世界著名大学核心教材（计算机类）

计算机图形学（影印版）
(OpenGL版)

Computer Graphics Using OpenGL
Second Edition

（美） F.S. Hill, JR. 著

Pearson Education
科学出版社
www.sciencep.com

世界著名大学核心教材（计算机类）

计算机图形学（OpenGL 版）

（影印版）

Computer Graphics Using OpenGL

Second Edition

（美） F.S. Hill, JR. 著

科学出版社

北京

图字：01-2003-7664 号

内 容 简 介

本书介绍了计算机图形学的原理和主要技术，并以涉及学生和职业人士在网上天天见到的和计算机合成电影的艺术级的图形为例。作者用完整的、综合的手法，写出饿实用性强而又容易理解的内容，它们都是经过精心选择的，优先数学解析，而且重要的概念都加以突出强调，

本书引导读者如何将数学转化为程序代码并显示效果。本书内容涵盖了计算机图形领域的最新信息。可作为计算机专业的本科生、研究生教材及相关专业、相关人员的参考用书。

图书在版编目(CIP)数据

计算机图形学（OpenGL 版）=Computer Graphics Using OpenGL/（美）希尔（F. S, Hill, Jr.）著.—影印本. —北京：科学出版社，2004

ISBN 7-03-012499-5

Ⅰ.计… Ⅱ.希… Ⅲ.计算机图形学—高等学校—教材—英文 Ⅳ.TP391.41

中国版本图书馆 CIP 数据核字（2003）第 103044 号

策划编辑：李佩乾 / 责任编辑：王日臣

责任印制：吕春珉 / 封面制作：东方人华平面设计室

科学出版社 出版

北京东黄城根北街16号

邮政编码:100717

http://www.sciencep.com

双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2004 年 1 月第 一 版 开本：787×960 1/16

2004 年 1 月第一次印刷 印张：59 1/4 插页：6

印数：1—3 000 字数：1 381 000

定价：95.00 元

（如有印装质量问题，我社负责调换〈环伟〉）

Preface

This book provides an introduction to computer graphics for students who wish to learn the basic principles and techniques of the field and who, in addition, want to write substantial graphics applications themselves. The field of computer graphics continues to enjoy tremendous vitality and growth. The ever-increasing number of feature-length animated movies has generated heady excitement about what graphics can do, and the ready access to graphics everyone now has through computer games and the Internet is stimulating people to learn how to do it themselves.

Graphics systems are getting better, faster, and cheaper at a bewildering rate, and many new techniques are emerging each year from researchers and practitioners around the world, but the underlying principles and approaches constitute a stable and coherent body of knowledge. Much of this knowledge can be acquired through a single course in graphics, and this book attempts to organize the ideas and methods to bring the reader from the beginning, with modest programming skills, to being able to design and produce significant graphics programs.

INTENDED AUDIENCE

The book is designed as a text for either a one- or two-semester course at the senior undergraduate or first-year graduate level. It can also be used for self-study. It is aimed principally at students majoring in computer science or engineering, but will also suit students in other fields, such as physics and mathematics.

Mathematical Background Required

The reader should have the equivalent of one year of college mathematics; knowledge of elementary algebra, geometry, trigonometry, and elementary calculus also is assumed. Some exposure to vectors and matrices is useful, but not essential, as vector and matrix techniques are introduced in the context of graphics as they are needed, and an appendix also summarizes the key ideas.

Computer graphics tends to use a lot of mathematics to express the geometric relationships between lines, surfaces, and the viewing eye. Although no single mathematical notion is difficult in itself, the sheer number of tools required can be daunting. The book places particular emphasis on revealing the reasons for using this or that technique and on showing how the objects of interest in a graphics program are properly described by the mathematical objects we use.

Computer Programming Background Required

In general, the reader should have at least one semester of experience writing computer programs in C, C++, or Java. A lot of the programming in graphics involves the direct translation of geometric relationships into code and so uses straightforward variables, functions, arrays, looping, and testing, which is similar from language to language. C++ is used throughout the book, but much of the material will be familiar to someone whose computer language background is only C.

It is helpful for the reader to have experience as well in manipulating `struct`'s in C or classes in C++.

