This excellent volume is unique in that it covers not only the basic techniques of computer
graphics and game development, but also provides a thorough and rigorous—yet very readable—
treatment of the underlying mathematics. Fledgling graphics and games developers will find it
a valuable introduction; experienced developers will find it an invaluable reference. Everything
is here, from the detailed numeric issues of IEEE floating point notation, to the correct way to
use quaternions and spherical linear interpolation to represent orientation, to the mathematics
of collision detection and rigid-body dynamics.

—David Luebke, University of Virginia,
co-author of *Level of Detail for 3D Graphics*

When it comes to software development for games or virtual reality, you cannot escape the math-
ematics. The best performance comes not from superfast processors and terabytes of memory,
but from well-chosen algorithms. With this in mind, the techniques most useful for developing
production-quality computer graphics for Hollywood blockbusters are not the best choice for
interactive applications. When rendering times are measured in milliseconds rather than hours,
you need an entirely different perspective.

*Essential Mathematics for Games and Interactive Applications* provides this perspective. While the mathematics are rigorous and perhaps challenging at times, Van Verth and Bishop
provide the context for understanding the algorithms and data structures needed to bring games
and VR applications to life. This may not be the only book you will ever need for games and VR
software development, but it will certainly provide an excellent framework for developing robust
and fast applications.

—Ian Ashdown, President, ByHeart Consultants Limited

With *Essential Mathematics for Games and Interactive Applications*, Van Verth and Bishop have
provided invaluable assistance for professional game developers looking to shore up weaknesses
in their mathematical training. Even if you never intend to write a renderer or tune a physics
engine, this book provides the mathematical and conceptual grounding needed to understand
many of the key concepts in rendering, simulation, and animation.

—Dave Weinstein, Microsoft, Red Storm Entertainment

Geometry, trigonometry, linear algebra, and calculus are all essential tools for 3D graphics. Math-
ematics courses in these subjects cover too much ground, while at the same time glossing over the
bread-and-butter essentials for 3D graphics programmers. In *Essential Mathematics for Games
and Interactive Applications*, Van Verth and Bishop bring just the right level of mathematics out
of the trenches of professional game development. This book provides an accessible and solid
mathematical foundation for interactive graphics programmers. If you are working in the area
of 3D games, this book is a “must have.”

—Jonathan Cohen, Department of Computer Science,
Johns Hopkins University,
co-author of *Level of Detail for 3D Graphics*

It’s the book with all the math you need for games.

—Neil Kirby, Bell Labs

As games become ever more sophisticated, mathematics and technical programming skills
become increasingly important to have in your toolbox. *Essential Math* provides a solid foun-
dation in many critical areas. You will find many topics covered in detail: from linear algebra
to calculus, from physics to rasterization. Some of this will be review material, but you will
undoubtedly learn something new and, most importantly, something useful.

—Erin Catto, Blizzard Entertainment
Dedications

To Mur and Fiona, for allowing me to slay the monster one more time. —Jim

To Jen, who constantly helps me get everything done; and to Nadia and Tasha, who constantly help me avoid getting any of it done on time. —Lars
About the Authors

James M. Van Verth is an OpenGL Software Engineer at NVIDIA, where he works on device drivers for NVIDIA GPUs. Prior to that, he was a founding member of Red Storm Entertainment, where he was a lead engineer for eight years. For the past nine years he also has been a regular speaker at the Game Developers Conference, teaching the all-day tutorials “Math for Game Programmers” and “Physics for Game Programmers,” on which this book is based. His background includes a B.A. in Math/Computer Science from Dartmouth College, an M.S. in Computer Science from the State University of New York at Buffalo, and an M.S. in Computer Science from the University of North Carolina at Chapel Hill.

Lars M. Bishop is an engineer in the Handheld Developer Technologies group at NVIDIA. Prior to joining NVIDIA, Lars was the Chief Technology Officer at Numerical Design Limited, leading the development of the Gamebryo3D cross-platform game engine. He received a BS in Math/Computer Science from Brown University and an MS in Computer Science from the University of North Carolina at Chapel Hill. His outside interests include photography, drumming, and playing bass guitar.
Contents

Preface xix

Introduction xxiii

Chapter 1

Real-World Computer Number Representation 1

1.1 Introduction 1

1.2 Representing Real Numbers 2
  1.2.1 Approximations 2
  1.2.2 Precision and Error 3

1.3 Floating-Point Numbers 4
  1.3.1 Review: Scientific Notation 4
  1.3.2 A Restricted Scientific Notation 5

1.4 Binary “Scientific Notation” 6

1.5 IEEE 754 Floating-Point Standard 9
  1.5.1 Basic Representation 9
  1.5.2 Range and Precision 11
  1.5.3 Arithmetic Operations 13
  1.5.4 Special Values 16
  1.5.5 Very Small Values 19
  1.5.6 Catastrophic Cancelation 22
  1.5.7 Double Precision 24

1.6 Real-World Floating-Point 25
  1.6.1 Internal FPU Precision 25
  1.6.2 Performance 26
  1.6.3 IEEE Specification Compliance 29
  1.6.4 Graphics Processing Units and Half-Precision Floating-Point Formats 31

