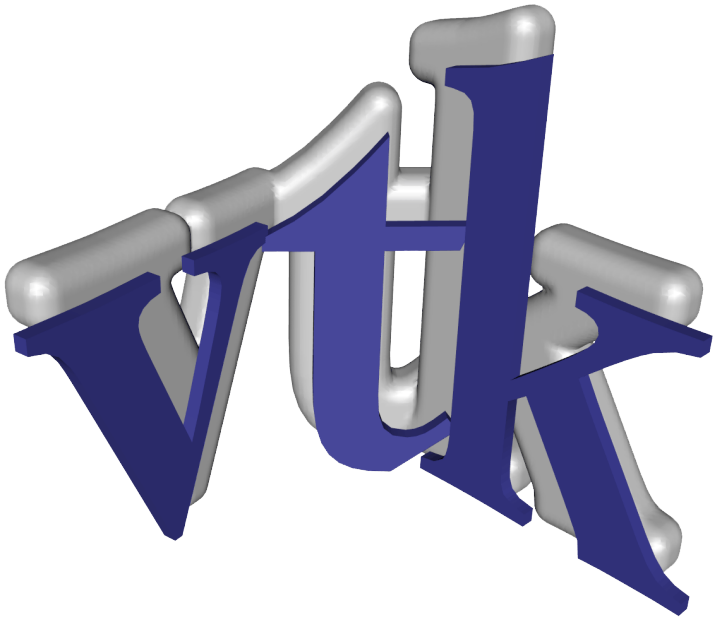
The Visualization Toolkit
An Object-Oriented Approach To 3D Graphics
Fourth Edition



Will Schroeder, Ken Martin, Bill Lorensen [1]

2006 Revised 26 April 2005

[1]with special contributors: Lisa Sobierajski Avila, Rick Avila, C. Charles Law

Contents

Preface

Visualization is a great field to work in these days. Advances in computer hardware and software have brought this technology into the reach of nearly every computer system. Even the ubiquitous personal computer now offers specialized 3D graphics hardware at discount prices. And with recent releases of the Windows operating systems such as XP, OpenGL has become the de facto standard API for 3D graphics.

We view visualization and visual computing as nothing less than a new form of communication. All of us have long known the power of images to convey information, ideas, and feelings. Recent trends have brought us 2D images and graphics as evidenced by the variety of graphical user interfaces and business plotting software. But 3D images have been used sparingly, and often by specialists using specialized systems. Now this is changing. We believe we are entering a new era where 3D images, visualizations, and animations will begin to extend, and in some cases, replace the current communication paradigm based on words, mathematical symbols, and 2D images. Our hope is that along the way the human imagination will be freed like never before.

This text and companion software offers one view of visualization. The field is broad, including elements of computer graphics, imaging, computer science, computational geometry, numerical analysis, statistical methods, data analysis, and studies in human perception. We certainly do not pretend to cover the field in its entirety. However, we feel that this text does offer you a great opportunity to learn about the fundamentals of visualization. Not only can you learn from the written word and companion images, but the included software will allow you to practice visualization. You can start by using the sample data we have provided here, and then move on to your own data and applications. We believe that you will soon appreciate visualization as much as we do.

In this, the third edition of Visualization Toolkit textbook, we have added several new features since the first and second editions. Volume rendering is now extensively supported, including the ability to combine opaque surface graphics with volumes. We have added an extensive image processing pipeline that integrates conventional 3D visualization and graphics with imaging. Besides several new filters such as clipping, smoothing, 2D/3D Delaunay triangulation, and new decimation algorithms, we have added several readers and writers, and better support net-based tools such as Java and VRML. VTK now supports cell attributes, and attributes have been generalized into data arrays that are labeled as being scalars, vectors, and so on. Parallel processing, both shared-memory and distributed models, is a major addition. For example, VTK has been used on a large 1024-processor computer at the US National Labs to process nearly a pit-a-pat of data. A suite of 3D widgets is now available in VTK, enabling powerful data interaction techniques. Finally, VTK's cross-platform support has greatly improved with the addition of CMake—a very nice tool for managing the compile process (CMake).

The additions of these features required the support of three special contributors to the text: Lisa Sobierajski Avila, Rick Avila, and C. Charles Law. Rick and Lisa worked

hard to create an object-oriented design for volume rendering, and to insure that the design and software is fully compatible with the surface-based rendering system. Charles is the principle architect and implementer for the imaging pipeline. We are proud of the streaming and caching capability of the architecture: It allows us to handle large data sets despite limited memory resources.

Especially satisfying has been the response from users of the text and software. Not only have we received a warm welcome from these wonderful people, but many of them have contributed code, bug fixes, data, and ideas that greatly improved the system. In fact, it would be best to categorize these people as co-developers rather than users of the system. We would like to encourage anyone else who is interested in sharing their ideas, code, or data to contact the VTK user community at VTK, or one of the authors. We would very much welcome any contributions you have to make. Contact us at Kitware.

Acknowledgments

During the creation of the Visualization Toolkit we were fortunate to have the help of many people. Without their aid this book and the associated software might never have existed. Their contributions included performing book reviews, discussing software ideas, creating a supportive environment, and providing key suggestions for some of our algorithms and software implementations.

We would like to first thank our management at the General Electric Corporate R&D Center who allowed us to pursue this project and utilize company facilities: Peter Meenan, Manager of the Computer Graphics and Systems Program, and Kirby Vosburgh, Manager of the Electronic Systems Laboratory. We would also like to thank management at GE Medical Systems who worked with us on the public versus proprietary software issues: John Lalonde, John Heinen, and Steve Roehm.

We thank our co-workers at the R&D Center who have all been supportive: Matt Turek, for proof reading much of the second edition; also Majeid Alyassin, Russell Blue, Jeanette Bruno, Shane Chang, Nelson Corby, Rich Hammond, Margaret Kelliher, Tim Kelliher, Joyce Langan, Paul Miller, Chris Nafis, Bob Tatar, Chris Volpe, Boris Yamrom, Bill Hoffman (now at Kitware), Harvey Cline and Siegwalt Ludke. We thank former co-workers Skip Montanaro (who created a FAQ for us), Dan McLachlan and Michelle Barry. We'd also like to thank our friends and co-workers at GE Medical Systems: Ted Hudacko (who managed the first VTK users mailing list), Darin Okerlund, and John Skinner. Many ideas, helpful hints, and suggestions to improve the system came from this delightful group of people.

The third edition is now published by Kitware, Inc. We very much appreciate the efforts of the many contributors at Kitware who have helped make VTK one of the leading visualization systems in the world today. Sébastien Barré, Andy Cedilnik, Berk Geveci, Amy Henderson, and Brad King have each made significant contributions. Thank also to the folks at GE Global Research such as Jim Miller who continue to push the quality of the system, particularly with the creation of the DART system for regression testing. The US National Labs, led by Jim Ahrens of Los Alamos, has been instrumental in adding parallel processing support to VTK. An additional special thanks to Kitware for accepting the challenge of publishing this book.

Many of the bug fixes and improvements found in the second and third editions came from talented people located around the world. Some of these people are acknowledged in the software and elsewhere in the text, but most of them have contributed their time, knowledge, code, and data without regard for recognition and acknowledgment. It is this exchange of ideas and information with people like this that makes the *Visualization Toolkit* such a fun and exciting project to work on. In particular we would like to thank John Biddiscombe, Charl P. Botha, David Gobbi, Tim Hutton, Dean Inglis, and Prabhu Ramachandran. Thank you very much.

A special thanks to the software and text reviewers who spent their own time to track

down some nasty bugs, provide examples, and offer suggestions and improvements. Thank you Tom Citriniti, Mark Miller, George Petras, Hansong Zhang, Penny Rheingans, Paul Hinker, Richard Ellson, and Roger Crawfis. We'd also like to mention that Tom Citriniti at Rensselaer, and Penny Rheingans at the University of Mississippi (now at the University of Maryland Baltimore County) were the first faculty members to teach from early versions of this text. Thank you Penny and Tom for your feedback and extra effort.

Most importantly we would like to thank our friends and loved ones who supported us patiently during this project. We know that you shouldered extra load for us. You certainly saw a lot less of us! But we're happy to say that we're back. Thank you.

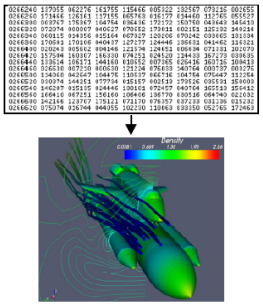


Figure 1.1: Visualization transforms numbers to images.

Visualization — "2: the act or process of interpreting in visual terms or of putting into visual form, Webster's Ninth New Collegiate Dictionary."

1.1 What Is Visualization?

Visualization is a part of our everyday life. From weather maps to the exciting computer graphics of the entertainment industry, examples of visualization abound. But what is visualization? Informally, visualization is the transformation of data or information into pictures. Visualization engages the primary human sensory apparatus, vision, as well as the processing power of the human mind. The result is a simple and effective medium for communicating complex and/or voluminous information

1.1.1 Terminology

Different terminology is used to describe visualization. Scientific visualization is the formal name given to the field in computer science that ncompasses user interface, data representation and processing algorithms, visual representations, and other sensory presentation such as sound or touch [3]. The term data visualization is another phrase used to describe visualization. Data visualization is generally interpreted to be more general than scientific visualization, since it implies treatment of data sources beyond the sciences and engineering. Such data sources include financial, marketing, or business data. In addition, the term data visualization is broad enough to include application of statistical methods and other standard data analysis techniques [2]. Another recently emerging term is information

visualization. This field endeavors to visualize abstract information such as hyper-text documents on the World Wide Web, directory/ file structures on a computer, or abstract data structures [15]. A major challenge facing information visualization researchers is to develop coordinate systems, transformation methods, or structures that meaningfully organize and represent data.

Another way to classify visualization technology is to examine the context in which the data exists. If the data is spatial-temporal in nature (up to three spatial coordinates and the time dimension) then typically methods from scientific visualization are used. If the data exists in higher dimensional spaces, or abstract spaces, then methods from information visualization are used. This distinction is important, because the human perceptual system is highly tuned to space-time relationships. Data expressed in this coordinate system is inherently understood with little need for explanation. Visualization of abstract data typically requires extensive explanations as to what is being viewed. This is not to say that there is no overlap between scientific and information visualization often the first step in the information visualization process is to project abstract data into the spatial-temporal domain, and then use the methods of scientific visualization to view the results. The projection process can be quite complex, involving methods of statistical graphics, data mining, and other techniques, or it may be as simple as selecting a lower-dimensional subset of the original data.

In this text we use the term data visualization instead of the more specific terms scientific visualization or information visualization. We feel that scientific visualization is too narrow a description of the field, since visualization techniques have moved beyond the scientific domain and into areas of business, social science, demographics, and information management in general. We also feel that the term data visualization is broad enough to encompass the term information visualization.

1.1.2 Examples of Visualization

Perhaps the best definition of visualization is offered by example. In many cases visualization is influencing peoples' lives and performing feats that a few years ago would have been unimaginable. A prime example of this is its application to modern medicine.

Computer imaging techniques have become an important diagnostic tool in the practice of modern medicine. These include techniques such as X-ray Computed Tomography (CT) and Magnetic Resonance Imaging (MRI). These techniques use a sampling or data acquisition process to capture information about the internal anatomy of a living patient. This information is in the form of slice-planes or cross-sectional images of a patient, similar to conventional photographic X-rays. CT imaging uses many pencil thin X-rays to acquire the data, while MRI combines large magnetic fields with pulsed radio waves. Sophisticated mathematical techniques are used to reconstruct the slice-planes. Typically, many such closely spaced slices are gathered together into a volume of data to complete the study.