These are used to capture the rather complicated structure of some graphical objects that reside in a scene, where the object (say, a castle or an airplane) consists of many parts and these parts themselves consist of complex subparts. Some experience with elementary linked data structures such as linked lists or trees is also desirable, but not essential.

A reader with knowledge of C but not C++ will need to pick up the basics of object-oriented programming. We define a number of useful classes (such as the

`Window`, `Mesh`, `Scene`, `Camera`, and `Texture` classes) and show why they are so convenient and usable. Some of the hallmarks of object-oriented programming, such as inheritance and polymorphism, are used in a few contexts to make the programmer's job easier, but we do not place inordinate emphasis on a pure object-oriented approach.

PHILOSOPHY

The book has been completely reorganized and rewritten from the first edition, but the basic philosophy remains: Computer graphics is learned by doing it: One must write and test real programs to comprehend fully what is going on. A principal goal of the book is to show readers how to translate a particular design "task" first into its underlying geometric components, to find a suitable mathematical representation for the objects involved, and finally to translate this representation into suitable algorithms and program code. Readers start by learning how to develop simple routines to produce pictures. Then methods for rendering drawings of ever more complex objects are presented in a step-by-step fashion.

Exercises and Problems

More than 440 drill exercises appear throughout the book. Most of these are of the "stop-and-think" variety that require no programming and that allow readers to test their grasp of the material themselves. Some urge the student to implement some of the new ideas in program code.

In addition, "case studies" appear at the end of each chapter, amounting to 100 in all. These exercises are normally programming projects suitable for homework assignments and range from the simple to the challenging. They expand on the material within their chapter and often extend ideas in new directions. Whether or not the case studies are actually carried out by students, they should be studied as an integral part of the chapter.

A suggested "level of effort" is associated with each case study, to indicate the approximate investment in time a student may need to accomplish the task. Programming is an unpredictable business and students' abilities vary, but the rough guide is as follows:

Level of Effort

I: A simple project that can be implemented in an evening, suitable to be made due at the next class meeting.

II: A more extensive project that might be assigned to be due in a week, so that a student has time to think about designing the program and has adequate time for the iterative (and sometimes frustrating) testing and debugging cycle that projects always seem to require.

III: A major project that might require three weeks to design and implement. Such a project requires substantial design effort and careful program layout, but would (correctly) be viewed as a major accomplishment by the student.

Use of OpenGL

A frequent stumbling block that appears as one first brushes up against computer graphics is getting started making pictures. It is easy enough to write a program, but there must be an underlying tool that ultimately draws the lines and curves on the screen. Fortunately, such a tool exists and is readily available. OpenGL emerged from Silicon Graphics, Inc., in 1992 and has become a widely adopted graphics application programming interface (API). It provides the actual drawing tools through a collection of functions that are called within an application. As described in Appendix 1, it

is available (usually through free downloads over the Internet) for all types of computer systems encountered in colleges, universities, and industry. OpenGL is easy to install and learn, and its longevity as a standard API is being nurtured by the OpenGL Architecture Review Board (ARB), an industry consortium responsible for guiding the evolution of the software.

One aspect of OpenGL that makes it so well suited for use in a computer graphics course is its "device independence," or portability. Many university computer laboratories contain a variety of different computers. A student can develop and run a program on any available computer. The program can then be run on a different computer, for testing or grading purposes perhaps, and the graphics will be the same on the two machines.

OpenGL offers a rich and highly usable API for 2D graphics and image manipulation, but its real power emerges with 3D graphics. Using OpenGL, students can progress rapidly and produce stunning animations in only a single-semester course.

Use of C++ as the Programming Language

C++ is now familiar enough to most students in engineering and computer science through a first programming course, that it is the natural choice of language to use. It offers several advantages over C, such as passing parameters to functions by reference, which reduces the need for explicit pointers and simplifies reading the code. File I/O also is greatly simplified through streams, and in general, the syntax for all kinds of I/O is clearer in C++ than in C. To keep things simple, in C++ no emphasis is placed on implementing operators.