1.7 Code 32

1.8 Chapter Summary 33

ix
## Chapter 2
### Vectors and Points

#### 2.1 Introduction 35

#### 2.2 Vectors 36
  - 2.2.1 Geometric Vectors 36
  - 2.2.2 Linear Combinations 39
  - 2.2.3 Vector Representation 40
  - 2.2.4 Basic Vector Class Implementation 42
  - 2.2.5 Vector Length 44
  - 2.2.6 Dot Product 47
  - 2.2.7 Gram-Schmidt Orthogonalization 51
  - 2.2.8 Cross Product 53
  - 2.2.9 Triple Products 56
  - 2.2.10 Real Vector Spaces 59
  - 2.2.11 Basis Vectors 62

#### 2.3 Points 63
  - 2.3.1 Points as Geometry 64
  - 2.3.2 Affine Spaces 66
  - 2.3.3 Affine Combinations 68
  - 2.3.4 Point Implementation 70
  - 2.3.5 Polar and Spherical Coordinates 72

#### 2.4 Lines 75
  - 2.4.1 Definition 75
  - 2.4.2 Parameterized Lines 76
  - 2.4.3 Generalized Line Equation 77
  - 2.4.4 Collinear Points 79

#### 2.5 Planes 80
  - 2.5.1 Parameterized Planes 80
  - 2.5.2 Generalized Plane Equation 80
  - 2.5.3 Coplanar Points 82

#### 2.6 Polygons and Triangles 82

#### 2.7 Chapter Summary 86

## Chapter 3
### Matrices and Linear Transformations

#### 3.1 Introduction 87

#### 3.2 Matrices 88
  - 3.2.1 Introduction to Matrices 88
  - 3.2.2 Simple Operations 90
  - 3.2.3 Vector Representation 92
  - 3.2.4 Block Matrices 92
  - 3.2.5 Matrix Product 94
  - 3.2.6 Identity Matrix 96
Contents

3.2.7 Performing Vector Operations with Matrices 97
3.2.8 Implementation 98

3.3 Linear Transformations 101
3.3.1 Definitions 101
3.3.2 Null Space and Range 103
3.3.3 Linear Transformations and Basis Vectors 104
3.3.4 Matrices and Linear Transformations 106
3.3.5 Combining Linear Transformations 108

3.4 Systems of Linear Equations 110
3.4.1 Definition 110
3.4.2 Solving Linear Systems 112
3.4.3 Gaussian Elimination 113

3.5 Matrix Inverse 117
3.5.1 Definition 117
3.5.2 Simple Inverses 120

3.6 Determinant 121
3.6.1 Definition 121
3.6.2 Computing the Determinant 123
3.6.3 Determinants and Elementary Row Operations 126
3.6.4 Adjoint Matrix and Inverse 128

3.7 Eigenvalues and Eigenvectors 129

3.8 Chapter Summary 130

Chapter 4

Affine Transformations 133

4.1 Introduction 133

4.2 Affine Transformations 134
4.2.1 Matrix Definition 134
4.2.2 Formal Definition 136
4.2.3 Formal Representation 138

4.3 Standard Affine Transformations 139
4.3.1 Translation 139
4.3.2 Rotation 141
4.3.3 Scaling 150
4.3.4 Reflection 151
4.3.5 Shear 154
4.3.6 Applying an Affine Transformation Around an
  Arbitrary Point 156
4.3.7 Transforming Plane Normals 158

4.4 Using Affine Transformations 159
4.4.1 Manipulation of Game Objects 159
4.4.2 Matrix Decomposition 164
4.4.3 Avoiding Matrix Decomposition 166
4.5 Object Hierarchies 169
4.6 Chapter Summary 171

Chapter 5
Orientation Representation 173
5.1 Introduction 173
5.2 Rotation Matrices 174
5.3 Fixed and Euler Angles 174
  5.3.1 Definition 174
  5.3.2 Format Conversion 177
  5.3.3 Concatenation 178
  5.3.4 Vector Rotation 178
  5.3.5 Other Issues 179
5.4 Axis–Angle Representation 181
  5.4.1 Definition 181
  5.4.2 Format Conversion 182
  5.4.3 Concatenation 184
  5.4.4 Vector Rotation 184
  5.4.5 Axis–Angle Summary 185
5.5 Quaternions 185
  5.5.1 Definition 185
  5.5.2 Quaternions as Rotations 186
  5.5.3 Addition and Scalar Multiplication 187
  5.5.4 Negation 188
  5.5.5 Magnitude and Normalization 188
  5.5.6 Dot Product 189
  5.5.7 Format Conversion 189
  5.5.8 Concatenation 193
  5.5.9 Identity and Inverse 195
  5.5.10 Vector Rotation 197
  5.5.11 Shortest Path of Rotation 199
  5.5.12 Quaternions and Transformations 200
5.6 Chapter Summary 201

Chapter 6
Viewing and Projection 203
6.1 Introduction 203
6.2 View Frame and View Transformation 205
  6.2.1 Defining a Virtual Camera 205
  6.2.2 Constructing the View-to-World Transformation 206
  6.2.3 Controlling the Camera 208
  6.2.4 Constructing the World-to-View Transformation 211
6.3 Projective Transformation 212
   6.3.1 Definition 212
   6.3.2 Normalized Device Coordinates 216
   6.3.3 View Frustum 216
   6.3.4 Homogeneous Coordinates 220
   6.3.5 Perspective Projection 221
   6.3.6 Oblique Perspective 228
   6.3.7 Orthographic Parallel Projection 231
   6.3.8 Oblique Parallel Projection 232
6.4 Culling and Clipping 235
   6.4.1 Why Cull or Clip? 235
   6.4.2 Culling 238
   6.4.3 General Plane Clipping 239
   6.4.4 Homogeneous Clipping 244
6.5 Screen Transformation 246
   6.5.1 Pixel Aspect Ratio 248
6.6 Picking 249
6.7 Management of Viewing Transformations 252
6.8 Chapter Summary 254

