Preface

[Just as] other information should be available to those who want to learn and understand, program source code is the only means for programmers to learn the art from their predecessors. It would be unthinkable for playwrights not to allow other playwrights to read their plays [or to allow them] at theater performances where they would be barred even from taking notes. Likewise, any good author is well read, as every child who learns to write will read hundreds of times more than it writes. Programmers, however, are expected to invent the alphabet and learn to write long novels all on their own. Programming cannot grow and learn unless the next generation of programmers has access to the knowledge and information gathered by other programmers before them. —Erik Naggum

Rendering is a fundamental component of computer graphics. At the highest level of abstraction, rendering is the process of converting a description of a three-dimensional scene into an image. Algorithms for animation, geometric modeling, texturing, and other areas of computer graphics all must pass their results through some sort of rendering process so that they can be made visible in an image. Rendering has become ubiquitous; from movies to games and beyond, it has opened new frontiers for creative expression, entertainment, and visualization.

In the early years of the field, research in rendering focused on solving fundamental problems such as determining which objects are visible from a given viewpoint. As effective solutions to these problems have been found and as richer and more realistic scene descriptions have become available thanks to continued progress in other areas of graphics, modern rendering has grown to include ideas from a broad range of disciplines, including physics and astrophysics, astronomy, biology, psychology and the study of perception, and pure and applied mathematics. The interdisciplinary nature of rendering is one of the reasons that it is such a fascinating area of study.

This book presents a selection of modern rendering algorithms through the documented source code for a complete rendering system. All of the images in this book, including the one on the front cover, were rendered by this software. All of the algorithms that came together to generate these images are described in these pages. The system, `pbrt`, is written using a programming methodology called *literate programming* that mixes prose describing the system with the source code that implements it. We believe that the literate programming approach is a valuable way to introduce ideas in computer graphics and computer science in general. Often, some of the subtleties of an algorithm can be unclear or hidden until it is implemented, so seeing an actual implementation is a good way to acquire a solid understanding of that algorithm's details. Indeed, we believe that deep understanding of a small number of algorithms in this manner provides a stronger base for further study of computer graphics than does superficial understanding of many.

In addition to clarifying how an algorithm is implemented in practice, presenting these algorithms in the context of a complete and nontrivial software system also allows us to address issues in the design and implementation of medium-sized rendering systems. The design of a rendering system's basic abstractions and interfaces has substantial implications for both the elegance of the implementation and the ability to extend it later, yet the trade-offs in this design space are rarely discussed.

`pbrt` and the contents of this book focus exclusively on *photorealistic rendering*, which can be defined variously as the task of generating images that are indistinguishable from those that a camera would capture in a photograph, or as the task of generating images that evoke the same response from a human observer as looking at the actual scene. There are many reasons to focus on photorealism. Photorealistic images are crucial for the movie special-effects industry because computer-generated imagery must often be mixed seamlessly with footage of the real world. In entertainment applications where all of the imagery is synthetic, photorealism is an effective tool for making the observer forget that he or she is looking at an environment that does not actually exist. Finally, photorealism gives a reasonably well-defined metric for evaluating the quality of the rendering system's output.

A consequence of our approach is that this book and the system it describes do not exhaustively cover the state-of-the-art in rendering; many interesting topics in photorealistic rendering will not be introduced either because they don't fit well with the architecture of the software system (e.g., finite-element radiosity algorithms) or because we believed that the pedagogical value of explaining the algorithm was outweighed by the complexity of its implementation. We will note these decisions as they come up and provide pointers to further resources so that the reader can follow up on topics of interest. Many other areas of rendering, including interactive rendering, visualization, and illustrative forms of rendering such as pen-and-ink styles, aren't covered in this book at all. Nevertheless, many of the algorithms and ideas in this system (e.g., algorithms for texture map antialiasing) are applicable to a wider set of rendering styles.

AUDIENCE

Our primary intended audience for this book is students in graduate or upper-level undergraduate computer graphics classes. This book assumes existing knowledge of computer graphics at the level of an introductory college-level course, although certain key concepts such as basic vector geometry and transformations will be reviewed here. For students who do not have experience with programs that have tens of thousands of lines of source code, the literate programming style gives a gentle introduction to this complexity. We pay special attention to explaining the reasoning behind some of the key interfaces and abstractions in the system in order to give these readers a sense of why the system is structured in the way that it is.

