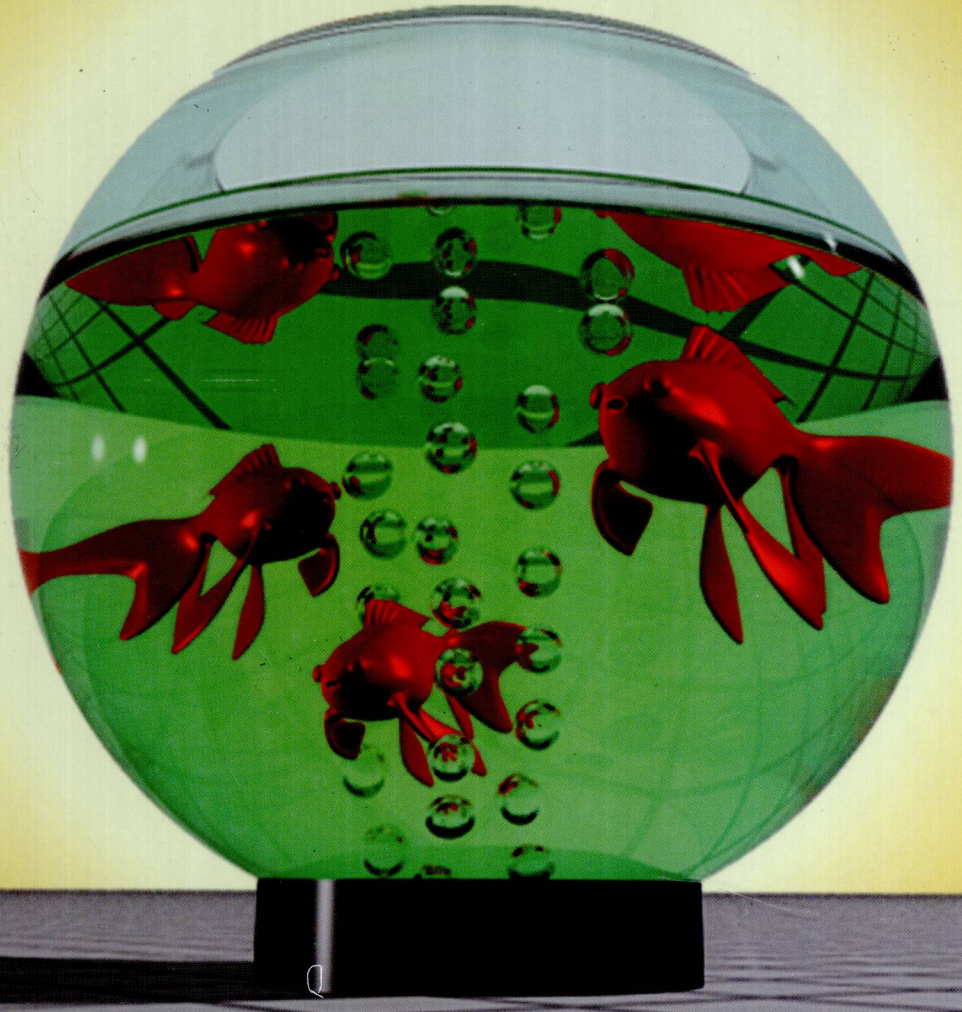
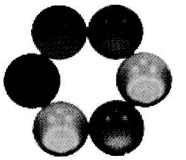
Ray Tracing from the Ground Up



Kevin Suffern



Ray Tracing from the Ground Up

Kevin Suffern

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

Suffern, Kevin G.
Ray tracing from the ground up / Kevin G. Suffern.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-56881-272-4 (alk. paper)
1. Computer graphics. I. Title.
T385.S7995 2007
006.6--dc22 2007021706

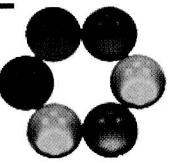
The image on the cover is based on the fishbowl object described in Chapter 28. The goldfish model, courtesy of James McNess, is rendered using a triangle mesh, as described in Chapters 22 and 23.