As acquired from the imaging system, a slice is a series of numbers representing the attenuation of X-rays (CT) or the relaxation of nuclear spin magnetization (MRI) [8]. On any given slice these numbers are arranged in a matrix, or regular array. The amount of data is large, so large that it is not possible to understand the data in its raw form. However, by assigning to these numbers a gray scale value, and then displaying the data on a computer screen, structure emerges. This structure results from the interaction of the human visual system with the spatial organization of the data and the gray-scale values we have chosen. What the computer represents as a series of numbers, we see as a cross section through the human body: skin, bone, and muscle. Even more impressive results are possible when we extend these techniques into three dimensions. Image slices can be

gathered into volumes and the volumes can be processed to reveal complete anatomical structures. Using modern techniques, we can view the entire brain, skeletal system, and vascular system on a living patient without interventional surgery. Such capability has revolutionized modern medical diagnostics, and will increase in importance as imaging and visualization technology matures.

Another everyday application of visualization is in the entertainment industry. Movie and television producers routinely use computer graphics and visualization to create entire worlds that we could never visit in our physical bodies. In these cases we are visualizing other worlds as we imagine them, or past worlds we suppose existed. It's hard to watch the movies such as JurassicPark and Toy Story and not gain a deeper appreciation for the awesome Tyrannosaurus Rex, or to be charmed by Toy Story's heroic Buzz Lightyear.

Morphing is another popular visualization technique widely used in the entertainment industry. Morphing is a smooth blending of one object into another. One common application is to morph between two faces. Morphing has also been used effectively to illustrate car design changes from one year to the next. While this may seem like an esoteric application, visualization techniques are used routinely to present the daily weather report. The use of isovalue, or contour, lines to display areas of constant temperature, rainfall, and barometric pressure has become a standard tool in the daily weather report.

Many early uses of visualization were in the engineering and scientific community. From its inception the computer has been used as a tool to simulate physical processes such as ballistic trajectories, fluid flow, and structural mechanics. As the size of the computer simulations grew, it became necessary to transform the resulting calculations into pictures. The amount of data overwhelmed the ability of the human to assimilate and understand it. In fact, pictures were so important that early visualizations were created by manually plotting data. Today, we can take advantage of advances in computer graphics and computer hardware. But, whatever the technology, the application of visualization is the same: to display the results of simulations, experiments, measured data, and fantasy; and to use these pictures to communicate, understand, and entertain.

1.2 Why Visualize?

Visualization is a necessary tool to make sense of the flood of information in today's world of computers. Satellites, supercomputers, laser digitizing systems, and digital data acquisition systems acquire, generate, and transmit data at prodigious rates. The Earth-Orbiting Satellite (EOS) transmits terabytes of data every day. Laser scanning systems generate over 500,000 points in a 15 second scan [19]. Supercomputers model weather patterns over the entire earth [5]. In the first four months of 1995, the New York Stock Exchange processed, on average, 333 million transactions per day [16]. Without visualization, most of this data would sit unseen on computer disks and tapes. Visualization offers some hope that we can extract the important information hidden within the data.

There is another important element to visualization: It takes advantage of the natural abilities of the human vision system. Our vision system is a complex and powerful part of our bodies. We use it and rely on it in almost everything we do. Given the environment in which our ancestors lived, it is not surprising that certain senses developed to help them survive. As we described earlier in the example of a 2D MRI scan, visual representations are easier to work with. Not only do we have strong 2D visual abilities, but also we are adept at integrating different viewpoints and other visual clues into a mental image of a 3D object or plot. This leads to interactive visualization, where we can manipulate our viewpoint. Rotating about the object helps to achieve a better understanding. Likewise, we

have a talent for recognizing temporal changes in an image. Given an animation consisting of hundreds of frames, we have an uncanny ability to recognize trends and spot areas of rapid change.

With the introduction of computers and the ability to generate enormous amounts of data, visualization offers the technology to make the best use of our highly developed visual senses. Certainly other technologies such as statistical analysis, artificial intelligence, mathematical filtering, and sampling theory will play a role in large-scale data processing. However, because visualization directly engages the vision system and human brain, it remains an unequaled technology for understanding and communicating data.

Visualization offers significant financial advantages as well. In today's competitive markets, computer simulation teamed with visualization can reduce product cost and improve time to market. A large cost of product design has been the expense and time required to create and test design prototypes. Current design methods strive to eliminate these physical prototypes, and replace them with digital equivalents. This digital prototyping requires the ability to create and manipulate product geometry, simulate the design under a variety of operating conditions, develop manufacturing techniques, demonstrate product maintenance and service procedures, and even train operators on the proper use of the product before it is built. Visualization plays a role in each case. Already CAD systems are used routinely to model product geometry and design manufacturing procedures. Visualization enables us to view the geometry, and see special characteristics such as surface curvature. For instance, analysis techniques such as finite element, finite difference, and boundary element techniques are used to simulate product performance; and visualization is used to view the results. Recently, human ergonomics and anthropometry are being analyzed using computer techniques in combination with visualization [9]. Three-dimensional graphics and visualization are being used to create training sequences. Often these are incorporated into a hypertext document or World Wide Web (WWW) pages. Another practical use of graphics and visualization has been in-flight simulators. This has been shown to be a significant cost savings as compared to flying real airplanes and is an effective training method.

1.3 Imaging, Computer Graphics, and Visualization

There is confusion surrounding the difference between imaging, computer graphics, and visualization. We offer these definitions.

- Imaging, or image processing, is the study of 2D pictures, or images. This includes techniques to transform (e.g., rotate, scale, shear), extract information from, analyze, and enhance images.

  The resulting data is mapped to a graphics system for display.

- Computer graphics is the process of creating images using a computer. This includes both 2D paint-and-draw techniques as well as more sophisticated 3D drawing (or rendering) techniques.

- Visualization is the process of exploring, transforming, and viewing data as images (or other sensory forms) to gain understanding and insight into the data.

Based on these definitions we see that there is overlap between these fields. The output of computer graphics is an image, while the output of visualization is often produced using computer graphics. Sometimes visualization data is in the form of an image, or we wish to visualize object geometry using realistic rendering techniques from computer graphics.

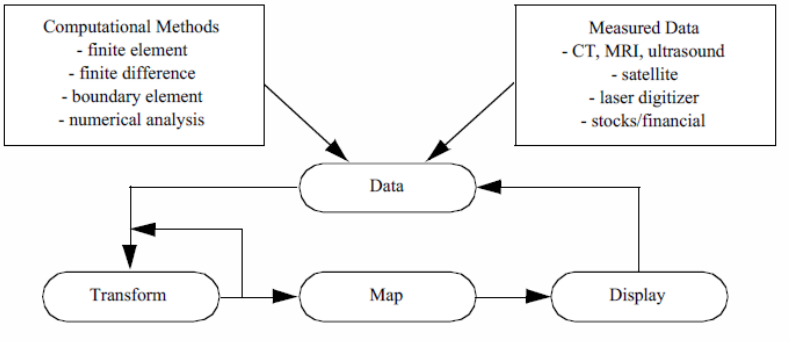


Figure 1.2: The visualization process. Data from various sources is repeatedly transformed to extract, derive, and enhance information. The resulting data is mapped to a graphics system for display.

Generally speaking we distinguish visualization from computer graphics and image processing in three ways.

1. The dimensionality of data is three dimensions or greater. Many well-known methods are available for data of two dimensions or less; visualization serves best when applied to data of higher dimension.

2. Visualization concerns itself with data transformation. That is, information is repeatedly created and modified to enhance the meaning of the data.

3. Visualization is naturally interactive, including the human directly in the process of creating, transforming, and viewing data.

Another perspective is that visualization is an activity that encompasses the process of exploring and understanding data. This includes both imaging and computer graphics as well as data processing and filtering, user interface methodology, computational techniques, and software design. Figure 1.2 depicts this process.

As this figure illustrates we see that the visualization process focuses on data. In the first step data is acquired from some source. Next, the data is transformed by various methods, and then mapped to a form appropriate for presentation to the user. Finally, the data is rendered or displayed, completing the process. Often, the process repeats as the data is better understood or new models are developed. Sometimes the results of the visualization can directly control the generation of the data. This is often referred to as analysis steering. Analysis steering is an important goal of visualization because it enhances the interactivity of the overall process.

1.4 Origins of Data Visualization

The origin of visualization as a formal discipline dates to the 1987 NSF report *Visualization in Scientific Computing* [3]. That report coined the term scientific visualization. Since then the field has grown rapidly with major conferences, such as IEEE Visualization, becoming well established. Many large computer graphics conferences, for example ACM SIGGRAPH, devote large portions of their program to visualization technology.

Of course, data visualization technology had existed for many years before the 1987 report referenced [18]. The first practitioners recognized the value of presenting data as images. Early pictorial data representations were created during the eighteenth century with the arrival of statistical graphics. It was only with the arrival of the digital computer and the development of the field of computer graphics, that visualization became a practicable discipline.

The future of data visualization and graphics appears to be explosive. Just a few decades ago, the field of data visualization did not exist and computer graphics was viewed as an offshoot of the more formal discipline of computer science. As techniques were created and computer power increased, engineers, scientists, and other researchers began to use graphics to understand and communicate data. At the same time, user interface tools were being developed. These forces have now converged to the point where we expect computers to adapt to humans rather than the other way around. As such, computer graphics and data visualization serve as the window into the computer, and more importantly, into the data that computers manipulate. Now, with the visualization window, we can extract information from data and analyze, understand, and manage more complex systems than ever before.

Dr. Fred Brooks, Kenan Professor of Computer Science at the University of North Carolina at Chapel Hill and recipient of the John von Neumann Medal of the IEEE, puts it another way. At the award presentation at the ACM SIGGRAPH '94, Dr. Brooks stated that computer graphics and visualization offer "intelligence amplification" (IA) as compared to artificial intelligence (AI). Besides the deeper philosophical issues surrounding this issue (e.g., human before computer), it is a pragmatic observation. While the long-term goal of AI has been to develop computer systems that could replace humans in certain applications, the lack of real progress in this area has lead some researchers to view the role of computers as amplifiers and assistants to humans. In this view, computer graphics and visualization play a significant role, since arguably the most effective human/ computer interface is visual. Recent gains in computer power and memory are only accelerating this trend, since it is the interface between the human and the computer that often is the obstacle to the effective application of the computer.

1.5 Purpose of This Book

There currently exist texts that define and describe data visualization, many of them using case studies to illustrate techniques and typical applications. Some provide high-level descriptions of algorithms or visualization system architectures. Detailed descriptions are left to academic journals or conference proceedings. What these texts lack is a way to practice visualization. Our aim in this text is to go beyond descriptions and provide tools to learn about and apply visualization to your own application area. In short, the purpose of the book is fourfold.

1. Describe visualization algorithms and architectures in detail.

2. Demonstrate the application of data visualization to a broad selection of case studies.

3. Provide a working architecture and software design for application of data visualization to real-world problems.

4. Provide effective software tools packaged in a C++ class library. We also provide language bindings for the interpreted languages Tcl, Python, and Java.

Taken together, we refer to the text and software as the Visualization Toolkit, or VTK for short. Our hope is that you can use the text to learn about the fundamental concepts of visualization, and then adapt the computer code to your own applications and data.

1.6 What This Book Is Not

The purpose of this book is not to provide a rigorous academic treatise on data visualization. Nor do we intend to include an exhaustive survey of visualization technology. Our goal is to bridge the formal discipline of data visualization with practical application, and to provide a solid technical overview of this emerging technology. In many cases we refer you to the included software to understand implementation details. You may also wish to refer to the appropriate references for further information.

