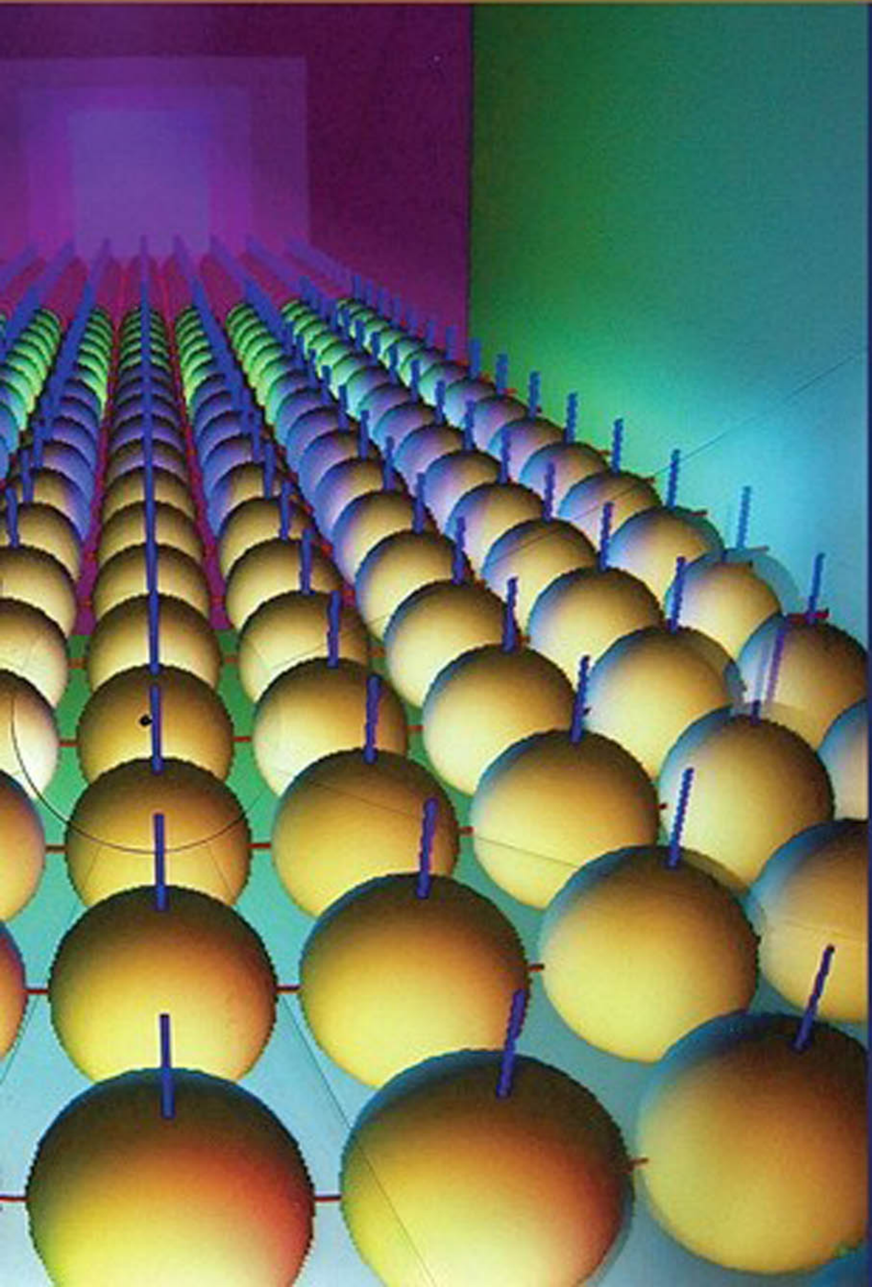
INTERACTIVE COMPUTER GRAPHICS

A Top-Down Approach with **WebGL**

7TH EDITION



EDWARD ANGEL

DAVE SHREINER

INTERACTIVE COMPUTER GRAPHICS

A Top-Down Approach with **WebGL** | 7TH EDITION

This page intentionally left blank

INTERACTIVE COMPUTER GRAPHICS

A Top-Down Approach with **WebGL** 7TH EDITION

EDWARD ANGEL
University of New Mexico

DAVE SHREINER
ARM, Inc.

PEARSON

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editorial Director, ECS Marcia Horton
Acquisitions Editor Matt Goldstein
Program Manager Kayla Smith-Tarbox
Director of Marketing Christy Lesko
Marketing Assistant Jon Bryant
Director of Production Erin Gregg
Senior Managing Editor Scott Disanno
Senior Project Manager Marilyn Lloyd
Manufacturing Buyer Linda Sager
Cover Designer Joyce Cosentino Wells
Manager, Text Permissions Tim Nicholls
Text Permission Project Manager William Opaluch
Media Project Manager Renata Butera
Full-Service Project Management Cypress Graphics, Paul C. Anagnostopoulos
Printer/Binder Courier Kendallville
Cover Printer Lehigh Phoenix-Color
Text Font Minion and Avenir

Cover Image: One frame of a particle physics simulation created with DomeGL, a version of OpenGL designed for generating images for multiprojector domed environments. Used with permission from Matthew Dosanjh, Jeff Bowles, and Joe Kniss, ARTS Lab, University of New Mexico.

Credits and acknowledgments borrowed from other sources and reproduced, with permission, appear on the appropriate page in the text.

Library of Congress Cataloging-in-Publication Data

Angel, Edward.
Interactive computer graphics : a top-down approach with WebGL / Edward Angel, Dave Shreiner. — 7th edition.
pages cm
Includes bibliographical references and indexes.
ISBN-13: 978-0-13-357484-5 (alkaline paper)
ISBN-10: 0-13-357484-9 (alkaline paper)
1. Computer graphics. 2. Interactive computer systems. 3. WebGL (Computer-program language) 4. OpenGL. I. Shreiner, Dave. II. Title.
T385.A5133 2014
006.6′633—dc23 2013050594

10 9 8 7 6 5 4 3 2 1—V011—18 17 16 15 14

PEARSON

ISBN 10: 0-13-357484-9
ISBN 13: 978-0-13-357484-5

To Rose Mary —E.A.