Printed in India
11 10 09 08 07 10 9 8 7 6 5 4 3 2 1

In memory of
Silvia Gladys Suffern, 1910-2003
Lucy Suffern, 1977-1997

To Eileen, I could not have written this without you.

Foreword



Computer graphics involves simulating the distribution of light in a 3D environment. There are only a few fundamentally different algorithms that have survived the test of time. They can be loosely classified into projective algorithms and image-space algorithms. The former class projects each geometric primitive onto the image plane, with local shading taking care of the appearance of objects. This class of algorithm is still widely used because it is amenable to pipeline processing and therefore to hardware implementation as evidenced by all modern graphics cards.

Image-space algorithms compute the color of each pixel by figuring out where the light came from for that pixel. Here, the basic operation is to determine the nearest object along a line of sight. Following light back along a line has given this basic operation and the associated image-synthesis algorithm their name: ray tracing.

In 1980, ray tracing was at the forefront of science. The quality of the images that can be computed with ray tracing was an eye opener, as it naturally includes light paths such as specular reflection and transmission, which are difficult to compute with projective algorithms. Some shapes are easier to intersect rays with than others, and in those early days, spheres featured heavily in ray-traced images. Hence, old images often contained shiny spheres to demonstrate the power of ray tracing.

A vast amount of research was then expended to make ray tracing both more tractable and to include more features. Variants were introduced, for instance, that compute diffuse inter-reflection, caustics, and/or participating media. To speed up image generation, many data structures were developed that spatially sort the 3D geometry. Spatial subdivision algorithms allow a very substantial reduction of the candidate set of objects that need to be

intersected to find the nearest object for each ray. Ray tracing is also amenable to parallel processing and has therefore attracted a substantial amount of research in that area.

All of this work moved ray tracing from being barely tractable, to just about doable for those who had state-of-the-art computers and plenty of time to kill. High-quality rendering tends to take a whole night to complete, mostly because this allows artists to start a new rendering before going home, to find the finished image ready when they arrive at work the next day. This, by the way, still holds true. For many practical applications, hardware and algorithmic improvements are used for rendering larger environments, or to include more advanced shading, rather than to reduce the computation time.

On the other hand, more than 25 years after its introduction, ray tracing has found a new lease on life in the form of interactive and real-time implementations. Such rendering speeds are obtained by using a combination of super-fast modern hardware, parallel processing, state-of-the-art algorithms, and a healthy dose of old-fashioned low-level engineering. Recent advances have enabled ray tracing to be a useful alternative for real-time rendering of animated scenes, as well as huge scenes that do not fit into main memory. In addition, there is a trend towards the development of dedicated hardware for ray tracing.

All of this research has helped to push ray tracing from an interesting esoteric technique for image synthesis to a seriously viable algorithm for practical applications. If necessary, ray tracing can operate in real time. If desired, ray tracing can be physically based and can therefore be used in predictive lighting simulations. As a result, ray tracing is now used in earnest in the movie industry, but also, for instance, in the automotive industry and in scientific visualization. In addition, it forms the basis for several other graphics algorithms, including radiosity and photon mapping.

The practical importance of ray tracing as a lighting-simulation technique means that ray tracing needs to be taught to students, as well as to practitioners in industry. In addition, ray tracing is sufficiently multifaceted that teaching students all aspects of the algorithm will give them all manner of additional benefits: 3D modeling skills, mathematics skills, software engineering skills (writing a ray tracer is for many students the first time that they will have to manage a sizeable chunk of code), hands-on experience in object-oriented programming, and deeper insights into the physics of light, as well as knowledge of the behavior of materials.

It would be ideal to present a ray-tracing course to students at the undergraduate level for all of the above reasons, but also because a deep understanding of ray tracing will make it easier to grasp other image-synthesis algorithms.

For this, a book is required that explains all facets of ray tracing at the right pace, assuming only a very moderate amount of background knowledge.