Chapter 7
Geometry and Programmable Shading 255
7.1 Introduction 255
7.2 Color Representation 257
   7.2.1 RGB Color Model 257
   7.2.2 Colors as “Vectors” 257
   7.2.3 Color Range Limitation 258
   7.2.4 Operations on Colors 259
   7.2.5 Alpha Values 260
   7.2.6 Color Storage Formats 264
7.3 Points and Vertices 266
   7.3.1 Per-Vertex Attributes 266
   7.3.2 An Object’s Vertices 267
7.4 Surface Representation 270
   7.4.1 Vertices and Surface Ambiguity 270
   7.4.2 Triangles 271
   7.4.3 Connecting Vertices into Triangles 271
   7.4.4 Drawing Geometry 274
7.5 Rendering Pipeline 275
   7.5.1 Fixed-Function versus Programmable Pipelines 277
7.6 Shaders 278
   7.6.1 Using Shaders to Move from Vertex to Triangle to Fragment 278
Contents

7.6.2 Shader Input and Output Values 279
7.6.3 Shader Operations and Language Constructs 280
7.7 Vertex Shaders 280
7.7.1 Vertex Shader Inputs 280
7.7.2 Vertex Shader Outputs 281
7.7.3 Basic Vertex Shaders 282
7.7.4 Linking Vertex and Fragment Shaders 282
7.8 Fragment Shaders 283
7.8.1 Fragment Shader Inputs 283
7.8.2 Fragment Shader Outputs 284
7.8.3 Compiling, Linking, and Using Shaders 284
7.8.4 Setting Uniform Values 286
7.9 Basic Coloring Methods 287
7.9.1 Per-Object Colors 288
7.9.2 Per-Vertex Colors 288
7.9.3 Per-Triangle Colors 290
7.9.4 Sharp Edges and Vertex Colors 290
7.9.5 More about Basic Shading 291
7.9.6 Limitations of Basic Shading Methods 292
7.10 Texture Mapping 292
7.10.1 Introduction 292
7.10.2 Shading via Image Lookup 293
7.10.3 Texture Images 294
7.10.4 Texture Samplers 297
7.11 Texture Coordinates 297
7.11.1 Mapping Texture Coordinates onto Objects 298
7.11.2 Generating Texture Coordinates 300
7.11.3 Texture Coordinate Discontinuities 301
7.11.4 Mapping Outside the Unit Square 302
7.11.5 Texture Samplers in Shader Code 309
7.12 The Steps of Texturing 309
7.12.1 Other Forms of Texture Coordinates 310
7.12.2 From Texture Coordinates to a Texture Sample Color 311
7.13 Limitations of Static Shading 312
7.14 Chapter Summary 313

Chapter 8
Lighting 315
8.1 Introduction 315
8.2 Basics of Light Approximation 316
8.2.1 Measuring Light 317
8.2.2 Light as a Ray 318
8.3 A Simple Approximation of Lighting 318
Contents

8.4  Types of Light Sources  319
  8.4.1  Directional Lights  320
  8.4.2  Point Lights  321
  8.4.3  Spotlights  327
  8.4.4  Other Types of Light Sources  330
8.5  Surface Materials and Light Interaction  331
8.6  Categories of Light  332
  8.6.1  Emission  332
  8.6.2  Ambient  332
  8.6.3  Diffuse  334
  8.6.4  Specular  338
8.7  Combined Lighting Equation  343
8.8  Lighting and Shading  348
  8.8.1  Flat-Shaded Lighting  349
  8.8.2  Per-Vertex Lighting  350
  8.8.3  Per-Fragment Lighting  354
8.9  Textures and Lighting  358
  8.9.1  Basic Modulation  359
  8.9.2  Specular Lighting and Textures  360
  8.9.3  Textures as Materials  362
8.10  Advanced Lighting  363
    8.10.1  Normal Mapping  363
8.11  Reflective Objects  366
8.12  Shadows  367
8.13  Chapter Summary  368

Chapter 9

Rasterization  369

9.1  Introduction  369
9.2  Displays and Framebuffers  370
9.3  Conceptual Rasterization Pipeline  371
    9.3.1  Rasterization Stages  372
9.4  Determining the Fragments: Pixels Covered by a Triangle  373
    9.4.1  Fragments  373
    9.4.2  Depth Complexity  373
    9.4.3  Converting Triangles to Fragments  375
    9.4.4  Handling Partial Fragments  376
9.5  Determining Visible Geometry  378
    9.5.1  Depth Buffering  378
    9.5.2  Depth Buffering in Practice  387
9.6  Computing Fragment Shader Inputs  388
    9.6.1  Uniform Values  389
    9.6.2  Per-Vertex Attributes  389
11.2 Probability 493
   11.2.1 Basic Probability 494
   11.2.2 Random Variables 497
   11.2.3 Mean and Standard Deviation 501
   11.2.4 Special Probability Distributions 502
11.3 Determining Randomness 505
   11.3.1 Chi-Square Test 506
   11.3.2 Spectral Test 512
11.4 Random Number Generators 513
   11.4.1 Linear Congruential Methods 516
   11.4.2 Lagged Fibonacci Methods 520
   11.4.3 Carry Methods 521
   11.4.4 Mersenne Twister 523
   11.4.5 Conclusions 526
11.5 Special Applications 527
   11.5.1 Integers and Ranges of Integers 527
   11.5.2 Floating-Point Numbers 528
   11.5.3 Nonuniform Distributions 528
   11.5.4 Spherical Sampling 530
   11.5.5 Disc Sampling 532
   11.5.6 Noise and Turbulence 534
11.6 Chapter Summary 538