Furthermore, it is easy to develop handy utility classes in C++, such as those for a 2D or 3D point, a line, a window, or a color, which make code simpler and more robust. Students see the benefit of hiding the details of a geometric object within the object itself and of imbuing the object with the ability to do things like draw itself or test whether it intersects another object. The `Canvas` class developed in Chapter 3 offers a good example, as it maintains its own notion of a window, a viewport, and a current position, and it can draw basic figures with very little programming effort.

Emphasis on 3D Computer Graphics

Because playing games on personal computers has become so popular, and so many dazzling animations are appearing in movies, students are particularly interested in developing 3D graphics applications. Accordingly, several chapters from the first edition have been rewritten and rearranged in order to get to topics in 3D graphics as quickly as possible. In a number of situations, concepts are presented for the 2D case and the 3D case together, which helps to clarify the distinctions between the two.

Describing 3D Scenes with the use of Scene Design Language

It can be very awkward and time consuming to design a scene of many 3D objects using "raw" OpenGL commands. So a simple Scene Design language (SDL) is introduced in Chapter 5 (and fully defined in an appendix). Using this language, students can describe scenes with familiar terms like "cube," "sphere," and "rotate" and can build files of such instructions that can be read into their program at run time. An appendix (and the book's Web Site) provides code for an interpreter that can read an SDL file and build a list of objects described in the file. It is then a simple matter to use OpenGL to draw the scene from the object list.

This same language and interpreter is put to fine use in Chapter 14, in which the student develops code for ray tracing a scene described using SDL. Students can therewith design and ray trace much more elaborate and interesting scenes than would be possible otherwise.

Optional Use of PostScript®

In recent years, PostScript has become a de facto standard page-layout language, offering a rich set of operators for drawing text and graphics in a device-independent manner. PostScript usually works invisibly within a laser printer, receiving commands from a word-processing or page-layout program and converting them to lines, dots, and characters. But it is possible for a student to prepare a "script" of PostScript commands and direct it to a printer, whereupon the onboard PostScript interpreter creates the intended graphics. Beautiful graphics can be created in this way. Therefore, PostScript provides an excellent example of a concise and powerful 2D graphics language, with many of the same capabilities as OpenGL to carry out transformations and perform rendering.

The PostScript language is introduced in an appendix, and students interested in approaching graphics this way are shown how to create interesting scripts that produce pleasing pictures. The appendix also shows how to download and work with GhostScript, which provides an on-screen PostScript interpreter, so that pictures can be easily previewed and debugged during their development.

ORGANIZATION OF THE BOOK AND COURSE PLANS

There is much more in this book than can be covered in a one-semester course or even in a two-semester course. The book has been arranged so that the instructor can select different groups of chapters for close study, depending on the length of the course and the interests and backgrounds of the students in the class. Several such paths through the book are suggested here, after the principal topics in each chapter are described.

Brief Overview of Each Chapter

Chapter 1. This chapter gives an overview of the computer graphics field, with examples of how various of its subfields are using graphics. The different kinds of graphics display systems available are described, along with the types of "primitives" (polygons, text, images, etc.) that a graphic system displays. The chapter also describes some of the many kinds of input devices (mouse, tablet, data glove, etc.) that are in common use.

Chapter 2. This chapter gets students started with writing graphics applications. Programming using OpenGL is described, and several complete line-drawing applications (including the popular Sierpinski gasket) are developed. Techniques are discussed for using OpenGL to draw various primitives such as polylines and polygons and for using the mouse and keyboard in an interactive graphics application. Case studies at the end of the chapter provide interesting programming projects to help students get a clear initial sense of how a graphics application is implemented.

Chapter 3. Chapter 3 develops the central notion of the window-to-viewport mapping, for sizing and positioning pictures on the display. Do-it-yourself management of windows and viewports is discussed, as is using OpenGL to handle the details. A first clipping algorithm is developed. Zooming, panning, and tilting to achieve interesting visual effects are described, as is the simple animation of figures. A `Canvas` class is developed that encapsulates all of the tools. The drawing of complex polygon-based figures, circles, and arcs is discussed, as is the parametric form for representing both 2D and 3D curves.

Chapter 4. This chapter reviews vectors and their basic operations and shows the great benefits to be gained by using vector tools in graphics. Students who are familiar with vectors can read the chapter quickly, focusing on how vectors describe relations between the geometric objects they manipulate in their programs. Where

possible, vector operations are treated without regard for the dimensionality of the space in question, but the use of the cross product in 3D is given special emphasis.

