Foreword

There have been tremendous advances in the realism of computer-generated images over the last twenty years. This is the result of a great deal of research and is documented in thousands of technical papers. While this effort has resulted in many algorithmic and mathematical tools, it has also resulted in a vast and somewhat impenetrable literature. This literature has conflicting terms, symbols, and often advocates approaches that are simply not practical. As a result, it is very difficult for new people to "get up to speed" and begin developing software to generate realistic images. The most technical part of realistic image generation is dealing with "global illumination." The word "global" refers to the fact that the appearance of an object depends on the light it receives from all other objects. So in this sense, computing lighting even at a single point requires computation using the entire model of the scene. While this might seem like overkill, the visual richness of an image created using a global illumination program is simply not possible with simpler local illumination programs.

This book breaks down the barrier to entry and describes global illumination concepts and algorithms from a modern viewpoint using consistent terms and symbols. While there are good books on specific global illumination topics, this is the first book to address global illumination techniques as a whole. The authors are ideal for such an ambitious project; they have a broad background in rendering and have done significant research in all of the major global illumination topics.

Most of the major theoretical advances in global illumination took place in the 1980s. These included the development of both radiosity and Monte Carlo ray tracing. In the 1990s, it became apparent that none of these algorithms were practical when applied in a straightforward manner. In that time, a more quiet revolution took place as techniques were developed

to make global illumination practical for real-world models. The authors were key players in that revolution, and this book stresses techniques that have been shown to work in the field. The approach of the book has been fine-tuned in a course on global illumination taught by the authors at the annual SIGGRAPH conference, and this has resulted in a clean progression of ideas.

Since *Advanced Global Illumination* was published, it has become my default reference for points related to advanced rendering. I also recommend it to new students at my university who need to absorb twenty years of rendering research without wading through hundreds of dense papers that often have conflicting terminology or, worse, advance concepts that have since been discredited. Rendering images with realistic illumination effects is very rewarding, and it is not hard once the basic concepts are clearly understood. This book describes all of those concepts, and it is a passport to making beautiful and realistic images. Enjoy!

Peter Shirley
May 2006

Table of Contents

Preface

This book is the result of our experience while teaching a course of the same name at the annual ACM SIGGRAPH conference during 2001 and 2002, as well as teaching various graduate-level courses and seminars covering advanced photorealistic rendering topics. When setting up these courses, we always felt that covering the fundamentals gives a much broader insight into how to design a global illumination algorithm, instead of presenting the student with a number of recipes and ready-to-compile code. By explaining the basic building blocks and underlying theory, the reader should be more able to design and implement his own photorealistic rendering algorithms.

We chose *Advanced Global Illumination* as the title because we present topics which are of a more fundamental nature than those which are usually understood by the term *global illumination* or *photorealistic rendering* by the computer graphics enthusiast. Too often, classic ray tracing with some extensions for handling area light sources, or with some heuristics added for indirect illumination, are categorized as global illumination algorithms. In order to know why such approaches fail to cover all possible illumination effects, and exactly why this is the case, it is necessary to understand the fundamental and advanced concepts of the most advanced global illumination algorithms available. The adjective "advanced" is to be read in this way. The professional researcher or Ph.D. student who spends several years of research studying global illumination algorithms may not judge the topics in this book to be "advanced" in this sense, but we think that the majority of computer graphics practitioners will discover many topics here not covered by other books.

However, this does not imply that this book only covers theoretical concepts. Various sections deal with practical issues on how to implement the different building blocks needed to build your own global illumination

algorithm. We hope that the researcher, the graduate and undergraduate student, and the computer graphics enthusiast will find this book interesting to read.

We would like to thank the people from A K Peters, who have been more than helpful during the process of publishing this book, and who have been very patient and encouraging. Especially, we would like to thank Alice Peters, Heather Holcombe, and Jonathan Peters for their understanding in us taking more time to finish this manuscript than originally intended.

We would also like to thank the various research groups and institutions, at which we found the time to work on this book and who gave us the opportunity to teach computer graphics and photorealistic rendering: The Program of Computer Graphics at Cornell University, USA; the Max Planck Institut für Informatik in Saarbrücken, Germany; the Department of Computer Science at the University of Leuven, Belgium; and the Expertise Centre for Digital Media at the University of Limburg, also in Belgium.

The students in our various computer graphics courses over the past years provided us with valuable additional insight on how to adequately explain various topics related to photorealistic rendering. Student feedback in an educational setting is very worthwhile, and we wish to thank them all. We also wish to thank the attendees of our global illumination courses at the ACM SIGGRAPH conferences for the criticism and encouragement they provided.

Last but not least, we would like to thank our families and close friends, who supported us throughout the huge task of writing this book.

Philip Dutré
Philippe Bekaert
Kavita Bala

Leuven, Hasselt, and Ithaca, January 2003

Preface to the Second Edition

Since the first edition of this book was published almost three years ago, we have received quite some feedback. We were very happy to hear that *Advanced Global Illumination* has been used as a textbook in various universities and are grateful for the constructive comments from our readers.

During the last years, the field of global illumination has expanded, and as a result, some new sections have been added. Chapter 5 contains a small section about environment maps. Moreover, we extended Chapters 7 and 8 to include some of the newest developments in scalable rendering and precomputed radiance transfer.

The most significant change probably is the inclusion of exercises at the end of each chapter. We often received requests about homeworks for courses using this book, and so we included a selection of homeworks we have used ourselves during the past years.

Specifically for this second edition of the book, we would like to thank all readers who have provided us with valuable feedback and criticism. Partly due to their comments, various sections in this book have been amended and improved. We especially thank the following persons: Tomas Akenine-Möller, Andreas Baerentzen, Dave Beveridge, Bryan Cool, Jeppe Revall Frisvad, Michael Goesele, Vlastimil Havran, Magnus Jonneryd, Jaroslav Křivánek, Nelson Max, Rick Speer, Derek Young, Koen Yskout. Our apologies to anyone who has contributed but is not mentioned in this list. We would like to thank Bruce Walter for providing us with the images for the book cover.

We would like to thank all the staff at A K Peters, and in particular Alice Peters, who has provided us with the opportunity to publish a second edition of this book. Also, we especially thank our editor Kevin Jackson-Mead, who has assisted us greatly in preparing the final manuscript and managed us skillfully throughout the whole process.

All three of us would also like to thank our families and close friends for making this a rewarding and fun experience; without their support this book would not have been possible.

Philip Dutré
Kavita Bala
Philippe Bekaert

Leuven, Ithaca, and Hasselt, March 2006

1

Introduction

1.1 What Is Realistic Image Synthesis?

Realistic image synthesis is a domain in the field of computer graphics that generates new images from other data. Typically, one starts from a complete description of a three-dimensional scene, specifying the size and location of objects, the material properties of all surfaces of solid objects in the scene, and the position and emission characteristics of light sources. From this data, a new picture is generated, as seen from a virtual camera placed in the scene. The aim is to make these pictures as photorealistic as possible, such that the difference with a real photograph (if the virtual scene would be constructed in reality) is not noticeable. This requires the underlying physical processes regarding materials and the behavior of light to be precisely modeled and simulated. Only by knowing exactly what one is trying to simulate does it become possible to know where simplifications can be introduced in the simulation and how this will affect the resulting pictures.

Generating photorealistic pictures is a very ambitious goal, and it has been one of the major driving forces in computer graphics over the last decades. Visual realism has always been a strong motivation for research in this field, and it is a selling point for many graphics-related, commercially available products. It is expected that this trend will continue in the coming years and that photorealism will remain one of the core fields in rendering.

Photorealistic rendering is not the only rendering paradigm that is used in computer graphics, nor is it the best solution for all rendering applications. Especially in the last couple of years, non-photorealistic rendering has become a field in itself, providing viable alternatives for the photorealistic rendering style. Non-photorealistic rendering (or NPR, as it is commonly called) uses a wide variety of drawing styles that are suited for a more artistic, technical, or educational approach. Drawing styles covered by NPR include pen-and-ink drawings, cartoon-style drawings, technical

illustrations, watercolor painting, and various artistic styles such as impressionism, pointillism, etc. The possibilities are virtually limitless, and the algorithms are driven by trying to recreate a certain style rather than by trying to simulate a physical process found in nature. While there is clearly room for NPR, a variety of applications are interested in the physical simulation of reality.

1.1.1 The Importance of Realistic Image Synthesis

Photorealistic rendering is a rendering style that has many applications in various fields. Early applications were limited by the amount of time it took to compute a single image (usually measured in hours), but recently, interactive techniques have broadened the scope of photorealistic image synthesis considerably.

Film and Visual Effects

Visual effects in the film industry have always been a driving force for the development of new computer graphics techniques. Depending on the rendering style used, feature animations can benefit from global illumination rendering, although this might be limited to a few scenes where more complex lighting configurations are used. Movies with live footage can benefit too, especially when virtual elements are added. In this case, a consistent lighting between the real and virtual elements in the same shot needs to be achieved, in order to avoid implausible lighting effects. Global illumination is necessary to compute the light interaction between those different elements.

Architecture

Architectural design is often quoted as one of the most beneficial applications of photorealistic rendering. It is possible to make visualizations, whether they be still images or interactive walk-throughs, of buildings yet to be constructed. Not only can indoor illumination due to interior lighting be simulated, but outdoor lighting can be considered as well, e.g., the building can be illuminated using various atmospheric conditions at different times of the year, or even various times during the day.