Chapter 12
Intersection Testing 541
12.1 Introduction 541
12.2 Closest Point and Distance Tests 542
   12.2.1 Closest Point on Line to Point 542
   12.2.2 Line–Point Distance 544
   12.2.3 Closest Point on Line Segment to Point 545
   12.2.4 Line Segment–Point Distance 546
   12.2.5 Closest Points Between Two Lines 548
   12.2.6 Line–Line Distance 550
   12.2.7 Closest Points Between Two Line Segments 551
   12.2.8 Line Segment–Line Segment Distance 553
   12.2.9 General Linear Components 554
12.3 Object Intersection 554
   12.3.1 Spheres 556
   12.3.2 Axis-Aligned Bounding Boxes 563
   12.3.3 Swept Spheres 571
   12.3.4 Object-Oriented Boxes 576
   12.3.5 Triangles 583
Contents

12.4 A Simple Collision System 588
   12.4.1 Choosing a Base Primitive 589
   12.4.2 Bounding Hierarchies 590
   12.4.3 Dynamic Objects 591
   12.4.4 Performance Improvements 593
   12.4.5 Related Systems 596
   12.4.6 Section Summary 599

12.5 Chapter Summary 599

Chapter 13
Rigid Body Dynamics 601

13.1 Introduction 601

13.2 Linear Dynamics 602
   13.2.1 Moving with Constant Acceleration 602
   13.2.2 Forces 605
   13.2.3 Linear Momentum 606
   13.2.4 Moving with Variable Acceleration 607

13.3 Numerical Integration 609
   13.3.1 Definition 609
   13.3.2 Euler's Method 611
   13.3.3 Runge-Kutta Methods 614
   13.3.4 Verlet Integration 616
   13.3.5 Implicit Methods 619
   13.3.6 Semi-Implicit Methods 621

13.4 Rotational Dynamics 622
   13.4.1 Definition 622
   13.4.2 Orientation and Angular Velocity 622
   13.4.3 Torque 625
   13.4.4 Angular Momentum and Inertia Tensor 626
   13.4.5 Integrating Rotational Quantities 628

13.5 Collision Response 630
   13.5.1 Contact Generation 630
   13.5.2 Linear Collision Response 634
   13.5.3 Rotational Collision Response 638
   13.5.4 Extending the System 640

13.6 Efficiency 643

13.7 Chapter Summary 645

Bibliography 647

Index 655

Trademarks 671

About the CD-ROM 672
Writing a book is an adventure. To begin with, it is a toy and an amusement; then it becomes a mistress, and then it becomes a master, and then a tyrant. The last phase is that just as you are about to be reconciled to your servitude, you kill the monster, and fling him out to the public. — Sir Winston Churchill

The Adventure Begins

As humorous as Churchill’s statement is, there is a certain amount of truth to it; writing this book was indeed an adventure. There is something about the process of writing, particularly a nonfiction work like this, that forces you to test and expand the limits of your knowledge. We hope that you, the reader, benefit from our hard work.

How does a book like this come about? Many of Churchill’s books began with his experience — particularly his experience as a world leader in wartime. This book had a more mundane beginning: Two engineers at Red Storm Entertainment, separately, asked Jim to teach them about vectors. These engineers were 2D game programmers, and 3D was not new, but was starting to replace 2D at that point. Jim’s project was in a crunch period, so he didn’t have time to do much about it until proposals were requested for the annual Game Developers Conference. Remembering the engineers’ request, he thought back to the classic “Math for SIGGRAPH” course from SIGGRAPH 1989, which he had attended and enjoyed. Jim figured that a similar course, at that time titled “Math for Game Programmers,” could help 2D programmers become 3D programmers.

The course was accepted, and together with a co-speaker, Marcus Nordenstam, Jim presented it at GDC 2000. The following years (2001–2002) Jim taught the course alone, as Marcus had moved from the game industry to the film industry. The subject matter changed slightly as well, adding more advanced material such as curves, collision detection, and basic physical simulation.

It was in 2002 that the seeds of what you hold in your hand were truly planted. At GDC 2002, another GDC speaker, whose name, alas, is lost to time, recommended that Jim turn his course into a book. This was an interesting idea, but how to get it published? As it happened, Jim ran into Dave Eberly at SIGGRAPH 2002, and he was looking for someone to write just that book for Morgan Kaufmann. At the same time, Lars, who was working at Numeric Design Limited at the time, was presenting some
of the basics of rendering on handheld devices as part of a SIGGRAPH course. Jim and Lars discussed the fact that handheld 3D rendering had brought back some of the “lost arts” of 3D programming, and that this might be included in a book on mathematics for game programming.