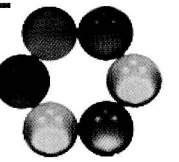
I'm positively delighted that such a book now exists. *Ray Tracing from the Ground Up* not only covers all aspects of ray tracing, but does so at a level that allows both undergraduate and graduate students to appreciate the beauty and algorithmic elegance of ray tracing. At the same time, this book goes into more than sufficient detail to deserve a place on the bookshelves of many professionals as a reference work.

Kevin was gracious enough to let me read early drafts of several chapters when I was teaching a graduate-level ray-tracing course at the University of Central Florida. This has certainly taught me many of the lesser-known intricacies of ray tracing. Kevin himself has taught ray tracing to undergraduate students for many years, and it shows. This book, which grew out of his course notes, is remarkably easy to follow, especially given the complexity of the subject matter.

As such, I can heartily recommend this book to both professionals as well as students and teachers. Whether you are only interested in rendering a collection of shiny spheres or want to create stunning images of highly complicated and realistic environments, this book will show you how. Whether its intended use is as a ray-tracing reference or as the basis of a course on ray tracing, this book is essential reading.

Erik Reinhard
University of Bristol
University of Central Florida

Preface



Where Did This Book Come From?

Since the early 1990s, I have had the privilege of teaching an introductory ray-tracing course at the University of Technology, Sydney, Australia. This book is the outcome of all of those years in the classroom. The ray tracer presented here has been developed and taught over the years, during which time my students have provided invaluable feedback, bug reports, ideas, and wonderful images. The book's manuscript has evolved from the teaching notes for the course and has been written (and re-written) chapter by chapter as the ray tracer, and my teaching of it, have developed. Writing in a teaching context with the feedback provided by students has helped me produce a book that is, I hope, much more understandable than it would have been had I written it in isolation. I like to call the iterative processes of programming, writing, and teaching ray tracing the *ray-tracing circle*.

What Is Ray Tracing?

Ray tracing is a computer-graphics technique that creates images by shooting rays. It's illustrated in Figure 1, which shows a camera, a window with pixels, two rays, and two objects. The rays go through pixels and are tested for intersection with the objects. When a ray hits an object, the ray tracer works out how much light is reflected back along the ray to determine the color of the pixel. By using enough pixels, the ray tracer can produce an image of the objects. If the objects are reflective, the rays can bounce off of them and hit other objects.

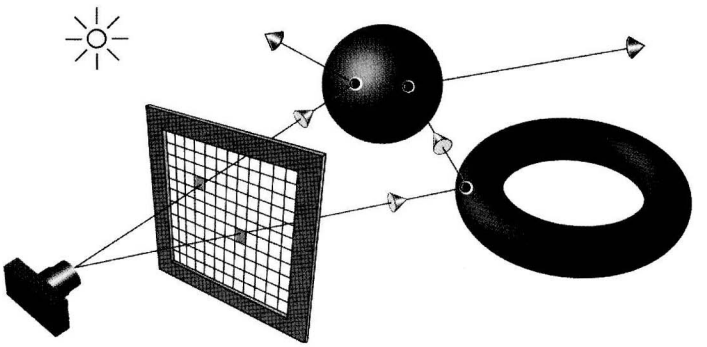


Figure 1. The ray-tracing process.

This process is conceptually simple, elegant, and powerful. For example, it allows ray tracing to accurately render reflections, transparent objects, shadows, and global illumination. Ray tracing can also render large triangle meshes more efficiently than other rendering techniques.

Why Is Ray Tracing Important?

The production of ever more realistic images is a trend of long standing in 3D computer graphics. This is a task at which ray tracing excels. A major application of ray tracing is the film industry, not just for visual effects, but for rendering whole movies. For example, the animated films *Ice Age*, *Ice Age 2*, and *Robots* were fully ray traced, as were the short films *Bunny* and *The Cathedral* (a.k.a. *Katedra*). *Ice Age* was nominated for the Academy Award for Best Animated Feature in 2002, *Bunny* won the Academy Award for Animated Short Film in 1998, and *The Cathedral* was nominated for the Academy Award for Animated Short Film in 2002. Ray tracing was also used in *Happy Feet* to render the penguins with ambient occlusion, and for reflection and refraction with the ocean surface (see Chapters 17, 27, and 28). *Happy Feet* won the Academy Award for Best Animated Feature in 2006.