1.7 Intended Audience

Our primary audience is computer users who create, analyze, quantify, and/or process data. We assume a minimal level of programming skill. If you can write simple computer code to import data and know how to run a computer program, you can practice data visualization with the software accompanying this book.

As we wrote this book we also had in mind educators and students of introductory computer graphics and visualization courses. In more advanced courses this text may not be rigorous enough to serve as sole reference. In these instances, this book will serve well as a companion text, and the software is well suited as a foundation for programming projects and class exercises.

Educators and students in other disciplines may also find the text and software to be valuable tools for presenting results. Courses in numerical analysis, computer science, business simulation, chemistry, dynamic systems, and engineering simulations, to name a few, often require large-scale programming projects that create large amounts of data. The software tools provided here are easy to learn and readily adapted to different data sources. Students can incorporate this software into their work to display and analyze their results.

1.8 How to Use This Book

There are a number of approaches you can take to make effective use of this book. The particular approach depends on your skill level and goals. Three likely paths are as follows:

Novice. You're a novice if you lack basic knowledge of graphics, visualization, or object-oriented principles. Start by reading Chapter 2 if you are unfamiliar with object-oriented principles, Chapter 3 if you are unfamiliar with computer graphics, and Chapter 4 if you are unfamiliar with visualization. Continue by reading the application studies in Chapter 12. You can then move on to the CD-ROM and try out some programming examples. Leave the more detailed treatment of algorithms and data representation until you are familiar with the basics and plan to develop your own applications.

Hacker. You're a hacker if you are comfortable writing your own code and editing other's. Review the examples in Chapter 3, Chapter 4, and Chapter 12. At this point you will want to acquire the companion software guide to this text (The VTK User's Guide) or become familiar with the programming resources at VTK. Then retrieve the examples from the CD-ROM and start practicing.

Researcher/Educator. You're a researcher if you develop computer graphics and/or visualization algorithms or if you are actively involved in using and evaluating such systems. You're an educator if you cover aspects of computer graphics and/or visualization within your courses. Start by reading Chapter 2, Chapter 3, and Chapter 4. Select appropriate algorithms from the text and examine the associated source code. If you wish to extend the system, we recommend that you acquire the companion software guide to this text (The VTK User's Guide) or become familiar with the programming resources at VTK.

1.9 Software Considerations and Example Code

In writing this book we have attempted to strike a balance between practice and theory. We did not want the book to become a user manual, yet we did want a strong correspondence between algorithmic presentation and software implementation. (Note: *The VTK User's Guide* published by Kitware, Inc. http://www.kitware.com is recommended as a companion text to this book.) As a result of this philosophy, we have adopted the following approach:

Application versus Design. The book's focus is the application of visualization techniques to real-world problems. We devote less attention to software design issues. Some of these important design issues include: memory management, deriving new classes, shallow versus deep object copy, single versus multiple inheritance, and interfaces to other graphics libraries. Software issues are covered in the companion text The VTK User's Guide published by Kitware, Inc.

Theory versus Implementation. Whenever possible, we separate the theory of data visualization from our implementation of it. We felt that the book would serve best as a reference tool if the theory sections were independent of software issues and terminology. Toward the end of each chapter there are separate implementation or example sections that are implementation specific. Earlier sections are implementation free.

Documentation. This text contains documentation considered essential to understanding the software architecture, including object diagrams and condensed object descriptions. More extensive documentation of object methods and data members is embedded in the software (in the.h header files) and on CD-ROM or online at VTK. In particular, the Doxygen generated manual pages contain detailed descriptions of class relationships, methods, and other attributes.

We use a number of conventions in this text. Imported computer code is denoted with a typewriter font, as are external programs and computer files. To avoid conflict with other C++ class libraries, all class names in VTK begin with the " vtk" prefix. Methods are differentiated from variables with the addition of the " ()" postfix. (Other conventions are listed in *VTK User's Guide*.)

All images in this text have been created using the *Visualization Toolkit* software and data found on the included CD-ROM or from the Web site http://www.vtk.org. In addition, every image has source code (sometimes in C++ and sometimes a Tcl script). We decided against using images from other researchers because we wanted you to be able to practice visualization with every example we present. Each computer generated image indicates the originating file. Files ending in.cxx are C++ code, files ending in.tcl are Tcl scripts. Hopefully these examples can serve as a starting point for you to create your own applications.

To find the example code you will want to search in one of three areas. The standard VTK distribution includes an VTK/Examples directory where many well-documented examples are found. The VTK testing directories VTK/Testing, for example, VTK/Graph-

ics/Testing/Tcl, contain some of the example code used in this text. These examples use the data found in the VTKData distribution. Finally, a separate software distribution, the VTKTextbook distribution, contains examples and data that do not exist in the standard VTK distribution. The VTK, VTKData, and VTKTextbook distributions are found on the included CD-ROM and/or on the web site at VTK.

1.10 Chapter-by-Chapter Overview

Chapter 2: Object-Oriented Design

This chapter discusses some of the problems with developing large and/or complex software systems and describes how object-oriented design addresses many of these problems. This chapter defines the key terms used in object-oriented modelling and design and works through a real-world example. The chapter concludes with a brief look at some object-oriented languages and some of the issues associated with object-oriented visualization.

Chapter 3: Computer Graphics Primer

Computer graphics is the means by which our visualizations are created. This chapter covers the fundamental concepts of computer graphics from an application viewpoint. Common graphical entities such as cameras, lights, and geometric primitives are described along with some of the underlying physical equations that govern lighting and image generation. Issues related to currently available graphics hardware are presented, as they affect how and what we choose to render. Methods for interacting with data are introduced.

Chapter 4: The Visualization Pipeline

This chapter explains our methodology for transforming raw data into a meaningful representation that can than be rendered by the graphics system. We introduce the notion of a visualization pipeline, which is similar to a data flow diagram from software engineering. The differences between process objects and data objects are covered, as well as how we resolved issues between performance and memory usage. We explain the advantages to a pipeline network topology regarding execution ordering, result caching, and reference counting.

Chapter 5: Basic Data Representation

There are many types of data produced by the variety of fields that apply visualization. This chapter describes the data objects that we use to represent and access such data. A flexible design is introduced where the programmer can interact with most any type of data using one consistent interface. The three high level components of data (structure, cells, and data attributes) are introduced, and their specific subclasses and components are discussed.

Chapter 6: Fundamental Algorithms

Where the preceding chapter deals with data objects, this one introduces process objects. These objects encompass the algorithms that transform and manipulate data. This chapter looks at commonly used techniques for isocontour extraction, scalar generation,

color mapping, and vector field display, among others. The emphasis of this chapter is to provide the reader with a basic understanding of the more common and important visualization algorithms.

Chapter 7: Advanced Computer Graphics

This chapter covers advanced topics in computer graphics. The chapter begins by introducing transparency and texture mapping, two topics important to the main thrust of the chapter: volume rendering. Volume rendering is a powerful technique to see inside of 3D objects, and is used to visualize volumetric data. We conclude the chapter with other advanced topics such as stereoscopic rendering, special camera effects, and 3D widgets.

Chapter 8: Advanced Data Representation

Part of the function of a data object is to store the data. The first chapter on data representation discusses this aspect of data objects. This chapter focuses on basic geometric and topological access methods, and computational operations implemented by the various data objects. The chapter covers such methods as coordinate transformations for data sets, interpolation functions, derivative calculations, topological adjacency operations, and geometric operations such as line intersection and searching.

Chapter 9: Advanced Algorithms

This chapter is a continuation of Fundamental Algorithms and covers algorithms that are either more complex or less widely used. Scalar algorithms such as dividing cubes are covered along with vector algorithms such as stream ribbons. A large collection of modelling algorithms is discussed, including triangle strip generation, polygon decimation, feature extraction, and implicit modelling. We conclude with a look at some visualization algorithms that utilize texture mapping.

Chapter 10: Image Processing

While 3D graphics and visualization is the focus of the book, image processing is an important tool for preprocessing and manipulating data. In this chapter we focus on several important image processing algorithms, as well as describe how we use a streaming data representation to process large datasets.

Chapter 11: Visualization on the Web

The Web is one of the best places to share your visualizations. In this chapter we show you how to write Java-based visualization applications, and how to create VRML (Virtual Reality Modelling Language) data files for inclusion in your own Web content.

Chapter 12: Application

In this chapter we tie the previous chapters together by working through a series of case studies from a variety of application areas. For each case, we briefly describe the application and what information we expect to obtain through the use of visualization. Then, we walk through the design and resulting source code to demonstrate the use of the tools described earlier in the text.

1.11 Legal Considerations

We make no warranties, expressly or implied, that the computer code contained in this text is free of error or will meet your requirements for any particular application. Do not use this code in any application where coding errors could result in injury to a person or loss of property. If you do use the code in this way, it is at your own risk. The authors and publisher disclaim all liability for direct or consequential damages resulting from your use of this code.

The computer code contained in this text is copyrighted. We grant permission for you to use, copy, and distribute this software for any purpose. However, you may not modify and then redistribute the software. Some of the algorithms presented here are implementations of patented software. If you plan to use this software for commercial purposes, please insure that applicable patent laws are observed.

Some of the data on the CD-ROM may be freely distributed or used (with appropriate acknowledgment). Refer to the local README files or other documentation for details.

Several registered trademarks are used in this text. UNIX is a trademark of UNIX System Laboratories. Sun Workstation and XGL are trademarks of Sun Microsystems, Inc. Microsoft, MS, MS-DOS, and Windows are trademarks of Microsoft Corporation. The X Window System is a trademark of the Massachusetts Institute of Technology. Starbase and HP are trademarks of Hewlett-Packard Inc. Silicon Graphics and OpenGL, are trademarks of Silicon Graphics, Inc. Mac-intosh is a trademark of Apple Computer. RenderMan is a trademark of Pixar.

1.12 Bibliographic Notes

A number of visualization texts are available. The first six texts listed in the reference section are good general references ([10], [11], [1], [21], [2], and[6]). Gallagher [6] is particularly valuable if you are from a computational background. Wolff and Yaeger [21] contains many beautiful images and is oriented towards Apple Macintosh users. The text includes a CD-ROM with images and software.

You may also wish to learn more about computer graphics and imaging. Foley and van Dam [7] is the basic reference for computer graphics.

Another recommended text is [4]. Suggested reference books on computer imaging are [12] and [20].

Two texts by Tufte [18] [17] are particularly impressive. Not only are the graphics superbly done, but the fundamental philosophy of data visualization is articulated. He also describes the essence of good and bad visualization techniques.

Another interesting text is available from Siemens, a large company offering medical imaging systems [8]. This text describes the basic concepts of imaging technology, including MRI and CT. This text is only for those users with a strong mathematical background. A less mathematical overview of MRI is available from [14].

To learn more about programming with Visualization Toolkit, we recommend the text The VTK User's Guide [13]. This text has an extensive example suite as well as descriptions of the internals of the software. Programming resources including a detailed description of API's, VTK file formats, and class descriptions are provided.

Bibliography

[1] K. W. Brodlie et al. Scientific Visualization Techniques and Applications. Springer-Verlag, Berlin, 1992.

[2] L. Rosenblum et al. Scientific Visualization Advances and Challenges. Harcourt Brace & Company, London, 1994.

[3] B. H. McCormick, T. A. DeFanti, and M. D. Brown. Visualization in Scientific Computing. Report of the NSF Advisory Panel on Graphics, Image Processing and Workstations, 1987.