Ergonomic Design of Buildings and Offices

Although not strictly a computer graphics application, the ergonomic design of office rooms or factory halls is very much related to global illumination. Given the design of a building, it is possible to compute the various illumination levels in different parts of the building (e.g., desks, workstations, etc.), and the necessary adjustments can be made to reach

the minimum legal or comfortable requirements by changing the color of paint on the walls, the repositioning of windows, or even the replacement of walls.

Computer Games

Most computer games revolve around fast and interactive action, coupled with a suspension of disbelief in a virtual world. As such, photorealistic rendered imagery probably is a strong cue to draw players into the environments in which their virtual characters are acting. Since interactivity is more important in a gaming context than realistic images, the use of global illumination in games is still somewhat limited but will undoubtedly become more important in the near future.

Lighting Engineering

The design of lights and light fixtures can also benefit from global illumination algorithms. Specific designs can be simulated in virtual environments, such that the effect of the emission patterns of light bulbs can be studied. This requires an accurate measurement and modeling of the characteristics of the emission of the light sources, which is a whole field of study by itself.

Predictive Simulations

Predictive simulations encompass much more than just simulating the look of buildings as described above. Other areas of design are important as well: car design, appliances, consumer electronics, furniture, etc. This all involves designing an object and then simulating how it will look in a real or virtual environment.

Flight and Car Simulators

Simulators used for training, such as flight and car simulators, benefit from having an as accurate as possible visual simulation, e.g., aspects of street lighting are important in car simulators, accurate atmospheric visual simulation is important when designing a flight simulator, etc. Other types of training simulators also use or might use realistic imagery in the future; armed combat, ship navigation, and sports are a few examples.

Advertising

Producing accurate imagery of yet-to-be-produced products is probably a good tool for the advertising world. Not only does the customer have the ability to see what the product looks like when generated using photorealistic rendering, but he would benefit if he could place the product in a known environment, e.g., furniture could be placed, with consistent illumination, in a picture of your own living room.

1.1.2 History of Photorealistic Rendering

This section provides a brief history of photorealistic rendering algorithms and the quest for visual realism. Some more extensive background and history on specific algorithms can also be found in the relevant chapters.

Photorealism in the Precomputer Age

The history of photorealistic rendering, or the quest for visual realism, can be traced throughout the history of art. Although we are mainly interested here in the computer-driven photorealistic rendering programs, it might be useful to look at how the understanding of realistic rendering evolved in the course of history. Medieval and premedieval art is very much iconic in nature: persons and objects are displayed in simplified, often two-dimensional forms, and sizes and shapes are used to reflect the importance of the person displayed, relative positioning in a scene, or other properties.

The real beginning of realistic rendering probably starts with the first use and study of perspective drawings. Especially in Italy during the Renaissance period, various artists were involved in discovering the laws of perspective. Brunelleschi (1377–1446), da Vinci (1452–1519), and Dürer (1471–1528) (to name a few) are well known for their contributions. Later, painters also started to pay attention to the shading aspects. By carefully studying shadows and light reflections, very accurate renderings of real scenes could be produced using classic artistic materials.

Much of the knowledge of photorealistic painting was collected by British landscape artist Joseph Turner (1775–1851), appointed Professor of Perspective at the Royal Academy of Arts in London. He designed a course of six lectures, covering principles such as accurate drawing of light, reflections, and refractions. Some of his sketches show reflections in mirrored and transparent spheres, a true precursor of ray tracing almost 300 years later. In his book, *Secret Knowledge*, British artist David Hockney [73] develops an interesting thesis: Starting in the 15th century, artists began using optical tools to display reality very accurately. Mostly using a camera lucida, they projected imagery onto the canvas and traced the silhouettes and outlines very carefully. Afterwards, several such drawings were composed in a bigger painting, which explains the different perspectives found in various well-known paintings of the era.

It is certainly no coincidence that the trend and developments towards more photorealism in art were somewhat halted with the invention of photography at the beginning of the 19th century (Nicéphore Niépce, 1765–1833). Capturing an image accurately is suddenly not a difficult process anymore. After the invention of photography, art evolved into modern art,

with its various novel ways, not necessarily photorealism, of looking at reality (pointillism, impressionism, cubism, etc.).

Primitive Shading Algorithms

The birth of computer graphics is usually accredited to *SketchPad* [188], the Ph.D. thesis of Ivan Sutherland at the Massachusetts Institute of Technology (M.I.T.) in 1963. Early computer graphics were mostly line drawings, but with the advent of raster graphics, shading algorithms became widely available. Primitive shading algorithms usually attributed a single color to a single polygon, the color being determined by the incident angle of light on the surface. This type of shading gives some cues about shape and orientation but is far away from realistically illuminated objects.

A breakthrough was achieved by Henri Gouraud and Bui Tui Phong, who realized that by interpolation schemes, additional realism in shading can be easily achieved. Gouraud shading [58] computes illumination values at vertices and interpolates these values over the area of a polygon. Phong shading [147] interpolates the normal vectors over the area of a polygon and computes illumination values afterwards, thus better preserving highlights caused by nondiffuse reflection functions. Both techniques are longstanding shading algorithms in computer graphics and are still widely used.

Another major breakthrough for more realism in computer-generated imagery was the use of texture mapping. Using a local two-dimensional coordinate system on an object, it is possible to index a texture map and attribute a color to the local coordinate. Integration in the rendering process involves a two-dimensional coordinate transform from the local coordinate system on the object to the local coordinate system of the texture map. Once texture mapping was able to change the color of points on a surface, it was fairly straightforward to change other attributes as well. Thus, the techniques of bumpmapping, displacement mapping, environment mapping, etc., were added. Texturing remains one of the building blocks for rendering in general.

Additional research was also performed in the area of light-source modeling. Originally only point light sources or directional light sources were used in the earliest rendering algorithms, but fairly soon spotlights, directional lights, and other types of light sources, sometimes emulating those found in lighting design, were introduced. Together with the modeling of light sources, the accurate portrayal of shadows has received much attention. When using point light sources, the computation of shadows can be reduced to a simple visibility problem from a single point of view, but the shadows are sharp and hard. The use of shadow volumes and shadow maps are among the best-known algorithms and still receive attention for improvement.

Ray Tracing

In 1980, *ray tracing*, probably the most popular algorithm in rendering, was introduced by Turner Whitted [194]. Although the principle of tracing rays was used before to generate correct perspective and shadows in the conventional arts, the tracing of rays in computer graphics was a major idea for generating all sorts of photorealistic effects. The original paper used rays for determining visibility through a single pixel (also known as *ray casting*) but also used rays to compute direct illumination and perfect specular and refractive illumination effects. As such, this seminal paper described a major new tool in generating images.

The ray-tracing algorithm has been researched and implemented extensively during the last two decades. Initially, much attention was on efficiency, using well-known techniques such as spatial subdivision and bounding volumes. More and more, the focus was also on lighting effects themselves. By treating ray tracing as a tool for computing integrals, effects such as diffuse reflections and refractions, motion blur, lens effects, etc. could be computed within a single framework. For a nice overview, the reader is referred to [52].

The original paper did not solve the entire global illumination problem but was very influential for later developments. To make a distinction with more modern ray-tracing algorithms, the first algorithm is sometimes referred to as *Whitted-style ray tracing* or *classic ray tracing*. Many present-day global illumination algorithms at the core are ray tracers, in the sense that the basic tool still is a procedure that traces a ray through a three-dimensional scene.

Since a basic ray tracer is rather easy to implement, it is a very popular algorithm to serve as the first step into photorealistic rendering. It is traditional to have undergraduate students implement a ray tracer in many graphics courses. Many computer graphics enthusiasts post their ray tracers on the internet, and many of the more popular rendering packages have ray-tracing roots.

Radiosity

With ray tracing being well underway in the first half of the eighties as the algorithm of choice for realistic rendering, it became clear that ray tracing also had severe limitations. Indirect illumination effects such as color bleeding and diffuse reflections were very difficult to simulate. It was clear that a solution needed to be found if one wanted to produce photorealistic pictures. The answer came in the form of a finite-element method called *radiosity*, named after the radiometric quantity that was computed. The algorithm was developed originally at Cornell University

[56] but, as was the case with ray tracing, spawned many research papers and received lots of attention.

One of the early advantages of radiosity was that it was a scene-based method, as opposed to ray tracing, which is an image-based method. In radiosity, the distribution of light is computed by subdividing the scene into surface elements and computing for each element the correct radiometric value. Once the radiosity value for each surface element was known, the solution could be displayed with existing graphics hardware, using Gouraud shading for smoothing out the radiosity values computed at each vertex or polygon. This made radiosity an algorithm of choice for interactive applications such as scene walk-throughs.

Early radiosity research was centered around computing a faster solution for the linear system of equations that expressed the equilibrium of the light distribution in the scene. Several relaxation techniques were introduced and more or less subdivided the radiosity solvers into *"gathering"* and *"shooting"* algorithms.

Early on, radiosity was limited to diffuse surfaces, and the accuracy of the method was set by the choice of surface elements. Finer details in the shading at a frequency higher than the initial mesh could not be displayed. Hierarchical radiosity proved to be a major step forwards, since the algorithm was now able to adapt its underlying solution mesh to the actual shading values found on those surfaces. Discontinuity meshing was similarly used to precompute accurate meshes that followed the discontinuity lines between umbra and penumbra regions caused by area light sources. The algorithm was also extended by subdividing the hemisphere around surfaces in a mesh as well, such that glossy surfaces could also be handled. On the other side of hierarchical radiosity, clustering algorithms were introduced to compute the illumination for disjunct objects in single clusters. Overall, radiosity has received wide attention comparable to ray tracing but, due to the somewhat more complex underlying mathematics, has not been as popular.