The major software packages used in the visual-effects industry have built-in ray tracers, and there are numerous state-of-the art ray tracers available as plug-ins or stand-alone applications. These include Brazil (http://www.splutterfish.com/), Mental Ray (http://www.mentalimages.com/), finalRender (http://www.finalrender.com/), and Maxwell Render (http://www.maxwellrender.com/). Cinema 4D (http://www.maxon.net/) also has a state-of-the art ray tracer.

Real-time 3D computer games also have an increasing demand for realism. Although current PCs are not fast enough for real-time ray-traced games, this is likely to change in the next few years. The introduction of chips with specialized graphics processors on multiple cores that can be programmed using existing programming tools will make this possible. Because each ray can be traced independently, ray tracing can trivially use as many processors as are available. In fact, ray tracing has been described as being "embarrassingly parallelizable."[1] These hardware advances should also result in ray tracing being used more frequently in the visual-effects industry.

All this means that ray tracing has a great future, and within a few years you should be able to use the techniques you will learn in this book to write real-time ray-tracing applications such as games.

Graphics education also benefits greatly. My experience has been that getting students to write a ray tracer is the best way for them to understand how rendering algorithms work. Ray tracing's flexibility and ease of programming is the primary reason for this.

In a more general context, ray tracing helps us understand the appearance of the world around us. Because it simulates geometric optics, ray tracing can be used to render many familiar optical phenomena. The appearance of the fish and bubbles on the front cover is an example.

Book Features

This book provides a detailed explanation of how ray tracing works, a task that's accomplished with a combination of text, code samples, about 600 ray-traced images, and over 300 illustrations. Full color is used throughout. Almost all of the ray-traced images were produced with the software discussed here. The book also showcases the work of about 25 students. You can develop the ray tracer chapter by chapter.

Most chapters have questions and exercises at the end. The questions often ask you to think about ray-traced images; the exercises cover the implementation of the ray tracer and suggest ways to extend it. There are almost 400 questions and exercises.

Shading is described rigorously as solutions to the rendering equation and is specified in radiometric terms such as radiance (see Chapter 13).

1. Alan Norton, circa 1984, personal communication.

The book's website (http://www.raytracegroundup.com) contains several animations that demonstrate effects and processes that are difficult or impossible to see with static images.

Pathways through This Book

You don't have to read the chapters in order, or read all of the material in every chapter, or read all of the chapters. For example, Chapters 2 and 20 cover some of the mathematics you need for ray tracing, and you may already be familiar with this; you can read Chapter 19 on ray-object intersections, or parts of it, when you need to.

Chapters 1–4, 9, and 13–16 cover ray-tracing fundamentals, perspective viewing with a pinhole camera, theoretical foundations, and basic shading. Chapter 13 is heavy going mathematically but provides the essential theoretical foundations for the following chapters on shading. The good news is that you don't have to master all of the material in Chapter 13. Most of the complicated integrals in this and the following chapters can be expressed in a few simple lines of code, which are in the book.

Chapter 24 covers mirror reflection, Chapters 27 and 28 cover transparency, and Chapters 29–31 cover texturing. Although You will find many interesting things to explore here, and you can read the texturing chapters first, if you want to.

If you read the sampling chapters, Chapters 5–7, you will have the background to understand the different camera models in Chapters 10–12, ambient occlusion in Chapter 17, area lights in Chapter 18, glossy reflection in Chapter 25, and global illumination in Chapter 26.

Chapter 21 explains how to ray trace transformed objects, and Chapter 22 covers grid acceleration, which is the tool for ray tracing triangle meshes in Chapter 23.

What Knowledge and Skills Do You Need?