[4] P. Burger and D. Gillies. Interactive Computer Graphics Functional, Procedural and Device-Level Methods. Addison-Wesley Publishing Company, Reading, MA, 1989.

[5] P. C. Chen. "A Climate Simulation Case Study". In: Proceedings of Visualization '93. IEEE Computer Society Press, Los Alamitos, 1993, pp. 391–401.

[6] R. S. Gallagher, ed. Computer Visualization Graphics Techniques for Scientific and Engineering Analysis. CRC Press, Boca Raton, FL, 1995.

[7] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. Computer Graphics Principles and Practice (2d Ed). Addison-Wesley, Reading, MA, 1990.

[8] E. Krestel, ed. Imaging Systems for Medical Diagnostics. Siemens-Aktienges, Munich, 1990.

[9] McDonnell Douglas Human Modeling System Reference Manual. Version 2.1. Report MDC 93K0281. Mc-Donnell Douglas Corporation, Human Factors Technology. July 1993.

[10] G. M. Nielson and B. Shriver, eds. Visualization in Scientific Computing. IEEE Computer Society Press, Los Alamitos, 1990.

[11] N. M. Patrikalakis, ed. Scientific Visualization of Physical Phenomena. Springer-Verlag, Berlin, 1991.

[12] T. Pavlidis. Graphics and Image Processing. Computer Science Press, Rockville, MD, 1982.

[13] Will Schroeder, Ken Martin, and Bill Lorensen. The VTK User's Guide. Ed. by W. Schroeder. Kitware, 2006. isbn: 1-930934-19-X. url: https://www.amazon.com/Visualization-Toolkit-Object-Oriented-Approach-Graphics/dp/193093419X?SubscriptionId=0JYN1NVW651KCA56C102 & tag = techkie - 20 & linkCode = xm2 & camp = 2025 & creative=165953&creativeASIN=193093419X.

[14] H. J. Smith and F. N. Ranallo. A Non-Mathematical Approach to Basic MRI. Medical Physics Publishing Corporation, Madison, WI, 1989.

[15] The First Information Visualization Symposium. IEEE Computer Society Press, 1995.

[16] The New York Times Business Day, Tuesday, May 2 (1990).

[17] E. R. Tufte. Envisioning Information. Graphics Press, Cheshire, CT, 1999.

[18] E. R. Tufte. The Visual Display of Quantitative Information. Graphics Press, Cheshire, CT, 1999.

[19] K. Waters and D. Terzopoulos. “Modeling and Animating Faces Using Scanned Data”. In: Visualization and Computer Animation. 2. 1991, pp. 123–128.

[20] G. Wolberg. Digital Image Warping. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[21] R. S. Wolff and L. Yaeger. Visualization of Natural Phenomena TELOS. Springer-Verlag, Santa Clara, CA, 1993.

2.0

Object-Oriented Design

Object-oriented systems are becoming widespread in the computer industry for good reason.
Object-oriented systems are more modular, easier to maintain, and easier to describe than traditional procedural systems. Since the Visualization Toolkit has been designed and implemented using object-oriented design, we devote this chapter to summarizing the concepts and practice of object-oriented design and implementation.

2.1 Introduction

Today's software systems try to solve complex, real-world problems. A rigorous software design and implementation methodology can ease the burden of this complexity. Without such a methodology, software developers can find it difficult to meet a system's specifications. Furthermore, as specifications change and grow, a software system that does not have a solid, underlying architecture and design will have difficulty adapting to these expanding requirements.

2.2 Bibliographic Notes

There are several excellent textbooks on object-oriented design. Both [12] and [10] present language-independent design methodologies. Both books emphasize modelling and diagramming as key aspects of design. [15] also describes the OO design process in the context of Eiffel, an OO language. Another popular book has been authored by Booch [6].

Anyone who wants to be a serious user of object-oriented design and implementation should read the books on Smalltalk [4] [11] by the developers of Smalltalk at Xerox Parc. In another early object-oriented programming book, [7] describes OO techniques and the programming language Objective-C. Objective-C is a mix of C and Smalltalk and was used by Next Computer in the implementation of their operating system and user interface.

There are many texts on object-oriented languages. CLOS [14] describes the Common List Object System. Eiffel, a strongly typed OO language is described by [15] . Objective-C [7] is a weakly typed language.

Since C++ has become a popular programming language, there now many class libraries available for use in applications. [13] describes an extensive class library for collections and arrays modeled after the Smalltalk classes described in [4] . [18] and [16] describe the Standard Template Library, a framework of data structures and algorithms that is now a part of the ANSI C++ standard. Open Inventor [1] is a C++ library supporting interactive 3D computer graphics. The Insight Segmentation and Registration Toolkit (ITK)

is a relatively newclass library often used in combination with VTK [21] for medical data processing. VXL is a C++library for computer vision research and implementation [22]. Several mathematical libraries such as VNL (a part of VXL) and Blitz++ [2] are also available. A wide variety of other C++ toolkits are available, Google searches [3] are the best way to find them.

C++ texts abound. The original description by the author of C++ [20] is a must for any serious C++ programmer. Another book [8] describes standard extensions to the language. These days the UML book series—of which [9] and [19] are quite popular—are highly recommended resources. Several books on generic programming [5] and STL [17] are also useful. Check with your colleagues for their favourite C++ book. To keep in touch with new developments there are conferences, journals, and Web sites. The strongest technical conference on object-oriented topics is the annual Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) conference. This is where researchers in the field describe, teach and debate the latest techniques in object-oriented technology. The bimonthly Journal of Object-Oriented Programming (JOOP) published by SIGS Publications, NY, presents technical papers, columns, and tutorials on the field. Resources on the World Wide Web include the Usenet newsgroups comp.object and comp.lang.c++ .

Bibliography

[1] url: http://oss.sgi.com/projects/inventor/.

[2] url: http://www.oonumerics.org/blitz/.

[3] url: https://www.google.com/.

[4] D. Robson A. Goldberg. Smalltalk-80: The Language and Its Implementation. Addison-Wesley, Reading, MA, 1983.

[5] M. H. Austern. Generic Programming and the STL. Addison-Wesley, 1999.

[6] G. Booch. Object-Oriented Design with Applications. Benjamin/Cummings Publishing Co., Redwood City, CA, 1991.

[7] B. J. Cox. Object-Oriented Programming: An Evolutionary Approach. Addison-Wesley, Reading, MA, 1986.

[8] M. Ellis and B. Stroustrup. The Annotated C++ Reference Manual. Addison-Wesley, Reading, MA, 1990.

[9] J. Rumbaugh G. Booch I. Jacobson. The Unified Modeling Language User Guide (Addison-Wesley Object Technology Series). Addison-Wesley Professional, 1998. isbn: 0201571684. url: https://www.amazon.com/Unified-Modeling-Language-Addison-Wesley-Technology/dp/0201571684?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0201571684.

[10] G. M. Birtwistle, O. Dahl, B. Myhrhaug, and K. Nygaard. Simula Begin. Chartwell-Bratt Ltd, England. 1979.

[11] A. Goldberg. Smalltalk-80: The Interactive Programming Environment. Addison-Wesley, Reading, MA, 1984.

[12] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, NJ, 1991.

[13] S. Orlow K. Gorlen and P. Plexico. Data Abstraction and Object-Oriented Programming. John Wiley & Sons, Ltd., Chichester, England, 1990.

[14] S. Keene. Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS. Addison-Wesley, Reading, MA, 1989.

[15] B. Meyer. Object-Oriented Software Construction. Prentice Hall International, Hertfordshire, England, 1988.

[16] D. Musser and A. Stepanov. "Algorithm-Oriented Generic Libraries". In: Software Practice and Experience 24.7 (July 1994), pp. 623–642.

[17] David R. Musser and Atul Saini. Stl Tutorial & Reference Guide: C++ Programming With the Standard Template Library (Addison-Wesley Professional Computing Series). Addison-Wesley, 1996. isbn: 0201633981. url: https://www.amazon.com/Stl-Tutorial-Reference-Guide-Addison-Wesley/dp/0201633981?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0201633981.

[18] P.J. Plauger et al. The C++ Standard Template Library. Prentice Hall, 2000. isbn: 978-0134376332. url: https://www.amazon.com/C-Standard-Template-Library/dp/0134376331?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0134376331.

[19] James Rumbaugh, Ivar Jacobson, and Grady Booch. The Unified Modeling Language Reference Manual. Addison-Wesley Professional. isbn: 020130998X. url: https://www.amazon.com/Unified-Modeling-Language-Reference-Manual/dp/020130998X?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=020130998X.

[20] Bjarne Stroustrup. The C++ Programming Language: Special Edition (3rd Edition). Addison-Wesley Professional, 2000. isbn: 0-201-70073-5. url: https://www.amazon.com/Programming-Language-Special-3rd/dp/0201700735?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0201700735.

[21] The Insight Software Consortium. url: https://www.itk.org.

[22] VXL. url: http://vxl.sourceforge.net/.

Computer graphics is the foundation of data visualization. Practically speaking, visualization is the process that transforms data into a set of graphics primitives. The methods of computer graphics are then used to convert these primitives into pictures or animations. This chapter discusses basic computer graphics principles. We begin by describing how lights and physical objects interact to form what we see. Next we examine how to simulate these interactions using computer graphics techniques. Hardware issues play an important role here since modern computers have built-in hardware support for graphics. The chapter concludes with a series of examples that illustrate our object-oriented model for 3D computer graphics.

3.1 Introduction

Computer graphics is the process of generating images using computers. We call this process rendering. There are many types of rendering processes, ranging from 2D paint programs to sophisticated 3D techniques. In this chapter we focus on basic 3D techniques for visualization.

We can view rendering as the process of converting graphical data into an image. In data visualization our goal is to transform data into graphical data, or graphics primitives, that are then rendered. The goal of our rendering is not so much photo realism as it is information content. We also strive for interactive graphical displays with which it is possible to directly manipulate the underlying data. This chapter explains the process of rendering an image from graphical data. We begin by looking at the way lights, cameras, and objects (or actors) interact in the world around us. From this foundation we explain how to simulate this process on a computer.

3.2 A Physical Description of Rendering

Figure 3.1 presents a simplified view of what happens when we look at an object, in this case a cube. Rays of light are emitted from a light source in all directions. (In this example we assume that the light source is the sun.) Some of these rays happen to strike the cube whose surface absorbs some of the incident light and reflects the rest of it. Some of this reflected light may head towards us and enter our eyes. If this happens, then we ”see” the object. Likewise, some of the light from the sun will strike the ground and some small percentage of it will be reflected into our eyes.

As you can imagine, the chances of a ray of light travelling from the sun through space to hit a small object on a relatively small planet are low. This is compounded by the

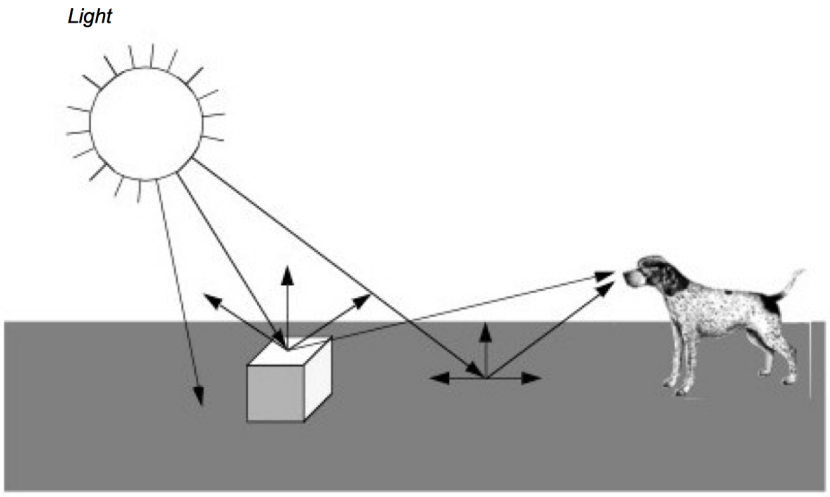


Figure 3.1: Physical generation of an image.

slim odds that the ray of light will reflect off the object and into our eyes. The only reason we can see is that the sun produces such an enormous amount of light that it overwhelms the odds. While this may work in real life, trying to simulate it with a computer can be difficult. Fortunately, there are other ways to look at this problem.

