10) = **M**(**Kx**) (10.11)

= (**MK**)**x** (10.12)

= *T***MK**(**x**). (10.13)

In other words, the composed transformation is also a matrix transformation, with matrix **MK**. Note that when we write *T***M**(*T***K**(**x**)), the transformation *T***K** is applied *first.* So, for example, if we look at the transformation *T*2 *◦T*3, it first shears the house and *then* scales the result nonuniformly.

| **Inline Exercise 10.8:** Describe the appearance of the house after transforming it by *T*1 *◦ T*2 and after transforming it by *T*2 *◦ T*1. |
| --- |

**10.3.4 Matrix Inverse and Inverse Functions Are Related**

A matrix **M** is **invertible** if there’s a matrix **B** with the property that **BM** = **MB** = **I**. If such a matrix exists, it’s denoted **M***−*1.

If **M** is invertible and *S*(**x**) = **M***−*1**x**, then *S* is the inverse function of *T***M**, that is,

*S*(*T***M**(**x**)) = **x** and (10.14)

*T***M**(*S*(**x**)) = **x**. (10.15)

| **Inline Exercise 10.9:** Using Equation 10.13, explain why Equation 10.15 holds. |
| --- |

If **M** is not invertible, then *T***M** has no inverse.

Let’s look at our examples. The matrix for *T*1 has an inverse: Simply replace 30 by *−*30 in all the entries. The resultant transformation rotates clockwise by 30*◦*; performing one rotation and then the other effectively does nothing (i.e., it is the identity transformation). The inverse for the matrix for *T*2 is diagonal, with entries

226 Transformations in Two Dimensions

3 and 12 . The inverse of the matrix for *T*3 is 1

1 *−*2 0 1

(note the negative sign).

The associated transformation also shears parallel to the *x*-axis, but vectors in the upper half-plane are moved to the *left,* which undoes the moving to the right done by *T*3.

For these first three it was fairly easy to guess the inverse matrices, because we could understand how to invert the transformation. The inverse of the matrix

for *T*4 is

1

4

2 1 *−*2 1

, (10.16)

which we computed using a general rule for inverses of 2 *×* 2 matrices (the only such rule worth memorizing):

*a b c d*

*−*1

= 1

*ad − bc*

*d −b −c a*

. (10.17)

Finally, for *T*5, the matrix has no inverse; if it did, the function *T*5 would be invertible: It would be possible to identify, for each point in the codomain, a single point in the domain that’s sent there. But we’ve already seen this isn’t possible.

| **Inline Exercise 10.10:** Apply the formula from Equation 10.17 to the matrix for *T*5 to attempt to compute its inverse. What goes wrong? |
| --- |

**10.3.5 Finding the Matrix for a Transformation**

We’ve said that every linear transformation really is just multiplication by some matrix, but how do we *find* that matrix? Suppose, for instance, that we’d like to find a linear transformation to flip our house across the *y*-axis so that the house ends up on the left side of the *y*-axis. (Perhaps you can guess the transformation that does this, and the associated matrix, but we’ll work through the problem directly.)

The key idea is this: If we know where the transformation sends **e**1 and **e**2, we know the matrix. Why? We know that the transformation must have the form

*T*

*x y*

=

*a b c d*

*x y*

; (10.18)

we just don’t know the values of *a*, *b*, *c*, and *d*. Well, *T*(**e**1) is then

*T*

1 0

=

*a b c d*

1 0

=

*a c*

. (10.19)

Similarly, *T*(**e**2) is the vector

*b*

. So knowing *T*(**e**1) and *T*(**e**2) tells us all the *d*

matrix entries. Applying this to the problem of flipping the house, we know that *T*(**e**1) = *−***e**1, because we want a point on the positive *x*-axis to be sent to the corresponding point on the negative *x*-axis, so *a* = *−*1 and *c* = 0. On the other hand, *T*(**e**2) = **e**2, because every vector on the *y*-axis should be left untouched, so *b* = 0 and *d* = 1. Thus, the matrix for the house-flip transformation is just

*−*1 0 0 1

. (10.20)

10.3 Important Facts about Transformations 227 v1

u2 e2 Mx x

u1e1

x Kx

v2

x

KM21x

M21x x

*Figure 10.6: Multiplication by the matrix* **M** *takes* **e**1 *and* **e**2 *to* **u**1 *and* **u**2*, respectively, so multiplying* **M***−*1 *does the opposite. Multiplying by* **K** *takes* **e**1 *and* **e**2 *to* **v**1 *and* **v**2*, so multiplying first by* **M***−*1 *and then by* **K***, that is, multiplying by* **KM***−*1*, takes* **u**1 *to* **e**1 *to* **v**1*, and similarly for* **u**2*.*

| 0    **Inline Exercise 10.11:** (a) Find a matrix transformation sending **e**1 to  and  4  1    **e**2 to  .  1  (b) Use the relationship of matrix inverse to the inverse of a transform, and the 0    formula for the inverse of a 2 *×* 2 matrix, to find a transformation sending 4  1    to **e**1 and  to **e**2 as well.  1 |
| --- |

As Inline Exercise 10.11 shows, we now have the tools to send the **standard basis vectors e**1 and **e**2 to any two vectors **v**1 and **v**2, and vice versa (provided that **v**1 and **v**2 are independent, that is, neither is a multiple of the other). We can combine this with the idea that composition of linear transformations (performing one after the other) corresponds to multiplication of matrices and thus create a solution to a rather general problem.

**Problem:** Given independent vectors **u**1 and **u**2 and any two vectors **v**1 and **v**2, find a linear transformation, in matrix form, that sends **u**1 to **v**1 and **u**2 to **v**2. **Solution:** Let **M** be the matrix whose columns are **u**1 and **u**2. Then

*T* : **R2** *→* **R2** : **x** *→* **Mx** (10.21)

sends **e**1 to **u**1 and **e**2 to **u**2 (see Figure 10.6). Therefore,

*S* : **R2** *→* **R2** : **x** *→* **M***−*1**x** (10.22)

sends **u**1 to **e**1 and **u**2 to **e**2.

Now let **K** be the matrix with columns **v**1 and **v**2. The transformation *R* : **R2** *→* **R2** : **x** *→* **Kx** (10.23)

sends **e**1 to **v**1 and **e**2 to **v**2.

If we apply first *S* and then *R* to **u**1, it will be sent to **e**1 (by *S*), and thence to **v**1 by *R*; a similar argument applies to **u**2. Writing this in equations,

*R*(*S*(**x**)) = *R*(**M***−*1**x**) (10.24)

= **K**(**M***−*1**x**) (10.25)

= (**KM***−*1)**x**. (10.26)

228 Transformations in Two Dimensions

Thus, the matrix for the transformation sending the **u**’s to the **v**’s is just **KM***−*1. Let’s make this concrete with an example. We’ll find a matrix sending

(10.27)

to

**u**1 =

2 3

and **u**2 =

1 *−*1

**v**1 =

1

and **v**2 =

1

2 *−*1

, (10.28)

respectively. Following the pattern above, the matrices **M** and **K** are

**M** = **K** =

2 1 3 *−*1 1 2 1 *−*1

(10.29)

. (10.30)

Using the matrix inversion formula (Equation 10.17), we find

**M***−*1 = *−*15*−*1 *−*1

*−*3 2

so that the matrix for the overall transformation is

(10.31)

**J** = **KM***−*1 =

1 2 1 *−*1

*·−*15*−*1 *−*1 *−*3 2

(10.32)

=

7*/*5 *−*3*/*5 *−*2*/*5 3*/*5

. (10.33)

As you may have guessed, the kinds of transformations we used in WPF in Chapter 2 are internally represented as matrix transformations, and transformation groups are represented by sets of matrices that are multiplied together to generate the effect of the group.

| **Inline Exercise 10.12:** Verify that the transformation associated to the matrix **J** in Equation 10.32 really does send **u**1 to **v**1 and **u**2 to **v**2. |
| --- |

| 1    1    **Inline Exercise 10.13:** Let **u**1 =  and **u**2 =  ; pick any two nonzero  3  4  vectors you like as **v**1 and **v**2, and find the matrix transformation that sends each **u***i* to the corresponding **v***i*. |
| --- |

The recipe above for building matrix transformations shows the following: Every linear transformation from **R2** to **R2** is determined by its values on two independent vectors. In fact, this is a far more general property: Any linear trans formation from **R2** to **R***k* is determined by its values on two independent vectors, and indeed, any linear transformation from **R***n* to **R***k* is determined by its values on *n* independent vectors (where to make sense of these, we need to extend our definition of “independence” to more than two vectors, which we’ll do presently).