Thus, a co-authorship was formed. Lars joined Jim in teaching the GDC 2003 version of what was now called “Essential Math for Game Programmers,” and simultaneously joined Jim to help with the book, helping to expand the topics covered to include numerical representations. As we began to flesh out the latter chapters of the outline, Lars was finding that the advent of programmable shaders on consumer 3D hardware was bringing more and more low-level lighting, shading, and texturing questions into his office at NDL. Accordingly, the planned single chapter on “texturing and antialiasing” became three, covering a wider selection of these rendering topics.

By early 2003, we were furiously typing the first full draft of the first edition of this book, and by GDC 2004 the book was out. Having defeated the dragon, we retired to our homes to live out the rest of our quiet little lives.

Or so we thought.

The Adventure Continues

Response to the first edition was quite positive, and the book continued to sell well beyond the initial release. Naturally, thoughts turned to what we could do to improve the book beyond what we already created.

In reviewing the topic list, it was obvious what the most necessary change was. Within a year or so of the publication of the first edition, programmable shading had revolutionized the creation of 3D applications on game consoles and on PC. While the first edition had provided readers with many of the fundamentals behind the mathematics used in shaders, it stopped short of actually discussing them in detail. It was clear that the second edition needed to embrace shaders completely, applying the mathematics of the earlier chapters to an entirely new set of rendering content. So the single biggest change in the second edition is a move to a purely shader-based rendering pipeline.

We also sent the book to reviewers to ask them what they would like to see added. The two most common requests were information about random numbers and the addition of problems and exercises. So we are providing both. A brand new chapter on probability and random numbers has been added, and problems and exercises for each chapter have been added to the CD in the back of the book. In addition, the entire book has been revised to add corrections and make the content flow better. We hope you’ll find our efforts worthwhile.

Both times, the experience was fascinating, sometimes frustrating, but ultimately deeply rewarding. Hopefully, this fascination and respect for the material will be conveyed to you, the reader. The topics in this book can each take a lifetime to study to a truly great depth; we hope you will be convinced to try just that, nonetheless!

Enjoy as you do so, as one of the few things more rewarding than programming and seeing a correctly animated, simulated, and rendered scene on a screen is the confidence of understanding how and why everything worked. When something in a 3D
system goes wrong (and it always does), the best programmers are never satisfied with “I fixed it, but I’m not sure how,” without understanding, there can be no confidence in the solution, and nothing new is learned. Such programmers are driven by the desire to understand what went wrong, how to fix it, and learning from the experience. No other tool in 3D programming is quite as important to this process than the mathematical bases\(^1\) behind it.

Those Who Helped Us Along the Road

In a traditional adventure the protagonists are assisted by various characters that pass in and out of the pages. Similarly, while this book bears the names of two people on the cover, the material between its covers bears the mark of many, many more. We would like to thank a few of them here.

The folks at our publisher, Elsevier, were extremely patient with both of us as we made up for being more experienced this time around by being more busy and less responsive! Chris Simpson, Laura Lewin, Georgia Kennedy, and Paul Gottehrer were all patient, professional, and flexible when we most needed it.

In addition, credit is still due to the folks at Morgan Kaufmann who helped us publish the first edition. Tim Cox, our editor, and Stacie Pierce and Richard Camp, his assistants, as well as Troy Lilly (in production) were patient and helpful in the daunting task of leading two first-time authors through the process. Special thanks are due to Dave Eberly, the series editor of our first edition, who read most of the book several times and provided great encouragement (and the occasional scolding) through the entire process, one he’s been through firsthand several times.

Our reviewers were top-notch. Together, Erin Catto and Chad Robertson reviewed the entire second edition of the book. Robert Brown, Matthew McCallus, Greg Stelmack, and Melinda Theilbar were invaluable for their comments on the random numbers chapter. Ian Ashdown, Steven Woodcock, John O’Brien, J.R. Parker, Neil Kirby, John Funge, Michael van Lent, Peter Norvig, Tomas Akenine-Möller, Wes Hunt, Peter Lipson, Jon McAllister, Travis Young, Clark Gibson, Joe Sauder, and Chris Stoy each reviewed parts of the first edition or the proposals for them. Despite having tight deadlines, they all provided page after page of useful feedback, keeping us honest and helping us generate a better arc to the material. Several of them went well above and beyond the call of duty, providing detailed comments and even re-reading sections of the book that required significant changes. Finally, thanks also to Victor Brueggemann and Garner Halloran, who asked Jim the questions that started this whole thing off five years ago.

Jim and Lars would like to acknowledge the folks at their jobs at NVIDIA Corporation, who were very understanding with respect to the time-consuming process of creating a book. Also, thanks to the talented engineers at this and previous companies who provided the probing discussions and great questions that led to and continually fed this book.

\(^1\) Vector or otherwise.
In addition, Jim would like to thank Mur and Fiona, his wife and daughter, who were willing to put up with this a second time after his long absences the first time through; his sister, Liz, who provided illustrations for an early draft of this text; and his parents, Jim and Pat, who gave him the resources to make it in the world and introduced him to the world of computers so long ago.

Lars would like to thank Jen, his wife, who somehow had the courage to survive a second edition of the book even after being promised that the first edition “was it;” and his parents, Steve and Helene, who supported, nurtured, and taught him so much about the value of constant learning and steadfast love.