A common and effective technique for 3D computer graphics is called ray-tracing or ray-casting. Ray-tracing simulates the interaction of light with objects by following the path of each light ray. Typically, we follow the ray backwards from the viewer's eyes and into the world to determine what the ray strikes. The direction of the ray is in the direction we are looking (i.e., the view direction) including effects of perspective (if desired). When a ray intersects an object, we can determine if that point is being lit by our light source. This is done by tracing a ray from the point of intersection towards the light. If the ray intersects the light, then the point is being lit. If the ray intersects something else before it gets to the light, then that light will not contribute to illuminating the point. For multiple light sources we just repeat this process for each light source. The total contributions from all the light sources, plus any ambient scattered light, will determine the total lighting or shadow for that point. By following the light's path backwards, ray tracing only looks at rays that end up entering the viewer's eyes. This dramatically reduces the number of rays that must be computed by a simulation program.

Having described ray tracing as a rendering process, it may be surprising that many members of the graphics community do not use it. This is because ray tracing is a relatively slow image generation method since it is typically implemented in software. Other graphics techniques have been developed that generate images using dedicated computer hardware. To understand why this situation has emerged, it is instructive to briefly examine the taxonomy and history of computer graphics.

3.2.1 Image-Order and Object-Order Methods

Rendering processes can be broken into two categories: image-order and object-order. Ray tracing is an image-order process. It works by determining what happens to each ray of light, one at a time. An object-order process works by rendering each object, one at a

time. In the above example, an object-order technique would proceed by first rendering the ground and then the cube.

To look at it another way consider painting a picture of a barn. Using an image-order algorithm you would start at the upper left corner of the canvas and put down a drop of the correct color paint. (Each paint drop is called a picture element or pixel.) Then you would move a little to the right and put down another drop of paint. You would continue until you reached the right edge of the canvas, then you would move down a little and start on the next row. Each time you put down a drop of paint you make certain it is the correct color for each pixel on the canvas. When you are done you will have a painting of a barn.

An alternative approach is based on the more natural (at least for many people) object-order process. We work by painting the different objects in our scene, independent of where the objects actually are located on the scene. We may paint from back to front, front-to-back, or in arbitrary order. For example, we could start by painting the sky and then add in the ground. After these two objects were painted we would then add in the barn. In the image-order process we worked on the canvas in a very orderly fashion; left to right, top to bottom. With an object-order process we tend to jump from one part of the canvas to another, depending on what object we are drawing.

The field of computer graphics started out using object-order processes. Much of the early work was closely tied to the hardware display device, initially a vector display. This was little more than an oscilloscope, but it encouraged graphical data to be drawn as a series of line segments. As the original vector displays gave way to the currently ubiquitous raster displays, the notion of representing graphical data as a series of objects to be drawn was preserved. Much of the early work pioneered by Bresenham [1] at IBM focused on how to properly convert line segments into a form that would be suitable for line plotters. The same work was applied to the task of rendering lines onto the raster displays that replaced the oscilloscope. Since then the hardware has become more powerful and capable of displaying much more complex primitives than lines.

It wasn't until the early 1980s that a paper by Turner Whitted [6] prompted many people to look at rendering from a more physical perspective. Eventually ray tracing became a serious competitor to the traditional object-order rendering techniques, due in part to the highly realistic images it can produce. Object-order rendering has maintained its popularity because there is a wealth of graphics hardware designed to quickly render objects. Ray tracing tends to be done without any specialized hardware and therefore is a time-consuming process.

3.2.2 Surface versus Volume Rendering

The discussion to this point in the text has tacitly assumed that when we render an object, we are viewing the surfaces of objects and their interactions with light. However, common objects such as clouds, water, and fog, are translucent, or scatter light that passes through them. Such objects cannot be rendered using a model based exclusively on surface interactions. Instead, we need to consider the changing properties inside the object to properly render them. We refer to these two rendering models as surface rendering (i.e., render the surfaces of an object) and volume rendering (i.e., render the surface and interior of an object).

Generally speaking, when we render an object using surface rendering techniques, we mathematically model the object with a surface description such as points, lines, triangles, polygons, or 2D and 3D splines. The interior of the object is not described, or only implicitly represented from the surface representation (i.e., surface is the boundary of the volume). Although techniques do exist that allow us to make the surface transparent or translucent,

there are still many phenomena that cannot be simulated using surface rendering techniques alone (e.g., scattering or light emission). This is particularly true if we are trying to render data interior to an object, such as X-ray intensity from a CT scan.

Volume rendering techniques allow us to see the inhomogeneity inside objects. In the prior CT example, we can realistically reproduce X-ray images by considering the intensity values from both the surface and interior of the data. Although it is premature to describe this process at this point in the text, you can imagine extending our ray tracing example from the previous section. Thus rays not only interact with the surface of an object, they also interact with the interior.

In this chapter we focus on surface rendering techniques. While not as powerful as volume rendering, surface rendering is widely used because it is relatively fast compared to volumetric techniques, and allows us to create images for a wide variety of data and objects. Chapter 7 describes volume rendering in more detail.

3.2.3 Visualization Not Graphics

Although the authors would enjoy providing a thorough treatise on computer graphics, such a discourse is beyond the scope of this text. Instead we make the distinction between visualization (exploring, transforming, and mapping data) and computer graphics (mapping and rendering). The focus will be on the principles and practice of visualization, and not on 3D computer graphics. In this chapter and Chapter 7 we introduce basic concepts and provide a working knowledge of 3D computer graphics. For those more interested in this field, we refer you to the texts recommended in the "Bibliographic Notes" on page 42 at the end of this chapter.

One of the regrets we have regarding this posture is that certain rendering techniques are essentially visualization techniques. We see this hinted at in the previous paragraph, where we use the term "mapping" to describe both visualization and computer graphics. There is not currently and will likely never be a firm distinction between visualization and graphics. For example, many researchers consider volume rendering to be squarely in the field of visualization because it addresses one of the most important forms of visualization data. Our distinction is mostly for our own convenience, and offers us the opportunity to finish this text. We recommend that a serious student of visualization supplement the material presented here with deeper books on computer graphics and volume rendering.

In the next few pages we describe the rendering process in more detail. We start by describing several color models. Next we examine the primary components of the rendering process. There are sources of light such as the sun, objects we wish to render such as a cube or sphere (we refer to these objects as actors), and there is a camera that looks out into the world. These terms are taken from the movie industry and tend to be familiar to most people. Actors represent graphical data or objects, lights illuminate the actors, and the camera constructs a picture by projecting the actors onto a view plane. We call the combination of lights, camera, and actors the scene, and refer to the rendering process as rendering the scene.

3.3 Color

The electromagnetic spectrum visible to humans contains wavelengths ranging from about 400 to 700 nanometers. The light that enters our eyes consists of different intensities of these wavelengths, an example of which is shown in Figure 3.2. This intensity plot defines the color of the light, therefore a different plot results in a different color. Unfortunately,

we may not notice the difference since the human eye throws out most of this information. There are three types of color receptors in the human eye called cones. Each type responds to a subset of the 400 to 700 nanometer wave-length range as shown in Figure 3.3. Any color we see is encoded by our eyes into these three overlapping responses. This is a great reduction from the amount of information that actually comes into our eyes. As a result, the human eye is incapable of recognizing differences in any colors whose intensity curves, when applied to the human eye's response curves, result in the same triplet of responses. This also implies that we can store and represent colors in a computer using a simplified form without the human eye being able to recognize the difference.

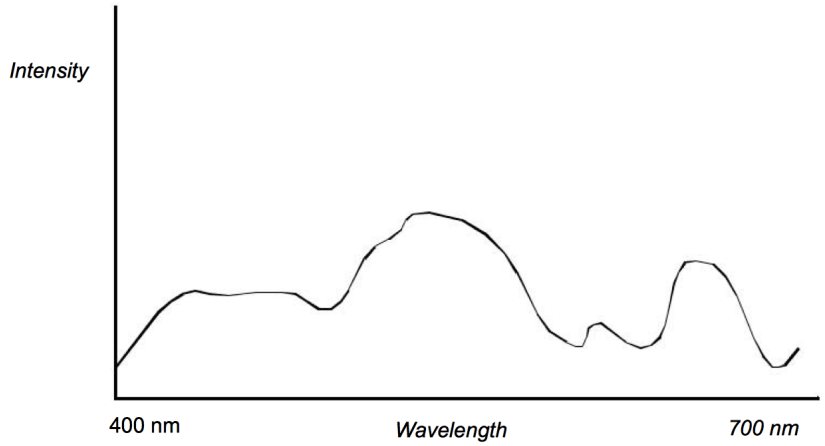


Figure 3.2: Wavelength versus Intensity plot.

The two simplified component systems that we use to describe colors are RGB and HSV color systems. The RGB system represents colors based on their red, green, and blue intensities. This can be thought of as a three dimensional space with the axes being red, green, and blue. Some common colors and their RGB components are shown in Table 3.1.

The HSV system represents colors based on their hue, saturation, and value. The value component is also known as the brightness or intensity component, and represents how much light is in the color. A value of 0.0 will always give you black and a value of 1.0 will give you something bright. The hue represents the dominant wavelength of the color and is often illustrated using a circle as in Figure 3.4. Each location on the circumference of this circle represents a different hue and can be specified using an angle. When we specify a hue we use the range from zero to one, where zero corresponds to zero degrees on the hue circle and one corresponds to 360 degrees. The saturation indicates how much of the hue is mixed into the color. For example, we can set the value to one, which gives us a bright color, and the hue to 0.66, to give us a dominant wavelength of blue. Now if we set the saturation to one, the color will be a bright primary blue. If we set the saturation to 0.5, the color will be sky blue, a blue with more white mixed in. If we set the saturation to zero, this indicates that there is no more of the dominant wavelength (hue) in the color than any other wavelength. As a result, the final color will be white (regardless of hue value). Table 3.1 lists HSV values for some common colors.

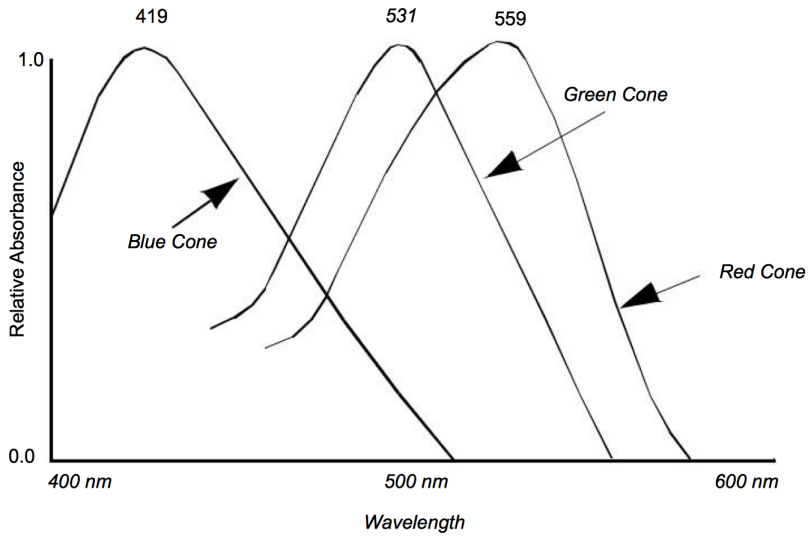


Figure 3.3: Relative absorbance of light by the three types of cones in the human retina.