10.3 Important Facts about Transformations 229

**10.3.6 Transformations and Coordinate Systems**

We tend to think about linear transformations as moving points around, but leaving the origin fixed; we’ll often use them that way. Equally important, however, is their use in changing coordinate systems. If we have two coordinate systems on **R2** with the same origin, as in Figure 10.7, then every arrow has coordinates in both the red and the blue systems. The two red coordinates can be written as a vector, as

**r u**

3

**s**

can the two blue coordinates. The vector **u**, for instance, has coordinates

in

2

the red system and approximately

*−*0.2 3.6

in the blue system.

| **Inline Exercise 10.14:** Use a ruler to find the coordinates of **r** and **s** in each of the two coordinate systems. |
| --- |

We could tabulate every imaginable arrow’s coordinates in the red and blue systems to convert from red to blue coordinates. But there is a far simpler way to achieve the same result. The conversion from red coordinates to blue coordinates is *linear* and can be expressed by a matrix transformation. In this example, the matrix is

*Figure 10.7: Two different coor dinate systems for* **R2***; the vector* **u***, expressed in the red coor dinate system, has coordinates* 3 *and* 2*, indicated by the dot ted lines, while the coordinates in the blue coordinate system are approximately −*0.2 *and* 3.6*, where we’ve drawn, in each case,*

**M** = 121 *−√*3 *√*3 1

. (10.34)

*the positive side of the first coor dinate axis in bold.*

Multiplying **M** by the coordinates of **u** in the red system gets us **v** = **Mu** (10.35)

= 121 *−√*3 *√*3 1

3 2

(10.36)

= 123 *−* 2*√*3 3*√*3 + 2

(10.37)

*≈*

*−*0.2 3.6

, (10.38)

which is the coordinate vector for **u** in the blue system.

| **Inline Exercise 10.15:** Confirm, for each of the other arrows in Figure 10.7, that the same transformation converts red to blue coordinates. |
| --- |

By the way, when creating this example we computed **M** just as we did at the start of the preceding section: We found the blue coordinates of each of the two basis vectors for the red coordinate system, and used these as the columns of **M**.

In the special case where we want to go from the usual coordinates on a vector to its coordinates in some coordinate system with basis vectors **u**1, **u**2, which are *unit vectors* and *mutually perpendicular,* the transformation matrix is one whose *rows* are the transposes of **u**1 and **u**2.

For example, if **u**1 =

3*/*5 4*/*5

and **u**2 =

*−*4*/*5 3*/*5

(check for yourself that

4

these are unit length and perpendicular), then the vector **v** = 2

**u**-coordinates, is

, expressed in

230 Transformations in Two Dimensions

3*/*5 4*/*5 *−*4*/*5 3*/*5

4 2

=

4 *−*2

. (10.39)

Verify for yourself that these really *are* the **u**-coordinates of **v**, that is, that the vector **v** really is the same as 4**u**1 + (*−*2)**u**2.

**10.3.7 Matrix Properties and the Singular Value Decomposition**

Because matrices are so closely tied to linear transformations, and because lin ear transformations are so important in graphics, we’ll now briefly discuss some important properties of matrices.

First, **diagonal** matrices—ones with zeroes everywhere except on the diag onal, like the matrix **M**2 for the transformation *T*2—correspond to remarkably simple transformations: They just scale up or down each axis by some amount (although if the amount is a negative number, the corresponding axis is also flipped). Because of this simplicity, we’ll try to understand other transformations in terms of these diagonal matrices.

Second, if the columns of the matrix **M** are **v**1, **v**2, *...* , **v***k ∈ Rn*, and they are pairwise orthogonal unit vectors, then **MTM** = **I***k*, the *k × k* identity matrix. In the special case where *k* = *n*, such a matrix is called **orthogonal.** If the determinant of the matrix is 1, then the matrix is said to be a **special orthogonal** matrix. In **R2**, such a matrix must be a rotation matrix like the one in *T*1; in **R3**, the transformation associated to such a matrix corresponds to rotation around some vector by some amount.1

Less familiar to most students, but of enormous importance in much graph ics research, is the **singular value decomposition (SVD)** of a matrix. Its exis tence says, informally, that if we have a transformation *T* represented by a matrix **M**, and if we’re willing to use new coordinate systems on both the domain and codomain, then the transformation simply looks like a nonuniform (or possibly uniform) scaling transformation. We’ll briefly discuss this idea here, along with the application of the SVD to solving equations; the web materials for this chapter show the SVD for our example transformations and some further applications of the SVD.

The singular value decomposition theorem says this:

Every *n × k* matrix **M** can be factored in the form

**M** = **UDVT**, (10.40)

where **U** is *n × r* (where *r* = min(*n*, *k*)) with orthonormal columns, **D** is *r × r* diagonal (i.e., only entries of the form *dii* can be nonzero), and **V** is *r × k* with orthonormal columns (see Figure 10.8).

By convention, the entries of **D** are required to be in nonincreasing order (i.e., *|d*1,1*|≥|d*2,2*|≥|d*3,3*| ...*) and are indicated by single subscripts (i.e., we write *d*1 instead of *d*1,1). They are called the **singular values** of **M**. It turns out that *M* is degenerate (i.e., singular) exactly if any singular value is 0. As a general

1. As we mentioned in Chapter 3, rotation about a vector in **R3** is better expressed as rotation *in a plane,* so instead of speaking about rotation about *z*, we speak of rotation in the *xy*-plane. We can then say that any special orthogonal matrix in **R**4 corresponds to a sequence of two rotations in two planes in 4-space.

10.3 Important Facts about Transformations 231

D V = t M U

=

M UD V = t =

(a) (b)

*Figure 10.8: (a) An n × k matrix, with n > k, factors as a product of an n × n matrix with orthonormal columns (indicated by the vertical stripes on the first rectangle), a diagonal k×k matrix, and a k×k matrix with orthonormal rows (indicated by the horizontal stripes), which we write as* **UDVT***, where* **U** *and* **V** *have orthonormal* columns. *(b) An n × k matrix with n < k is written as a similar product; note that the diagonal matrix in both cases is square, and its size is the smaller of n and k.*

guideline, if the ratio of the largest to the smallest singular values is very large (say, 106), then numerical computations with the matrix are likely to be unstable.

| **Inline Exercise 10.16:** The singular value decomposition is not unique. If we negate the first row of **VT** and the first column of **U** in the SVD of a matrix **M**, show that the result is still an SVD for **M**. |
| --- |

In the special case where *n* = *k* (the one we most often encounter), the matri ces **U** and **V** are both square and represent change-of-coordinate transformations in the domain and codomain. Thus, we can see the transformation

*T*(**x**) = **Mx** (10.41)

as a sequence of three steps: (1) Multiplication by **VT** converts **x** to **v**-coordinates; (2) multiplication by **D** amounts to a possibly nonuniform scaling along each axis; and (3) multiplication by **U** treats the resultant entries as coordinates in the **u**-coordinate system, which then are transformed back to standard coordinates.

**10.3.8 Computing the SVD**

How do we find **U**, **D**, and **V**? In general it’s relatively difficult, and we rely on numerical linear algebra packages to do it for us. Furthermore, the results are by no means unique: A single matrix may have multiple singular value decompositions. For instance, if **S** is *any n × n* matrix with orthonormal columns, then

**I** = **SIST** (10.42)

is one possible singular value decomposition of the identity matrix. Even though there are many possible SVDs, the singular values are the same for all decompo sitions.