And lastly, we would once again like to thank you, the reader, for joining us on this adventure. May the teeth of this monster find fertile ground in your minds, and yield a new army of 3D programmers.
Introduction

The (Continued) Rise of 3D Games

Over the past decade or so (driven by increasingly powerful computer and video game console hardware), three-dimensional (3D) games have expanded from custom-hardware arcade machines to the realm of hardcore PC games, to consumer set-top video game consoles, and even to handheld devices such as personal digital assistants (PDAs) and cellular telephones. This explosion in popularity has lead to a corresponding need for programmers with the ability to program these games. As a result, programmers are entering the field of 3D games and graphics by teaching themselves the basics, rather than a classic college-level graphics and mathematics education. At the same time, many college students are looking to move directly from school into the industry. These different groups of programmers each have their own set of skills and needs in order to make the transition. While every programmer’s situation is different, we describe here some of the more common situations.

Many existing, self-taught 3D game programmers have strong game experience and an excellent practical approach to programming, stressing visual results and strong optimization skills that can be lacking in college-level computer science programs. However, these programmers are sometimes less comfortable with the conceptual mathematics that form the underlying basis of 3D graphics and games. This can make developing, debugging, and optimizing these systems more of a trial-and-error exercise than would be desired.

Programmers who are already established in other specializations in the game industry, such as networking or user interfaces, are now finding that they want to expand their abilities into core 3D programming. While having experience with a wide range of game concepts, these programmers often need to learn or refresh the basic mathematics behind 3D games before continuing on to learn the applications of the principles of rendering and animation.

On the other hand, college students entering (or hoping to enter) the 3D games industry often ask what material they need to know in order to be prepared to work on these games. Younger students often ask what courses they should attend in order to gain the most useful background for a programmer in the industry. Recent graduates, on the other hand, often ask how their computer graphics knowledge best relates to the way games are developed for today’s computers and game consoles.

We have designed this book to provide something for each of these groups of readers. We attempt to provide readers with a conceptual understanding of the
mathematics needed to create 3D games, as well as an understanding of how these mathematical bases actually apply to games and graphics. The book provides not only theoretical mathematical background, but also many examples of how these concepts are used to affect how a game looks (how it is rendered) and plays (how objects move and react to users). Each type of reader is likely to find sections of the book that, for them, provide mainly refresher courses, a new understanding of the applications of basic mathematical concepts, or even completely new information. The specific sections that fall into each category for a particular reader will, of course, depend on the reader.

**How to Read This Book**

Perhaps the best way to discuss any reader's approach to reading this book is to think in terms of how a 3D game or other interactive application works at the highest level. Most readers of this book likely intend to apply what they learn from it to create, extend, or fix a 3D game or other 3D application. Each chapter in this book deals with a different topic that has applicability to some or all of the major parts of a 3D game.

**Game Engines**

An interactive 3D application such as a game requires quite a large amount of code to do all of the many things asked of it. This includes representing the virtual world, animating parts of it, drawing that virtual world, and dealing with user interaction in a game-relevant manner. The bulk of the code required to implement these features is generally known as a game engine. Game engines have evolved from small, simple, low-level rendering systems in the early 1990s to massive and complex software systems in modern games, capable of rendering detailed and expansive worlds, animating realistic characters, and simulating complex physics. At their core, these game engines are really implementations of the concepts discussed throughout this book.

Initially, game engines were custom affairs, written for a single use as a part of the game itself, and thrown away after being used for that single game project. Today, game developers have several options when considering an engine. They may purchase a commercial engine from another company and use it unmodified for their project. They may purchase an engine and modify it very heavily to customize their application. Finally, they may write their own, although most programmers choose to use such an internally developed engine for multiple games to offset the large cost of creating the engine.

In any of these cases, the developer must still understand the basic concepts of the game engine. Whether as a user, a modifier, or an author of a game engine, the developer must understand at least a majority of the concepts presented in this book. To better understand how the various chapters in this book surface in game engines, we first present a common main loop as it might appear in a game engine:

1. Draw the current configuration of the game's scene to the screen.
2. Animate the characters in the scene based on animator-created sequences (e.g., soccer players running downfield).
3. Detect collisions between the characters and objects (e.g., the soccer ball entering the goal or two players sliding into one another).

4. React to these collisions and basic forces such as gravity in the scene in a physically correct manner (e.g., the soccer ball in flight).

All of these steps will need to be done for each frame to present the player with a convincing game experience. Thus, the code to implement the steps above must be correct and optimal.

**Chapters 1–5: The Basics**

Perhaps the most core parts of any game engine are the low-level mathematical and geometric representations and algorithms. The pieces of code will be used by each and every step listed above. Chapter 1 provides the lowest-level basis for this. It discusses the practicalities of representing real numbers on a computer, with a focus on the issues most likely to affect the development of a 3D game engine for a PC, console, or handheld device.

Chapter 2 provides a focused review of vectors and points, objects that are used in all game engines to represent locations, directions, velocities, and other geometric quantities in all aspects of a 3D application. Chapters 3 and 4 review the basics of linear and affine algebra as they relate to orienting, moving, and distorting the objects and spaces that make up a virtual world. Finally, Chapter 5 introduces the quaternion, a very powerful nonmatrix representation of object orientation that will be pivotal to the later chapters on animation and simulation.