3.4 Lights

One of the major factors controlling the rendering process is the interaction of light with the actors in the scene. If there are no lights, the resulting image will be black and rather uninformative. To a great extent it is the interaction between the emitted light and the surface (and in some cases the interior) of the actors in the scene that defines what we see. Once rays of light interact with the actors in a scene, we have something for our camera to view.

Of the many different types of lights used in computer graphics, we will discuss the simplest, the infinitely distant, point light source. This is a simplified model compared to the lights we use at home and work. The light sources that we are accustomed to typically radiate from a region in space (a filament in an incandescent bulb, or a light-emitting gas

Color	RGB	HSV
Black	0, 0, 0	$*, *, 0$
White	1, 1, 1	$*, 0, 1$
Red	1, 0, 0	0, 1, 1
Green	0, 1, 0	1/3, 1, 1
Blue	0, 0, 1	2/3, 1, 1
Cyan	0, 1, 1	1/2, 1, 1
Magenta	1, 0, 1	5/6, 1, 1
Yellow	1, 1, 0	1/6, 1, 1
Sky Blue	1/2, 1/2, 1	2/3, 1/2, 1

Table 3.1: Common colors in RGB and HSV space

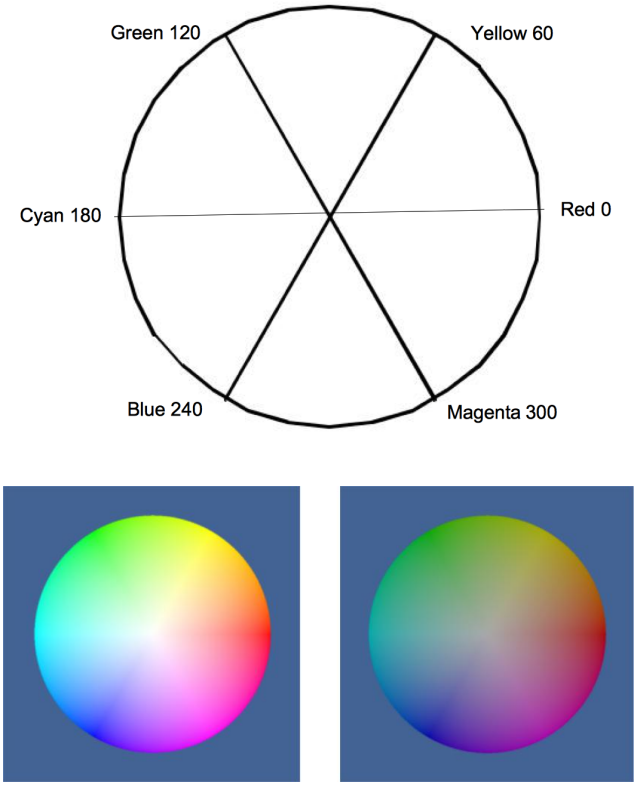


Figure 3.4: On the top, circular representation of hue. The other two images on the bottom are slices through the HSV color space. The first slice has a value of 1.0, the other has a value of 0.5.

in a fluorescent light). The point source lighting model assumes that the light is emitted in all directions from a single point in space. For an infinite light source, we assume that it is positioned infinitely far away from what it is illuminating. This is significant because it implies that the incoming rays from such a source will be parallel to each other. The emissions of a local light source, such as a lamp in a room, are not parallel. Figure 3.5 illustrates the differences between a local light source with a finite volume, versus an infinite point light source. The intensity of the light emitted by our infinite light sources also remains constant as it travels, in contrast to the actual $1/distance^2$ relationship physical lights obey. As you can see this is a great simplification, which later will allow us to use less complex lighting equations.

3.5 Surface Properties

As rays of light travel through space, some of them intersect our actors. When this happens, the rays of light interact with the surface of the actor to produce a color. Part of this resulting color is actually not due to direct light, but rather from ambient light that is being reflected or scattered from other objects. An ambient lighting model accounts for

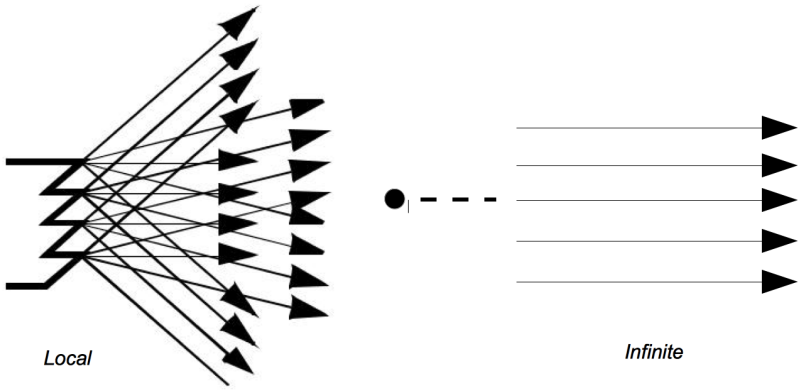


Figure 3.5: Local light source with a finite volume versus an infinite point light source.

this and is a simple approximation of the complex scattering of light that occurs in the real world. It applies the intensity curve of the light source to the color of the object, also expressed as an intensity curve. The result is the color of the light we see when we look at that object. With such a model, it is important to realize that a white light shining on a blue ball is indistinguishable from a blue light shining on a white ball. The ambient lighting equation is

$$R_a = L_c \cdot O_a \tag{3.1}$$

where R_a is the resulting intensity curve due to ambient lighting, L_c is the intensity curve of the ambient light, and O_a is the color curve of the object. To help keep the equations simple we assume that all of the direction vectors are normalized (i.e., have a magnitude of one).

Two components of the resulting color depend on direct lighting. Diffuse lighting, which is also known as Lambertian reflection, takes into account the angle of incidence of the light onto an object. Figure 3.6 shows the image of a cylinder that becomes darker as you move laterally from its center. The cylinder's color is constant; the amount of light hitting the surface of the cylinder changes. At the center, where the incoming light is nearly perpendicular to the surface of the cylinder, it receives more rays of light per surface area. As we move towards the side, this drops until finally the incoming light is parallel to the side of the cylinder and the resulting intensity is zero.

The contribution from diffuse lighting is expressed in Equation 3.2 and illustrated in Figure 3.7.

$$R_d = L_c O_d[\overrightarrow{O}_n \cdot (-\overrightarrow{L}_n)] \tag{3.2}$$

where R_d is the resulting intensity curve due to diffuse lighting, L_c is the intensity curve for the light, and O_c is the color curve for the object. Notice that the diffuse light is a function of the relative angle between incident light vector and $\overrightarrow{L}_n$ and the surface normal of the object $\overrightarrow{O}_n$. As a result diffuse lighting is independent of viewer position.

Specular lighting represents direct reflections of a light source off a shiny object. Figure 3.9 shows a diffusely lit ball with varying specular reflection. The specular intensity (which varies between the top and bottom rows) controls the intensity of the specular

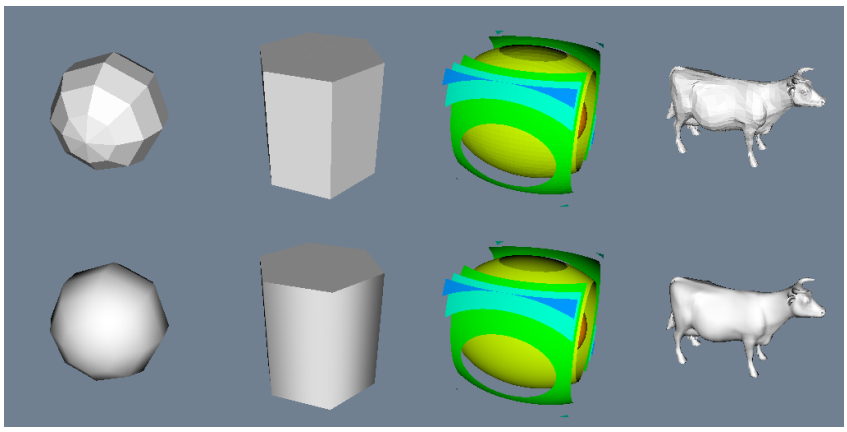


Figure 3.6: Flat and Gouraud shading. Different shading methods can dramatically improve the look of an object represented with polygons. On the top, flat shading uses a constant surface normal across each polygon. On the bottom, Gouraud shading interpolates normals from polygon vertices to give a smoother look.

lighting. The specular power, O_sp, indicates how shiny an object is, more specifically it indicates how quickly specular reflections diminish as the reflection angles deviate from a perfect reflection. Higher values indicate a faster drop off, and therefore a shinier surface. Referring to Figure 3.8, the equation for specular lighting is

$$R_s. = L_c O_s [\vec{S} \cdot (-\vec{C}_n)]^{O_{sp}} \vec{S} = 2[\vec{O}_n \cdot (-\vec{L}_n)]\vec{O}_n + \vec{L}_n \tag{3.3}$$

where $\vec{C}_n$ is the direction of projection for the camera and $\vec{S}$ is the direction of specular reflection.

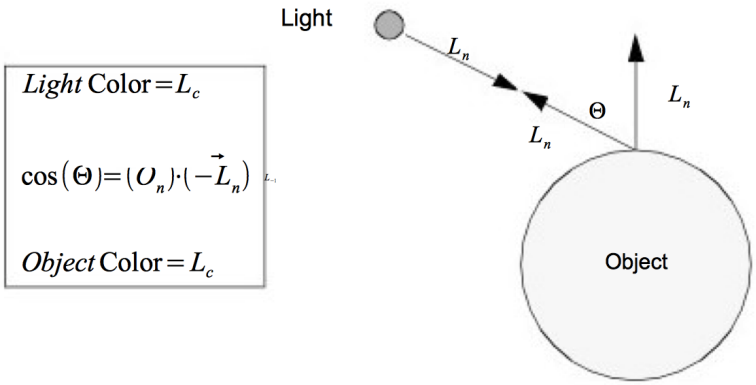


Figure 3.7: Diffuse lighting

We have presented the equations for the different lighting models independently. We can apply all lighting models simultaneously or in combination. Equation 3.4 combines ambient, diffuse and specular lighting into one equation.