Because the ray tracer is written in C++, you should be reasonably proficient at C++ programming. A first course in C++ should be sufficient preparation, but there is a heavy emphasis on inheritance, dynamic binding, and polymorphism right from the start. That's a critical design element, as I'll explain in Chapter 1.

You should also be familiar with coordinate geometry, elementary trigonometry, and elementary vector and matrix algebra. Although there is some

calculus in the book, I usually just quote the results and give you the relevant code.

You don't need previous studies in computer graphics because the book is self-contained in this regard. Chapters 3, 8, and 13 cover the necessary graphics background material. As far as graphics output is concerned, ray tracing is a simple as possible—you just draw pixels into a window on your computer screen.

Intended Audience

The book is intended for computer-graphics students who have had at least an introductory course in C++. The book is suitable for both undergraduate and graduate courses.

It's also intended for anyone with the required background who wants to write a ray tracer or who wants find out how ray tracing works. This includes people working in the computer-graphics industry.

Online Resources

The book's website is at http://www.raytracegroundup.com, where you will find:

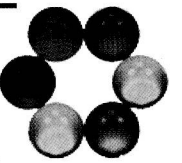
- the skeleton ray tracer described in Chapter 1;
- sample code;
- triangle mesh files in PLY format;
- image files in PPM format;
- the ray-traced images in JPEG format;
- additional images;
- C++ code for constructing scenes;
- animations;
- a place where you can post errata;
- useful links.

Topics Not Covered

Due to time and space constraints, here are some of the many topics that I have not discussed: high dynamic range (HDR) imaging with local tone-map-

ping operators; comparisons between grid acceleration and other acceleration schemes such as bounding volume hierarchies; an efficient global illumination algorithm; an efficient technique for rendering caustics; sub-surface scattering; bump mapping; the volumetric rendering of participating media. Although HDR imaging would require some serious retrofitting, the other topics could be implemented as add-ons. Any of these could be student assignments or projects.

Acknowledgments



It's traditional for authors to start by thanking their long-suffering families. This is not a tradition I'm about to break. First, my thanks go to my wife, Eileen, without whose support this book would not have been written. Eileen read the whole book about three times, and then the proofs, corrected innumerable grammar mistakes, improved my writing, and did a lot of work on the index. She did all the cooking and housework through this period, while completing her own diploma studies, and watched in despair as the number of urgent repair jobs on the house grew and grew. As a result, I probably have about a year's work of repairs awaiting me.

My son Chris provided expert help with Adobe Illustrator and did all the curves in the figures. Chris also did the photography and graphic design for Figure 29.26. I'm very grateful for Chris's help. My son Tim checked the references, for which I am also very grateful. And finally, my four-year-old grandson Broden gave me permission to use his photograph in Figure 29.26.

Paul Perry, a friend of 40 years, provided on more than one occasion advice and a quiet refuge in Melbourne for writing.

I would like to thank Erik Rienhard, who acted as my reviewer. Erik read the whole manuscript, found many mistakes, and made many suggestions that have improved this book. I'm honored that he also wrote the foreword. Frequent phone conversations with him were very helpful.

I would also like to thank Pete Shirley, who has helped and inspired me in several ways. For example, Pete suggested that I use orthonormal bases and use instances for ray tracing transformed objects. There is a lot of Pete's work in this book, and it is the better for it.

The following people trialed early versions of the manuscript in their ray-tracing courses: Dave Breen at Drexel University, Bob Futrelle at Northeastern University, Steve Parker at the University of Utah, Erik Reinhard at the University of Central Florida, and Pete Shirley at the University of Utah. I wish to thank all of them.

Special thanks must go to Alice Peters at A K Peters, who believed in this book, right from the start, was amazingly patient during interminable delays on my part, and worked extremely hard to get it published for SIGGRAPH 2007. I particularly wish to thank Kevin Jackson-Mead, Senior Editor at A K Peters, who expertly edited the manuscript. This was a huge job. I also thank the typesetter, Erica Schultz, for an expert job.

Thanks go to Darren Wotherspoon at Skye Design for the beautiful cover design.