The Rendering Equation

One of the most important concepts for global illumination algorithms, the rendering equation, was introduced by Kajiya in 1986 [85], although in a different form than is used today. In this seminal paper, for the first time, the complete transport equation describing the distribution of light in a scene was described in a computer graphics context. The importance of the rendering equation is that all light transport mechanisms are described using a recursive integral equation, whose kernel contains the various material properties and the visibility function.

Formulating the global illumination problem as the rendering equation allows for a unified approach when computing images. It now became possible to apply any sort of integration mechanism to numerically evaluate the rendering equation. Also, because the recursive nature of the rendering equation required recursive algorithms, and thus stopping conditions, it was more obvious which successive light reflections were ignored, approximated only up to a certain depth, etc. Also, ray-tracing and radiosity algorithms could now be considered as different integration procedures trying to solve the rendering equation. Ray tracing basically could be written down as a succession of recursive quadrature rules, and radiosity algorithms expressed a finite element solution to the same equation.

One of the most influential consequences of the rendering equation was the development of stochastic ray tracing or Monte Carlo ray tracing. Monte Carlo integration schemes use random numbers to evaluate integrals, but they have the nice property that the expected value of the result equals the exact value of the integral. Thus, it became possible, in theory, to compute correct photorealistic images, assuming the algorithm ran long enough.

Multipass Methods

At the end of the eighties, there were two big families of global illumination algorithms: those that used a ray-tracing approach, computing a single color for every pixel on the screen, and those that were based on the radiosity approach, computing a scene-based solution, only generating an image as a post-process. The first class of algorithms is good for mostly specular and refractive indirect illumination, while the second class is better suited for computing diffuse interreflections and allows interactive manipulation.

It was therefore inevitable that a *"best-of-both-worlds"* approach would be developed, using ray tracing and radiosity characteristics in the same algorithm. These algorithms usually consist of multiple passes, hence the name *multipass methods*. Many different variants have been published (e.g., [24], [209], [171]). A multipass method usually consists of a radiosity pass computing the indirect diffuse illumination, followed by a ray-tracing pass computing the specular light transport, while picking up radiosity values from the first pass. Care has to be taken that some light transport modes are not computed twice, otherwise the image would be too bright in some areas. More than two passes are possible, each pass dedicated to computing a specific aspect of the total light transport.

Algorithms that store partial solutions of the light distribution in the scene, such as the *RADIANCE* algorithm [219] or photon mapping, can be considered multipass algorithms as well. The photon mapping algorithm has especially received a lot of attention in research literature and is widely

considered to be an efficient and accurate algorithm to solve the global illumination problem.

Current Developments

Currently, lots of attention is given to interactive applications using global illumination algorithms. These usually involve a clever combination of storage and reuse of partial solutions, multipass algorithms, etc.

Also, more and more use is made of photographs of real objects or scenes, which are integrated into virtual environments. The problem is that one wants to keep a consistent illumination, and image-based lighting techniques have proposed some elegant solutions.

As far as the authors can see, global illumination and photorealistic rendering will likely remain a major influence in computer graphics developments of the future.

1.1.3 A Framework for Global Illumination Algorithms

When looking at the development of global illumination algorithms over the past 20 years, one sees a collection of widely different approaches, as well as variants of the same approach. Especially for the light transport simulation, one can make a distinction between different paradigms: pixel-oriented versus scene-oriented, diffuse versus specular surfaces, deterministic versus Monte Carlo integration, shooting versus gathering, etc. These differences are important because they affect the accuracy of the final image, but a wider framework for a complete global illumination pipeline also involves other aspects such as data acquisition and image display.

A framework for realistic image synthesis that combines these different aspects was described in a paper by the same name by Greenberg et al. [59]. The framework presented in this paper encompasses different aspects of a full photorealistic rendering system and provides a general overview of how photorealistic rendering algorithms have evolved over time.

A photorealistic rendering system can be thought of as consisting of three main stages: measurement and acquisition of scene data, the light transport simulation, and the visual display.

Measurement and Acquisition

This part of the framework includes measuring and modeling the BRDF of materials to be used in the virtual scene, as well as emission characteristics of light sources. By comparing the goniometric data, one is able to verify the accuracy of the models and measurements.

Light Transport

The light transport phase takes the data describing the geometry of the scene, materials, and light sources and computes the distribution of light in the scene. This is the main bulk of what is usually called a global illumination algorithm. The result is radiometric values in the image plane, which can be verified by, for example, comparing real photographs with computed pictures.

Visual Display

The matrix of radiometric values needs to be displayed on a screen or printer. A tone-mapping operator is necessary to transform the raw radiometric data into pixel colors. This transformation uses a model of the human visual system, such that the same visual sensation is caused by looking at the displayed picture as by looking at the real scene.

If it is known what error can be tolerated in last stage, this error can be translated into tolerances for the light transport phase, and eventually to the measurement phase. The critical notion of this framework is that perceptual accuracy on the part of the human observer, not radiometric accuracy, should be the driving force when designing algorithms.

1.2 Structure of this Book

As mentioned before, the content of this book is geared towards understanding the fundamental principles of global illumination algorithms. The division of the content into several chapters reflects this. We strongly believe that only by treating the fundamental and basic building blocks in a thorough way can a full understanding of photorealistic rendering be achieved.

The chapters are organized as follows:

- Chapter 1 provides a general introduction to global illumination, outlines the importance of global illumination in the field of computer graphics, and provides a short history of global illumination algorithms.

- Radiometry and the rendering equation are covered in Chapter 2. A good understanding of radiometry is necessary for understanding global illumination algorithms. We only cover those aspects that we need to design global illumination software. The characteristics and nature of the bidirectional reflectance distribution function (BRDF)

are covered in detail, as well as how the definition of the BRDF gives rise to the rendering equation.

- Chapter 3 explains the principle of Monte Carlo integration, a versatile technique that is used in almost all recent global illumination algorithms. The key concepts are explained, but again, only to the level that we need to understand and adequately explain the chapters that follow.

- Chapter 4 puts the rendering equation in a somewhat broader context and gives some general insights into several strategies on how a global illumination algorithm can be designed.

- Chapter 5 gives all the details about stochastic ray tracing. Starting from the rendering equation and using Monte Carlo integration as a tool, several algorithms are deduced for computing various lighting effects. Special attention is given to the computation of direct illumination.

- Stochastic radiosity is covered in Chapter 6 and complements the previous chapter. It offers a very profound overview on the various Monte Carlo radiosity approaches that matured only recently.

- Chapter 7 provides an overview of hybrid methods, which builds on the principles of stochastic ray tracing and radiosity. Various algorithms are explained in detail, with references for further study.

- Chapter 8 covers a number of topics that receive attention in current research, including participating media, subsurface scattering, tone mapping, human visual perception, and strategies for computing global illumination very rapidly.

- Appendix A describes an API for global illumination, a set of object classes that encapsulates and hides the technical details of material and geometry representation and ray casting. This API allows concise and efficient implementations of the algorithms discussed in this book. An example implementation of a light tracer, a path tracer, and a bidirectional path tracer are given.

- Appendix B gives a review of solid angles and hemispherical geometry.

- Appendix C contains technical details omitted from Chapter 6.

1.3 How to Use this Book

This book is the result of teaching various classes about advanced rendering algorithms, and we think that if this book is used as a textbook, it should be a course at the graduate level.

Students that wish to take a class that uses this book should have taken at least one other computer graphics course. One course might be a general introduction to computer graphics, while another course might be project-oriented and focus on some aspects of animation, ray tracing, or modeling. Also, familiarity with probability theory and calculus is required, since otherwise the concepts of the rendering equation and Monte Carlo integration will be hard to explain. Some knowledge about physics might help, although we usually found it was not strictly necessary.

We have added exercises to each chapter in this edition. These exercises are based on assignments we have used ourselves when teaching this course at the graduate level and so have gone through some scrutiny as to whether they have the appropriate difficulty level.

In all of our assignments for our own courses, we provided the students with a basic ray-tracing framework. This skeleton ray tracer is kept very simple, such that the focus can be put entirely on implementing physically correct algorithms. Having the students themselves implement a (basic) ray tracer from scratch is, in our opinion, not a good assignment, since students will be mostly bothered by the nuts and bolts of ray-object intersections, parsing an input file, image viewing, etc.

In case the instructor wants to put together his or her own assignments, here are some suggestions based on our experience:

- Homework 1 might include some problems on radiometry to make students familiar with the concepts of radiometry and make them think about the definition of radiance. A typical exercise could be to compute the radiance reaching earth from the sun, or the radiosity value incident on a square surface under various conditions.

- Homework 2 could be a programming exercise in which students are provided with a basic ray-tracing program. The students would then have to add a specific BRDF model and render a few pictures.

- Homework 3 would extend on the ray tracer from Homework 2. Students could be allowed to add specific lighting effects, such as various ways of computing direct illumination. Also, they could be asked to experiment with different sampling techniques and see what the effect is on the resulting images.

- A number of problems about which global illumination algorithm to use in specific situations could be the subject of Homework 4. For example, scenes could be given with a high number of light sources, a significant amount of specular materials, some unusual geometric configuration, etc. This could be a written exercise, in which the student does not necessarily have to implement his or her ideas, but merely sketch them on paper. Thus, students can design any algorithm they wish without the burden of actually implementing it.
- Studying and presenting a recent research paper would be a good topic for Homework 5 and would also be a good conclusion of the entire course.

