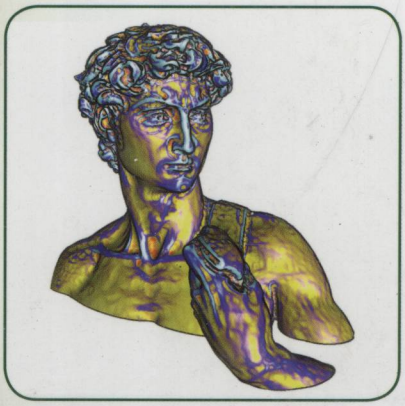
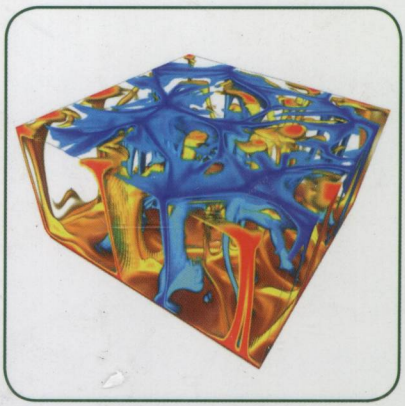
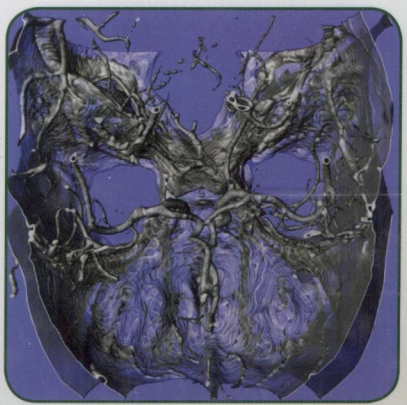
Real-Time VOLUME GRAPHICS







Klaus Engel
Markus Hadwiger
Joe M. Kniss
Christof Rezk-Salama
Daniel Weiskopf

Real-Time Volume Graphics

Klaus Engel
Markus Hadwiger
Joe M. Kniss
Christof Rezk-Salama
Daniel Weiskopf

A K Peters, Ltd.
Wellesley, Massachusetts

Editorial, Sales, and Customer Service Office

A K Peters, Ltd.
888 Worcester Street, Suite 230
Wellesley, MA 02482
www.akpeters.com

Library of Congress Cataloging-in-Publication Data

Real-time volume graphics / Klaus Engel ... [et al.].
p. cm.
Includes bibliographical references and index.
ISBN 13: 978-1-56881-266-3 (alk. paper)
ISBN 10: 1-56881-266-3 (alk. paper)
1. Computer graphics. 2. Three-dimensional display systems. I. Engel, Klaus, 1969-

T385.R43414 2006
006.6'93–dc22

2006041662

Printed in India
10 09 08 07 06

10 9 8 7 6 5 4 3 2 1

Real-Time Volume Graphics

Für Monika
—Klaus Engel

For Pilar
—Markus Hadwiger

To K. P.
—Joe M. Kniss

For Malak and Helene
—Christof Rezk-Salama

Für Bettina
—Daniel Weiskopf

Preface

IN TRADITIONAL COMPUTER GRAPHICS, 3D objects are created using high-level surface representations such as polygonal meshes, NURBS (nonuniform rational B-spline) patches, or subdivision surfaces. Using this modeling paradigm, visual properties of surfaces, such as color, roughness, and reflectance, are described by means of a shading algorithm, which might be as simple as the Lambertian diffuse reflection model or as complex as a fully-featured shift-variant anisotropic BRDF.[1] Because light transport is evaluated only at points on the surface, these methods usually lack the ability to account for light interaction that takes place in the atmosphere or in the interior of an object.

Compared with surface rendering, volume rendering describes a wide range of techniques for generating images from 3D scalar data. These techniques are originally motivated by scientific visualization, where volume data is acquired by measurement or generated by numerical simulation. Typical examples are medical data of the interior of the human body obtained by computerized tomography (CT) or magnetic resonance imaging (MRI). Other examples are data from computational fluid dynamics (CFD), geological and seismic data, and abstract mathematical data such as the 3D probability distribution of a random number, implicit surfaces, or any other 3D scalar function.

It did not take long for volume-rendering techniques to find their way into visual arts. Artists were impressed by the expressiveness and beauty of the resulting images. With the evolution of efficient rendering techniques, volume data is also becoming more and more important for applications in computer games. Volumetric models are ideal for describing fuzzy objects, such as fluids, gases, and natural phenomena like clouds, fog, and fire.