Three-dimensional engine code that implements all of these fundamental objects must be built carefully and with a good understanding of both the underlying mathematics and programming issues. Otherwise, the game engine built on top of these basic objects or functions will be based upon a poor foundation. Many game programmers’ multiday debugging sessions have ended with the realization that the complex bug was rooted in an error in the engine’s basic mathematics code.

Some readers will have a passing familiarity with the topics in these chapters. However, most readers will want to start with these chapters, as many of the topics are covered in more conceptual detail than is often discussed in basic graphics texts. Readers new to the material will want to read in detail, while those who already know some linear algebra can use the chapters to fill in any missing background. All of these chapters form a basis for the rest of the book, and an understanding of these topics, whether existing or new, will be key to successful 3D programming.

**Chapters 6–9: Rendering**

Chapters 6–9 apply the foundational objects detailed in Chapters 1–5 to explain step 1 of the game engine main loop: the rendering or drawing pipeline, perhaps the best-known part of any game engine. In some game engines, more time and effort is spent designing, programming, and tuning the rendering pipeline than the rest of the engine in its entirety. Chapter 6 describes the mathematics and geometry behind the virtual cameras used to view the scene or game world. Chapter 7 describes the representation
Introduction

of color and the concept of shaders, which are short programs that allow modern graphics hardware to draw the scene objects to the display device. Chapter 8 explains how to use these programmable shaders to implement simple approximations of real-world lighting. The rendering section concludes with Chapter 9, which details the methods used by low-level rendering systems to draw to the screen. An understanding of these details can allow programmers to create much more efficient and artifact-free rendering code for their engines.

Chapters 10–13: Animation and Physics

The game engine loop’s step 2, animating characters and other objects based on data created by computer animators or motion-captured data, is introduced in Chapter 10. This chapter discusses methods for smoothly animating the position, orientation, and appearance of objects in the virtual game world. The importance of good, complex character and object animation in modern engines continues to grow as new games attempt to create smoother, more convincing representations of athletes, rock stars, soldiers, and other human characters.

Chapter 11 covers another element for adding realism to games: random numbers. Everything up to this point has been carefully determined and planned by the programmer or artist. Adding randomness adds the unexpected behavior that we see in real life. Gunshots are not always exact, clouds are not perfectly spherical, and walls are not pristine. This chapter discusses how to handle randomness in a game, and how we can get effects such as those discussed above.

Step 3, detecting collisions, is discussed in Chapter 12. This chapter describes the mathematics and programming behind detecting when two game objects touch, intersect, or penetrate. Many genres of game have exacting requirements when it comes to collision, be it a racing game, a sports title, or a military simulation.

Finally, step 4, reacting in a realistic manner to physical forces and collisions, is covered in Chapter 13. This chapter describes how to make game objects behave and react in physically convincing ways.

Put together, the chapters that form this book are designed to give a good basis for the foundations of a game engine, details on the ways that engines represent and draw their virtual worlds, and an introduction to making those worlds seem real and active.

Interactive Demo Applications

Three-dimensional games and graphics are, by their nature, not only visual but dynamic. While figures are indeed a welcome necessity in a book about 3D applications, interactive demos can be even more important. It is difficult to truly understand such topics as lighting, quaternion interpolation, or physical simulation without being able to see them work firsthand and to interact with these complex systems. This book includes a CD-ROM of source code and demonstrations that are designed
to illustrate the concepts in a way that is analogous to the static figures in the
book itself. Throughout the book, you will find references to interactive demos that
may be found on the CD-ROM. Whenever a topic is illustrated with an interactive
demo, a special icon like the one seen next to this paragraph will appear in the
margin.

**Support Libraries**

In addition to the source code for each of the demos, the CD-ROM includes the sup-
porting libraries used to create the demos, with full source code. Often, code from
these supporting libraries is excerpted in the book itself in order to explain how the
particular concept is implemented. In such situations, an icon will appear in the mar-
gin to note where the library code may be found on the CD-ROM. This source code
is designed to allow readers to modify and experiment themselves, as a way of better
understanding the way the code works.

The source code is written entirely in C++, a language that is likely to be familiar
to most game developers. C++ was chosen because it is one of the most commonly
used languages in 3D game development and because vectors, matrices, quaternions,
and graphics algorithms decompose very well into C++ classes. In addition, C++’s
support of operator overloading means that the math library can be implemented
in a way that makes the code look very similar to the mathematical derivations in
the text. However, in some sections of the text, the class declarations as printed
in the book are not complete with respect to the code on the CD-ROM. Often,
class members that are not relevant to the particular discussion (especially mem-
ber variable accessor and “housekeeping” functions) have been omitted for clarity.
These other functions may be found in the full class declarations/definitions on the
CD-ROM.

Note that we have modified our mathematical notation slightly to allow our equa-
tions to be as compatible as possible with the code. Mathematicians normally start
indexing with 1, for example, $P_1, P_2, \ldots, P_n$. This does not match how indexing is
done in C++: $P[0]$ is the first element in the array $P$. To avoid this disconnect, in our
equations we will be using the convention that the starting element in a list is indexed
as 0; thus, $P_0, P_1, \ldots, P_{n-1}$. This should allow for a direct translation from equation to
code.