The **rank** of the matrix **M**, which is defined as the number of linearly inde pendent columns, turns out to be exactly the number of nonzero singular values.

**10.3.9 The SVD and Pseudoinverses**

Again, in the special case where *n* = *k* so that **U** and **V** are square, it’s easy to compute **M***−*1 if you know the SVD:

**M***−*1 = **V***D−*1**UT**, (10.43)

232 Transformations in Two Dimensions

where *D−*1 is easy to compute—you simply invert all the elements of the diagonal. If one of these elements is zero, the matrix is singular and no such inverse exists; in this case, the **pseudoinverse** is also often useful. It’s defined as

**M***†* = **V***D†***UT**, (10.44)

where *D†* is just *D* with every nonzero entry inverted (i.e., you try to invert the diagonal matrix *D* by inverting diagonal elements, and every time you encounter a zero on the diagonal, you ignore it and simply write down 0 in the answer). The definition of the pseudoinverse makes sense even when *n* = *k*; the pseudoinverse can be used to solve “least squares” problems, which frequently arise in graphics.

**The Pseudoinverse Theorem:**

(a) If **M** is an *n × k* matrix with *n > k*, the equation **Mx** = **b** generally represents an overdetermined system of equations2 which may have no solution. The vector

**x**0 = **M***†***b** (10.45)

represents an optimal “solution” to this system, in the sense that **Mx**0 is as close to **b** as possible.

(b) If **M** is an *n × k* matrix with *n < k*, and rank *n*, the equation **Mx** = **b** represents an underdetermined system of equations.3 The vector

**x**0 = **M***†***b** (10.46)

represents an optimal solution to this system, in the sense that **x**0 is the *shortest* vector satisfying **Mx** = **b**.

Here are examples of each of these cases.

**Example 1: An overdetermined system**

The system

*t*= 43(10.47)

2

1

has *no* solution: There’s simply no number *t* with 2*t* = 4 and 1*t* = 3 (see Fig

2

ure 10.9). But among all the multiples of **M** = 1

, there *is* one that’s closest to

(4*,* 3)

the vector **b** =

4 3

, namely 2.2

2 1

=

4. 4 2.2

, as you can discover with elemen

2

tary geometry. The theorem tells us we can compute this directly, however, using the pseudoinverse. The SVD and pseudoinverse of **M** are

**M** = **UDVT** = ( 1*~~√~~*5 21) *√*51(10.48)

1

*Figure 10.9: The equations*

*t*

2 1

=

4 3

*have no common*

**M***†* = **VD***†***U** = 11*/√*5( 1*~~√~~*52 1) (10.49) = 0.4 0.2. (10.50)

2. In other words, a situation like “five equations in three unknowns.” 3. That is, a situation like “three equations in five unknowns.”

*solution. But the multiples of the vector* [2 1]**T** *form a line in the plane that passes by the point* (4, 3)*, and there’s a point of this line (shown in a red circle on the topmost arrow) that’s as close to* (4, 3) *as possible.*

10.4 Translation 233

And the solution guaranteed by the theorem is

*t* = **M***†***b** = 0.4 0.2  43= 2.2. (10.51)

**Example 2: An underdetermined system**

The system

1 3 *xy*= 4 (10.52)

has a great many solutions; any point (*x*, *y*) on the line *x* + 3*y* = 4 is a solution

*y* = 3/4

*x* + 3*y* = 4

*x* = 4

(see Figure 10.10). The solution that’s *closest to the origin* is the point on the line *x* + 3*y* = 4 that’s as near to (0, 0) as possible, which turns out to be *x* = 0.4; *y* =

1.2. In this case, the matrix **M** is 1 3; its SVD and pseudoinverse are simply **M** = **UDVT** = 1 *√*101*/√*10 3*/√*10and (10.53)

*Figure 10.10: Any point of the blue line is a solution; the red point is closest to the origin.*

**M***†* = **VD***†***U** =

1*/√*10 3*/√*10

1*/√*101=1*/*10 3*/*10

. (10.54)

And the solution guaranteed by the theorem is

**M***†***b** =

1*/*10 3*/*10

4=0.4 1.2

. (10.55)

Of course, this kind of computation is much more interesting in the case where the matrices are much larger, but all the essential characteristics are present even in these simple examples.

A particularly interesting example arises when we have, for instance, two polyhedral models (consisting of perhaps hundreds of vertices joined by trian gular faces) that might be “essentially identical”: One might be just a translated, rotated, and scaled version of the other. In Section 10.4, we’ll see how to represent translation along with rotation and scaling in terms of matrix multiplication. We can determine whether the two models are in fact essentially identical by listing the coordinates of the first in the columns of a matrix **V** and the coordinates of the second in a matrix **W**, and then seeking a matrix **A** with

**AV** = **W**. (10.56)

This amounts to solving the “overconstrained system” problem; we find that **A** = **V***†***W** is the best possible solution. If, having computed **A**, we find that

**AV** = **W**, (10.57)

then the models are essentially identical; if the left and right sides differ, then the models are not essentially identical. (This entire approach depends, of course, on corresponding vertices of the two models being listed in the corresponding order; the more general problem is a lot more difficult.)

**10.4 Translation**

We now describe a way to apply linear transformations to generate *translations,* and at the same time give a nice model for the points-versus-vectors ideas we’ve espoused so far.

234 Transformations in Two Dimensions

The idea is this: As our Euclidean plane (our set of *points*), we’ll take the plane *w* = 1 in *xyw*-space (see Figure 10.11). The use of *w* here is in preparation for what we’ll do in 3-space, which is to consider the three-dimensional set defined by *w* = 1 in *xyzw*-space.

Having done this, we can consider transformations that multiply such vectors by a 3 *×* 3 matrix **M**. The only problem is that the result of such a multiplication may not have a 1 as its last entry. We can restrict our attention to those that do:

*w*

*y x*

*Figure 10.11: The w* = 1 *plane in xyw-space.*

⎡

⎣*abc*

*def pqr*

⎤ ⎦

⎡

⎣*xy* 1

⎤

⎦ =

⎡

⎣*xy* 1

⎤

⎦ . (10.58)

For this equation to hold for every *x* and *y*, we must have *px* + *qy* + *r* = 1 for all *x*, *y*. This forces *p* = *q* = 0 and *r* = 1.

Thus, we’ll consider transformations of the form

⎡

⎣*abc*

*def* 001

⎤ ⎦

⎡

⎣*xy* 1

⎤

⎦ =

⎡

⎣*xy* 1

⎤

⎦ . (10.59)

If we examine the special case where the upper-left corner is a 2 *×* 2 identity matrix, we get

⎡

⎣1 0 *c*

0 1 *f* 001

⎤ ⎦

⎡

⎣*xy* 1

⎤

⎦ =

⎡

⎣*x* + *c*

*y* + *f* 1

⎤

⎦ . (10.60)

As long as we pay attention only to the *x*- and *y*-coordinates, this looks like a translation! We’ve added *c* to each *x*-coordinate and *f* to each *y*-coordinate (see Figure 10.12). Transformations like this, restricted to the plane *w* = 1, are called **affine transformations** of the plane. Affine transformations are the ones most often used in graphics.

On the other hand, if we make *c* = *f* = 0, then the third coordinate becomes irrelevant, and the upper-left 2*×*2 matrix can perform any of the operations we’ve seen up until now. Thus, with the simple trick of adding a third coordinate and requiring that it always be 1, we’ve managed to unify rotation, scaling, and all the other linear transformations with the new class of transformations, *translations,* to get the class of affine transformations.

**10.5 Points and Vectors Again**

Back in Chapter 7, we said that points and vectors could be combined in certain ways: The difference of points is a vector, a vector could be added to a point

*T*

*Figure 10.12: The house figure, before and after a translation generated by* shearing *par allel to the w* = 1 *plane.*

10.6 Why Use 3 *×* 3 Matrices Instead of a Matrix and a Vector? 235

to get a new point, and more generally, affine combinations of points, that is, combinations of the form

*α*1*P*1 + *α*2*P*2 + *...* + *αkPk*, (10.61)

were allowed if and only if *α*1 + *α*2 + *...* + *αk* = 1.