[1] BRDF = bidirectional reflection distribution function: a function used to describe complex optical material properties.

Many artists and researchers have generated volume data synthetically to supplement their traditional surface models. They have found that volume-rendering techniques are useful for producing a large variety of impressive visual effects.

Although, at first glance, volumetric data sets seem to be more difficult to visualize than surfaces, it is both worthwhile and rewarding to render them as truly 3D entities without falling back to 2D subsets. Efficient rendering techniques that generate high-quality images of volumetric objects including local and global illumination effects in real time, or at least at interactive frame rates, are the topic of this book.

Intended Audience

This book is intended for two groups of readers. The first group comprises members of the scientific community, such as computer scientists, engineers, physicists, and medical imaging professionals. The other group comprises game developers, visual artists and animators, technical directors, and all people that are concerned with the development of multimedia and visual-entertainment applications. For scientists, the clarity and the accuracy of the visual representation of their data is essential. The entertainment community will focus more on artistic merits and creative aspects such as aesthetics, expressiveness, and everything that helps them communicate with the audience and tell their story. Both groups will find that interactivity is essential.

Although most of the topics covered in this book deal with the programming of computer-graphics applications, the book is not solely intended for software developers or computer scientists. Content creators and visual artists, whose primary concern is usually not software development, will find out that volume graphics is not as difficult to realize as they might think. They will learn expressive and powerful techniques for creating visual effects that are hard to realize with traditional surface modeling. From our experience with various application areas, we know that there are also many people from scientific disciplines who need customized methods for visualizing their scientific data. They often find themselves writing programs to visually display their abstract data without really having a pool of working methods that they can build upon. For those people, this book will provide effective solutions, important concepts, and ideas for tailoring their applications to their specific needs.

How to Read This Book

From the didactic point of view, the best way to read this book is from cover to cover. Having said that, we encourage you to browse through

the book and start reading wherever a passage or a figure catches your attention. As we know, many readers prefer to skip parts of the text and jump back and forth through the different chapters. In this section, we want to give you some hints about what you will find in which parts of the book and which chapters are built upon other chapters.

The first two chapters cover the basic prerequisites for the rest of the book. Chapter 1 explains the physical basics of light transport and lays the theoretical groundwork for later chapters. If you already feel familiar with optics and light transfer, or if you are more interested in practical implementation than theory, you can skip this chapter for now and return to it later. Chapter 2 gives an overview of programmable graphics hardware and its most important features. We assume that you are already familiar with graphics programming to a certain extent, and this chapter is only meant as a refresher.

The next few chapters are essential for all readers, regardless of whether you're interested in scientific visualization, visual arts, or games. Chapter 3 starts with a practical introduction to different approaches to texture-based volume rendering. After having worked through this chapter, you should be able to implement your first completely functional volume-rendering system. Some of the techniques described in this chapter do not even require programmable graphics hardware, but the algorithms are essential for the rest of the book. Chapter 4 introduces transfer functions, which are used to specify the optical properties based on your underlying volumetric data. You will learn different mechanisms to perform color mapping and understand their influence on image quality.

With the next two chapters, we increase the level of realism by integrating different aspects of light-matter interaction. Chapter 5 shows how to adapt popular local illumination techniques to volumetric data. This is important for applications both in science and entertainment. Chapter 6 introduces global illumination techniques such as shadows, scattering, and translucency. These advanced illumination effects are clearly motivated by visual arts, but scientific applications will also benefit from shadows and improved realism.

Although graphics hardware has been designed for object-order approaches, modern techniques also allow image-order approaches such as ray casting to be implemented. Chapter 7 explains GPU-based implementations of ray casting, including optimization techniques such as space leaping and early ray termination.

The next two chapters cover optimization strategies, which are important for all application areas. Chapter 8 analyzes rendering speed and covers effective techniques to get the maximum performance out of your graphics board. Chapter 9 provides methods to improve the visual quality of your images. Different types of visual artifacts and their real causes are analyzed, and efficient countermeasures are introduced. Chapter 10

revisits transfer functions and extends them to multiple dimensions and multivariate data. User interfaces for intuitive classification and guidance are demonstrated. These three chapters together are essential for implementing a state-of-the-art volume-rendering system.