Additionally, various problems discussed in the different chapters can be used as homework assignments or can serve as a problem to start a class discussion.

2

The Physics of Light Transport

The goal of rendering algorithms is to create images that accurately represent the appearance of objects in scenes. For every pixel in an image, these algorithms must find the objects that are visible at that pixel and then display their "appearance" to the user. What does the term "appearance" mean? What quantity of light energy must be measured to capture "appearance"? How is this energy computed? These are the questions that this chapter will address.

In this chapter, we present key concepts and definitions required to formulate the problem that global illumination algorithms must solve. In Section 2.1, we present a brief history of optics to motivate the basic assumptions that rendering algorithms make about the behavior of light (Section 2.2). In Section 2.3, we define radiometric terms and their relations to each other. Section 2.4 describes the sources of lights in scenes; in Section 2.5, we present the bidirectional distribution function, which captures the interaction of light with surfaces. Using these definitions, we present the rendering equation in Section 2.6, a mathematical formulation of the equilibrium distribution of light energy in a scene. We also formulate the notion of importance in Section 2.7. Finally, in Section 2.8, we present the measurement equation, which is the equation that global illumination algorithms must solve to compute images. In the rest of this book, we will discuss how global illumination algorithms solve the measurement equation.

2.1 Brief History

The history of the science of optics spans about three thousand years of human history. We briefly summarize relevant events based mostly on the history included by Hecht and Zajac in their book *Optics* [68]. The Greek philosophers (around 350 B.C.), including Pythagoras, Democritus, Empedocles, Plato, and Aristotle among others, evolved theories of the nature

of light. In fact, Aristotle's theories were quite similar to the ether theory of the nineteenth century. However, the Greeks incorrectly believed that vision involved emanations from the eye to the object perceived. By 300 B.C. the rectilinear propagation of light was known, and Euclid described the law of reflection. Cleomedes (50 A.D.) and Ptolemy (130 A.D.) did early work on studying the phenomenon of refraction.

The field of optics stayed mostly dormant during the Dark Ages with the exception of the contribution of Ibn-al-Haitham (also known as Al-hazen); Al-hazen refined the law of reflection specifying that the angles of incidence and reflection lie in the same plane, normal to the interface. In fact, except for the contributions of Robert Grosseteste (1175–1253) and Roger Bacon (1215–1294) the field of optics did not see major activity until the seventeenth century.

Optics became an exciting area of research again with the invention of telescopes and microscopes early in the seventeenth century. In 1611, Johannes Kepler discovered total internal reflection and described the small angle approximation to the law of refraction. In 1621, Willebrord Snell made a major discovery: the *law of refraction*; the formulation of this law in terms of sines was later published by René Descartes. In 1657, Pierre de Fermat rederived the law of refraction from his own *principle of least time*, which states that a ray of light follows the path that takes it to its destination in the shortest time.

Diffraction, the phenomenon where light "bends" around obstructing objects, was observed by Grimaldi (1618–1683) and Hooke (1635–1703). Hooke first proposed the wave theory of light to explain this behavior. Christian Huygens (1629–1695) considerably extended on the wave theory of light. He was able to derive the laws of reflection and refraction using this theory; he also discovered the phenomenon of *polarization* during his experiments.

Contemporaneously, Isaac Newton (1642–1727) observed *dispersion*, where white light splits into its component colors when it passes through a prism. He concluded that sunlight is composed of light of different colors, which are refracted by glass to different extents. Newton, over the course of his research, increasingly embraced the emission (corpuscular) theory of light over the wave theory.

Thus, in the beginning of the nineteenth century, there were two conflicting theories of the behavior of light: the particle (emission/corpuscular) theory and the wave theory. In 1801, Thomas Young described his *principle of interference* based on his famous double-slit experiment, thus providing experimental support for the wave theory of light. However, due to the weight of Newton's influence, his theory was not well-received. Independently, in 1816, Augustin Jean Fresnel presented a rigorous treatment of

3

Monte Carlo Methods

This chapter introduces the concept of Monte Carlo integration and reviews some basic concepts in probability theory. We also present techniques to create better distributions of samples. More details on Monte Carlo methods can be found in Kalos and Whitlock [86], Hammersley and Handscomb [62], and Spanier and Gelbard [183]. References on quasi–Monte Carlo methods include Niederreiter [132].

3.1 Brief History

The term "Monte Carlo" was coined in the 1940s, at the advent of electronic computing, to describe mathematical techniques that use statistical sampling to simulate phenomena or evaluate values of functions. These techniques were originally devised to simulate neutron transport by scientists such as Stanislaw Ulam, John von Neumann, and Nicholas Metropolis, among others, who were working on the development of nuclear weapons. However, early examples of computations that can be defined as Monte Carlo exist, though without the use of computers to draw samples. One of the earliest documented examples of a Monte Carlo computation was done by Comte de Buffon in 1677. He conducted an experiment in which a needle of length L was thrown at random on a horizontal plane with lines drawn at a distance d apart ($d > L$). He repeated the experiment many times to estimate the probability P that the needle would intersect one of these lines. He also analytically evaluated P as

$$P = \frac{2L}{\pi d}.$$

Laplace later suggested that this technique of repeated experimentation could be used to compute an estimated value of π. Kalos and Whitlock [86] present early examples of Monte Carlo methods.

3.2 Why Are Monte Carlo Techniques Useful?

Consider a problem that must be solved, for example, computing the value of the integration of a function with respect to an appropriately defined measure over a domain. The Monte Carlo approach to solving this problem would be to define a random variable such that the expected value of that random variable would be the solution to the problem. Samples of this random variable are then drawn and averaged to compute an estimate of the expected value of the random variable. This estimated expected value is an approximation to the solution of the problem we originally wanted to solve.

One major strength of the Monte Carlo approach lies in its conceptual simplicity; once an appropriate random variable is found, the computation consists of sampling the random variable and averaging the estimates obtained from the sample. Another advantage of Monte Carlo techniques is that they can be applied to a wide range of problems. It is intuitive that Monte Carlo techniques would apply to problems that are stochastic in nature, for example, transport problems in nuclear physics. However, Monte Carlo techniques are applicable to an even wider range of problems, for example, problems that require the higher-dimensional integration of complicated functions. In fact, for these problems, Monte Carlo techniques are often the only feasible solution.

One disadvantage of Monte Carlo techniques is their relatively slow convergence rate of $\frac{1}{\sqrt{N}}$, where N is the number of samples (see Section 3.4). As a consequence, several variance reduction techniques have been developed in the field, discussed in this chapter. However, it should be noted that despite all these optimizations, Monte Carlo techniques still converge quite slowly and, therefore, are not used unless there are no viable alternatives. For example, even though Monte Carlo techniques are often illustrated using one-dimensional examples, they are not typically the most efficient solution technique for problems of this kind. But there are problems for which Monte Carlo methods are the only feasible solution technique: higher-dimensional integrals and integrals with nonsmooth integrands, among others.

3.3 Review of Probability Theory

In this section, we briefly review important concepts from probability theory. A Monte Carlo process is a sequence of random events. Often, a numerical outcome can be associated with each possible event. For exam-

4

Strategies for Computing Light Transport

4.1 Formulation of the Rendering Equation

The global illumination problem is basically a transport problem. Energy emitted by light sources is transported by means of reflections and refractions in a three-dimensional environment. We are interested in the energy equilibrium of the illumination in the environment. Since the human eye is sensitive to radiance values, and since we want to compute photorealistic images, we are primarily interested in radiance values or average radiance values computed over certain areas and solid angles in the scene. The latter means that we should compute flux values for several areas of interest, which will be referred to as sets. The exact geometric shape of these sets can vary substantially, depending on the requested level of accuracy. As will be explained in subsequent chapters, ray tracing algorithms define sets as surface points visible through a pixel, with regard to the aperture of the eye. Radiosity algorithms often define sets as surface patches with the reflecting hemisphere as the directional component (Figure 4.1). Other algorithms might follow different approaches, but the common factor is always that for a number of finite surface elements and solid angle combinations, average radiance values need to be computed.

As explained in Chapter 2, the fundamental transport equation used to describe the global illumination problem is called the rendering equation and was first introduced into the field of computer graphics by Kajiya [85]. The rendering equation describes the transport of radiance through a three-dimensional environment. It is the integral equation formulation of the definition of the BRDF and adds the self-emittance of surface points at light sources as an initialization function. The self-emitted energy of light sources is necessary to provide the environment with some starting energy.

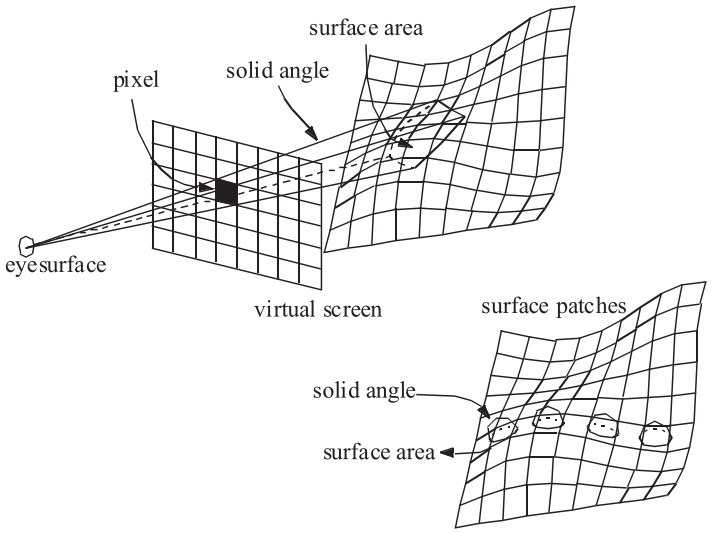

Figure 4.1. Sets of surface points and directions for ray tracing and radiosity algorithms.