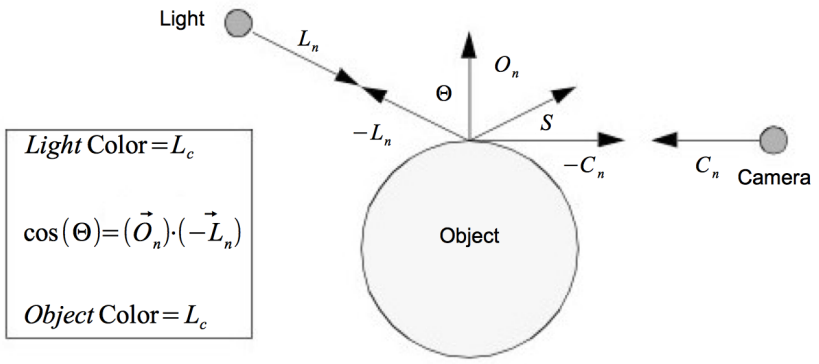


Figure 3.8: Specular lighting.

$$R_c = O_{ai}O_{ac}L_c - O_{di}O_{dc}L_c(\vec{O}_n \cdot \vec{L}_n) + O_{si}O_{sc}L_c[\vec{S} \cdot (-\vec{C}_n)]^{O_{sp}} \tag{3.4}$$

The result is a color at a point on the surface of the object. The constants O_{ai}, O_{di}, and O_{si} control the relative amounts of ambient, diffuse and specular lighting for an object. The constants O_{ac} O_{dc} and O_{sc} specify the colors to be used for each type of lighting. These six constants along with the specular power are part of the surface material properties. (Other properties such as transparency will be covered in later sections of the text.) Different combinations of these property values can simulate dull plastic and polished metal. The equation assumes an infinite point light source as described in "Lights" on 3.4. However the equation can be easily modified to incorporate other types of directional lighting.

3.6 Cameras

We have light sources that are emitting rays of light and actors with surface properties. At every point on the surface of our actors this interaction results in some composite color (i.e., combined color from light, object surface, specular, and ambient effects). All we need now to render the scene is a camera. There are a number of important factors that determine how a 3D scene gets projected onto a plane to form a 2D image (see Figure 3.10). These are the position, orientation, and focal point of the camera, the method of camera projection , and the location of the camera clipping planes.

The position and focal point of the camera define the location of the camera and where it points. The vector defined from the camera position to the focal point is called the direction of projection . The camera image plane is located at the focal point and is typically perpendicular to the projection vector. The camera orientation is controlled by the position and focal point plus the camera view-up vector. Together these completely define the camera view.

The method of projection controls how the actors are mapped to the image plane. Orthographic projection is a parallel mapping process. In orthographic projection (or parallel projection) all rays of light entering the camera are parallel to the projection vector. Perspective projection occurs when all light rays go through a common point (i.e., the viewpoint or center of projection). To apply perspective projection we must specify a perspective angle or camera view angle.

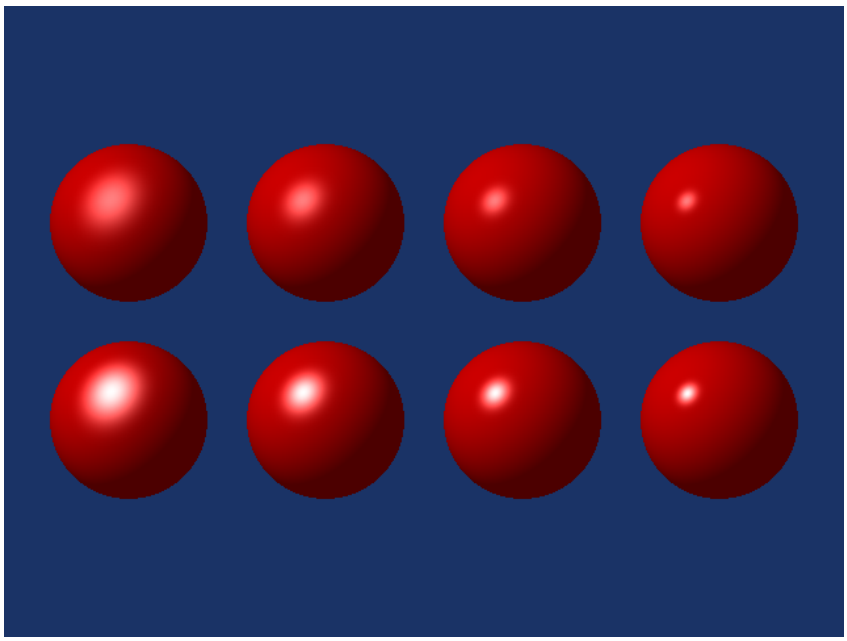


Figure 3.9: Effects of specular coefficients. Specular coefficients control the apparent "shininess" of objects. The top row has a specular intensity value of 0.5; the bottom row 1.0. Along the horizontal direction the specular power changes. The values (from left to right) are 5, 10, 20, and 40.

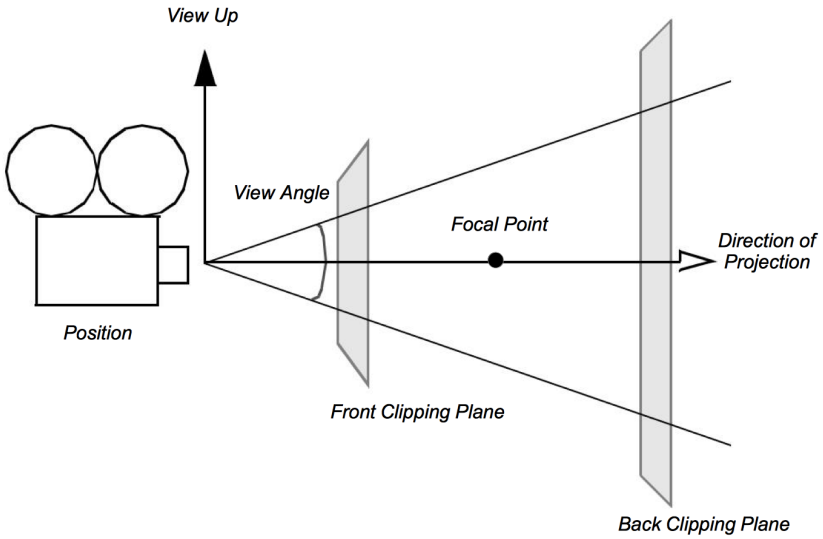


Figure 3.10: Camera attributes.

The front and back clipping planes intersect the projection vector, and are usually perpendicular to it. The clipping planes are used to eliminate data either too close to the camera or too far away. As a result only actors or portions of actors within the clipping

planes are (potentially) visible. Clipping planes are typically perpendicular to the direction of projection. Their locations can be set using the camera's clipping range. The location of the planes are measured from the camera's position along the direction of projection. The front clipping plane is at the minimum range value, and the back clipping plane is at the maximum range value. Later on in Chapter 7 , when we discuss stereo rendering, we will see examples of clipping planes that are not perpendicular to the direction of projection.

Taken together these camera parameters define a rectangular pyramid, with its apex at the camera's position and extending along the direction of projection. The pyramid is truncated at the top with the front clipping plane and at the bottom by the back clipping plane. The resulting view frustum defines the region of 3D space visible to the camera.

While a camera can be manipulated by directly setting the attributes mentioned above, there are some common operations that make the job easier. Figure 3.11 and Figure 3.12 will help illustrate these operations. Changing the azimuth of a camera rotates its position around its view up vector, centered at the focal point . Think of this as moving the camera to the left or right while always keeping the distance to the focal point constant. Changing a camera's elevation rotates its position around the cross product of its direction of projection and view up centered at the focal point. This corresponds to moving the camera up and down. To roll the camera, we rotate the view up vector about the view plane normal. Roll is sometimes called twist.

The next two motions keep the camera's position constant and instead modify the focal point. Changing the yaw rotates the focal point about the view up centered at the camera's position. This is like an azimuth, except that the focal point moves instead of the position. Changes in pitch rotate the focal point about the cross product of the direction of projection and view up centered at the camera's position. Dollying in and out moves the camera's position along the direction of projection, either closer or farther from the focal point. This operation is specified as the ratio of its current distance to its new distance. A value greater than one will dolly in, while a value less than one will dolly out. Finally, zooming changes the camera's view angle, so that more or less of the scene falls within the view frustum.

Once we have the camera situated, we can generate our 2D image. Some of the rays of light traveling through our 3D space will pass through the lens on the camera. These rays then strike a flat surface to produce an image. This effectively projects our 3D scene into a 2D image. The camera's position and other properties determine which rays of light get captured and projected. More specifically, only rays of light that intersect the camera's position, and are within its viewing frustum, will affect the resulting 2D image.

This concludes our brief rendering overview. The light has traveled from its sources to the actors, where it is reflected and scattered. Some of this light gets captured by the camera and produces a 2D image. Now we will look at some of the details of this process.

3.7 Coordinate Systems

There are four coordinate systems commonly used in computer graphics and two different ways of representing points within them (Figure 3.13). While this may seem excessive, each one serves a purpose. The four coordinate systems we use are: model, world , view, and display.

The model coordinate system is the coordinate system in which the model is defined, typically a local Cartesian coordinate system. If one of our actors represents a football, it will be based on a coordinate system natural to the football geometry (e.g., a cylindrical system). This model has an inherent coordinate system determined by the decisions of

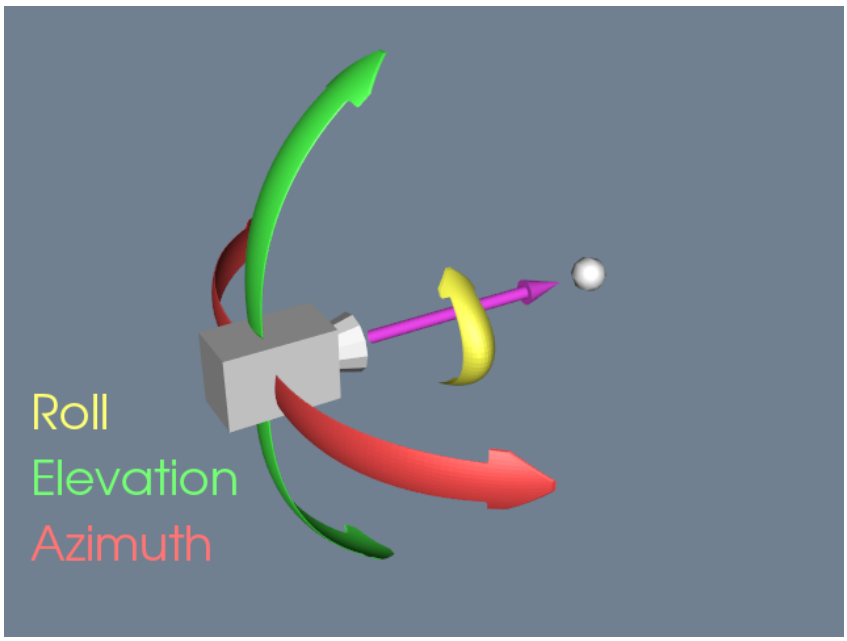


Figure 3.11: Camera movements around the focal point.

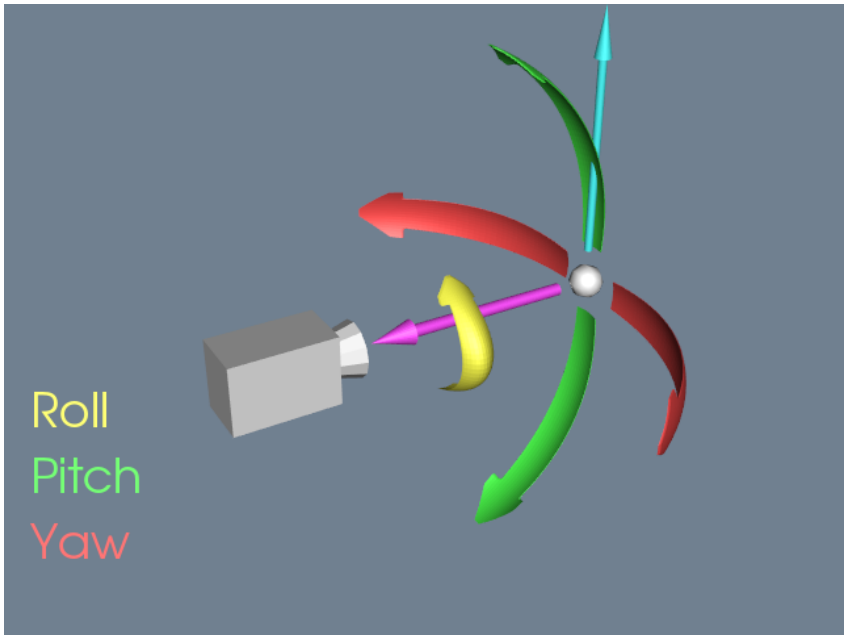


Figure 3.12: Camera movements around the camera position.

whoever generated it. They may have used inches or meters as their units, and the football may have been modelled with any arbitrary axis as its major axis.