Chapter 11 is a guide to volume-rendering techniques for game programmers. It discusses the value of volume-graphics techniques for games and compares them to traditional techniques. It explains how to seamlessly integrate volume graphics into a game engine. The next two chapters focus on visual arts. Chapter 12 covers practical techniques for generating volumetric models from scratch using polygonal surfaces and procedural techniques. Chapter 13 discusses techniques for volumetric deformation and animation. These techniques can be used to sculpt volumetric models or to deform measured data. Apart from visual arts, fast deformation techniques are important for scientific applications such as computer-assisted surgery.

Chapter 14 deals with illustrative volume-rendering techniques and non-photorealistic rendering. The goal of such approaches is to create contours and cutaways to convey the important information by *amplification through simplification*. The chapter covers approaches such as importance-driven visualization, focus-and-context techniques, and non-photorealistic shading, which are mainly important for scientific visualization. Chapter 15 explains a variety of interactive clipping techniques, which facilitate the exploration of volume data in scientific data analysis. Segmented volume data is often used in medical scenarios, where certain inner organs or anatomical structures are marked explicitly by different tags. Chapter 16 covers techniques for integrating segmentation data into our volume-rendering framework. Finally, with respect to the ongoing trend toward huge data sets, Chapter 17 introduces effective strategies to overcome memory and bandwidth limitations for rendering of large volume data.

Graphics Programming

Only a couple of years ago, real-time volume graphics was restricted to expensive graphics workstations and large rendering clusters. The past couple of years, however, have seen a breathtaking evolution of consumer graphics hardware from traditional *fixed-function* architectures (up to 1998) to *configurable* pipelines to *fully programmable* floating-point graphics processors with hundreds of millions of transistors. The first step toward a fully programmable GPU was the introduction of configurable rasterization and vertex processing in late 1999. Prominent examples are NVIDIA's *register combiners* and ATI's *fragment shader* OpenGL extensions. Unfortunately, at the time, it was not easy to access these vendor-specific features in a uniform way.

The major innovation provided by today's graphics processors is the introduction of true programmability. This means that user-specified microprograms can be uploaded to graphics memory and executed directly by the vertex processor (*vertex programs*) and the fragment processor (*fragment programs*).[2] Vertex and fragment programs consist of assembler-like instructions from the limited instruction set understood by the graphics processor (`MOV`, `MAD`, `LERP`, and so on). To spare the user the tedious task of writing assembler code, high-level shading languages for GPU programming have been introduced. They provide an additional layer of abstraction and allow access to the capabilities of different graphics chips in an almost uniform way. Popular examples of high-level shading languages are GLSL, the shading language introduced with the OpenGL 2.0 specification, and Cg, introduced by NVIDIA, which is derived from the *Stanford Shading Language*. HLSL, the high-level shading language introduced in Microsoft's DirectX 9.0 SDK, uses a syntax very similar to Cg.

We believe that code samples are essential for conveying algorithms. Throughout this book, we provide code samples that concretely illustrate our rendering algorithms. We have made an effort to keep the samples simple and easy to understand, and we have taken our choice of programming languages seriously. Unless stated otherwise, the samples in this book are written in C/C++ with OpenGL as the graphics API and Cg as the shading language.

C++ is the most popular programming-language choice of graphics programmers. There are many introductory textbooks on C++ programming, including [257]. The reason for choosing OpenGL as the graphics API is that it is consistently supported on the largest number of different platforms and operating systems. At this point, we assume that you already have a basic knowledge of graphics programming and OpenGL. If you are not familiar with OpenGL, we suggest studying the OpenGL Red Book [240] first. However, we do not expect that readers who are more familiar with the DirectX API will have major problems when adapting the code samples. The reason for choosing Cg as the high-level shading language rather than OpenGL's built-in shading language GLSL is that Cg can be used directly with both OpenGL and DirectX, and the current version of the Cg compiler is also able to generate GLSL code. The syntax of Cg should be intelligible to anyone familiar with C/C++, and even a less experienced programmer should not have major problems understanding the code and adapting the samples to any high-level shading language. Introductory material and sample code using Cg can be found on the NVIDIA developer site [34].

[2]The terms *vertex shader* and *vertex program* and also *fragment shader* and *fragment program* have the same meaning, respectively. We usually prefer the term *program* because a major part of the code is not related to shading at all.

Acknowledgments

This book has evolved as a result of several courses and tutorials held at ACM SIGGRAPH, IEEE Visualization, and Eurographics conferences in the past couple of years. We are indebted to many people who helped make it possible in one way or another.