The radiance leaving some point x, in direction Θ, is written as:

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi . \tag{4.1}$$

The rendering equation tells us that the exitant radiance emitted by a point x in a direction Θ equals the self-emitted exitant radiance at that point and in that direction, plus any incident radiance from the illuminating hemisphere that is reflected at x in direction Θ. This is illustrated in Figure 4.2.

Emission can result from various physical processes, e.g., heat or chemical reactions. The emission can also be time-dependent for a single surface point and direction, as is the case with phosphorescence. In the context of global illumination algorithms, one usually is not interested in the nature of the source of the self-emitted radiance of surfaces. Self-emitted radiance is merely considered as a function of position and direction.

As was shown in Chapter 2, it is possible to transform the rendering equation from an integral over the hemisphere to an integral over all surfaces in the scene. Both the hemispherical and area formulation contain exitant and incident radiance functions. We know that radiance remains unchanged along straight paths, so we can easily transform exitant radi-

5

Stochastic Path-Tracing Algorithms

This chapter discusses a class of algorithms for computing global illumination pictures known as path-tracing algorithms[1]. The common aspect of these algorithms is that they generate light transport paths between light sources and the points in the scene for which we want to compute radiance values. Another, although less vital, characteristic is that they usually compute radiance values for each individual pixel directly. As such, these algorithms are pixel-driven, but many of the principles outlined here can be equally applied to other classes of light transport algorithms, such as finite element techniques (to be discussed in the next chapter).

First, we present a brief history of path-tracing algorithms in the context of global illumination algorithms (Section 5.1). Then, we discuss the camera set-up that is common to most pixel-driven rendering algorithms (Section 5.2) and introduce a simple path-tracing algorithm in Section 5.3. In Section 5.4, we introduce various methods for computing the direct illumination in a scene, followed by similar sections for the special case of environment map illumination (Section 5.5) and indirect illumination (Section 5.6). Finally, in Section 5.7, the light-tracing algorithm is discussed, which is the dual algorithm of ray tracing.

5.1 Brief History

Path-tracing algorithms for global illumination solutions started with the seminal paper on ray tracing by Whitted [194]. This paper described a novel way for extending the ray-casting algorithm to determine visible surfaces in a scene [4] to include perfect specular reflections and refractions.

[1] The terms ray tracing and path tracing are often used interchangeably in literature. Some prefer to use the term path tracing for a variant of ray tracing where rays do not split into multiple rays at surface points.

At the time, ray tracing was a very slow algorithm due to the number of rays that had to be traced through the scene, such that many techniques were developed for speeding up the ray-scene intersection test (see [52] for a good overview).

In 1984, Cook et al. [34] described stochastic ray tracing. Rays were distributed over several dimensions, such that glossy reflections and refractions, and other effects such as motion blur and depth of field, could be simulated in a coherent framework.

The paper of Kajiya [85] applied ray tracing to the rendering equation, which described the physical transport of light (see Chapter 2). This technique allowed full global illumination effects to be rendered, including all possible interreflections between any types of surfaces.

Other Monte Carlo sampling techniques were applied to the rendering equation, the most complete being bidirectional ray tracing, introduced by Lafortune [100] and Veach [200].

5.2 Ray-Tracing Set-Up

In order to compute a global illumination picture, we need to attribute a radiance value L_{pixel} to each pixel in the final image. This value is a weighted measure of radiance values incident on the image plane, along a ray coming from the scene, passing through the pixel, and pointing to the eye (Figure 5.1). This is best described by a weighted integral over the image plane:

$$\begin{aligned} L_{pixel} &= \int_{imageplane} L(p \rightarrow eye)h(p)dp \\ &= \int_{imageplane} L(x \rightarrow eye)h(p)dp, \end{aligned} \tag{5.1}$$

with p being a point on the image plane, and $h(p)$ a weighting or filtering function. x is the visible point seen from the *eye* through p. Often, $h(p)$ equals a simple box filter such that the final radiance value is computed by uniformly averaging the incident radiance values over the area of the pixel. A more complex camera model is described in [95].

The complete ray-tracing set-up refers to the specific configuration of scene, camera, and pixels, with the specific purpose to compute radiance values for each pixel directly. We need to know the camera position and orientation, and the resolution of the target image. We assume the image is centered along the viewing axis. To evaluate $L(p \rightarrow eye)$, a ray is cast from the eye through p, in order to find x. Since $L(p \rightarrow eye) = L(x \rightarrow \overrightarrow{xp})$, we can compute this radiance value using the rendering equation.

6

Stochastic Radiosity

The algorithms discussed in the previous chapter directly compute the intensity of light passing though the pixels of the virtual screen. In contrast, this chapter covers methods that compute a so-called *world space* representation of the illumination in a three-dimensional scene. Very often, this object space representation consists of the average diffuse illumination on triangles or convex quadrilaterals into which a three-dimensional model has been tessellated. There are, however, plenty of other possibilities, too. Since diffuse illumination is best modeled by a quantity called *radiosity* (see Section 2.3.1), such methods are usually called *radiosity methods*.

The main advantage of computing the illumination in object space is that generating new views of a model takes less work, compared to rendering from scratch. For instance, graphics hardware can be used for real-time rendering of an "illuminated" model, with colors derived from the precomputed average diffuse illumination. Also, path tracing can be augmented to exploit precomputed illumination in object space, allowing very high image quality. The combination of path tracing after a radiosity method is an example of a two-pass method. Two-pass methods, and other hybrid methods, are the topic of Chapter 7.

The most well-known algorithm for computing an object space representation of illumination is the *classic radiosity method* [56, 28, 133]. In this chapter, we will present a brief overview of the classic radiosity method (Section 6.1). More introductory or more in-depth coverage of the classic radiosity method can be found in textbooks such as [29, 172]. We will focus on a range of radiosity methods that matured only recently, since the publication of these books. In particular, we describe three classes of radiosity algorithms, based on stochastic sampling, introduced in Chapter 3.

The first class, called *stochastic relaxation* methods (Section 6.3), is based on stochastic adaptations of classic iterative solution methods for linear systems such as the Jacobi, Gauss-Seidel, or Southwell iterative methods.

The solution of linear systems, such as those that occur in the classic radiosity method, is one of the earliest applications of the Monte Carlo method [50, 224]. They are based on the notion of a *discrete random walk*. Their application to radiosity, which leads to algorithms we call *discrete random walk radiosity* methods, is discussed in Section 6.4.

The third class of Monte Carlo radiosity methods (Section 6.5) is very similar to the random walk methods for linear systems but solves the radiosity or rendering integral equation directly, rather than the radiosity linear system. The random walks of these methods are nothing but simulated photon trajectories. The density of surface hit points of such trajectories will be shown to be proportional to radiosity. Various *density estimation* methods known from statistics [175] can be used in order to estimate radiosity from the photon trajectory hit points.

These three classes of Monte Carlo radiosity methods can be made more efficient by applying *variance-reduction techniques* and *low-discrepancy sampling*, which have been discussed in general in Chapter 3. The main techniques are covered in Section 6.6.

This chapter concludes with a discussion of how adaptive meshing, hierarchical refinement, and clustering techniques can be incorporated into Monte Carlo radiosity (Section 6.7). Combined with adaptive meshing, hierarchical refinement, and clustering, Monte Carlo radiosity algorithms allow us to precompute, on a state-of-the-art PC, the illumination in three-dimensional scenes consisting of milions of polygons, such as models of large and complex buildings.

Monte Carlo radiosity methods all share one very important feature: unlike other radiosity algorithms, they do not require the computation and storage of so-called *form factors* (Section 6.1). This is possible because form factors can be interpreted as probabilities that can be sampled efficiently (Section 6.2). The photon density estimation algorithms in Section 6.5 do not even require form factors at all. Because the nasty problems of accurate form factor computation and their storage are avoided, Monte Carlo radiosity methods can handle much larger models in a reliable way. They are also significantly easier to implement and use than other radiosity methods. In addition, they provide visual feedback very early on and converge gracefully. Often, they are much faster, too.

In this chapter, we will place a large number of (at first sight) unrelated algorithms in a common perspective and compare them to each other. We will do so by analyzing the variance of the underlying Monte Carlo estimators (Section 3.4.4). The same techniques can be used to analyze other Monte Carlo rendering algorithms, but they are easier to illustrate for diffuse illumination, as is done in this chapter.

7

Hybrid Algorithms

Chapters 5 and 6 described two of the most popular global illumination algorithms: ray tracing and radiosity. These algorithms have evolved significantly since they were first introduced, but mainly, the core ideas for both are still the same: a ray-tracing algorithm computes radiance values for every pixel in the final image by generating paths between the pixel and the light sources; a radiosity algorithm computes a radiance value for every mesh element in the scene, after which this solution is displayed using any method that can project polygons to the screen.

This chapter focuses on algorithms that try to combine the best of both worlds. These algorithms often use various elements from the previously mentioned approaches, and therefore we call them hybrid algorithms.