The notion of a coordinate frame is introduced, and it is shown how such frames make it natural to work with homogeneous coordinates. Affine combinations of points are discussed to clarify the difference between vectors and points (to help avoid a common pitfall that arises when one writes graphics applications). Several applications involving interpolation, elementary Bezier curves, and line intersections are developed. The fundamental algorithm to clip a line against a convex polygon is developed in detail, and more advanced clipping algorithms are addressed in the case studies. (An interesting project for "2D ray tracing" is suggested in one case study.)

Chapter 5. Transformations are of central importance in computer graphics, and students sometimes have difficulty developing intuitions about them—particularly about 3D transformations. This chapter develops the underlying theory of transforming figures and coordinate systems using affine transformations in both the 2D and 3D cases. Homogeneous coordinates are employed from the start for describing transformations. Special care is given to rotations in 3D, which are notoriously difficult to visualize.

Tools are added to the `Canvas` class set forth in Chapter 3 to shift, scale, and rotate figures through the "current transformation," and OpenGL's matrix operations are enlisted to facilitate this feature. An overview of the OpenGL viewing pipeline is then developed, and the roles of the modelview, projection, and viewport transformations are described. The drawing of 3D objects using OpenGL's tools is developed. The use of Scene Description Language (SDL) is introduced, and it is shown how to use the SDL interpreter to read in a description of a 3D scene from a file and to draw the objects represented in the file.

Chapter 6. In this chapter, tools are developed for modeling and drawing complicated mesh objects. Sample meshes are developed, including polyhedra such as the dodecahedron and buckyball and more complex shapes such as arches, domes, "tubes" that undulate through space, and surfaces of revolution. Techniques are developed for rendering these objects either with flat or smooth shading.

Chapter 7. This chapter develops tools for the flexible viewing of 3D scenes. The "synthetic camera" that forms perspectival views is defined, and its relationship to the low-level viewing tools OpenGL provides is discussed. A convenient `Camera` class is built that encapsulates the details of manipulating the camera and makes it easy to "fly" the camera through a scene in an animation.

The mathematics of perspectival projections is then developed in detail, along with a discussion of how OpenGL produces perspectival views through matrix manipulations. The clipping algorithm that operates in homogeneous coordinate space (which OpenGL also uses) is developed in detail. Methods for producing stereo views are introduced. The chapter closes with a taxonomy of the many kinds of projections used in art, architecture, and engineering and shows how to produce each kind of projection in a program.

Chapter 8. Chapter 8 tackles ways to make pictures of 3D scenes more realistic. Shading models are developed that compute the various light components that reflect off of objects that are bathed in light. Methods for using OpenGL to set up light sources and alter the surface material properties of objects are described. OpenGL's depth-buffer method of removing hidden surfaces is described in detail. Techniques for "painting" texture onto the surface of an object to make it more realistic are developed, for both procedural and "image" textures. Finally, methods for adding simple shadows to pictures are presented.

Chapter 9. This chapter delves into the fascinating area of fractals and ways to generate images of them. Methods are presented for refining a curve's shape to maintain "self-similarity," which, in the limit, produces a fractal. Methods are also presented for

drawing very complex curves based on a small set of "string-replacement" rules. Tiling the plane with a small set of shapes, including "reptiles," is described.

Methods are described for drawing complex images known as "strange attractors." These methods use the repeated application of a few affine transformations. The inverse problem of how to find a set of affine transformations whose attractor is a given image is presented and leads to a discussion of fractal image compression that exploits the technique. The celebrated Mandelbrot set and Julia sets are introduced, and tools to draw them are developed.

Chapter 10. Chapter 10 discusses powerful graphics methods for manipulating images formed on a raster display.

The basic pix map is revisited as a fundamental object for storing and manipulating images, and a number of operations for manipulating pix maps are developed. The classical Bresenham's algorithm for drawing lines is described in detail. Ways to describe "regions" in a pix map and to fill them with a color or pattern are developed. Particular attention is given to filling a polygonal region. The phenomenon of aliasing that plagues graphics programmers is discussed, and some techniques for reducing aliasing are developed. The techniques of dithering and error diffusion that produce the effect of more colors than a device can display also are described.