We now have a situation in which these distinctions make sense in terms of familiar mathematics: We can regard *points* of the plane as being elements of **R3** whose third coordinate is 1, and *vectors* as being elements of **R3** whose third coordinate is 0.

With this convention, it’s clear that the difference of points is a vector, the sum of a vector and a point is a point, and combinations like the one in Equation 10.61 yield a point if and only if the sum of the coefficients is 1 (because the third coordinate of the result will be exactly the sum of the coefficients; for the sum to be a *point,* this third coordinate is required to be 1).

You may ask, “Why, when we’re already familiar with vectors in 3-space, should we bother calling some of them ‘points in the Euclidean plane’ and others ‘two-dimensional vectors’?” The answer is that the distinctions have geometric significance when we’re using this subset of 3-space as a model for 2D transfor mations. Adding vectors in 3-space is defined in linear algebra, but adding together two of our “points” gives a location in 3-space that’s not on the *w* = 1 plane or the *w* = 0 plane, so we don’t have a name for it at all.

Henceforth we’ll use *E*2 (for “Euclidean two-dimensional space”) to denote this *w* = 1 plane in *xyw*-space, and we’ll write (*x*, *y*) to mean the point of *E*2

corresponding to the 3-space vector

⎡

⎣*xy* 1

⎤

⎦. It’s conventional to speak of an affine

transformation as acting on *E*2, even though it’s defined by a 3 *×* 3 matrix.

**10.6 Why Use 3** *×* **3 Matrices Instead of a Matrix and a Vector?**

Students sometimes wonder why they can’t just represent a linear transformation plus translation in the form

*T*(**x**) = **Mx** + **b**, (10.62)

where the matrix **M** represents the linear part (rotating, scaling, and shearing) and **b** represents the translation.

First, you *can* do that, and it works just fine. You might save a tiny bit of storage (four numbers for the matrix and two for the vector, so six numbers instead of nine), but since our matrices always have two 0s and a 1 in the third column, we don’t really need to store that column anyhow, so it’s the same. Otherwise, there’s no important difference.

Second, the reason to unify the transformations into a single matrix is that it’s then very easy to take multiple transformations (each represented by a matrix) and **compose** them (perform one after another): We just multiply their matrices together in the right order to get the matrix for the composed transformation. You can do this in the matrix-and-vector formulation as well, but the programming is slightly messier and more error-prone.

236 Transformations in Two Dimensions

There’s a third reason, however: It’ll soon become apparent that we can also work with triples whose third entry is neither 1 nor 0, and use the operation of **homogenization** (dividing by *w*) to convert these to points (i.e., triples with *w* = 1), except when *w* = 0. This allows us to study even more transformations, one of which is central to the study of perspective, as we’ll see later.

The singular value decomposition provides the tool necessary to decompose not just linear transformations, but affine ones as well (i.e., combinations of linear transformations and translations).

**10.7 Windowing Transformations**

*v*

As an application of our new, richer set of transformations, let’s examine **window ing transformations,** which send one axis-aligned rectangle to another, as shown

(*u*2, *v*2)

in Figure 10.13. (We already discussed this briefly in Chapter 3.) We’ll first take a direct approach involving a little algebra. We’ll then examine a more automated approach.

We’ll need to do essentially the same thing to the first and second coordinates, so let’s look at how to transform the first coordinate only. We need to send *u*1 to *x*1 and *u*2 to *x*2. That means we need to scale up any coordinate *difference* by the factor *x*2*−x*1

*u*2*−u*1. So our transformation for the first coordinate has the form *t → x*2 *− x*1

*u*2 *− u*1*t* + something. (10.63)

If we apply this to *t* = *u*1, we know that we want to get *x*1; this leads to the equation

(*u*1, *v*1) (*u*2, *v*1)

*y*

(*x*2, *y*2)

*u x*

*x*2 *− x*1

*u*2 *− u*1*u*1 + something = *x*1. (10.64)

Solving for the missing offset gives

(*x*1, *y*1)

(*x*2, *y*1)

*x*1 *− x*2 *− x*1

*u*2 *− u*1*u*1 = *x*1*u*2 *− u*1

*u*2 *− u*1*− x*2 *− x*1

*u*2 *− u*1*u*1 (10.65)

= *x*1*u*2 *− x*1*u*1 *− x*2*u*1 + *x*1*u*1

*u*2 *− u*1(10.66)

= *x*1*u*2 *− x*2*u*1

*u*2 *− u*1, (10.67)

so that the transformation is

*t → x*2 *− x*1

*u*2 *− u*1*t* + *x*1*u*2 *− x*2*u*1

*u*2 *− u*1. (10.68)

Doing essentially the same thing for the *v* and *y* terms (i.e., the second coordi nate) we get the transformation, which we can write in matrix form:

*T*(**x**) = **Mx**, (10.69)

where

*Figure 10.13: Window transfor mation setup. We need to move the uv-rectangle to the xy rectangle.*

⎡

⎤

*x*2*−x*1

*u*2*−u*1 0 *x*1*u*2*−x*2*u*1

**M** =

⎣

0 *y*2*−y*1 *v*2*−v*1

*u*2*−u*1

*y*1*v*2*−y*2*v*1 *v*2*−v*1

⎦ . (10.70)

00 1

10.8 Building 3D Transformations 237

| **Inline Exercise 10.17:** Multiply the matrix **M** of Equation 10.70 by the vector *u*1 *v*1 1 **T** to confirm that you do get *x*1 *y*1 1 **T**. Do the same for the opposite corner of the rectangle. |
| --- |

We’ll now show you a second way to build this transformation (and many others as well).

**10.8 Building 3D Transformations**

Recall that in 2D we could send the vectors **e**1 and **e**2 to the vectors **v**1 and **v**2 by building a matrix **M** whose columns were **v**1 and **v**2, and then use two such matrices (inverting one along the way) to send any two independent vectors **v**1 and **v**2 to any two vectors **w**1 and **w**2. We can do the same thing in 3-space: We can send the standard basis vectors **e**1, **e**2, and **e**3 to any three other vectors, just by using those vectors as the columns of a matrix. Let’s start by sending **e**1, **e**2, and **e**3 to three corners of our first rectangle—the two we’ve already specified and the lower-right one, at location (*u*2, *v*1). The three vectors corresponding to these

points are⎡ ⎣*u*1*v*1

1

⎤

⎦ ,

⎡

⎣*u*2

*v*2

1

⎤

⎦ , and

⎡

⎣*u*2

*v*1

1

⎤

⎦ . (10.71)

Because the three corners of the rectangle are not collinear, the three vectors are independent. Indeed, this is our definition of independence for vectors in *n*-space: Vectors **v**1, *...* , **v**k are independent if there’s no (*k−*1)-dimensional subspace con taining them. In 3-space, for instance, three vectors are independent if there’s no plane through the origin containing all of them.

So the matrix

**M**1 =

⎡

⎣*u*1 *u*2 *u*2

*v*1 *v*2 *v*1 111

⎤

⎦ , (10.72)

which performs the desired transformation, will be invertible. We can similarly build the matrix **M**2, with the corresponding *x*s and *y*s in it. Finally, we can compute

**M**2**M***−*1

1 , (10.73)

which will perform the desired transformation. For instance, the lower-left cor ner of the starting rectangle will be sent, by **M***−*1

1 , to **e**1 (because **M**1 sent **e**1 to

the lower-left corner); multiplying **e**1 by **M**2 will send it to the lower-left corner of the target rectangle. A similar argument applies to all three corners. Indeed, if we compute the inverse algebraically and multiply out everything, we’ll once again arrive at the matrix given in Equation 10.7. But we don’t need to do so: We know that this must be the right matrix. Assuming we’re willing to use a matrix inversion routine, there’s no need to think through anything more than “I want these three points to be sent to these three other points.”

Summary: Given any three noncollinear points *P*1, *P*2, *P*3 in *E*2, we can find a matrix transformation and send them to any three points *Q*1, *Q*2, *Q*3 with the procedure above.