7.1 Final Gathering

Once a radiosity solution is computed and an image of the scene is generated, Gouraud shading is often used to interpolate between radiance values at vertices of the mesh, thus obtaining a smoothly shaded image. This technique can miss significant shading features. It is often difficult to generate accurate shadows; shadows may creep under surfaces (shadow leaks and light leaks), Mach-band effects may occur, and other secondary illumination effects containing features with a frequency higher than that which the mesh can represent are also possible.

One way of solving this is to consider the radiosity solution to be a coarse precomputed solution of the light distribution in the scene. During a second phase, when the image is actually generated, a more accurate per-pixel illumination value is computed, which is based on the ray-tracing algorithm.

As was explained in Chapter 5, the ray-tracing set-up for computing the radiance for a pixel is given by

$$L_{pixel} = \int_{imageplane} L(p \rightarrow eye)h(p)dp.$$

$L(p \rightarrow eye)$ equals $L(x \rightarrow \Theta)$ with x being the visible point in the scene and Θ the direction from x towards the eye. Suppose we have a precomputed radiance solution in a diffuse scene, given by $\widetilde{L}(y)$ for every surface point y. We can then acquire the value of $L(x \rightarrow \Theta)$ by writing the rendering equation, approximating the radiance distribution in the kernel of the transport equation by $\widetilde{L}(y)$:

$$L(x \rightarrow \Theta) = L(x) = L_e(x) + f_r(x) \int_A \widetilde{L}(y)G(x,y)V(x,y)dA_y \qquad (7.1)$$

or equivalently, using integration over the hemisphere,

$$L(x) = L_e(x) + f_r(x) \int_{\Omega_x} \widetilde{L}(r(x,\Psi)) \cos(N_x, \Psi)d\omega_\Psi. \qquad (7.2)$$

This integral can now be evaluated using Monte Carlo integration. The main difference with the stochastic ray-tracing algorithm is that there is no recursive evaluation of the radiance distribution, since it is substituted by the precomputed radiosity solution. Thus, one gains the advantage of using an accurate per-pixel method, using a fast precomputed finite element method.

Various sampling strategies can now be used to evaluate either Equation 7.1 or 7.2. In a diffuse scene, with a constant radiance value $\widetilde{L}_j$ for each surface element j, the above equation can also be rewritten as

$$L(x) = L_e(x) + f_r(x) \sum_j \widetilde{L}_j \int_{A_j} G(x,y)V(x,y)dA_y. \qquad (7.3)$$

7.1.1 Simple Hemisphere Sampling

The most straightforward approach is to sample random directions over the hemisphere and evaluate $\widetilde{L}$ at the nearest intersection point. This strategy is very similar to simple stochastic ray tracing (Section 5.3) and will result in a lot of noise in the final image. The reason is the same as with stochastic ray tracing: light sources will be missed by just randomly sampling the hemisphere. Therefore, splitting the integral into a direct and indirect term is a good approach for increasing the accuracy.

To save time, only the direct illumination can be computed using a per-pixel gathering step [167], and the indirect illumination can be read out

8

The Quest for Ultimate Realism and Speed

In this last chapter, we cover a number of topics that are the subject of ongoing research. Indeed, the quest for realism and speed has not yet come to an end.

While deriving the rendering equation in Chapter 2, several restrictions were imposed on light transport. We assumed that wave effects could be ignored and that radiance is conserved along its path between mutually visible surfaces. We also assumed that light scattering happens instantaneously; that scattered light has the same wavelength as the incident beam; and that it scatters from the same location where it hits a surface. This is not always true. We start this chapter with a discussion of how to deal with participating media, translucent objects, and phenomena such as polarization, diffraction, interference, fluorescence, and phosphorescence, which do not fall within our assumptions. We need to refine our light transport model in order to obtain high realism when these phenomena come into play. Fortunately, most of the algorithms previously covered in this book can be extended rather easily to handle these phenomena, although some new and specific approaches exist as well.

Radiometry is, however, only part of the story, albeit an important part. Most often, computer graphics images are consumed by human observers, looking at a printed picture or a computer screen, or watching a computer graphics movie in a movie theater. Unfortunately, current display systems are not nearly capable of reproducing the wide range of light intensities that occurs in nature and that results from our accurate light transport simulations. These radiometric values need to be transformed in some way to display colors. For good realism, this transformation should take into account the response of the human vision system, which is known to be sophisticated and highly nonlinear. Human visual perception can also be

exploited to avoid computing detail that one wouldn't notice anyway, thus saving computation time.

The last part of this chapter deals with rendering speed. We cover how frame-to-frame coherence can be exploited in order to more rapidly render computer animation movies or walk-throughs of nondiffuse static environments. Very recently, a number of approaches have appeared that go even further on this track and achieve interactive global illumination, without predefined animation script or camera path.

8.1 Beyond the Rendering Equation

8.1.1 Participating Media

We assumed in Chapter 2 that radiance is conserved along its path between unoccluded surfaces. The underlying idea was that all photons leaving the first surface needed to land on the second one because nothing could happen to them along their path of flight. As everyone who has ever been outside in mist or foggy weather conditions knows, this is not always true. Photons reflected or emitted by a car in front of us on the road for instance, will often not reach us. They will rather be absorbed or scattered by billions of tiny water or fog droplets immersed in the air. At the same time, light coming from the sky above will be scattered towards us. The net effect is that distant objects fade away in gray. Even clear air itself causes photons to be scattered or absorbed. This is evident when looking at a distant mountain range, and it causes an effect known as *aerial perspective*. Clouds in the sky scatter and absorb sunlight strongly, although they don't have a real surface boundary separating them from the air around. Surfaces are also not needed for light emission, as in the example of a candle flame.

Our assumption of radiance conservation between surfaces is only true in a vacuum. In that case, the relation between emitted radiance and incident radiance at mutually visible surface points x and y along direction Θ is given by the simple relation

$$L(x \rightarrow \Theta) = L(y \leftarrow -\Theta). \tag{8.1}$$

If a vacuum is not filling the space between object surfaces, this will cause photons to change direction and to transform into other forms of energy. In the case of the candle flame, other forms of energy are also transformed into visible light photons. We now discuss how these phenomena can be integrated into our light transport framework. We start by studying how they affect the relation (Equation 8.1) between emitted radiance and incident radiance at mutually visible surface points x and y.

9

Conclusion

9.1 Achievements of Photorealistic Rendering

Photorealistic rendering and global illumination algorithms have come a long way since the publication of the first recursive ray-tracing algorithm in 1979. There has been a gradual evolution from simple algorithms, some of them deemed to be hacks by today's standards, to very advanced, fully physically based rendering algorithms.

It is now possible, within a reasonable amount of time, to generate an image that is indistinguishable from a photograph of a real scene. This has been achieved by carefully researching the physical processes that form the basis of photorealistic rendering: light-material interaction, light transport, and the psychophysical aspects of the human visual system. In each of these domains, extensive research literature is available. In this book, we have tried to give an overview of some of these aspects, mostly focusing on the light transport mechanism. As in most modern algorithms, we strongly believe that a good understanding of all fundamental issues is the key to well-designed global illumination light transport algorithms.

Global illumination has not yet found its way to many mainstream applications, but some use has already been made in feature-animation films and to a limited extent in some computer games. High-quality rendering of architectural designs has become more common (although still unusual), and car manufacturers have become more aware of the possibilities of rendering cars in real virtual environments for glossy advertisements. Moreover, recent advances have indicated that full interactive ray tracing is already a possibility for specialist applications and machinery.

As such, photorealistic rendering has certainly propelled forward the development of high-quality visualization techniques.

9.2 Unresolved Issues in Photorealistic Rendering

Research in photorealistic rendering is still alive and well, with a large number of publications devoted to the topic every year. There are still a number of unresolved issues, which will undoubtedly form the topic of future research. We have tried to compile a few topics we think will become heavily researched in the near future:

Acquisition and modeling of BRDFs. There has been quite some effort to measure the BRDF of real materials and to design usable models for use in computer graphics, but this whole field still needs a lot of research to provide us with reliable, accurate, and cheap ways to evaluate BRDF models. Measuring devices such as gonio-reflectometers should be made adaptive, such that they can measure more samples in those areas of the BRDF where more accuracy is needed. Image-based acquisition techniques will be used much more often, driven by cheaper digital cameras.

Acquisition of geometry and surface appearance. Computer vision has developed several techniques for acquiring the geometry of real objects from camera images, but it is still a major problem when the surface of the object is nondiffuse or when the nature of the illumination on the object is unknown. Surface appearance, such as textures and local BRDFs, has recently been captured based on photographs as well. Combining these two fields in order to build an integrated scanner seems a very promising research area. Also, emphasis should be placed on in-hand scanning, where the user manipulates an object in front of a camera and all relevant characteristics are captured.

Self-adaptive light transport. The light transport simulation algorithms outlined in this book come in many different flavors and varieties. Some algorithms perform better in specific situations than others (e.g., radiosity-like algorithms behave better in pure diffuse environments, ray tracing works well in highly specular scenes, etc.) Little effort has been made so far to try to make an overall global illumination algorithm that behaves in an adaptive way in these various situations. Such an algorithm would pick the right mode of simulating the light transport, depending on the nature of the surfaces, the frequency of the geometry, the influence on the final image, etc. Also, partially computed illumination results should always be stored and available for future use by different light transport modes.