Our secondary, but equally important, audiences are advanced graduate students and researchers, software developers in industry, and individuals interested in the fun of writing their own rendering systems. Although many of the ideas in this book will likely be familiar to these readers, seeing explanations of the algorithms presented in the literate style may provide new perspectives. `pbrt` includes implementations of a number of

advanced and/or difficult-to-implement algorithms and techniques, such as subdivision surfaces, Monte Carlo light transport and Metropolis sampling, subsurface scattering, and precomputed light transport algorithms; these should be of particular interest to experienced practitioners in rendering. We hope that delving into one particular organization of a complete and nontrivial rendering system will also be thought provoking to this audience.

OVERVIEW AND GOALS

`pbrt` is based on the *ray-tracing* algorithm. Ray tracing is an elegant technique that has its origins in lens making; Carl Freidrich Gauss traced rays through lenses by hand in the 19^{th} century. Ray-tracing algorithms on computers follow the path of infinitesimal rays of light through the scene until they intersect a surface. This approach gives a simple method for finding the first visible object as seen from any particular position and direction, and is the basis for many rendering algorithms.

`pbrt` was designed and implemented with three main goals in mind: it should be *complete*, it should be *illustrative*, and it should be *physically based*.

Completeness implies that the system should not lack key features found in high-quality commercial rendering systems. In particular, it means that important practical issues, such as antialiasing, robustness, and the ability to efficiently render complex scenes, should all be addressed thoroughly. It is important to consider these issues from the start of the system's design, since these features can have subtle implications for all components of the system and can be quite difficult to retrofit into the system at a later stage of implementation.

Our second goal means that we tried to choose algorithms, data structures, and rendering techniques with care and with an eye toward readability and clarity. Since their implementations will be examined by more readers than is the case for many other rendering systems, we tried to select the most elegant algorithms that we were aware of and implement them as well as possible. This goal also required that the system be small enough for a single person to understand completely. We have implemented `pbrt` using an extensible architecture, with the core of the system implemented in terms of a set of carefully-designed abstract base classes, and as much of the specific functionality as possible in implementations of these base classes. The result is that one doesn't need to understand all of the specific implementations in order to understand the basic structure of the system. This makes it easier to delve deeply into parts of interest and skip others, without losing sight of how the overall system fits together.

There is a tension between the two goals of being complete and being illustrative. Implementing and describing every possible useful technique would not only make this book extremely long, but also would make the system prohibitively complex for most readers. In cases where `pbrt` lacks a particularly useful feature, we have attempted to design the architecture so that the feature could be added without altering the overall system design.

The basic foundations for physically based rendering are the laws of physics and their mathematical expression. `pbrt` was designed to use the correct physical units and concepts for the quantities it computes and the algorithms it implements. When configured

to do so, pbrt can compute images that are *physically correct*; they accurately reflect the lighting as it would be in a real-world version of the scene. One advantage of the decision to use a physical basis is that it gives a concrete standard of program correctness: for simple scenes, where the expected result can be computed in closed form, if pbrt doesn't compute the same result, we know there must be a bug in the implementation. Similarly, if different physically based lighting algorithms in pbrt give different results for the same scene, or if pbrt doesn't give the same results as another physically based renderer, there is certainly an error in one of them. Finally, we believe that this physically based approach to rendering is valuable because it is rigorous. When it is not clear how a particular computation should be performed, physics gives an answer that guarantees a consistent result.

Efficiency was given lower priority than these three goals. Since rendering systems often run for many minutes or hours in the course of generating an image, efficiency is clearly important. However, we have mostly confined ourselves to *algorithmic* efficiency rather than low-level code optimization. In some cases, obvious micro-optimizations take a backseat to clear, well-organized code, although we did make some effort to optimize the parts of the system where most of the computation occurs.

In the course of presenting pbrt and discussing its implementation, we hope to convey some hard-learned lessons from years of rendering research and development. There is more to writing a good renderer than stringing together a set of fast algorithms; making the system both flexible and robust is a difficult task. The system's performance must degrade gracefully as more geometry or light sources are added to it, or as any other axis of complexity is pushed. Numeric stability must be handled carefully, and algorithms that don't waste floating-point precision are critical.

The rewards for developing a system that addresses all these issues are enormous—it is a great pleasure to write a new renderer or add a new feature to an existing renderer and use it to create an image that couldn't be generated before. Our most fundamental goal in writing this book was to bring this opportunity to a wider audience. Readers are encouraged to use the system to render the example scenes in the pbrt software distribution as they progress through the book. Exercises at the end of each chapter suggest modifications to the system that will help clarify its inner workings, and more complex projects to extend the system by adding new features.

The website for this book is located at *www.pbrt.org*. The latest version of the pbrt source code is available from this site and we will also post errata and bug fixes, additional scenes to render, and supplemental utilities. Any bugs in pbrt or errors in this text that are not listed at the website can be reported to the email address *bugs@pbrt.org*. We greatly value your feedback!