To Vicki, Bonnie, Bob, Cookie, and Goatee —D.S.

This page intentionally left blank

CONTENTS

PREFACE

This book is an introduction to computer graphics with an emphasis on applications programming. The first edition, which was published in 1997, was somewhat revolutionary in using OpenGL and a top-down approach. Over the succeeding 16 years and 6 editions, this approach has been adopted by most introductory classes in computer graphics and by virtually all the competing textbooks.

The sixth edition reflected the recent major changes in graphics software due to major changes in graphics hardware. In particular, the sixth edition was fully shader-based, enabling readers to create applications that could fully exploit the capabilities of modern GPUs. We noted that these changes are also part of OpenGL ES 2.0, which is being used to develop applications for embedded systems and handheld devices, such as cell phones and tablets, and of WebGL, its JavaScript implementation. At the time, we did not anticipate the extraordinary interest in WebGL that began as soon as web browsers became available that support WebGL through HTML5.

As we continued to write our books, teach our SIGGRAPH courses, and pursue other graphics-related activities, we became aware of the growing excitement about WebGL. WebGL applications were running everywhere, including on some of the latest smart phones, and even though WebGL lacks some of the advanced features of the latest versions of OpenGL, the ability to integrate it with HTML5 opened up a wealth of new application areas. As an added benefit, we found it much better suited than desktop OpenGL for teaching computer graphics. Consequently, we decided to do a seventh edition that uses WebGL exclusively. We believe that this edition is every bit as revolutionary as any of the previous editions.

New to the Seventh Edition

- WebGL is used throughout.
- All code is written in JavaScript.
- All code runs in recent web browsers.
- A new chapter on interaction is included.
- Additional material on render-to-texture has been added.
- Additional material on displaying meshes has been added.
- An efficient matrix–vector package is included.
- An introduction to agent-based modeling has been added.

A Top-Down Approach

Recent advances and the success of the first six editions continue to reinforce our belief in a top-down, programming-oriented approach to introductory computer graphics. Although many computer science and engineering departments now support more than one course in computer graphics, most students will take only a single course. Such a course usually is placed in the curriculum after students have already studied programming, data structures, algorithms, software engineering, and basic mathematics. Consequently, a class in computer graphics allows the instructor to build on these topics in a way that can be both informative and fun. We want these students to be programming three-dimensional applications as soon as possible. Low-level algorithms, such as those that draw lines or fill polygons, can be dealt with later, after students are creating graphics.

When asked "why teach programming," John Kemeny, a pioneer in computer education, used a familiar automobile analogy: You don't have to know what's under the hood to be literate, but unless you know how to program, you'll be sitting in the back seat instead of driving. That same analogy applies to the way we teach computer graphics. One approach—the algorithmic approach—is to teach everything about what makes a car function: the engine, the transmission, the combustion process. A second approach—the survey approach—is to hire a chauffeur, sit back, and see the world as a spectator. The third approach—the programming approach that we have adopted here—is to teach you how to drive and how to take yourself wherever you want to go. As the old auto rental commercial used to say, "Let us put *you* in the driver's seat."

Programming with WebGL and JavaScript

When Ed began teaching computer graphics 30 years ago, the greatest impediment to implementing a programming-oriented course, and to writing a textbook for that course, was the lack of a widely accepted graphics library or application programming interface (API). Difficulties included high cost, limited availability, lack of generality, and high complexity. The development of OpenGL resolved most of the difficulties many of us had experienced with other APIs and with the alternative of using home-brewed software. OpenGL today is supported on all platforms and is widely accepted as a cross-platform standard.

A graphics class teaches far more than the use of a particular API, but a good API makes it easier to teach key graphics topics, including three-dimensional graphics, lighting and shading, client–server graphics, modeling, and implementation algorithms. We believe that OpenGL's extensive capabilities and well-defined architecture lead to a stronger foundation for teaching both theoretical and practical aspects of the field and for teaching advanced concepts, including texture mapping, compositing, and programmable shaders.

Ed switched his classes to OpenGL about 18 years ago and the results astounded him. By the middle of the semester, *every* student was able to write a moderately complex three-dimensional application that required understanding of three-dimensional viewing and event-driven input. In the previous years of teaching computer graphics, he had never come even close to this result. That class led to the first edition of this book.

This book is a textbook on computer graphics; it is not an OpenGL or WebGL manual. Consequently, it does not cover all aspects of the WebGL API but rather explains only what is necessary for mastering this book's contents. It presents WebGL at a level that should permit users of other APIs to have little difficulty with the material.

Unlike previous editions, this one uses WebGL and JavaScript for all the examples. WebGL is a JavaScript implementation of OpenGL ES 2.0 and runs in most recent browsers. Because it is supported by HTML5, not only does it provide compatibility with other applications but also there are no platform dependences; WebGL runs within the browser and makes use of the local graphics hardware. Although JavaScript is not the usual programming language with which we teach most programming courses, it is the language of the Web. Over the past few years, JavaScript has become increasingly more powerful and our experience is that students who are comfortable with Java, C, or C++ will have little trouble programming in JavaScript.

All the modern versions of OpenGL, including WebGL, require every application to provide two shaders written in the OpenGL Shading Language (GLSL). GLSL is similar to C but adds vectors and matrices as basic types, along with some C++ features such as operator overloading. We have added a JavaScript library **MV.js** that supports both our presentation of graphics functions and the types and operations in GLSL.

Intended Audience

This book is suitable for advanced undergraduates and first-year graduate students in computer science and engineering and for students in other disciplines who have good programming skills. The book also will be useful to many professionals. Between us, we have taught well over 100 short courses for professionals; our experiences with these nontraditional students have had a great influence on what we chose to include in the book.

Prerequisites for the book are good programming skills in JavaScript, C, C++, or Java; an understanding of basic data structures (linked lists, trees); and a rudimentary knowledge of linear algebra and trigonometry. We have found that the mathematical backgrounds of computer science students, whether undergraduates or graduates, vary considerably. Hence, we have chosen to integrate into the text much of the linear algebra and geometry that is required for fundamental computer graphics.

