

---

# 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.

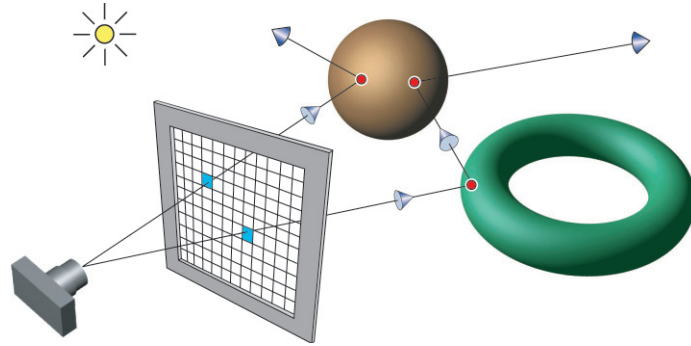


Figure 1. The ray-tracing process.

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.maxwell-render.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.”<sup>1</sup> 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 as 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.