CHANGES SINCE THE FIRST EDITION

Six years have passed since the publication of the first edition of this book. In that time, thousands of copies of the book have been sold and the pbrt software has been downloaded thousands of times from the book's website. The pbrt user base has given

us a significant amount of feedback and encouragement, and our experience with the system guided many of the decisions we made in making changes between the version of pbrt presented in the first edition and the updated version described here. In addition to a number of bug fixes, we also made several significant design changes and enhancements:

1. Removal of the plugin architecture. The first version of pbrt used a run-time plugin architecture to dynamically load code for implementations of objects like shapes, lights, integrators, cameras, and other objects that were used in the scene currently being rendered. This approach allowed users to extend pbrt with new object types (e.g., new shape primitives) without recompiling the entire rendering system. This approach initially seemed elegant, but it complicated the task of supporting pbrt on multiple platforms and it made debugging more difficult. The only new usage scenario that it truly enabled (binary-only distributions of pbrt or binary plugins) was actually contrary to our pedagogical and open-source goals. Therefore, the plugin architecture was dropped in this edition.
2. Removal of the image processing pipeline. The first version of pbrt provided a tone-mapping interface that converted high dynamic range (HDR) floating point output images directly into low dynamic range TIFFs for display. This functionality made sense in 2004, as support for HDR images was still sparse. In 2010, however, advances in digital photography have made HDR images commonplace. Although the theory and practice of tone mapping are elegant and worth learning, we decided to focus the new book exclusively on the process of image formation and ignore the topic of image display. Interested readers should read the book written by Reinhard et al. (2005) for a thorough and modern treatment of the HDR image display process.
3. Task parallelism. Multicore architectures are now ubiquitous, and we felt that pbrt would not remain relevant without the ability to scale to the number of locally available cores. We also hope that the parallel programming implementation details documented in this book will help graphics programmers understand some of the subtleties and complexities in writing scalable parallel code (e.g., choosing appropriate task granularities or mutex types), which is still a difficult and too-infrequently taught topic.
4. Appropriateness for "production" rendering. The first version of pbrt was intended exclusively as a pedagogical tool and a stepping-stone for rendering research. Indeed, we made a number of decisions in preparing the first edition that were contrary to use in a production environment, such as limited support for image-based lighting, no support for motion blur, and a photon mapping implementation that wasn't robust in the presence of complex lighting. With much improved support for these features as well as support for subsurface scattering and Metropolis light transport, we feel that pbrt is now much more suitable for rendering very high-quality images of complex environments as it is presented here. The tradeoff in making these improvements is that as the system becomes more feature-complete, it may be harder for instructors to use the new software to create manageable assignments for students. While this is a real concern, we had similar reservations about the first version of pbrt "relieving" students of the burden and benefits of writing their own ray-tracing system from scratch. With

experience from the first edition being used at many universities, we have come to believe that this tradeoff was a good one, and we hope and expect that the new edition will continue to enable high-quality rendering courses.

ACKNOWLEDGMENTS

Pat Hanrahan has contributed to this book in more ways than we could hope to acknowledge; we owe a profound debt to him. He tirelessly argued for clean interfaces and finding the right abstractions to use throughout the system, and his understanding of and approach to rendering deeply influenced its design. His willingness to use `pbrt` and this manuscript in his rendering course at Stanford was enormously helpful, particularly in the early years of its life when it was still in very rough form; his feedback throughout this process has been crucial for bringing the text to its current state. Finally, the group of people that Pat helped assemble at the Stanford Graphics Lab, and the open environment that he fostered, made for an exciting, stimulating, and fertile environment. We feel extremely privileged to have been there.

We owe a debt of gratitude to the many students who used early drafts of this book in courses at Stanford and the University of Virginia between 1999 and 2004. These students provided an enormous amount of feedback about the book and `pbrt`. The teaching assistants for these courses deserve special mention: Tim Purcell, Mike Cammarano, Ian Buck, and Ren Ng at Stanford, and Nolan Goodnight at Virginia. A number of students in those classes gave particularly valuable feedback and sent bug reports and bug fixes; we would especially like to thank Evan Parker and Phil Beatty. A draft of the manuscript of this book was used in classes taught by Bill Mark and Don Fussell at the University of Texas, Austin, and Raghu Machiraju at Ohio State University; their feedback was invaluable, and we are grateful for their adventurousness in incorporating this system into their courses, even while it was still being edited and revised.