Organization of the Book

The book is organized as follows. Chapter 1 provides an overview of the field and introduces image formation by optical devices; thus, we start with three-dimensional concepts immediately. Chapter 2 introduces programming using WebGL. Although the first example program that we develop (each chapter has one or more complete programming examples) is two-dimensional, it is embedded in a three-dimensional setting and leads to a three-dimensional extension. We introduce interactive graphics in Chapter 3 and develop event-driven graphics within the browser environment. Chapters 4 and 5 concentrate on three-dimensional concepts. Chapter 4 is concerned with defining and manipulating three-dimensional objects, whereas Chapter 5 is concerned with viewing them. Chapter 6 introduces light–material interactions and shading. Chapter 7 introduces many of the new discrete capabilities that are now supported in graphics hardware and by WebGL. All these techniques involve working with various buffers. These chapters should be covered in order and can be taught in about 10 weeks of a 15-week semester.

The last five chapters can be read in almost any order. All five are somewhat open-ended and can be covered at a survey level, or individual topics can be pursued in depth. Chapter 8 surveys implementation. It gives one or two major algorithms for each of the basic steps, including clipping, line generation, and polygon fill. Chapter 9 includes a number of topics that fit loosely under the heading of hierarchical modeling. The topics range from building models that encapsulate the relationships between the parts of a model, to high-level approaches to graphics over the Internet. Chapter 9 also includes an introduction to scene graphs. Chapter 10 introduces a number of procedural methods, including particle systems, fractals, and procedural noise. Curves and surfaces, including subdivision surfaces, are discussed in Chapter 11. Chapter 12 surveys alternate approaches to rendering. It includes expanded discussions of ray tracing and radiosity, and an introduction to image-based rendering and parallel rendering.

Appendix A presents the details of the WebGL functions needed to read, compile, and link the application and shaders. Appendices B and C contain a review of the background mathematics. Appendix D discusses sampling and aliasing starting with Nyquist's theorem and applying these results to computer graphics.

Changes from the Sixth Edition

The reaction of readers to the first six editions of this book was overwhelmingly positive, especially to the use of OpenGL and the top-down approach. In the sixth edition, we abandoned the fixed-function pipeline and went to full shader-based OpenGL. In this edition, we move to WebGL, which is not only fully shader-based—each application must provide at least a vertex shader and a fragment shader–but also a version that works within the latest web browsers.

Applications are written in JavaScript. Although JavaScript has its own idiosyncrasies, we do not expect that students with experience in a high-level language, such as Java, C, or C++, will experience any serious problems with it.

As we pointed out earlier in this preface, every application must provide its own shaders. Consequently, programmable shaders and GLSL need to be introduced in Chapter 2. Many of the examples produce the same output as in previous editions, but the code is very different.

In the sixth edition, we eliminated a separate chapter on input and interaction, incorporating the material in other chapters. With this edition, we revert to a separate chapter. This decision is based on the ease and flexibility with which we can integrate event-driven input with WebGL through HTML5.

We have added additional material on off-screen rendering and render-to-texture. These techniques have become fundamental to using GPUs for a variety of compute-intensive applications such as image processing and simulation.

Given the positive feedback we've received on the core material from Chapters 1–6 in previous editions, we've tried to keep the changes to those chapters to a minimum. We see Chapters 1–7 as the core of any introductory course in computer graphics. Chapters 8–12 can be used in almost any order, either as a survey in a one-semester course or as the basis of a two-semester sequence.

Support Materials

The support for the book is on the Web, both through the author's website www.cs.unm.edu/~angel and at www.pearsonhighered.com. Support material that is available to all readers of this book includes

- Sources of information on WebGL
- Program code
- Solutions to selected exercises
- PowerPoint lectures
- Figures from the book

Additional support materials, including solutions to all the nonprogramming exercises, are available only to instructors adopting this textbook for classroom use. Please contact your school's Pearson Education representative or visit www.pearsonhighered.com/irc for information on obtaining access to this material.

Acknowledgments

Ed has been fortunate over the past few years to have worked with wonderful students at the University of New Mexico. They were the first to get him interested in OpenGL, and he has learned much from them. They include Ye Cong, Pat Crossno, Tommie Daniel, Chris Davis, Lisa Desjarlais, Kim Edlund, Lee Ann Fisk, Maria Gallegos, Brian Jones, Christopher Jordan, Takeshi Hakamata, Max Hazelrigg, Sheryl Hurley, Thomas Keller, Ge Li, Pat McCormick, Al McPherson, Ken Moreland, Martin Muller, David Munich, Jim Pinkerton, Jim Prewett, Dave Rogers, Hal Smyer, Dave Vick, Hue (Bumgarner-Kirby) Walker, Brian Wylie, and Jin Xiong. Many of the examples in the color plates were created by these students.

The first edition of this book was written during Ed's sabbatical; various parts were written in five different countries. The task would not have been accomplished without the help of a number of people and institutions that made their facilities available to him. He is greatly indebted to Jonas Montilva and Chris Birkbeck of the Universidad de los Andes (Venezuela), to Rodrigo Gallegos and Aristides Novoa of the Universidad Tecnologica Equinoccial (Ecuador), to Long Wen Chang of the National Tsing Hua University (Taiwan), and to Kim Hong Wong and Pheng Ann Heng of the Chinese University of Hong Kong. Ramiro Jordan of ISTEC and the University of New Mexico made possible many of these visits. John Brayer and Jason

Stewart at the University of New Mexico and Helen Goldstein at Addison-Wesley somehow managed to get a variety of items to him wherever he happened to be. His website contains a description of his adventures writing the first edition.

David Kirk and Mark Kilgard at NVIDIA were kind enough to provide graphics cards for testing many of the algorithms. A number of other people provided significant help. Ed thanks Ben Bederson, Gonzalo Cartagenova, Tom Caudell, Kathi Collins, Kathleen Danielson, Roger Ehrich, Robert Geist, Chuck Hansen, Mark Henne, Bernard Moret, Dick Nordhaus, Helena Saona, Dave Shreiner, Vicki Shreiner, Gwen Sylvan, and Mason Woo. Mark Kilgard, Brian Paul, and Nate Robins are owed a great debt by the OpenGL community for creating software that enables OpenGL code to be developed over a variety of platforms.