Gordon Kindlmann and Aaron Lefohn have contributed significant parts to the text and to the original SIGGRAPH course notes. Gordon's work on curvature-based classification and Aaron's ideas on efficient data structures are essential parts of the book.

This book reflects the collective work of many researchers over several years and would not exist without the wealth of experience provided to us. Many of these researches have also supported the writing of this book by generously providing their material, especially images and data sets. We would like to thank (in alphabetical order): Dörte Apelt, Anna Vilanova í Bartroli, Christoph Berger, Stefan Bruckner, Katja Bühler, Min Chen, Roger Crawfis, Paul Debevec, Helmut Doleisch, Knut E. W. Eberhardt, David S. Ebert, Laura Fritz, Markus Gross, Stefan Guthe, Peter Hastreiter, Jiří Hladůvka, Shoukat Islam, Mark Kilgard, Andrea Kratz, Martin Kraus, Caroline Langer, Bob Laramee, Torsten Möller, Lukas Mroz, André Neubauer, Bernhard Preim, Werner Purgathofer, Stefan Röttger, Henning Scharsach, Christian Sigg, Wolfgang Straßer, Nikolai A. Svakhine, Thomas Theußl, Bernd F. Tomandl, Ivan Viola, Manfred Weiler, Rüdiger Westermann, and Xiaoru Yuan.

Volume data sets were generously provided by the Digital Morphology Project at the University of Texas at Austin, the Department of Neuroradiology at the University of Erlangen-Nuremberg, the University of Minnesota at Twin Cities, the University of North Carolina at Chapel Hill, Siemens Medical Solutions, Stanford University, the Universities of Tübingen and Stuttgart (Deutsche Forschungsgesellschaft, SFB 382), Tiani Medgraph, the United States National Library of Medicine, and the ETH Zürich.

Our deep respect is due to Tom Ertl, Günther Greiner, Meister Eduard Gröller, Charles Hanson, Helwig Hauser, Chris Johnson, and Rüdiger Westermann. They provided encouragement and valuable feedback throughout the years. It has been a pleasure working with you.

It is impossible to develop efficient graphics algorithms without the cooperative work of the hardware manufacturers. We wish to thank ATI and NVIDIA for their continuous support in knowledge and hardware, especially Mike Doggett and Mark Segal from ATI and Mark Kilgard, David Kirk, and Nick Triantos from NVIDIA.

Klaus Engel would like to thank everybody at Siemens Corporate Research for their input and support, particularly James Williams, Gianluca Paladini, Thomas Möller, Daphne Yu, John Collins, and Wei Li.

We are grateful to our students and all the attendees of our courses, who provided valuable feedback and suggestions to improve both the course and the book.

Kevin Jackson-Mead, Alice Peters, and all the staff at A K Peters have done a great job in making this book. We wish to thank you for your care and your patient attention.

Finally, and most of all, we wish to express our love and gratitude to our families for their support and for giving us the quiet time we needed to finish the book.

Additional Resources

Further information, sample programs, data sets, and links to other online resources can be found at http://www.real-time-volume-graphics.org.

Contents

1

Theoretical Background and Basic Approaches

THIS BOOK COVERS two seemingly very different applications of volume graphics: on the one hand, "special effects" and realistic rendering of clouds, smoke, fire, and similar effects for computer games, movie production, and so forth; on the other hand, the scientific visualization of volumetric data. How do these different fields fit together, and why are they covered in the same text?

The simple answer is that both fields rely on the same underlying physical models and therefore use identical, or at least very similar, rendering techniques. This chapter focuses on the physical model for volume rendering, discussing its fundamental mathematical description and its approximations typically used for real-time volume rendering. The basic idea is to model light transport in gaseous materials such as clouds or fog. Therefore, volume graphics targets the same goal as computer graphics in general: the simulation of light propagation in order to produce images as recorded by a virtual camera.

The specific challenge for volume graphics is the interaction between light and the participating media. Light may be absorbed, scattered, or emitted by the gaseous materials that virtually "participate" in light propagation. This interaction needs to be evaluated at all positions in the 3D volume filled by the gas, making volume rendering a computationally intensive task. Therefore, the techniques discussed throughout this book address the issue of efficient volume rendering. The remainder of this chapter focuses on the theoretical foundation for these rendering methods, and it provides a general overview of the volume-rendering process.

We have decided to lay out a theoretical background for volume rendering in the beginning of this book. Our motivation is to provide a sound foundation for the various algorithms that are presented in later chapters. However, for readers who would like to start with practical issues of volume