Matt Pharr would like to acknowledge colleagues and co-workers in rendering-related endeavors who have been a great source of education and who have substantially influenced his approach to writing renderers and his understanding of the field. Particular thanks go to Craig Kolb, who provided a cornerstone of Matt's early computer graphics education through the freely available source code to the `rayshade` ray-tracing system, and Eric Veach, who has also been generous with his time and expertise. Thanks also to Doug Shult and Stan Eisenstat for formative lessons in mathematics and computer science during high school and college, respectively, and most importantly to Matt's parents, for the education they've provided and continued encouragement along the way. Finally, thanks also to Nick Triantos, Jayant Kolhe, and NVIDIA for their understanding and support through the final stages of the preparation of the first edition of the book.

Greg Humphreys is very grateful to all the professors and TAs who tolerated him when he was an undergraduate at Princeton. Many people encouraged his interest in graphics, specifically Michael Cohen, David Dobkin, Adam Finkelstein, Michael Cox, Gordon Stoll, Patrick Min, and Dan Wallach. Doug Clark, Steve Lyon, and Andy Wolfe also supervised various independent research boondoggles without even laughing once. Once, in a group meeting about a year-long robotics project, Steve Lyon became exasperated and yelled, "Stop telling me why it can't be done, and figure out how to do it!"—an im-

promptu lesson that will never be forgotten. Eric Ristad fired Greg as a summer research assistant after his freshman year (before the summer even began), pawning him off on an unsuspecting Pat Hanrahan and beginning an advising relationship that would span 10 years and both coasts. Finally, Dave Hanson taught Greg that literate programming was a great way to work, and that computer programming can be a beautiful and subtle art form.

We are also grateful to Don Mitchell, for his help with understanding some of the details of sampling and reconstruction; Thomas Kollig and Alexander Keller, for explaining the finer points of low-discrepancy sampling; and Dave Eberly, "Just d'FAQs," Hans-Bernhard Broeker, Steve Westin, and Gernot Hoffmann, for many interesting threads on *comp.graphics.algorithms*. Christer Ericson had a number of suggestions for improving our kd-tree implementation. Christophe Hery helped us with understanding the nuances of subsurface scattering and Peter-Pike Sloan was kind enough to carefully review Chapter 17 on precomputed light transport algorithms.

Many people and organizations have generously supplied us with scenes and models for use in this book and the `pbrt` distribution. Their generosity has been invaluable in helping us create interesting example images throughout the text. The bunny, Buddha, and dragon models are courtesy of the Stanford Computer Graphics Laboratory's scanning repository at *graphics.stanford.edu/data/3Dscanrep/*. The ecosystem scene was created by Oliver Deussen and Bernd Lintermann for a paper by them and collaborators (Deussen, Hanrahan, Lintermann, Mech, Pharr, and Prusinkiewicz 1998). The "killeroo" model is included with permission of Phil Dench and Martin Rezard (3D scan and digital representations by headus, design and clay sculpt by Rezard). The physically accurate smoke data sets were created by Duc Nguyen and Ron Fedkiw. Nolan Goodnight created environment maps with a realistic skylight model. The Cornell Program of Computer Graphics Light Measurement Laboratory allowed us to include measured BRDF data, and Paul Debevec provided numerous high dynamic-range environment maps. Marc Ellens provided spectral data for a variety of light sources, and the spectral RGB measurement data for a variety of displays is courtesy of Tom Lianza at X-Rite.

We are particularly grateful to Guillermo M. Leal Llaguno of Evolución Visual, *www.evvisual.com*, who modeled and rendered the San Miguel scene featured on the cover and in numerous figures in the book. We would also especially like to thank Marko Dabrovic (*www.3lhd.com*) and Mihovil Odak at RNA Studios (*www.rna.hr*), who supplied a bounty of excellent models and scenes, including the Sponza atrium, the Sibenik cathedral, and the Audi TT car model. Many thanks are also due to Florent Boyer (*www.florentboyer.com*), who provided the contemporary house scene used in many of the images in Chapter 15.

We would also like to thank the book's reviewers, all of whom had insightful and constructive feedback about the manuscript at various stages of its progress. We'd particularly like to thank the reviewers who provided feedback on both editions of the book: Ian Ashdown, Per Christensen, Doug Epps, Dan Goldman, Eric Haines, Erik Reinhard, Pete Shirley, Peter-Pike Sloan, Greg Ward, and a host of anonymous reviewers. For the second edition, Janne Kontkanen, Nelson Max, Bill Mark, and Eric Tabellion also contributed numerous helpful suggestions.