At the University of New Mexico, the Art, Research, Technology, and Science Laboratory (ARTS Lab) and the Center for High Performance Computing have provided support for many of Ed's projects. The Computer Science Department, the Arts Technology Center in the College of Fine Arts, the National Science Foundation, Sandia National Laboratories, and Los Alamos National Laboratory have supported many of Ed's students and research projects that led to parts of this book. David Beining, formerly with the Lodestar Astronomy Center and now at the ARTS Lab, has provided tremendous support for the Fulldome Project. Sheryl Hurley, Christopher Jordan, Laurel Ladwig, Jon Strawn and Hue (Bumgarner-Kirby) Walker provided some of the images in the color plates through Fulldome projects. Hue Walker has done the wonderful covers for previous editions and some of the examples in the Color Plates.

Ed would also like to acknowledge the informal group that started at the Santa Fe Complex, including Jeff Bowles, Ruth Chabay, Stephen Guerin, Bruce Sherwood, Scott Wittenberg, and especially JavaScript evangelist Owen Densmore, who convinced him to teach a graphics course in Santa Fe in exchange for getting him involved with JavaScript. We've all gained by the experience.

Dave would like first to thank Ed for asking him to participate in this project. We've exchanged ideas on OpenGL and how to teach it for many years, and it's exciting to advance those concepts to new audiences. Dave would also like to thank those who created OpenGL, and who worked at Silicon Graphics Computer Systems, leading the way in their day. He would like to recognize the various Khronos working groups who continue to evolve the API and bring graphics to unexpected places. Finally, as Ed mentioned, SIGGRAPH has featured prominently in the development of these materials, and is definitely owed a debt of gratitude for providing access to enthusiastic test subjects for exploring our ideas.

Reviewers of the manuscript drafts provided a variety of viewpoints on what we should include and what level of presentation we should use. These reviewers for previous editions include Gur Saran Adhar (University of North Carolina at Wilmington), Mario Agrular (Jacksonville State University), Michael Anderson (University of Hartford), Norman I. Badler (University of Pennsylvania), Mike Bailey (Oregon State University), Marty Barrett (East Tennessee State University), C. S. Bauer (University of Central Florida), Bedrich Benes (Purdue University), Kabekode V. Bhat (The Pennsylvania State University), Isabelle Bichindaritz (University of Washington,

Tacoma), Cory D. Boatright (University of Pennsylvania), Eric Brown, Robert P. Burton (Brigham Young University), Sam Buss (University of California, San Diego), Kai H. Chang (Auburn University), James Cremer (University of Iowa), Ron DiNapoli (Cornell University), John David N. Dionisio (Loyola Marymount University), Eric Alan Durant (Milwaukee School of Engineering), David S. Ebert (Purdue University), Richard R. Eckert (Binghamton University), W. Randolph Franklin (Rensselaer Polytechnic Institute), Natacha Gueorguieva (City University of New York/College of Staten Island), Jianchao (Jack) Han (California State University, Dominguez Hills), Chenyi Hu (University of Central Arkansas), George Kamberov (Stevens Institute of Technology), Mark Kilgard (NVIDIA Corporation), Lisa B. Lancor (Southern Connecticut State University), Chung Lee (California State Polytechnic University, Pomona), John L. Lowther (Michigan Technological University), R. Marshall (Boston University and Bridgewater State College), Hugh C. Masterman (University of Massachusetts, Lowell), Bruce A. Maxwell (Swathmore College), Tim McGraw (West Virginia University), James R. Miller (University of Kansas), Rodrigo Obando (Columbus State University), Jon A. Preston (Southern Polytechnic State University), Andrea Salgian (The College of New Jersey), Lori L. Scarlatos (Brooklyn College, CUNY), Han-Wei Shen (The Ohio State University), Oliver Staadt (University of California, Davis), Stephen L. Stepoway (Southern Methodist University), Bill Toll (Taylor University), Michael Wainer (Southern Illinois University, Carbondale), Yang Wang (Southern Methodist State University), Steve Warren (Kansas State University), Mike Way (Florida Southern College), George Wolberg (City College of New York), Xiaoyu Zhang (California State University San Marcos), Ye Zhao (Kent State University). and Ying Zhu (Georgia State University). Although the final decisions may not reflect their views—which often differed considerably from one another—each reviewer forced us to reflect on every page of the manuscript.

The reviewers for this edition were particularly supportive. They include Mike Bailey (Oregon State University), Patrick Cozzi (University of Pennsylvania and Analytic Graphics, Inc) and Jeff Parker (Harvard University). All of them were familiar with previous editions and excited about the potential of moving their classes to WebGL.

We would also like to acknowledge the entire production team at Addison-Wesley. Ed's editors, Peter Gordon, Maite Suarez-Rivas, and Matt Goldstein, have been a pleasure to work with through seven editions of this book and the OpenGL primer. For this edition, Marilyn Lloyd and Kayla Smith-Tarbox at Pearson have provided considerable help. Through seven editions, Paul Anagnostopoulos at Windfall Software has always been more than helpful in assisting with $\TeX$ problems. Ed is especially grateful to Lyn Dupré. If the readers could see the original draft of the first edition, they would understand the wonders that Lyn does with a manuscript.

Ed wants to particularly recognize his wife, Rose Mary Molnar, who did the figures for his first graphics book, many of which form the basis for the figures in this book. Probably only other authors can fully appreciate the effort that goes into the book production process and the many contributions and sacrifices our partners make to that effort. The dedication to this book is a sincere but inadequate recognition of all of Rose Mary's contributions to Ed's work.