Richard Raban also deserves special thanks. Richard, as my Department Head at the University of Technology, Sydney, gave me a lot of support and a working environment that made writing easier. Chris W. Johnson and Helen Lu at UTS also read several chapters and pointed out mistakes.

The following students graciously provided permission to use their ray-traced images: Steve Agland, Nathan Andrews, John Avery, Peter Brownlow, David Gardner, Peter Georges, Mark Howard, Tania Humphreys, Daniel Kaestli, Adeel Khan, Mark Langsworth, Lisa Lönroth, Alistair McKinley, Jimmy Nguyen, Riley Perry, Duy Tran, and Ving Wong. Thank you all for your beautiful images, which have enhanced this book.

Thanks also go to Tania Humphreys for modeling and rendering the Mirage 2000 device by Opti-Gone Associates, to Alistair McKinley for implementing the ray-visualization software that made Figure 27.24(b) possible, and to Jimmy Nguyen for performing a number of file conversions.

The following students pointed out errors in the manuscript: Deepak Chaudhary, Tim Cooper, Ronnie Sunde, Ksenia Szeweva, and Mark White.

I must thank Naomi Hatchman for allowing me to use her penguin model in Chapters 23 and 29 and for a lot of work in supplying files in various formats and triangle resolutions.

I also thank James McNess for permission to use his goldfish model, shown on the cover and in Chapters 23 and 28.

There are two critical pieces of software associated with this book. One is the skeleton ray tracer. I'd like to thank Peter Kohout and James McGregor for producing early versions. I particularly thank Sverre Kvåle for writing the current version. Sverre also converted all the code files to PC format and tested the sample code. I am most grateful for all his work and his great programming skills.

The other is the website, built by HwaLeon (Ayo) Lee. Thank you, Ayo, for your generosity and professional development skills.

Ayo, James McGregor, Naomi, Peter, and Sverre have been students of mine; James McNess has been a student of Naomi.

Another student, Hong Son Nguyen, kindly produced a number of computer animations for the book and allowed me to put these on the website. Thank you, Hong.

My students, Naomi Hatchman, André Mazzone, Glen Sharah, Rangi Sutton, and the "UTS Amigos": Steve Agland, Peter Brownlow, Chris Cooper, Peter Georges, Justen Marshall, Adrian Paul, and Brian Smith, have inspired me by earning credits on Academy Award winning films for their outstanding visual-effects work. The films include *Happy Feet*, *The Lord of the Rings: The Return of the King*, and *The Matrix*.

I have also benefited greatly from discussions with Paul Bourke about nonlinear projections and stereoscopy. Paul read Chapters 11 and 12, and as a result of his feedback, these chapters are now more relevant to real-world applications. He also permitted me to use material in these chapters from his website at http://local.wasp.uwa.edu.au/~pbourke/. I thank Paul for all his help.

I also thank Paul Debevec for permission to use the Uffizi probe image from his website at http://www.debevec.org/Probes/ and for providing advice on light-probe mappings.

Henrik Jensen provided advice on various aspects of ray tracing.

The following people and organizations allowed me to use various information. Stephen Addleman from Cyberware: images of the horse, Isis, and Ganesh models from http://www.cyberware.com/; James Hastings-Trew: Earth images in Chapter 29 from his website at http://planetpixelemporium.com/planets.html; Phillipe Hurbain: sky images from http://www.philohome.com/ in Chapters 11 and 12; Michael Levin from Opti-Gone International: material from http://www.optigone.com/ about the Mirage 2000 in Chapter 12; Ric Lopez-Fabrega at Lopez-Fabrega Design: sky images from http://www.lfgrafix.com in Chapter 29; Morgan Kaufmann Publishers: images and code in Chapter 31; Steve Parker at the University of Utah: the Cornell-box image in Chapter 26; Greg Turk: PLY code and various PLY models of the Stanford bunny at http://www.cc.gatech.edu/projects/large_models/index.html. I thank you all.

Finally, I wish to thank Eric Haines and Pete Shirley for graciously providing endorsements for the book.