We'd like to thank the faculty members at various universities who have used pbrt in their courses, including Emmanuel Agu, Dirk Arnold, Stephen Chenney, Yung-Yu Chuang, Don Fussell, Pat Hanrahan, Bill Mark, Nelson Max, Gary Meyer, Torsten Möller, Rick Parent, Sumanta Pattanaik, and Luiz Velho.

Many people have contributed to not only pbrt but to our own better understanding of rendering through bug reports, patches, and suggestions about better implementation aproaches. A few have made particularly substantial contributions—we would especially like to thank Kevin Egan, John Danks, Volodymyr Kachurovskyi, Solomon Boulos, and Stephen Chenney. In addition, we would also like to thank Rachit Agrawal, Frederick Akalin, Mark Bolstad, Thomas de Bodt, Brian Budge, Mark Colbert, Shaohua Fan, Nigel Fisher, Jeppe Revall Frisvad, Robert G. Graf, Asbjørn Heid, Keith Jeffery, Greg Johnson, Aaron Karp, Donald Knuth, Martin Kraus, Murat Kurt, Larry Lai, Craig McNaughton, Swaminathan Narayanan, Anders Nilsson, Jens Olsson, Vincent Pegoraro, Nils Thuerey, Xiong Wei, Wei-Wei Xu, Arek Zimny, and Matthias Zwicker for their suggestions and bug reports. Finally, we would like to thank the *LuxRender* developers and the *LuxRender* community, particularly Terrence Vergauwen, Jean-Philippe Grimaldi, and Asbjørn Heid; it has been a delight to see the rendering system they have built from pbrt's foundation, and we have learned from reading their source code and implementations of new rendering algorithms.

For the production of the first edition, we would also like to thank Tim Cox (senior editor), for his willingness to take on this slightly unorthodox project and for both his direction and patience throughout the process. We are very grateful to Elisabeth Beller (project manager), who has gone well beyond the call of duty for this book; her ability to keep this complex project in control and on schedule has been remarkable, and we particularly thank her for the measurable impact she has had on the quality of the final result. Thanks also to Rick Camp (editorial assistant) for his many contributions along the way. Paul Anagnostopoulos and Jacqui Scarlott at Windfall Software did the book's composition; their ability to take the authors' homebrew literate programming file format and turn it into high-quality final output while also juggling the multiple unusual types of indexing we asked for is greatly appreciated. Thanks also to Ken DellaPenta (copyeditor) and Jennifer McClain (proofreader) as well as to Max Spector at Chen Design (text and cover designer), and Steve Rath (indexer).

For the second edition, we'd like to thank Greg Chalson who talked us into expanding and updating the book; Greg also ensured that Paul Anagnostopoulos at Windfall Software would again do the book's composition. We'd like to thank Paul again for his efforts in working with this book's production complexity. Finally, we'd also like to thank Todd Green, Paul Gottehrer, and Heather Scherer at Elsevier.

ABOUT THE COVER

The "San Miguel" scene on the cover of the book was modeled and then rendered by Guillermo M. Leal Llaguno of Evolución Visual, *www.evvisual.com*, based on a hacienda that he visited in San Miguel de Allende, Mexico. The scene was modeled in *3ds max* and exported to the pbrt file format with a custom script written by Guillermo. The scene features just over 2.5 million unique triangles and has a total geometric complexity of

10.7 million triangles due to the use of object instancing; the `pbrt` files that describe the scene geometry require 620 MB of on-disk storage. There are a total of 354 texture maps, representing 293 MB of texture data. Final rendering of the cover image at 1496 by 2235 resolution using `pbrt` took over 40 hours of computation on an eight-core Mac Pro computer. The scene is available in the `scenes/sanmiguel` directory of the `pbrt` distribution.

ADDITIONAL READING

Donald Knuth's article *Literate Programming* (Knuth 1984) describes the main ideas behind literate programming as well as his `web` programming environment. The seminal TEX typesetting system was written with `web` and has been published as a series of books (Knuth 1986, Knuth 1993a). More recently, Knuth has published a collection of graph algorithms in literate format in *The Stanford GraphBase* (Knuth 1993b). These programs are enjoyable to read and are excellent presentations of their respective algorithms. The website *www.literateprogramming.com* has pointers to many articles about literate programming, literate programs to download, and a variety of literate programming systems; many refinements have been made since Knuth's original development of the idea.

The only other literate programs we know of that have been published as books are the implementation of the `lcc` compiler, which was written by Christopher Fraser and David Hanson and published as *A Retargetable C Compiler: Design and Implementation* (Fraser and Hanson 1995), and Martin Ruckert's book on the *mp3* audio format, *Understanding MP3* (Ruckert 2005).