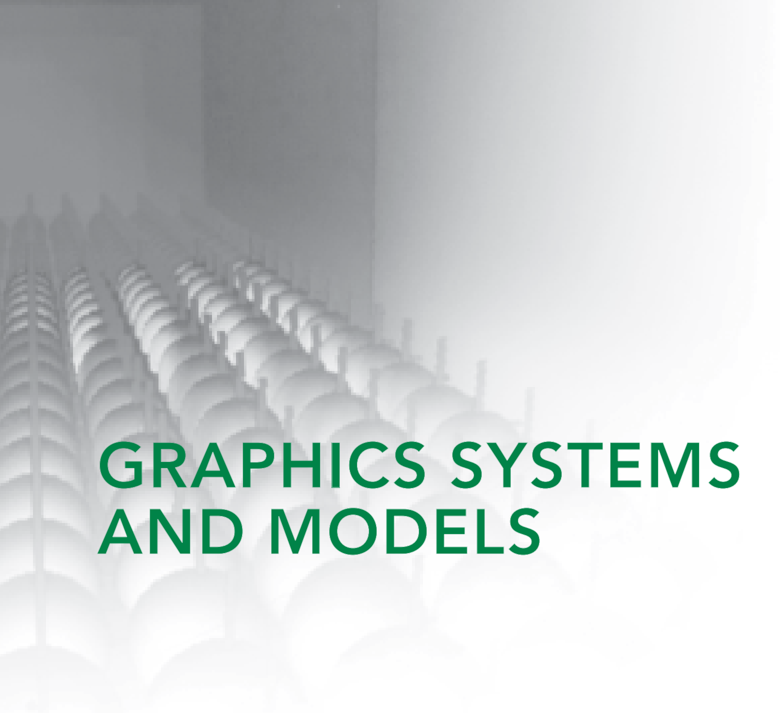
Dave would like to recognize the support and encouragement of Vicki, his wife, without whom creating works like this would never occur. Not only does she provide warmth and companionship but also provides invaluable feedback on our presentation and materials. She's been a valuable, unrecognized partner in all of Dave's OpenGL endeavors.

Ed Angel
Dave Shreiner

INTERACTIVE COMPUTER GRAPHICS

A Top-Down Approach with **WebGL** | 7TH EDITION

This page intentionally left blank



CHAPTER 1

GRAPHICS SYSTEMS AND MODELS

It would be difficult to overstate the importance of computer and communication technologies in our lives. Activities as wide-ranging as filmmaking, publishing, banking, and education have undergone revolutionary changes as these technologies alter the ways in which we conduct our daily activities. The combination of computers, networks, and the complex human visual system, through computer graphics, has been instrumental in these advances and has led to new ways of displaying information, seeing virtual worlds, and communicating with both other people and machines.

Computer graphics is concerned with all aspects of producing pictures or images using a computer. The field began humbly 50 years ago, with the display of a few lines on a cathode-ray tube (CRT); now, we can generate images by computer that are indistinguishable from photographs of real objects. We routinely train pilots with simulated airplanes, generating graphical displays of a virtual environment in real time. Feature-length movies made entirely by computer have been successful, both critically and financially.

In this chapter, we start our journey with a short discussion of applications of computer graphics. Then we overview graphics systems and imaging. Throughout this book, our approach stresses the relationships between computer graphics and image formation by familiar methods, such as drawing by hand and photography. We will see that these relationships can help us to design application programs, graphics libraries, and architectures for graphics systems.

In this book, we will use **WebGL,** a graphics software system supported by most modern web browsers. WebGL is a version of OpenGL, which is the widely accepted standard for developing graphics applications. WebGL is easy to learn, and it possesses most of the characteristics of the full (or desktop) OpenGL and of other important graphics systems. Our approach is top-down. We want you to start writing, as quickly as possible, application programs that will generate graphical output. After you begin writing simple programs, we shall discuss how the underlying graphics library and the hardware are implemented. This chapter should give a sufficient overview for you to proceed to writing programs.

1.1 APPLICATIONS OF COMPUTER GRAPHICS

The development of computer graphics has been driven both by the needs of the user community and by advances in hardware and software. The applications of computer graphics are many and varied; we can, however, divide them into four major areas:

1. Display of information
2. Design
3. Simulation and animation
4. User interfaces

Although many applications span two or more of these areas, the development of the field was based largely on separate work in each.

1.1.1 Display of Information

Classical graphics techniques arose as a medium to convey information among people. Although spoken and written languages serve a similar purpose, the human visual system is unrivaled both as a processor of data and as a pattern recognizer. More than 4000 years ago, the Babylonians displayed floor plans of buildings on stones. More than 2000 years ago, the Greeks were able to convey their architectural ideas graphically, even though the related mathematics was not developed until the Renaissance. Today, the same type of information is generated by architects, mechanical designers, and draftspeople using computer-based drafting systems.

For centuries, cartographers have developed maps to display celestial and geographical information. Such maps were crucial to navigators as these people explored the ends of the earth; maps are no less important today in fields such as geographic information systems. Now, maps can be developed and manipulated in real time over the Internet.

During the past 100 years, workers in the field of statistics have explored techniques for generating plots that aid the viewer in determining the information in a set of data. Now, we have computer plotting packages that provide a variety of plotting techniques and color tools that can handle multiple large data sets. Nevertheless, it is still the human ability to recognize visual patterns that ultimately allows us to interpret the information contained in the data. The field of information visualization is becoming increasingly more important as we have to deal with understanding complex phenomena, from problems in bioinformatics to detecting security threats.

Medical imaging poses interesting and important data analysis problems. Modern imaging technologies—such as computed tomography (CT), magnetic resonance imaging (MRI), ultrasound, and positron-emission tomography (PET)—generate three-dimensional data that must be subjected to algorithmic manipulation to provide useful information. Color Plate 20 shows an image of a person's head in which the skin is displayed as transparent and the internal structures are displayed as opaque. Although the data were collected by a medical imaging system, computer graphics produced the image that shows the structures.

Supercomputers now allow researchers in many areas to solve previously intractable problems. The field of scientific visualization provides graphical tools that help these researchers interpret the vast quantity of data that they generate. In fields such as fluid flow, molecular biology, and mathematics, images generated by conversion of data to geometric entities that can be displayed have yielded new insights into complex processes. For example, Color Plate 19 shows fluid dynamics in the mantle of the earth. The system used a mathematical model to generate the data. We present various visualization techniques as examples throughout the rest of the text.