Chapter 11. This chapter is devoted to the design and drawing of "smooth" curves and surfaces. The theory of Bezier and B-spline curves is described, along with that for rational B-splines, which leads to a discussion of NURBS curves. Interactive curve design is presented, wherein a designer specifies a set of "control points" with a mouse and uses a curve-generation algorithm to preview the curve associated with those points. The curve may either interpolate the points or merely be attracted to them.

Complex surface design using Bezier, B-spline, and NURBS patches is also developed, and the issue of joining two patches together seamlessly is addressed.

Chapter 12. This chapter examines some intricacies of the human color vision system and addresses the problem of representing colors numerically. The CIE standard chromaticity diagram is described, along with various ways to use it in color calculations. The color gamuts of various devices also are discussed, as are different color spaces and conversions of colors between them. The problem of efficient color quantization, which attempts to reduce the number of different colors in an image without destroying its visual quality, is developed.

Chapter 13. In this chapter, several methods are developed for performing proper hidden surface removal (HSR) in pictures of 3D scenes. The difference between "image-precision" and "object-precision" algorithms is discussed, along with ways to preprocess the polygonal faces in a scene for rapid HSR. The depth-buffer method first seen in Chapter 8 is examined more deeply.

Several HSR methods based on sorting the list of faces to allow rapid rendering, including the binary space partition approach, are discussed. A scan-line HSR method also is developed, and its advantages over the depth-buffer method are described. Further HSR methods based on a "divide-and-conquer" approach are also discussed.

Chapter 14. Chapter 14 introduces the powerful ray-tracing approach to rendering scenes with high realism. Working through this chapter, the student can first develop a primitive, but simple, ray tracer and then add on capabilities to ultimately produce a full ray tracer that can generate dazzling images. Methods to intersect rays with various shapes are described, followed by ways to render the objects using different shading models. The physically based Cook–Torrance reflection model, which OpenGL cannot provide, is developed for use in ray tracing. Techniques for painting texture onto ray-traced surfaces—both 3D textures such as marble and image-based textures—are described in detail. Methods to speed up ray tracing using bounding boxes are also developed.

A great advantage of ray tracing is that it automatically performs HSR and makes it easy to create exact shadows of objects. In addition, it allows one to simulate the reflection of light from shiny surfaces, as well as the refraction of light through transparent objects. Methods to accomplish each of these aims are described. The chapter ends with a thorough discussion of ray tracing complex objects formed by using "constructive solid geometry."

Suggested Paths through the Book

All suggested paths through the book include Chapters 1 through 5 as fundamental, although Chapter 4 can be perused independently by students who are familiar with vectors. Chapter 9 can be tackled after Chapter 5 with no loss in continuity, as can Chapter 10. The 2D parts of Chapter 11 also may be studied after Chapter 5.

Possible Course Plans

- For a *one-semester undergraduate course* in which interest is highest in **3D graphics**: Chapters 1 through 5, with parts of Chapter 6 and Chapter 7 and parts of Chapter 9.
- If *extending the material to a two-semester course*, add the rest of Chapter 7 and parts of Chapters 8, 10, and 11.
- For a *one-semester undergraduate course* in which interest is highest in **2D and raster graphics**: Chapters 1 through 3, along with the POSTSCRIPT appendix and parts of Chapters 4 and 5. Also include Chapter 9.
- If extending the material to a *two-semester course*, add parts of Chapters 7 and 8, and include Chapters 10 and 11 and parts of Chapter 12.
- For a *one-semester graduate course* in which interest is highest in **3D graphics**: Chapters 1 through 7, with parts of Chapters 8 and 9.
- If *extending the material to a two-semester course*, add the rest of Chapter 8, and include parts of Chapters 10 and all of Chapters 11 through 14.
- For a *one-semester graduate course* in which interest is highest in **2D and raster graphics**: Chapters 1 through 3, along with the POSTSCRIPT appendix and parts of Chapters 4 through 8. Include Chapter 9 and 10 as well.
- If *extending the material to a two-semester course*, add Chapters 11 and 12 and parts of Chapters 13 and 14.