238 Transformations in Two Dimensions

**10.9 Another Example of Building a 2D Transformation**

Suppose we want to find a 3*×*3 matrix transformation that rotates the entire plane 30*◦* counterclockwise around the point *P* = (2, 4), as shown in Figure 10.14. As you’ll recall, WPF expresses this transformation via code like this:

| <RotateTransform Angle="-30" CenterX="2" CenterY="4"/> |
| --- |

An implementer of WPF then must create a matrix like the one we’re about to build.

Here are two approaches.

*y*

*x*

*y*

(2, 4)

First, we know how to rotate about the origin by 30*◦*; we can use the transfor

(2, 4)

mation *T*1 from the start of the chapter. So we can do our desired transformation

in three steps (see Figure 10.15).

1. Move the point (2, 4) to the origin.

*x*

2. Rotate by 30*◦*.

3. Move the origin back to (2, 4).

The matrix that moves the point (2, 4) to the origin is

⎡

⎣1 0 *−*2

0 1 *−*4 00 1

⎤

⎦ . (10.74)

*Figure 10.14: We’d like to rotate the entire plane by* 30*◦ counter clockwise about the point P* = (2, 4)*.*

The one that moves it back is similar, except that the 2 and 4 are not negated. And the rotation matrix (expressed in our new 3 *×* 3 format) is

⎡ ⎣

cos 30*◦ −* sin 30*◦* 0 sin 30*◦* cos 30*◦* 0 0 01

⎤

⎦ . (10.75)

The matrix representing the entire sequence of transformations is therefore

⎡

⎣102

014 001

⎤

⎡

⎤

⎣cos 30*◦ −* sin 30*◦* 0 ⎦

⎦

sin 30*◦* cos 30*◦* 0 0 01

⎡

⎣1 0 *−*2

0 1 *−*4 00 1

⎤

⎦ . (10.76)

| **Inline Exercise 10.18:** (a) Explain why this is the correct order in which to multiply the transformations to get the desired result.  (b) Verify that the point (2, 4) is indeed left unmoved by multiplying 241 **T** by the sequence of matrices above. |
| --- |

The second approach is again more automatic: We find three points whose target locations we know, just as we did with the windowing transformation above. We’ll use *P* = (2, 4), *Q* = (3, 4) (the point one unit to the right of *P*), and *R* = (2, 5) (the point one unit above *P*). We know that we want *P* sent to *P*, *Q* sent to (2+ cos 30*◦*, 4+ sin 30*◦*), and *R* sent to (2*−*sin 30*◦*, 4+ cos 30*◦*). (Draw a picture to convince yourself that these are correct). The matrix that achieves this is just

10.9 Another Example of Building a 2D Transformation 239

⎤

⎡

⎣2 2 + cos 30*◦* 4 *−* sin 30*◦* ⎦

4 4 + sin 30*◦* 4 + cos 30*◦* 111

⎡

⎣232

445 111

⎤

*−*1

*y*

⎦

. (10.77)

Both approaches are reasonably easy to work with.

*x*

There’s a third approach—a variation of the second—in which we specify

where we want to send a point and two vectors, rather than three points. In this

case, we might say that we want the point *P* to remain fixed, and the vectors **e**1

and **e**2 to go to

⎡

⎣cos 30*◦*

sin 30*◦* 0

⎤

⎦ and

⎡

⎣*−* sin 30*◦*

cos 30*◦*

0

⎤

⎦ , (10.78)

*y*

respectively. In this case, instead of finding matrices that send the vectors **e**1, **e**2,

*x*

and **e**3 to the desired three points, before and after, we find matrices that send those

vectors to the desired point and two vectors, before and after. These matrices are

⎡ ⎣

210 401 100

⎤

⎦ and

⎡

⎣2 cos 30*◦ −* sin 30*◦*

4 sin 30*◦* cos 30*◦* 10 0

⎤

⎦ , (10.79)

*y*

so the overall matrix is

⎡

⎤

⎣2 cos 30*◦ −* sin 30*◦*

⎦

4 sin 30*◦* cos 30*◦*

10 0

⎡

⎣210

401 100

⎤

*−*1

*x*

⎦

. (10.80)

These general techniques can be applied to create any linear-plus-translation transformation of the *w* = 1 plane, but there are some specific ones that are good to know. Rotation in the *xy*-plane, by an amount *θ* (rotating the positive *x*-axis toward the positive *y*-axis) is given by

*Figure 10.15: The house after translating* (2, 4) *to the origin, after rotating by* 30*◦, and after*

*Rxy*(*θ*) =

⎡

⎣cos *θ −* sin *θ* 0

sin *θ* cos *θ* 0 0 01

⎤

⎦ . (10.81)

*translating the origin back to* (2, 4)*.*

In some books and software packages, this is called **rotation around *z*;** we prefer the term “rotation in the *xy*-plane” because it also indicates the direction of rotation (from *x*, toward *y*). The other two standard rotations are

⎤

⎦ (10.82)

and

*Ryz*(*θ*) =

⎡

⎣10 0

0 cos *θ −* sin *θ* 0 sin *θ* cos *θ*

*Rzx*(*θ*) =

⎡

⎣cos *θ* 0 sin *θ*

01 0

*−* sin *θ* 0 cos *θ*

⎤

⎦ ; (10.83)

note that the last expression rotates *z* toward *x*, and *not* the opposite. Using this naming convention helps keep the pattern of plusses and minuses symmetric.

240 Transformations in Two Dimensions

**10.10 Coordinate Frames**

In 2D, a linear transformation is completely specified by its values on two indepen dent vectors. An affine transformation (i.e., linear plus translation) is completely specified by its values on any three noncollinear points, or on any point and pair of independent vectors. A projective transformation on the plane (which we’ll dis cuss briefly in Section 10.13) is specified by its values on four points, no three collinear, or on other possible sets of points and vectors. These facts, and the cor responding ones for transformations on 3-space, are so important that we enshrine them in a principle:

| **THE TRANSFORMATION UNIQUENESS PRINCIPLE:** For each class of transformations—linear, affine, and projective—and any corresponding coor dinate frame, and any set of corresponding target elements, there’s a unique transformation mapping the frame elements to the correponding elements in the target frame. If the target elements themselves constitute a frame, then the transformation is invertible. |
| --- |

To make sense of this, we need to define a **coordinate frame.** As a first exam ple, a coordinate frame for linear transformations is just a “basis”: In two dimen sions, that means “two linearly independent vectors in the plane.” The elements of the frame are the two vectors. So the principle says that if **u** and **v** are linearly independent vectors in the plane, and **u**and **v**are any two vectors, then there’s a unique linear transformation sending **u** to **u**and **v** to **v**. It further says that if **u**and **v**are independent, then the transformation is invertible.

More generally, a **coordinate frame** is a set of geometric elements rich enough to uniquely characterize a transformation in some class. For linear transformations of the plane, a coordinate frame consists of two independent vectors in the plane, as we said; for affine transforms of the plane, it consists of three noncollinear points in the plane, *or* of one point and two independent vectors, etc.

In cases where there are multiple kinds of coordinate frames, there’s always a way to convert between them. For 2D affine transformations, the three non collinear points *P*, *Q*, and *R* can be converted to *P*, **v**1 = *Q − P*, and **v**2 = *R − P*; the conversion in the other direction is obvious. (It may not be obvious that the vectors **v**1 and **v**2 are linearly independent. See Exercise 10.4.)

There’s a restricted use of “coordinate frame” for affine maps that has some advantages. Based on the notion that the origin and the unit vectors along the posi tive directions for each axis form a frame, we’ll say that a **rigid coordinate frame** for the plane is a triple (*P*, **v**1, **v**2), where *P* is a point and **v**1 and **v**2 are *perpendic ular* unit vectors with the rotation from **v**1 toward **v**2 being counterclockwise (i.e.,

with

0 *−*1 1 0

**v**1 = **v**2). The corresponding definition for 3-space has one point

and three mutually perpendicular unit vectors forming a right-hand coordinate system. Transforming one rigid coordinate frame (*P*, **v**1, **v**2) to another (*Q*, **u**1, **u**2) can always be effected by a sequence of transformation,

*TQ ◦ R ◦ T−*1

*P* , (10.84)

where *TP*(*A*) = *A*+*P* is translation by *P*, and similarly for *TQ*, and *R* is the rotation given by

*R* = [**u**1; **u**2] *·* [**v**1; **v**2]**T**, (10.85)

10.11 Application: Rendering from a Scene Graph 241

where the semicolon indicates that **u**1 is the first column of the first factor, etc.