1.1.2 Design

Professions such as engineering and architecture are concerned with design. Starting with a set of specifications, engineers and architects seek a cost-effective and aesthetic solution that satisfies the specifications. Design is an iterative process. Rarely in the real world is a problem specified such that there is a unique optimal solution. Design problems are either *overdetermined*, such that they possess no solution that satisfies all the criteria, much less an optimal solution, or *underdetermined*, such that they have multiple solutions that satisfy the design criteria. Thus, the designer works in an iterative manner. She generates a possible design, tests it, and then uses the results as the basis for exploring other solutions.

The power of the paradigm of humans interacting with images on the screen of a CRT was recognized by Ivan Sutherland over 50 years ago. Today, the use of interactive graphical tools in computer-aided design (CAD) pervades fields such as architecture and the design of mechanical parts and of very-large-scale integrated (VLSI) circuits. In many such applications, the graphics are used in a number of distinct ways. For example, in a VLSI design, the graphics provide an interactive interface between the user and the design package, usually by means of such tools as menus and icons. In addition, after the user produces a possible design, other tools analyze the design and display the analysis graphically. Color Plates 9 and 10 show two views of the same architectural design. Both images were generated with the same CAD system. They demonstrate the importance of having the tools available to generate different images of the same objects at different stages of the design process.

1.1.3 Simulation and Animation

Once graphics systems evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators. One of the most important uses has been in the training of pilots. Graphical flight simulators have proved both to increase safety and to reduce training expenses. The use of special VLSI chips has led to a generation of arcade games as sophisticated as flight simulators. Games and educational software for home computers are almost as impressive.

The success of flight simulators led to the use of computer graphics for animation in the television, motion picture, and advertising industries. Entire animated movies can now be made by computer at a cost less than that of movies made with traditional hand-animation techniques. The use of computer graphics with hand animation allows the creation of technical and artistic effects that are not possible with either alone. Whereas computer animations have a distinct look, we can also generate photorealistic images by computer. Images that we see on television, in movies, and in magazines often are so realistic that we cannot distinguish computer-generated or computer-altered images from photographs. In Chapter 6, we discuss many of the lighting effects used to produce computer animations. Color Plates 15 and 23 show realistic lighting effects that were created by artists and computer scientists using animation software. Although these images were created for commercial animations, interactive software to create such effects is widely available.

The field of virtual reality (VR) has opened up many new horizons. A human viewer can be equipped with a display headset that allows her to see separate images with her right eye and her left eye so that she has the effect of stereoscopic vision. In addition, her body location and position, possibly including her head and finger positions, are tracked by the computer. She may have other interactive devices available, including force-sensing gloves and sound. She can then act as part of a computer-generated scene, limited only by the image generation ability of the computer. For example, a surgical intern might be trained to do an operation in this way, or an astronaut might be trained to work in a weightless environment. Color Plate 22 shows one frame of a VR simulation of a simulated patient used for remote training of medical personnel.

Simulation and virtual reality have come together in many exciting ways in the film industry. Recently, stereo (3D) movies have become both profitable and highly acclaimed by audiences. Special effects created using computer graphics are part of virtually all movies, as are more mundane uses of computer graphics such as removal of artifacts from scenes. Simulations of physics are used to create visual effects ranging from fluid flow to crowd dynamics.

1.1.4 User Interfaces

Our interaction with computers has become dominated by a visual paradigm that includes windows, icons, menus, and a pointing device, such as a mouse. From a user's perspective, windowing systems such as the X Window System, Microsoft Windows, and the Macintosh Operating System differ only in details. More recently, millions of people have become users of the Internet. Their access is through graphical network browsers, such as Firefox, Chrome, Safari, and Internet Explorer, that use these same interface tools. We have become so accustomed to this style of interface that we often forget that what we are doing is working with computer graphics.

Although personal computers and workstations evolved by somewhat different paths, at present they are indistinguishable. When you add in smart phones, tablets, and game consoles, we have an incredible variety of devices with considerable computing power, all of which can access the World Wide Web through a browser. For lack of a better term, we will tend to use *computer* to include all these devices.

Color Plate 13 shows the interface used with a high-level modeling package. It demonstrates the variety of tools available in such packages and the interactive devices the user can employ in modeling geometric objects. Although we are familiar with this style of graphical user interface, devices such as smart phones and tablets have popularized touch-sensitive interfaces that allow the user to interact with every pixel on the display.

1.2 A GRAPHICS SYSTEM

A computer graphics system is a computer system; as such, it must have all the components of a general-purpose computer system. Let us start with the high-level view of a graphics system, as shown in the block diagram in Figure 1.1. There are six major elements in our system:

1. Input devices
2. Central Processing Unit
3. Graphics Processing Unit
4. Memory
5. Framebuffer
6. Output devices

This model is general enough to include workstations and personal computers, interactive game systems, mobile phones, GPS systems, and sophisticated image generation systems. Although most of the components are present in a standard computer, it is the way each element is specialized for computer graphics that characterizes this diagram as a portrait of a graphics system. As more and more functionality can be included in a single chip, many of the components are not physically separate. The CPU and GPU can be on the same chip and their memory can be shared. Nevertheless, the model still describes the software architecture and will be helpful as we study the various parts of computer graphics systems.