SUPPLEMENTS

An accompanying instructor's manual provides solutions to most of the exercises and suggests additional projects. Complete demonstration programs of techniques developed in the text are explained and listed.

Materials are also available through the book's site on the Internet: `http://www.prenhall.com/hill`.

Many samples of code and utility libraries are available here as well, as are images and textures. All may be used freely.

ACKNOWLEDGMENTS

This book and the first edition have grown out of notes used in courses I have been teaching at the University of Massachusetts for the last 19 years. During this time, a large number of students have helped to develop demonstrations and make suggestions for improving the courses. They have also produced many exquisite graphical samples, some of which appear here. Some students who have been particularly helpful in the first and second editions are Tarik Abou-Raya, Earl Billingsley, Dennis Chen, Daniel Dee, Brett Diamond, Jay Greco, Tom Kopec, Adam Lavine, Russell

Turner, Bill Verts, Shel Walker, Noel Llopis, Russell Swan, A. Chandrashekhara, Emmanuel Agu, Tom Laramee, Chang Su, Xiongzi Li, Jung-Yao Huang, Anjul Srivastava, Steve Morin, and Elwood Anderson. I apologize for any inadvertent omissions.

Several colleagues have provided inspiration and guidance during the germination of the book. I am particularly grateful to Charles Hutchinson for his support in starting the graphics effort at the university, to Michael Wozny for his enthusiasm and encouragement in the development of that effort, and to Charle Rupp for the many creative ideas in graphics he passed on to me. I would especially like to thank Daniel Bergeron, who made substantial contributions to the coherence and readability of the first edition.

I would also like to thank the following individuals, and many others who are not mentioned by name, for their advice and help: Edward Hammerand, Arkansas State University; Deborah Walters, SUNY at Buffalo; Suzanne M. Lea, University of North Carolina at Greensboro; John Neitzke, Northeast Missouri State University; Norman Hosay, University of New Haven; David F. McAllister, North Carolina State University; John DeCatrel, Florida State University; Steve Cunningham, California State University, Stanislaus; Paul Heckbert, Carnegie Mellon University; Angelo Yfantis, University of Nevada; Lee H. Tichenor, Western Illinois University; Norman Wittels, Worcester Polytechnic Institute; Edward Angel, University of New Mexico; Matthew Ward, Worcester Polytechnic Institute; Richard E. Neapolitan, Northeastern Illinois University; Jack E. Bresenham, Winthrop University; Michael Goss, Colorado State University; Bikash Sabata, Wayne State University; and Paul T. Barham, North Carolina State University.

Portions of the book were written while I was on sabbatical working with Dr. Hermann Maurer at the Institute for Information Processing and Computer Supported Media, Technical University Graz, in Graz, Austria, and portions were written while I was on a Fulbright grant at the Indian Institute of Science in Bangalore. I am grateful for the stimulation and support I received during these visits.

Special thanks to my project manager Ana Arias Terry, for her guidance and encouragement during the preparation of the book, and to Irwin Zucker, the production editor, whose expertise and care during production have markedly improved it. Finally, thanks to my parents, to my wife Merilee, and to Greta, Jessie, and Rosy, for all their patience and support while this book slowly took shape.

NOTE TO THE READER: HOW TO VIEW THE STEREO PICTURES

Several stereoscopic figures appear in the book to clarify discussions of 3D situations. They appear as a pair of nearly identical figures placed side by side. To gain the full value of these pictures, coerce your left eye to look at the left-hand picture alone and your right eye to look at the right-hand one alone. This may take practice: Some people catch on quickly, others only after many bleary-eyed attempts, and some people never. Of course, the figures still help to clarify the discussion even without the stereo effect.

One way to practice viewing these figures is to hold the index fingers of each hand upright in front of you, about 2 inches apart, and to stare "through them" at a blank wall in the distance. Each eye, naturally, sees two fingers, but they seem to overlap in the middle. This overlap is precisely what is desired when looking at stereo figures: Each eye sees two fingers, but the "middle ones" are brought into perfect overlap. When the "middle fingers" fuse together like this, the brain constructs a single 3D image out of them. Some people find it helpful to place a piece of white cardboard between the two figures and to rest their nose on it. The cardboard barrier prevents each eye from seeing the image intended for the other eye.

Contents