The G3D library, which we use in examples in Chapters 12, 15, and 32, uses rigid coordinate frames extensively in modeling, encapsulating them in a class, CFrame.

**10.11 Application: Rendering from a Scene Graph**

We’ve discussed affine transformations on a two-dimensional affine space, and how, once we have a coordinate system and can represent points as triples, as in **x** = *x y* 1 **T**, we can represent a transformation by a 3 *×* 3 matrix **M**. We transform the point **x** by multiplying it on the left by **M** to get **Mx**. With this in mind, let’s return to the clock example of Chapter 2 and ask how we could start from a WPF description and convert it to an image, that is, how we’d do some of



*Figure 10.16: Our clock model. y* 5 21

the work that WPF does. You’ll recall that the clock shown in Figure 10.16 was created in WPF with code like this,

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

*x*

| <Canvas ... >  <Ellipse  Canvas.Left="-10.0" Canvas.Top="-10.0"  Width="20.0" Height="20.0"  Fill="lightgray" />  <Control Name="Hour Hand" .../>  <Control Name="Minute Hand" .../>  <Canvas.RenderTransform>  <TransformGroup>  <ScaleTransform ScaleX="4.8" ScaleY="4.8" />  <TranslateTransform X="48" Y="48" />  </TransformGroup>  </Canvas.RenderTransform>  </Canvas> |
| --- |

1

2

3

4

5

6

7

8

9

10

*y* 5 9

11

*y*

12

13

14

where the code for the hour hand is

*Figure 10.17: The clock-hand template.*

| <Control Name="HourHand" Template="{StaticResource ClockHandTemplate}"> <Control.RenderTransform>  <TransformGroup>  <ScaleTransform ScaleX="1.7" ScaleY="0.7" />  <RotateTransform Angle="180"/>  <RotateTransform x:Name="ActualTimeHour" Angle="0"/>  </TransformGroup>  </Control.RenderTransform>  </Control> |
| --- |

1

2

3

4

5

6

7

8

9

and the code for the minute hand is similar, the only differences being that ActualTimeHour is replaced by ActualTimeMinute and the scale by 1.7 in *X* and 0.7 in *Y* is omitted.

The ClockHandTemplate was a polygon defined by five points in the plane: (*−*0. 3, *−*1),(*−*0.2, 8),(0, 9),(0.2, 8), and (0. 3, *−*1) (see Figure 10.17). We’re going to slightly modify this code so that the clock face and clock hands are both described in the same way, as polygons. We *could* create a polygonal version of the circular face by making a regular polygon with, say, 1000 vertices, but to keep the code simple and readable, we’ll make an octagonal approximation of a circle instead.

242 Transformations in Two Dimensions Now the code begins like this:

| <Canvas ...  <Canvas.Resources>  <ControlTemplate x:Key="ClockHandTemplate">  <Polygon  Points="-0.3,-1 -0.2,8 0,9 0.2,8 0.3,-1"  Fill="Navy"/>  </ControlTemplate>  <ControlTemplate x:Key="CircleTemplate">  <Polygon  Points="1,0 0.707,0.707 0,1 -.707,.707  -1,0 -.707,-.707 0,-1 0.707,-.707"  Fill="LightGray"/>  </ControlTemplate>  </Canvas.Resources> |
| --- |

1

2

3

4

5

6

7

8

9

10

11

12

13

14

This code defines the geometry that we’ll use to create the face and hands of the clock. With this change, the circular clock face will be defined by transforming a template “circle,” represented by eight evenly spaced points on the unit circle. This form of specification, although not idiomatic in WPF, is quite similar to scene specification in many other scene-graph packages.

The actual creation of the scene now includes building the clock face from the CircleTemplate, and building the hands as before.

| <!- 1. Background of the clock ->  <Control Name="Face"  Template="{StaticResource CircleTemplate}">  <Control.RenderTransform>  <ScaleTransform ScaleX="10" ScaleY="10" />  </Control.RenderTransform>  </Control>  <!- 2. The minute hand ->  <Control Name="MinuteHand"  Template="{StaticResource ClockHandTemplate}">  <Control.RenderTransform>  <TransformGroup>  <RotateTransform Angle="180" />  <RotateTransform x:Name="ActualTimeMinute" Angle="0" />  </TransformGroup>  </Control.RenderTransform>  </Control>  <!- 3. The hour hand ->  <Control Name="HourHand" Template="{StaticResource ClockHandTemplate}"> <Control.RenderTransform>  <TransformGroup>  <ScaleTransform ScaleX="1.7" ScaleY="0.7" />  <RotateTransform Angle="180" />  <RotateTransform x:Name="ActualTimeHour"  Angle="0" />  </TransformGroup>  </Control.RenderTransform>  </Control> |
| --- |

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

All that remains is the transformation from Canvas to WPF coordinates, and the timers for the animation, which set the ActualTimeMinute and ActualTimeHour values.

10.11 Application: Rendering from a Scene Graph 243

| <Canvas.RenderTransform>  ...same as before...  </Canvas.RenderTransform>  <Canvas.Triggers>  <EventTrigger RoutedEvent="FrameworkElement.Loaded"> <BeginStoryboard>  <Storyboard>  <DoubleAnimation  Storyboard.TargetName="ActualTimeHour"  Storyboard.TargetProperty="Angle"  From="0.0" To="360.0"  Duration="00:00:01:0" RepeatBehavior="Forever"  />  <DoubleAnimation  Storyboard.TargetName="ActualTimeMinute"  Storyboard.TargetProperty="Angle"  From="0.0" To="4320.0"  Duration="00:00:01:0" RepeatBehavior="Forever"  />  </Storyboard>  </BeginStoryboard>  </EventTrigger>  </Canvas.Triggers>  </Canvas> |
| --- |

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

As a starting point in transforming this scene description into an image, we’ll

assume that we have a basic graphics library that, given an array of points rep resenting a polygon, can draw that polygon. The points will be represented by a 3*×k* array of homogeneous coordinate triples, so the first column of the array will be the homogeneous coordinates of the first polygon point, etc.

We’ll now explain how we can go from something like the WPF description to a sequence of drawPolygon calls. First, let’s transform the XAML code into a tree structure, as shown in Figure 10.18, representing the scene graph (see Chapter 6).

We’ve drawn transformations as diamonds, geometry as blue boxes, and named parts as beige boxes. For the moment, we’ve omitted the matter of instanc ing of the ClockHandTemplate and pretended that we have two separate identical copies of the geometry for a clock hand. We’ve also drawn next to each transfor mation the matrix representation of the transformation. We’ve assumed that the angle in ActualTimeHour is 15*◦* (whose cosine and sine are approximately 0.96 and 0.26, respectively) and the angle in ActualTimeMinutes is 180*◦* (i.e., the clock is showing 12:30).

| **Inline Exercise 10.19:** (a) Remembering that rotations in WPF are specified in degrees and that they rotate objects in a *clockwise* direction, check that the matrix given for the rotation of the hour hand by 15*◦* is correct. (b) If you found that the matrix was wrong, recall that in WPF *x* increases to the right and *y* increases *down.* Does this change your answer? By the way, if you ran this program in WPF and debugged it and printed the matrix, you’d find the negative sign on the (2, 1) entry instead of the (1, 2) entry. That’s because WPF internally uses row vectors to represents points, and multiplies them by transformation matrices on the right. |
| --- |

244 Transformations in Two Dimensions

WPF 



13 4

Trans 48, 48 



1

48 48 1

Scale 

4.8, 4.8





Canvas 



4.8 3 4

4.8

1







Face

Minute hand

Hour hand

10 3 4 1 

Scale 10 10

–1 3 4 1

Rot 180 Rot 15

10

–1

1

–1 3 4 

0.96 –0.26 3 4 0.26 0.96



Circle

Rot 180

13 4 1 –1

Rot 180

–1

1



Hand 1

Scale

1.7 3 4 0.7

1.7, 0.7 1

Hand 2

*Figure 10.18: A scene-graph representation of the XAML code for the clock.*

The order of items in the tree is a little different from the textual order, but there’s a natural correspondence between the two. If you consider the hour hand and look at all transformations that occur in its associated render transform or in the render transform of anything containing it (i.e., the whole clock), those are exactly the transforms you encounter as you read from the leaf node corresponding to the hour hand up toward the root node.

| **Inline Exercise 10.20:** Write down all transformations applied to the circle template that’s used as the clock face by reading the XAML program. Confirm that they’re the same ones you get by reading upward from the “Circle” box in Figure 10.18. |
| --- |

In the scene graph we’ve drawn, the transformation matrices are the most important elements. We’re now going to discuss how these matrices and the coor dinates of the points in the geometry nodes interact.