Scalable and robust rendering. Scenes that include very high complexity in illumination, materials, and geometry remain challenging. Better and cheaper acquisition technology is driving the demand for rendering such complex scenes in the future. Currently, a user has to manually pick approximations, rendering algorithms, and levels of detail to achieve reason-

able quality and performance for such scenes. But this manual approach is clearly not desirable, particularly when we get to the realm of applications such as games where players interact with dynamically varying scenes while generating content on the fly. Robust algorithms that can scale to complex scenes and can automatically handle scene complexity without user intervention will be critical in the future.

Geometry-independent rendering. Current light transport algorithms assume that the geometry of the scene is known and explicitly compute a huge number of ray-object intersections in order to know where light gets reflected off surfaces. In the future, it is likely that primitives, whose geometry is not explicitly known, will be used in scenes to be rendered. Such primitives might be described by a light field, or another implicit description of how light interacts with the object (e.g., a series of photographs). Incorporating such objects in a global illumination algorithm will pose new problems and challenges. Also, storing partial illumination solutions independent of the underlying geometry (e.g., photon mapping) should be researched further.

Psychoperceptual rendering. Radiometric accuracy has been the main driving force for global illumination algorithms, but since most images are to be viewed by human observers, it is usually not necessary to compute up to this level of accuracy. New rendering paradigms should be focused around rendering perceptually correct images. A perceptually correct image does not necessarily have all the radiometric details, but a viewer might still judge the image to be realistic. It might be possible not to render certain shadows, or to drop certain highlights, or even simplify geometry, if this would not harm the human observer judging the image as being realistic. Radiometric accuracy is best judged by comparing a rendered image with a reference photograph and measuring the amount of error. Psychoperceptual accuracy is probably best judged by having a human look at the rendered picture and asking whether the picture looks "realistic." However, at this point, very little research is available about how this could be done.

Integration with real elements. It is likely that more integration between real and virtual environments will become an integral part of many applications. This does not only entail putting real objects in virtual scenes, but also putting virtual elements in real scenes, e.g., by using projectors or holography. A perfect blend between the real and virtual elements becomes a major concern. This blend includes geometric alignment of real and virtual elements, but also consistent illumination. For example, a virtual element could throw shadows on real objects and vice versa. Developing a good framework for achieving such an integrated rendering system will probably evolve into a major research field during subsequent years.

As a major theme covering all these issues, one can think, or dream, about what the ultimate photorealistic rendering would look like in the future. It is very hard to make any predictions about any specific algorithmic techniques, but it is nevertheless possible to list a few of the requirements or features such a rendering tool should possess:

Interactivity. Any rendering algorithm of the future should be able to render scenes at interactive speeds, irrespective of scene or illumination complexity.

Any material, any geometry. All possible materials, from pure diffuse to pure specular, should be handled efficiently and accurately. Moreover, any type of geometry should be handled as well, whether it is a low-complexity polygon model or a scanned model containing millions of sample points.

Many different input models. It should be possible to take any form of input, whether it is a virtual model or a model based on acquisition from the real world. This probably means leaving the classic polygon model and texture maps for describing geometry and surface appearance and adapting other forms of geometry representation.

Realism slider. Depending on the application, one might settle for different styles of realism: for example, realistic lighting as one would experience in real life; studio-realism with lots of artificial lighting designed to eliminate unwanted shadows; lighting designed for optimally presenting products and prototypes, etc. This should be possible without necessarily altering the scene input or configuration of the light sources.

9.3 Concluding Remarks

Computer graphics is a very exciting field in which to work and is probably one of the most challenging research areas in computer science because it has links with many other disciplines, many of them outside the traditional computer science community. It is exactly this mix with disciplines such as art, psychology, filmmaking, biology, etc. that makes computer graphics very attractive to many students and enthusiasts.

The authors have an accumulated experience of more than 40 years in this field, but we still have the ability to be amazed and surprised by many of the new exciting ideas that are being developed each year. By writing this book, we hope to have made a small contribution in keeping people motivated and enthusiastic about computer graphics, and we can only hope that someday in the future, an exciting new computer graphics technique will develop from some of the ideas presented here.

A

A Class Library for Global Illumination

Global illumination is all about generating paths connecting a virtual camera with a light source. In this appendix, we propose a library of software classes that will facilitate generating such paths in a computer program, by hiding the details of geometry and materials representation and ray casting from a higher-level algorithm implementation.

The library offers the following building blocks:

- Classes for representing **path nodes**, such as a point on a light source, a surface scattering point, the viewing position, etc. (Section A.1).
- Classes for **light source sampling**. These classes generate path nodes that serve as the head of light paths (Section A.2).
- **Support classes**, representing a virtual screen buffer, classes for doing tone mapping, etc. (Section A.3).

The relationship between the classes is depicted in Figure A.1. Some example code fragments, illustrating the use of these classes, are presented in Section A.4.

The interface we describe here does not include a representation of geometry or materials itself. Such a representation is, of course, required in an actual implementation. Our implementation, on top of a VRML-based scene graph management library, is available from this book's website (http://www.advancedglobalillumination.com). In our experience, it is easy to port the class library to other global illumination platforms. Algorithms implemented on top of this interface may be portable to other global illumination systems supporting this interface almost without modifications. Our

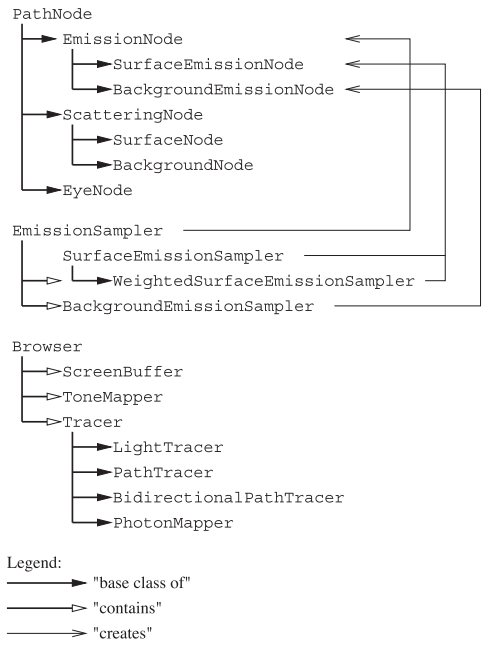

Figure A.1. Graphical overview of the classes contained in the library described here.

experiments have indicated that the additional computation cost caused by the interface is relatively small: on the order of 10% to 20% of the rendering time at most, even if the underlying scene graph management, shader implementation, and ray-tracing kernel are highly optimized. The programming language we used in our implementation is C++. The same interface can obviously also be realized using a different object-oriented programming language.

A.1 Path Node Classes

A.1.1 Overview

All the algorithms described in this book require that light paths or eye paths are generated stochastically. These paths have an associated value and a probability density (PDF). In order to form an image, average ratios are computed of path values over their PDFs.

The value associated with a path is always the product of values associated with the nodes in the path and transition factors such as $\text{vis}(x, y)\cos\theta_y / r_{xy}^2$ between subsequent nodes x and y. The value associated with a path node depends on the type of node. For instance, for a surface scattering event, it is the BSDF times the outgoing cosine; for a light source node, it is the self-emitted radiance, etc.

The PDF indicates the chance that a particular path is being generated. It is also the product of PDFs associated with each node in the path and transition factors. The PDF associated with a surface scattering node, for instance, is the probability by which a scattered light direction is sampled; for a light source node, it is the probability of sampling a light emission direction on a light source.

We call every event at which a path is generated, or its trajectory changed, a path node. The library contains a representation for a variety of path nodes corresponding to:

- Emission of an eye ray through a virtual screen pixel, that is, emission of potential: see `EyeNode` class (Section A.1.3).
- Emission of light, at a surface or from the background (for instance, a model for sky illumination or a high dynamic range environment map): see `EmissionNode` class (Section A.1.4).
- Scattering of light or potential at a surface, or light/potential disappearing into the background: see `ScatteringNode` class (Section A.1.5).

A full path corresponds to a list of such path nodes.

A.1.2 Common Interface: The `PathNode` Base Class

All path node classes inherit from a single `PathNode` base class. The `PathNode` class encapsulates the common properties of all path nodes and provides a uniform interface, so that complete algorithms can be implemented without having to know what types of path nodes may be generated. The main members of the `PathNode` class are:

- The cumulative *probability density* by which a path up to a given node has been generated.
- The cumulative *value* associated with the path up to a given node.
- An `eval()` member function for querying the value (BSDF, EDF, etc.), path survival PDF, the PDF of sampling a given outgoing direction, and the outgoing cosine factor (if applicable) associated with the path node.

- A `sample()` function that calculates from two random numbers whether or not a path at a node shall be expanded and, if so, in what direction.
- A `trace()` function that returns a new path node resulting from tracing a ray into a given direction. The resulting node is always a scattering node (see Section A.1.5). Its precise type depends on the event that occurs next: If the ray hits a surface, a `SurfaceNode` is returned. If the ray disappears to the background, a `BackgroundNode` is returned. The `trace()` function also computes geometric factors associated with the transition to the new path node and properly initializes the cumulative PDF and value of the resulting path node.