**Math Libraries**

All of the demos use a shared core math library called IvMath, which includes C++
classes that implement vectors and matrices of different dimensions, along with a few
other basic mathematical objects discussed in the book. This library is designed to be
useful to readers beyond the examples supplied with the book, as the library includes
a wide range of functions and operators for each of these objects, some of which are
beyond the scope of the book’s demos.
The animation demos use a shared library called IvCurves, which includes classes that implement spline curves, the basic objects used to animate position. IvCurves is built upon IvMath, extending this basic functionality to include animation. As with IvMath, the IvCurves library is likely to be useful beyond the scope of the book, as these classes are flexible enough to be used (along with IvMath) in other applications.

Finally, the simulation demos use a shared library called IvCollision, which implements basic object intersection (collision) data structures and algorithms. Building on the IvMath library, this set of classes and functions forms not only the basis for the later demos in the book, but also is an excellent starting point for experimentation with other forms of object collision and physics modeling.

**Engine and Rendering Libraries**

In addition to the math libraries, the CD-ROM includes a set of classes that implement a simple game-like application framework, basic rendering, input handling, and timer functionality. All of these functions are grouped under the heading of game engine functionality, and are located in the IvEngine library. The engine’s rendering code takes the form of a set of renderer-abstraction classes that simplify the interfaces between the C++ classes in IvMath and the C-based, low-level rendering application programmer interfaces (APIs). This code is included as a part of the rendering library IvGraphics. It includes renderer setup, basic render-state management, and rendering of simple geometric primitives, such as spheres, cubes, and boxes.

Furthermore, a set of basic classes that implement a simple hierarchical data structure called a scene graph are included in the library IvScene. The classes in IvScene use and depend on the functionality of the IvCollision library. As a result, to avoid unnecessary code dependencies, the scene graph classes were placed in their own library, rather than in IvEngine.

Since this book focuses on the mathematics and concepts behind 3D games, we chose not to center the discussion around a large-scale, general 3D rendering engine. Doing so would introduce an extra layer of indirection that would not serve the conceptual requirements of the book. Valuable real estate in the rendering chapters would be spent on background in the use of a particular engine — the one written for the book. For an example and discussion of a full, hierarchical rendering engine, the reader is encouraged to read David Eberly’s *3D Game Engine Design* [25].

We have opted to implement our rendering system and examples using two standard SDKs: the multiplatform OpenGL [83] and the popular Direct3D DX9 [47]. We also use the utility toolkits provided with these SDKs (OpenGL’s GLUT and Direct3D’s D3DX) to implement cross-platform renderer setup and input handling, neither of which are core topics of this book.

**Exercises and Supplementary Material**

In addition to the sample code, we have included some useful reading material on the CD-ROM for those who haven’t absorbed enough of our luminous prose. Each chapter
has an associated set of exercises, ranging from easy to hard questions, that should help those readers interested in testing their understanding of the material within. Certain chapters also have supplemental material that unfortunately didn’t make its way into the book proper due to space considerations. Those chapters have notes at their end indicating that such material is available on the CD-ROM.

References and Further Reading

Hopefully, this book will leave readers with a desire to learn even more details and the breadth of the mathematics involved in creating high-performance, high-quality 3D games. Wherever possible, we have included references to other books, articles, papers, and websites that detail particular subtopics that fall outside the scope of this book. The full set of references may be found at the back of the book.

We have attempted to include references that the vast majority of readers should be able to locate. When possible, we have referenced recent and/or standard industry texts and well-known conference proceedings. However, in some cases we have included references to older magazine articles and technical reports when we found those references to be particularly complete, seminal, or well written. In some cases, older references can be easier for the less-experienced reader to understand, as they often tend to assume less common knowledge when it comes to computer graphics and game topics.

In the past, older magazine articles and technical reports were notoriously difficult for the average reader to locate. However, the Internet and digital publishing have made great strides toward reversing this trend. For example, the following sources have made several classes of resources far more accessible:

- The magazine most commonly referenced in this book, *Game Developer*, offers CD-ROMs that contain every issue of the magazine ever published. Copies of these CD-ROMs are available from www.gdmag.com. Several other technical magazines also offer such CD-ROMs.

- Technical societies are now placing major historical publications into their “digital libraries,” which are often made accessible to members. The Association for Computing Machinery (ACM) has done this via their ACM Digital Library, which is available to ACM members. As an example, the full text of the entire collection of papers from all SIGGRAPH conferences (the conference proceedings most frequently referenced in this book) is available electronically to ACM SIGGRAPH members.

- Other papers and technical reports are often available on the Internet. The two most common methods of finding these resources are via publication portals such as Citeseer (www.citeseer.com) and via the authors’ personal homepages (if they have them). Most of the technical reports referenced in this book are available online from such sources. Owing to the dynamic nature of the Internet, we suggest using a search engine if the publication portals do not succeed in finding the desired article.
Introduction

For further reading, we suggest several books that cover topics related to this book in much greater detail. In most cases they assume that the reader is familiar with the concepts discussed in this book. David Eberly’s *3D Game Engine Design* [25] discusses the design and implementation of a full game engine, focusing mostly on graphics and animation. Books by Gino van den Bergen [108] and Christer Ericson [32] cover topics in interactive collision detection. Finally, Eberly [27] and Millington [76] provide a more advanced discussion of a wide range of physical simulation topics.