Recall that there are two ways to think about transformations. The first is to say that the minute hand, for instance, has a rotation operation applied to each of its points, creating a new minute hand, which in turn has a translation applied to each point, creating yet another new minute hand, etc. The tip of the minute hand is at location (0, 9), once and for all. The tip of the *rotated* minute hand is somewhere else, and the tip of the translated and rotated minute hand is somewhere else again. It’s common to talk about all of these as if they were the same thing (“*Now* the tip of the minute hand is at (3, 17). . . ”), but that doesn’t really make sense—the tip of the minute hand cannot be in two different places.

10.11 Application: Rendering from a Scene Graph 245

The second view says that there are several different coordinate systems, and that the transformations tell you how to get from the tip’s coordinates in one sys tem to its coordinates in another. We can then say things like, “The tip of the minute hand is at (0, 9) in **object space** or **object coordinates,** but it’s at (0, *−*9) in canvas coordinates.” Of course, the position in canvas coordinates depends on the amount by which the tip of the minute hand is rotated (we’ve assumed that the ActualTimeMinute rotation is 180*◦*, so it has just undergone two 180*◦* rotations). Similarly, the WPF coordinates for the tip of the minute hand are computed by fur ther scaling each canvas coordinate by 4.8, and then adding 48 to each, resulting in WPF coordinates of (48, 4.8).

| The terms **object space, world space, image space,** and **screen space** are frequently used in graphics. They refer to the idea that a single point of some object (e.g., “Boston” on a texture-mapped globe) starts out as a point on a unit sphere (object space), gets transformed into the “world” that we’re going to render, eventually is projected onto an image plane, and finally is displayed on a screen. In some sense, all those points refer to the same thing. But each point has different coordinates. When we talk about a certain point “in world space” or “in image space,” we really mean that we’re working with the coordinates of the point in a coordinate system associated with that space. In image space, those coordinates may range from *−*1 to 1 (or from 0 to 1 in some systems), while in screen space, they may range from 0 to 1024, and in object space, the coordinates are a triple of real numbers that are typically in the range [*−*1, 1] for many standard objects like the sphere or cube. |
| --- |

For this example, we have seven coordinate systems, most indicated by pale green boxes. Starting at the top, there are WPF coordinates, the coordinates used by drawPolygon(). It’s possible that internally, drawPolygon() must convert to, say, pixel coordinates, but this conversion is hidden from us, and we won’t dis cuss it further. Beneath the WPF coordinates are canvas coordinates, and within the canvas are the clock-face coordinates, minute-hand coordinates, and hour-hand coordinates. Below this are the hand coordinates, the coordinate system in which the single prototype hand was created, and circle coordinates, in which the pro totype octagonal circle approximation was created. Notice that in our model of the clock, the clock-face, minute-hand, and hour-hand coordinates all play similar roles: In the hierarchy of coordinate systems, they’re all children of the canvas coordinate system. It might also have been reasonable to make the minute-hand and hour-hand coordinate systems children of the clock-face coordinate system. The advantage of doing so would have been that translating the clock face would have translated the whole clock, making it easier to adjust the clock’s position on the canvas. Right now, adjusting the clock’s position on the canvas requires that we adjust three different translations, which we’d have to add to the face, the minute hand, and the hour hand.

We’re hoping to draw each shape with a drawPolygon() call, which takes an array of point coordinates as an argument. For this to make sense, we have to declare the coordinate system in which the point coordinates are valid. We’ll assume that drawPolygon() expects WPF coordinates. So when we want to tell it about the tip of the minute hand, we’ll need the numbers (48, 4.8) rather than (0, 9).

246 Transformations in Two Dimensions

Here’s a strawman algorithm for converting a scene graph into a sequence of drawPolygon() calls. We’ll work with 3 *× k* arrays of coordinates, because we’ll represent the point (0, 9) as a homogeneous triple (0, 9, 1), which we’ll write vertically as a column of the matrix that represents the geometry.

| for each polygonal geometry element, *g*  let *v* be the 3 *× k* array of vertices of *g*  let *n* be the parent node of *g*  let **M** be the 3 *×* 3 identity matrix  while (*n* is not the root)  if *n* is a transformation with matrix **S**  **M** = **SM**  *n* = parent of *n*  *w* = **M***v*  drawPolygon(*w*) |
| --- |

1

2

3

4

5

6

7

8

9

10

11

As you can see, we multiply together several matrices, and then multiply the

result (the **composite transformation matrix**) by the vertex coordinates to get the WPF coordinates for each polygon, which we then draw.

| **Inline Exercise 10.21:** (a) How many elementary operations are needed, approximately, to multiply a 3 *×* 3 matrix by a 3 *× k* matrix?  (b) If **A** and **B** are 3*×*3 and **C** is 3*×*1000, would you rather compute (**AB**)**C** or **A**(**BC**), where the parentheses are meant to indicate the order of calculations that you perform?  (c) In the code above, should we have multiplied the vertex coordinates by each matrix in turn, or was it wiser to accumulate the matrix product and only multiply by the vertex array at the end? Why? |
| --- |

If we hand-simulate the code in the clock example, the circle template coordi nates are multiplied by the matrix

⎡

⎣1 0 48

0 1 48 00 1

⎤ ⎦

⎡

⎣4.8 0 0

0 4.8 0 0 01

⎤ ⎦

⎡

⎣10 0 0

0 10 0 0 01

⎤

⎦ . (10.86)

The minute-hand template coordinates are multiplied by the matrix

⎡

⎣1 0 48

0 1 48 00 1

⎤ ⎦

⎡

⎣4.8 0 0

0 4.8 0 0 01

⎤ ⎦

⎡

⎣*−*1 00

0 *−*1 0 0 01

⎤ ⎦

⎡

⎣*−*1 00

0 *−*1 0 0 01

⎤

⎦ . (10.87)

And the hour-hand template coordinates are multiplied by the matrix

⎡

⎣1 0 48

0 1 48 00 1

⎤ ⎦

⎡

⎣4.8 0 0

0 4.8 0 0 01

⎤

⎤

⎡

⎣0.96 *−*0.26 0 ⎦

⎦

0.26 0.96 0 0 01

*·*

⎡ ⎣

*−*1 00 0 *−*1 0 0 01

⎤ ⎦

⎡ ⎣

1.7 0 0 0 0.7 0 0 01

⎤

⎦ . (10.88)

10.11 Application: Rendering from a Scene Graph 247

| **Inline Exercise 10.22:** Explain where each of the matrices for the minute hand arose. |
| --- |

Notice how much of this matrix multiplication is *shared.* We could have com puted the product for the circle and reused it in each of the others, for instance. For a large scene graph, the overlap is often much greater. If there are 70 transfor mations applied to an object with only five or six vertices, the cost of multiplying matrices together far outweighs the cost of multiplying the composite matrix by the vertex coordinate array.