1.2.1 Pixels and the Framebuffer

Virtually all modern graphics systems are raster based. The image we see on the output device is an array—the **raster**—of picture elements, or **pixels**, produced by the graphics system. As we can see from Figure 1.2, each pixel corresponds to a location, or small area, in the image. Collectively, the pixels are stored in a part of

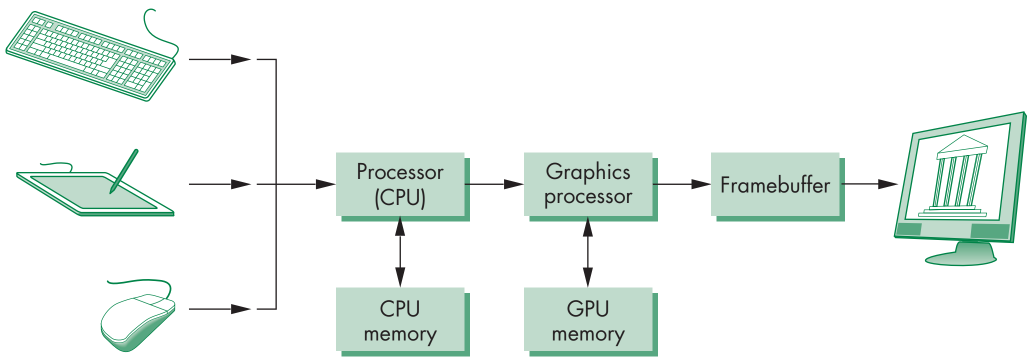


FIGURE 1.1 A graphics system.

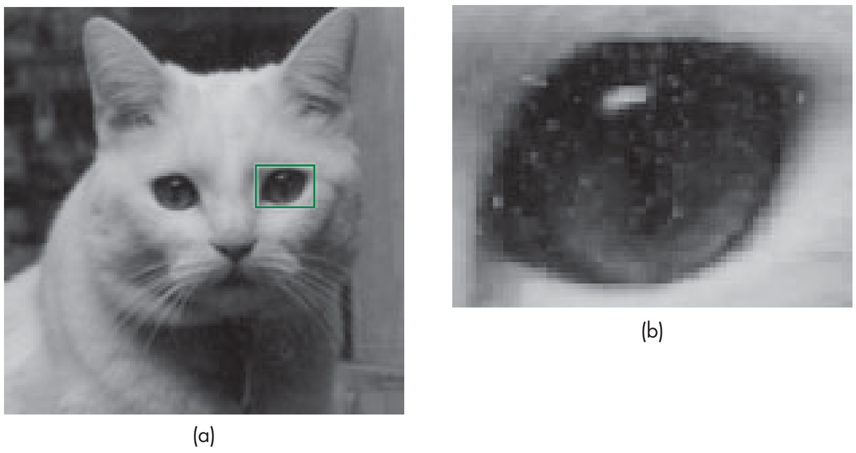


FIGURE 1.2 Pixels. (a) Image of Yeti the cat. (b) Detail of area around one eye showing individual pixels.

memory called the **framebuffer**.[1] The framebuffer can be viewed as the core element of a graphics system. Its **resolution**—the number of pixels in the framebuffer—determines the detail that you can see in the image. The **depth**, or **precision**, of the framebuffer, defined as the number of bits that are used for each pixel, determines properties such as how many colors can be represented on a given system. For example, a 1-bit-deep framebuffer allows only two colors, whereas an 8-bit-deep framebuffer allows 2^8 (256) colors. In **full-color** systems, there are 24 (or more) bits per pixel. Such systems can display sufficient colors to represent most images realistically. They are also called **true-color** systems, or **RGB color** systems, because individual groups of bits in each pixel are assigned to each of the three primary colors—red, green, and blue—used in most displays. **High dynamic range** (HDR) systems use 12 or more bits for each color component. Until recently, framebuffers stored colors in integer formats. Recent framebuffers use floating point and thus support HDR colors more easily.

In a simple system, the framebuffer holds only the colored pixels that are displayed on the screen. In most systems, the framebuffer holds far more information, such as depth information needed for creating images from three-dimensional data. In these systems, the framebuffer comprises multiple buffers, one or more of which are **color buffers** that hold the colored pixels that are displayed. For now, we can use the terms *framebuffer* and *color buffer* synonymously without confusion.

1.2.2 The CPU and the GPU

In a simple system, there may be only one processor, the **central processing unit** (**CPU**), which must perform both the normal processing and the graphical process-

1. Some references use *frame buffer* rather than *framebuffer*.

ing. The main graphical function of the processor is to take specifications of graphical primitives (such as lines, circles, and polygons) generated by application programs and to assign values to the pixels in the framebuffer that best represent these entities. For example, a triangle is specified by its three vertices, but to display its outline by the three line segments connecting the vertices, the graphics system must generate a set of pixels that appear as line segments to the viewer. The conversion of geometric entities to pixel colors and locations in the framebuffer is known as **rasterization** or **scan conversion**. In early graphics systems, the framebuffer was part of the standard memory that could be directly addressed by the CPU. Today, virtually all graphics systems are characterized by special-purpose **graphics processing units** (**GPUs**), custom-tailored to carry out specific graphics functions. The GPU can be located on the motherboard of the system or on a graphics card. The framebuffer is accessed through the graphics processing unit and usually is on the same circuit board as the GPU.

GPUs have evolved to the point where they are as complex or even more complex than CPUs. They are characterized both by special-purpose modules geared toward graphical operations and by a high degree of parallelism—recent GPUs contain over 100 processing units, each of which is user programmable. GPUs are so powerful that they can often be used as mini supercomputers for general-purpose computing. We will discuss GPU architectures in more detail in Section 1.7.

1.2.3 Output Devices

Until recently, the dominant type of display (or **monitor**) was the **cathode-ray tube** (**CRT**). A simplified picture of a CRT is shown in Figure 1.3. When electrons strike the phosphor coating on the tube, light is emitted. The direction of the beam is controlled by two pairs of deflection plates. The output of the computer is converted, by digital-to-analog converters, to voltages across the *x* and *y* deflection plates. Light appears on the surface of the CRT when a sufficiently intense beam of electrons is directed at the phosphor.