The world coordinate system is the 3D space in which the actors are positioned. One of the actor's responsibilities is to convert from the model's coordinates into world

coordinates. Each model may have its own coordinate system but there is only one world coordinate system. Each actor must scale, rotate, and translate its model into the world coordinate system. (It may also be necessary for the modeller to transform from its natural coordinate system into a local Cartesian system. This is because actors typically assume that the model coordinate system is a local Cartesian system.) The world coordinate system is also the system in which the position and orientation of cameras and lights are specified.

The view coordinate system represents what is visible to the camera. This consists of a pair of x and y values, ranging between (-1,1), and a z depth coordinate. The x, y values specify location in the image plane, while the z coordinate represents the distance, or range, from the camera. The camera's properties are represented by a four by four transformation matrix (to be described shortly), which is used to convert from world coordinates into view coordinates. This is where the perspective effects of a camera are introduced.

The display coordinate system uses the same basis as the view coordinate system, but instead of using negative one to one as the range, the coordinates are actual x, y pixel locations on the image plane. Factors such as the window's size on the display determine how the view coordinate range of (-1,1) is mapped into pixel locations. This is also where the viewport comes into effect.

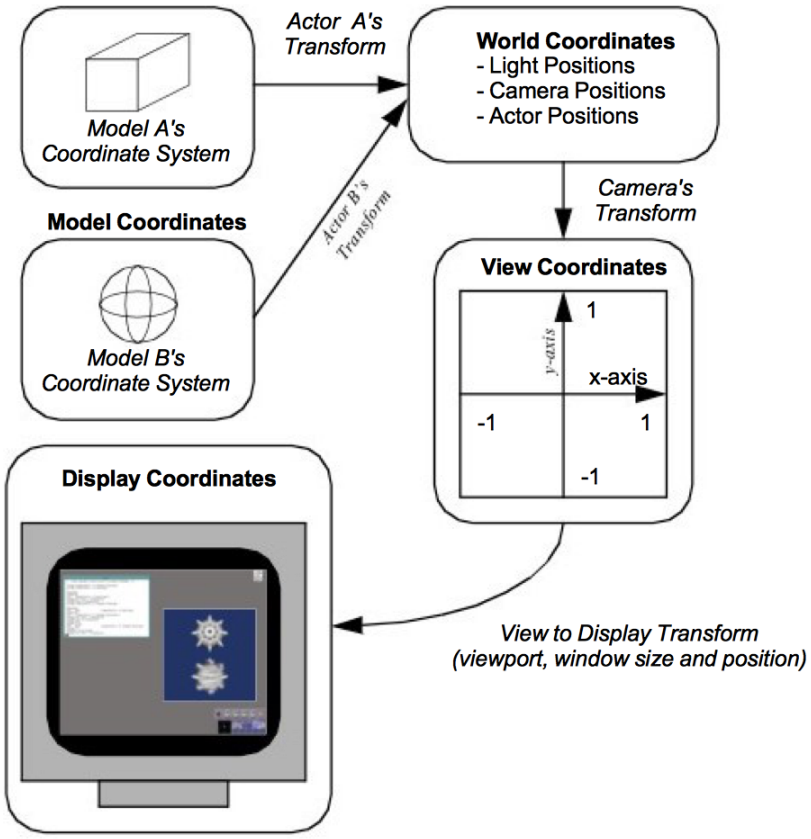


Figure 3.13: Modelling, world, view and display coordinate system.

You may want to render two different scenes, but display them in the same window. This can be done by dividing the window into rectangular viewports. Then, each renderer

can be told what portion of the window it should use for rendering. The viewport ranges from (0,1) in both the x and y axis. Similar to the view coordinate system, the z-value in the display coordinate system also represents depth into the window. The meaning of this z-value will be further described in the section titled "Z-Buffer" on page 41.

3.8 Coordinate Transformation

When we create images with computer graphics, we project objects defined in three dimensions onto a two-dimensional image plane. As we saw earlier, this projection naturally includes perspective. To include projection effects such as vanishing points we use a special coordinate system called homogeneous coordinates.

The usual way of representing a point in 3D is the three element Cartesian vector (x, y, z). Homogeneous coordinates are represented by a four element vector $(x_h h, y_h, z_h, w_h)$. The conversion between Cartesian coordinates and homogeneous coordinates is given by:

$$x = \frac{x_h}{w_h} \qquad x = \frac{y_h}{w_h} \qquad x = \frac{z_h}{w_h} \tag{3.5}$$

Using homogeneous coordinates we can represent an infinite point by setting w h to zero. This capability is used by the camera for perspective transformations. The transformations are applied by using a 4x4 transformation matrix . Transformation matrices are widely used in computer graphics because they allow us to perform translation, scaling, and rotation of objects by repeated matrix multiplication. Not all of these operations can be performed using a 3x3 matrix.

For example, suppose we wanted to create a transformation matrix that translates a point (x, y, z) in Cartesian space by the vector (t_x, t_y, t_z). We need only construct the translation matrix given by

$$T_T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

and then postmultiply it with the homogeneous coordinate (x_h, y_h, z_h, w_h). To carry this example through, we construct the homogeneous coordinate from the Cartesian coordinate (x, y, z) by setting $w_h = 1$ to yield $(x, y, z, 1)$. Then to determine the translated point (x', y', z') we premultiply current position by the transformation matrix T_T to yield the translated coordinate. Substituting into Equation 3.6 we have the result

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.7}$$

Converting back to Cartesian coordinates via Equation3.5 we have the expected solution

$$x' = x + t_x y' = y + t_y z' = z + t_z \tag{3.8}$$

The same procedure is used to scale or rotate an object. To scale an object we use the transformation matrix

$$T_s = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

where the parameters s_x, s_y, and s_z are scale factors along the x, y, z axes. Similarly we can rotate an object around the x axes by angle θ using the matrix

$$T_{R_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

Around the y axis we use

$$TT_{R_y} = \begin{bmatrix} \cos\theta & 0 & \sin\theta 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

and around the z axis we use

$$T_{R_z} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

Another useful rotation matrix is used to transform one coordinate axes $x - y - z$ to another coordinate axes $x' - y' - z'$. To derive the transformation matrix we assume that the unit x'. To derive the transformation matrix we assume that the unit x' axis make the angles $(\theta_{x'x}, \theta_{x'y}, \theta_{x'z})$ around the x-y-z axes (these are called direction cosines). Similarly, the unit y' axis makes the angles $(\theta_{y'x}, \theta_{y'y}, \theta_{y'z})$ and the unit z' axis makes the angles $(\theta_{z'x}, \theta_{z'y}, \theta_{z'z})$. The resulting rotation matrix is formed by placing the direction cosines along the rows of the transformation matrix as follows

$$T_R = \begin{bmatrix} \cos\theta_{x'x} & \cos\theta_{x'y} & \cos\theta_{x'z} & 0 \\ \cos\theta_{y'x} & \cos\theta_{z'y} & \cos\theta_{y'z} & 0 \\ \cos\theta_{z'x} & \cos\theta_{z'y} & \cos\theta_{z'z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

Rotations occur about the coordinate origin. It is often more convenient to rotate around the center of the object (or a user-specified point). Assume that we call this point the object's center . To rotate around we Oc must first translate the object to the Oorigin, apply rotations, and then translate the object back.

Transformation matrices can be combined by matrix multiplication to achieve combinations of translation, rotation, and scaling. It is possible for a single transformation matrix to represent all types of transformation simultaneously. This matrix is the result of repeated matrix multiplications. A word of warning: The order of the multiplication is important. For example, multiplying a translation matrix by a rotation matrix will not yield the same result as multiplying the rotation matrix by the translation matrix.

3.9 Actor Geometry

Modelling

Actor Location and Orientation

3.10 Graphics Hardware

Raster Devices

Interfacing to the Hardware

Rasterization

Z-Buffer

3.11 Putting It All Together

3.12 Chapter Summary

Rendering is the process of generating an image using a computer. Computer graphics is the field of study that encompasses rendering techniques, and forms the foundation of data visualization.

Three-dimensional rendering techniques simulate the interaction of lights and cameras with objects, or actors, to generate images. A scene consists of a combination of lights, cameras, and actors. Object-order rendering techniques generate images by rendering actors in a scene in order. Image-order techniques render the image one pixel at a time. Polygon based graphics hardware is based on object-order techniques. Ray tracing or ray-casting is an image-order technique.

Lighting models require a specification of color. We saw both the RGB (red-green-blue) and HSV (hue-saturation-value) color models. The HSV model is a more natural model than the RGB model for most users. Lighting models also include effects due to ambient, diffuse, and specular lighting.

There are four important coordinate systems in computer graphics. The model system is the 3D coordinate system where our geometry is defined. The world system is the global Cartesian system. All modelled data is eventually transformed into the world system. The view coordinate system represents what is visible to the camera. It is a 2D system scaled from (-1,1). The display coordinate system uses actual pixel locations on the computer display.

Homogeneous coordinates are a 4D coordinate system in which we can include the effects of perspective transformation. Transformation matrices are 4×4 matrices that operate on homogeneous coordinates. Transformation matrices can represent the effects of translation, scaling, and rotation of an actor. These matrices can be multiplied together to give combined transformations.

Graphics programming is usually implemented using higher-level graphics libraries and specialized hardware systems. These dedicated systems offer better performance and easier implementation of graphics applications. Common techniques implemented in these systems include dithering and z-buffering. Dithering is a technique to simulate colors by mixing combinations of available colors. Z-buffering is a technique to perform hidden-line and hidden-surface removal.

3.13 Bibliographic Notes

This chapter provides the reader with enough information to understand the basic issues and terms used in computer graphics. There are a number of good text books that cover computer graphics in more detail and are recommended to readers who would like a more thorough understanding. The bible of computer graphics is [4]. For those wishing for less intimidating books [2] and [5] are also useful references. You also may wish to peruse proceedings of the ACM SIGGRAPH conferences. These include papers and references to other papers for some of the most important work in computer graphics. [3] provides a good introduction for those who wish to learn more about the human vision system.

Bibliography

[1] J. E. Bresenham. “Algorithm for Computer Control of a Digital Plotter”. In: IBM Systems Journal 4.1 (Jan. 1965), pp. 25–30.

[2] P. Burger and D. Gillies. Interactive Computer Graphics Functional, Procedural and Device-Level Methods. Addison-Wesley Publishing Company, Reading, MA, 1989.

[3] N. R. Carlson. Physiology of Behaviour (3d Edition). Allyn and Bacon Inc., Newton, MA, 1985.

[4] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. Computer Graphics Principles and Practice (2d Ed). Addison-Wesley, Reading, MA, 1990.

[5] A. Watt. 3D Computer Graphics (2d Edition). Addison-Wesley, Reading, MA, 1993.

[6] T. Whitted. “An Improved Illumination Model for Shaded Display”. In: Communications of the ACM 23.6 (1980), pp. 343–349.

List of Figures

List of Tables