We can avoid duplicated work by revising our strawman algorithm. We per form a depth-first traversal of the scene graph, maintaining a stack of matrices as we do so. Each time we encounter a new transformation with matrix **M**, we mul tiply **M** by the current transformation matrix **C** (the one at the top of the stack) and push the result, **MC**, onto the stack. Each time our traversal rises up through a transformation node, we pop a matrix from the stack. The result is that whenever we encounter geometry (like the coordinates of the hand points, or of the ellipse points), we can multiply the coordinate array on the left by the current transfor mation to get the WPF coordinates of those points. In the pseudocode below, we assume that the scene graph is represented by a Scene class with a method that returns the root node of the graph, and that a transformation node has a matrix method that returns the matrix for the associated transformation, while a geometry node has a vertexCoordinateArray method that returns a 3 *× k* array containing the homogeneous coordinates of the *k* points in the polygon.

| void drawScene(Scene myScene)  s = *empty Stack*  s.push( 3 *×* 3 *identity matrix* )  explore(myScene.rootNode(), s)  void explore(Node n, Stack& s)  if n *is a transformation node*  *push* n.matrix() \* s.top() *onto* s  else if n *is a geometry node*  drawPolygon(s.top() \* n.vertexCoordinateArray())  *foreach child* k *of* n  explore(k, s)  if n *is a transformation node*  *pop top element from* s |
| --- |

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

In some complex models, the cost of matrix multiplications can be enormous.

If the same model is to be rendered over and over, and none of the transformations change (e.g., a model of a building in a driving-simulation game), it’s often worth it to use the algorithm above to create a list of polygons in world coordinates that can be redrawn for each frame, rather than reparsing the scene once per frame. This is sometimes referred to as **prebaking** or **baking** a model.

The algorithm above is the core of the standard one used for scene traversals in scene graphs. There are two important additions, however.

First, geometric transformations are not the only things stored in a scene graph—in some cases, attributes like color may be stored as well. In a simple

248 Transformations in Two Dimensions

version, each geometry node has a color, and the drawPolygon procedure is passed both the vertex coordinate array and the color. In a more complex version, the color attribute may be set at some node in the graph, and that color is used for all the geometry “beneath” that node. In this latter form, we can keep track of the color with a parallel stack onto which colors are pushed as they’re encountered, just as transformations are pushed onto the transformation stack. The difference is that while transformations are multiplied by the previous composite transforma tion before being pushed on the stack, the colors, representing an absolute rather than a relative attribute, are pushed without being combined in any way with pre vious color settings. It’s easy to imagine a scene graph in which color-alteration nodes are allowed (e.g., “Lighten everything below this node by 20%”); in such a structure, the stack would have to accumulate color transformations. Unless the transformations are quite limited, there’s no obvious way to combine them except to treat them as a sequence of transformations; matrix transformations are rather special in this regard.

Second, we’ve studied an example in which the scene graph is a *tree,* but depth-first traversal actually makes sense in an arbitrary directed acyclic graph (DAG). And in fact, our clock model, in reality, *is* a DAG: The geometry for the two clock hands is shared by the hands (using a WPF StaticResource). During the depth-first traversal we arrive at the hand geometry twice, and thus render two different hands. For a more complex model (e.g., a scene full of identical robots) such repeated encounters with the same geometry may be very frequent: Each robot has two identical arms that refer to the same underlying arm model; each arm has three identical fingers that refer to the same underlying finger model, etc. It’s clear that in such a situation, there’s some lost effort in retraversal of the arm model. Doing some analysis of a scene graph to detect such retraversals and avoid them by prebaking can be a useful optimization, although in many of today’s graphics applications, scene traversal is only a tiny fraction of the cost, and lighting and shading computations (for 3D models) dominate. You should avoid optimizing the scene-traversal portions of your code until you’ve verified that they are the expensive part.

**10.11.1 Coordinate Changes in Scene Graphs**

Returning to the scene graph and the matrix products, the transformations applied to the minute hand to get WPF coordinates,

⎡ ⎣

1 0 48 0 1 48 00 1

⎤ ⎦

⎡

⎣4.8 0 0

0 4.8 0 0 01

⎤ ⎦

⎡

⎣*−*1 00

0 *−*1 0 0 01

⎤ ⎦

⎡

⎣*−*1 00

0 *−*1 0 0 01

⎤

⎦ , (10.89)

represent the transformation from minute-hand coordinates to WPF coordinates. To go from WPF coordinates to minute-hand coordinates, we need only apply the inverse transformation. Remembering that (**AB**)*−*1 = **B***−*1**A***−*1, this inverse transformation is

⎡

⎣*−*1 00

0 *−*1 0 0 01

⎤ ⎦

⎡

⎣*−*1 00

0 *−*1 0 0 01

⎤

⎡

⎤

⎣1*/*4.8 0 0 ⎦

⎦

0 1*/*4.8 0

0 01

⎡

⎣1 0 *−*48

0 1 *−*48 00 1

⎤

⎦ . (10.90)

You can similarly find the coordinate transformation matrix to get from any one coordinate system in a scene graph to any other. Reading upward, you accumulate

10.11 Application: Rendering from a Scene Graph 249

the matrices you encounter, with the first matrix being farthest to the right; reading downward, you accumulate their inverses in the opposite order. When we build scene graphs in 3D, exactly the same rules apply.

For a 3D scene, there’s the description not only of the model, but also of how to transform points of the model into points on the display. This latter description is provided by specifying a camera. But even in 2D, there’s something closely analogous: The Canvas in which we created our clock model corresponds to the “world” of a 3D scene; the way that we transform this world to make it appear on the display (scale by (4.8, 4.8) and then translate by (48, 48)) corresponds to the viewing transformation performed by a 3D camera.

Typically the polygon coordinates (the ones we’ve placed in templates) are called modeling coordinates. Given the analogy to 3D, we can call the canvas coordinates world coordinates, while the WPF coordinates can be called image coordinates. These terms are all in common use when discussing 3D scene graphs.

As an exercise, let’s consider the tip of the hour hand; in modeling coordinates (i.e., in the clock-hand template) the tip is located at (0, 9). In the same way, the tip of the minute hand, in modeling coordinates, is at (0, 9). What are the Canvas coordinates of the tip of the hour hand? We must multiply (reading from leaf toward root) by all the transformation matrices from the hour-hand template up to the Canvas, resulting in

⎡

⎤

⎣0.96 *−*0.26 0 ⎦

0.26 0.96 0 0 01

⎡

⎣*−*1 00

0 *−*1 0 0 01

⎤ ⎦

⎡

⎣1.7 0 0

0 0.7 0 0 01

⎤ ⎦

⎡

⎣09 1

⎤

⎦ (10.91)

=

⎡ ⎣

⎤

*−*1.64 *−*.18 0 ⎦

*−*0.44 *−*0.68 0 001

⎡ ⎣

0 9 1

⎤

⎦ =

⎡ ⎣

1.63

*−*6.09 1

⎤

⎦ , (10.92)

where all coordinates have been rounded to two decimal places for clarity. The Canvas coordinates of the tip of the minute hand are

⎡

⎣*−*1 00

0 *−*1 0 0 01

⎤ ⎦

⎡

⎣*−*1 00

0 *−*1 0 0 01

⎤ ⎦

⎡

⎣09 1

⎤

⎦ =

⎡

⎣09 1

⎤

⎦ . (10.93)

We can thus compute a vector from the hour hand’s tip to the minute hand’s tip by subtracting these two, getting *−*1.63 15.08 0 **T**. The result is the homogeneous-coordinate representation of the vector *−*1.63 15.08 **T** in Canvas coordinates.

Suppose that we wanted to know the direction from the tip of the minute hand to the tip of the hour hand *in minute-hand coordinates.* If we knew this direction, we could add, within the minute-hand part of the model, a small arrow that pointed toward the hour-hand. To find this direction vector, we need to know the coordi nates of the tip of the hour hand in minute-hand coordinates. So we must go from hour-hand coordinates to minute-hand coordinates, which we can do by working up the tree from the hour hand to the Canvas, and then back down to the minute hand. The location of the hour-hand tip, in minute-hand coordinates, is given by