If the voltages steering the beam change at a constant rate, the beam will trace a straight line, visible to the viewer. Such a device is known as the **random-scan**,

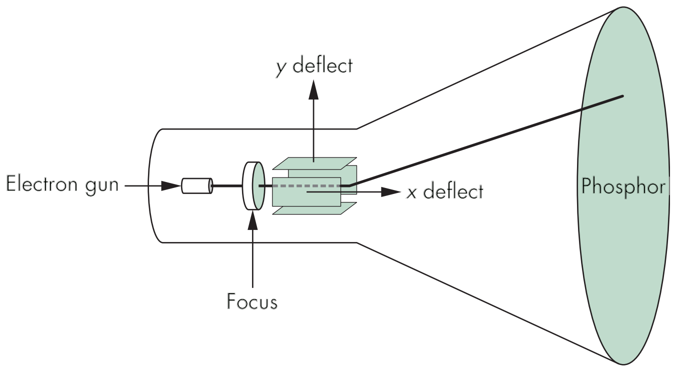


FIGURE 1.3 The cathode-ray tube (CRT).

calligraphic, or **vector** CRT, because the beam can be moved directly from any position to any other position. If intensity of the beam is turned off, the beam can be moved to a new position without changing any visible display. This configuration was the basis of early graphics systems that predated the present raster technology.

A typical CRT will emit light for only a short time—usually, a few milliseconds—after the phosphor is excited by the electron beam. For a human to see a steady, flicker-free image on most CRT displays, the same path must be retraced, or **refreshed**, by the beam at a sufficiently high rate, the **refresh rate**. In older systems, the refresh rate is determined by the frequency of the power system, 60 cycles per second or 60 hertz (Hz) in the United States and 50 Hz in much of the rest of the world. Modern displays are no longer coupled to these low frequencies and operate at rates up to about 85 Hz.

In a raster system, the graphics system takes pixels from the framebuffer and displays them as points on the surface of the display in one of two fundamental ways. In a **noninterlaced** system, the pixels are displayed row by row, or scan line by scan line, at the refresh rate. In an **interlaced** display, odd rows and even rows are refreshed alternately. Interlaced displays are used in commercial television. In an interlaced display operating at 60 Hz, the screen is redrawn in its entirety only 30 times per second, although the visual system is tricked into thinking the refresh rate is 60 Hz rather than 30 Hz. Viewers located near the screen, however, can tell the difference between the interlaced and noninterlaced displays. Noninterlaced displays are becoming more widespread, even though these displays must process pixels at twice the rate of the interlaced display.

Color CRTs have three different-colored phosphors (red, green, and blue), arranged in small groups. One common style arranges the phosphors in triangular groups called **triads**, each triad consisting of three phosphors, one of each primary. Most color CRTs have three electron beams, corresponding to the three types of phosphors. In the shadow-mask CRT (Figure 1.4), a metal screen with small holes—the **shadow mask**—ensures that an electron beam excites only phosphors of the proper color.

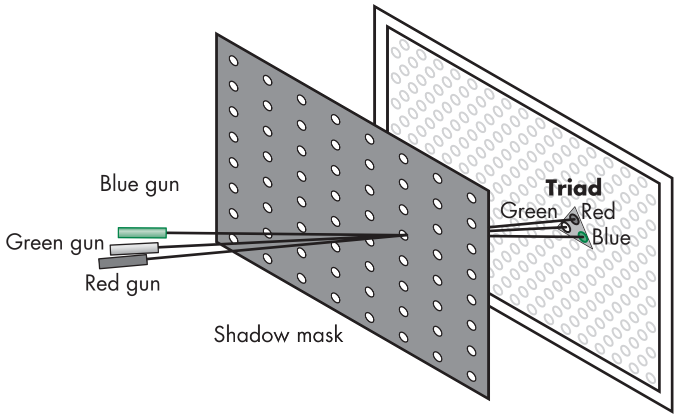


FIGURE 1.4 Shadow-mask CRT.

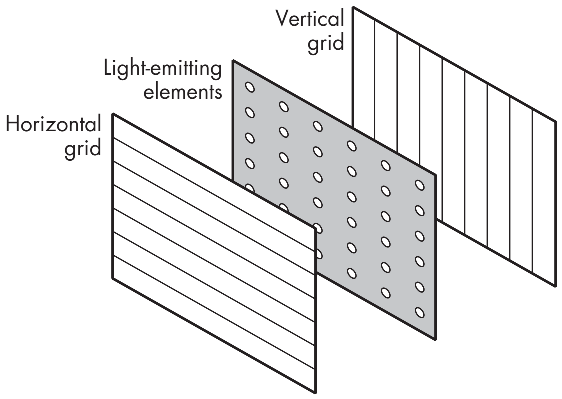


FIGURE 1.5 Generic flat-panel display.

Although CRTs are still common display devices, they are rapidly being replaced by flat-screen technologies. Flat-panel monitors are inherently raster based. Although there are multiple technologies available, including light-emitting diodes (LEDs), liquid-crystal displays (LCDs), and plasma panels, all use a two-dimensional grid to address individual light-emitting elements. Figure 1.5 shows a generic flat-panel monitor. The two outside plates each contain parallel grids of wires that are oriented perpendicular to each other. By sending electrical signals to the proper wire in each grid, the electrical field at a location, determined by the intersection of two wires, can be made strong enough to control the corresponding element in the middle plate. The middle plate in an LED panel contains light-emitting diodes that can be turned on and off by the electrical signals sent to the grid. In an LCD display, the electrical field controls the polarization of the liquid crystals in the middle panel, thus turning on and off the light passing through the panel. A plasma panel uses the voltages on the grids to energize gases embedded between the glass panels holding the grids. The energized gas becomes a glowing plasma.

Most projection systems are also raster devices. These systems use a variety of technologies, including CRTs and digital light projection (DLP). From a user perspective, they act as standard monitors with similar resolutions and precisions. Hard-copy devices, such as printers and plotters, are also raster based but cannot be refreshed.

Stereo (3D) television displays use alternate refresh cycles to switch the display between an image for the left eye and an image for the right eye. The viewer wears special glasses that are coupled to the refresh cycle. 3D movie projectors produce two images with different polarizations. The viewer wears polarized glasses so that each eye sees only one of the two projected images. As we shall see in later chapters, producing stereo images is basically a matter of changing the location of the viewer for each frame to obtain the left- and right-eye views.

1.2.4 Input Devices

Most graphics systems provide a keyboard and at least one other input device. The most common input devices are the mouse, the joystick, and the data tablet. Each