`eval()`, `sample()`, and `trace()` are virtual member functions, implemented in children classes of `PathNode`. We choose to provide a single `eval()` function, for evaluating everything related to a path node, in order to minimize the number of virtual function calls and in order to make it easier to share the calculation of certain partial results between the value and the PDF. The latter can result in significant time savings. For instance, PDFs are quite often very similar to values. Results are filled in objects pointed to by pointers passed as parameters to the `eval()` function. If null pointers are passed, corresponding quantities (value, survival or direction sampling PDF, outgoing cosine) are not computed if not needed for other results. In the same spirit, the `sample()` and `trace()` functions can also return values and PDFs that are computed on the fly if nonnull pointer arguments are passed for filling in such side results. The `trace()` function optionally accepts a set of pointers to path node objects of each type that can be returned, in order to avoid dynamic storage allocation and to allow easy type checking afterwards. This will be illustrated in Section A.4.

Besides the above members, the `PathNode` base class also maintains and offers:

- The depth of the path node in its path: 0 for the head of a path, depth of the parent node plus 1 for nonhead path nodes.
- The light emission and scattering modes to take into account for evaluation and sampling (diffuse/glossy/specular emission/reflection/refraction);
- A pointer to the parent node in the path.
- Various flags: whether the path node belongs to a light path or eye path (required for making certain corrections due to nonsymmetric light scattering [203]), whether the path node is at the end of a subpath, whether it has a finite position in space, or whether it is located "at infinity" (for instance: background emission nodes).

- Member functions for accessing the position of a path node in space, or the geometry at that location, or for obtaining the head of the path, or direction and distance (taken to be 1 for background nodes) to another path node, or for computing visibility with regard to another node.

- Static member variables indicating the minimum and maximum path depth for generating paths. These values affect survival probabilities computed in the `sample()` and `eval()` functions.

- Some more member functions for convenience: `scatter()` computes the radiance or potential accumulated along a path and scattered into a given direction. The `expand()` member function combines `sample()` and `trace()` in a single function.

A.1.3 Pixel Filtering and Sampling: The `EyeNode` Class

The `EyeNode` class represents the head of eye paths. The position of an `EyeNode` object is the position of the pinhole camera used to view a scene. `EyeNode` objects are associated with a virtual screen pixel. They encapsulate pixel filtering and sampling. The value returned by `EyeNode::eval()` is the pixel measurement function of a given direction (see Section 5.7.1). `EyeNode::sample()` will select a direction through the associated virtual screen pixel for shooting an eye ray. Currently, a simple box pixel filter is implemented.

A.1.4 Light Emission: The `EmissionNode` Classes

An `EmissionNode` object represents the head of a light path. It corresponds with a point on a surface light source (`SurfaceEmissionNode` sub-class) or a direction towards the background for background illumination such as sky illumination or a high dynamic range environment map (`BackgroundEmissionNode` subclass). The value associated with an emission node is the self-emitted radiance into a given direction. The `sample()` member function will sample a direction according to the directional emission distribution at a surface emission location. For background emission nodes, where the emission direction is encoded in the node, `sample()` will select a point on the projection of the scene bounding box perpendicular to the emission direction. In both cases, `sample()` results in a point and a direction, enough for constructing a ray to shoot self-emitted radiance along.

Emission nodes can be generated by means of the `EmissionSampler` classes described in Section A.2.

A.1.5 Light and Potential Scattering: The `ScatteringNode` Classes

The `trace()` function of any path node usually results in a new `ScatteringNode` object representing surface scattering (`SurfaceNode`) or light or potential that disappears into the background (`BackgroundNode`).

Surface Scattering: `SurfaceNode` Class

The position of a `SurfaceNode` object is the position on the surface of an object in the scene at which a light path or eye path can be reflected, refracted, or absorbed. The value associated with such a node is the BSDF for a given direction. By default, the survival probability is computed based on the fraction of incident illumination or potential that will be scattered rather than absorbed. It depends on the direction of incidence and is, of course, affected by the currently required minimum and maximum path length. The "outgoing cosine" computed by `SurfaceNode::eval()` is the absolute value of the cosine between a given outgoing direction and the shading normal at the scattering location. The `sample()` member function samples an outgoing direction ideally according to the BSDF times the outgoing cosine. `SurfaceNode` objects know whether they belong to a light path or eye path, and appropriate correction factors for nonsymmetric scattering due to bump mapping or normal interpolation are applied on the BSDF [203]. There is also a version of `SurfaceNode::eval()` that allows us to specify incident directions other than the one for which the path node was constructed.

Occasionally, a path will hit a surface light source. In order to evaluate self-emitted radiance at a scattering location, and to compute the probability of obtaining the surface location by means of surface emission sampling (with a `SurfaceEmissionSampler` object, see Section A.2), appropriate `source_radiance()` and `source_pdf()` member functions are provided. Some algorithms, like bidirectional path tracing, require more complex operations if a path hits a light source. A conversion from the `SurfaceNode` class to the `SurfaceEmissionNode` class is provided in order to meet such requirements. An `on_light_source()` member function returns whether or not a `SurfaceNode` lays on a light source.

Paths Disappearing into the Background: `BackgroundNode` Class

If a path doesn't hit a surface, it's said to disappear into the background. A special `BackgroundNode` marks the end of such paths. The `BackgroundNode` class inherits from the `ScatteringNode` base class, but of course, no scattering happens: A path that disappears into the background is always terminated. The value and PDFs returned by `BackgroundNode::eval()`

are always zero, and the `BackgroundNode::sample()` member function will always result in an error. The `trace()` function returns a null result.

If background illumination has been modeled in a scene to be rendered, however, the `BackgroundNode::source_radiance()` and `BackgroundNode ::source_pdf()` member functions will compute the self-emitted radiance received from the background along the path direction, as well as the probability of sampling that direction using a `BackgroundEmissionSampler` object. Also for background "scattering," a conversion from the class `BackgroundNode` to the class `BackgroundEmissionNode` is provided so all queries for self-emitted illumination can be performed at a background "scattering" node.

A.2 Light Source Sampling Classes

A scene can contain both a number of surfaces that emit light spontaneously, as well as a model for background illumination such as sky light or a high dynamic range environment map. A second set of classes provided by the library will select either a position on a light source surface (`SurfaceEmissionSampler` class) or a direction for background illumination (`BackgroundEmissionSampler` class). Unlike path node objects, which are very frequently created and destroyed during the global illumination computations, there is normally only a single surface and background emission sampler active while rendering a frame.

A.2.1 Surface Emission Sampling: The `SurfaceEmissionSampler` and `WeightedSurfaceEmissionSampler` Classes

A `SurfaceEmissionSampler` class object maintains a list (or better, an array) of light source surfaces in the scene. Our current implementation assumes scenes modeled out of triangles, so our `SurfaceEmissionSampler`s will contain a list of pointers to light-emitting triangles. It is straightforward to extend the interface to handle curved light sources, too. Besides member functions for building up such a list, the main member functions are:

- A `sample()` function that will select a triangle from the list and return a point on the selected triangle as a `SurfaceEmissionNode`. Triangles are selected with a probability proportional to their self-emitted power. Points are selected uniformly on a triangle.
- A `pdf()` member function returns the probability density of sampling a given point on a given triangle using `sample()`.

The `pdf()` member function assumes an index mechanism for quickly locating a given triangle in the list of light source triangles. Our `SurfaceEmissionNode`s and `SurfaceNode`s contain a pointer to the surface triangle on which they are located. This allows us to find out easily whether a `SurfaceNode` is located on a light source, or to calculate all relevant light source quantities.

Weighted Surface Emission Sampling

Sometimes, surface emission sampling according to emitted power is not optimal, and other probabilities for selecting light source triangles are required. One example of such a case is view-importance–driven light source sampling (Section 5.4.5), when a light source needs to be selected according to its estimated impact on a particular view. A powerful, but distant or occluded light source for instance, receives a lower probability of being selected than a less powerful, but nearby, light source. The `WeightedSurfaceEmissionSampler` subclass of `SurfaceEmissionSampler` allows us to enable/disable light source triangles from a list and to attach weights to light source triangles in a very general way. For convenience, a member function is provided that will assign weights according to light source distance and orientation with regard to a specified point and normal. Our implementation also contains an adapted version of a light-path tracer that estimates the light flux each light source contributes to the current view and that assigns light source weights proportional to these fluxes eventually.

A.2.2 Background Emission Sampling: The `BackgroundEmissionSampler` Class

The `BackgroundEmissionSampler` class works in a very similar way to the `SurfaceEmissionSampler` class, except that usually, the number of background light sources is small, and it returns a sampled direction to the background in the form of a `BackgroundEmissionNode`. Background directions are selected with a probability that reflects the intensity of self-emitted radiance received from the direction. It is much harder to take into account surface orientation here so there is no class for weighted background emission sampling.

A.2.3 The `EmissionSampler` Wrapper Class

The library provides an `EmissionSampler` wrapper class that contains a pointer to a `WeightedSurfaceEmissionSampler` and to a `BackgroundEmissionSampler` for the scene. By default, surface emission

sampling and background emission sampling receive a weight proportional to the total emitted power from surfaces and the background, respectively. In order to calculate these weights, it is necessary to know in what length units a scene has been modeled. The default weights can, of course, be modified in a program. Our adapted light tracer, described above, does so after measuring the light flux contributed to the current view by surfaces and background.

The public implementation provides only triangle light sources and background emission. Other light sources, such as spherical or disc light sources, can easily be added in the form of additional emission sampler classes. The `EmissionSampler` wrapper class shall contain a reference to all light source samplers, with proper weights, so that it can hide the variety of light sources in a scene from the implementation of global illumination algorithms by providing a single `sample()` function for any kind of light emission.

A.3 Support Classes

The path node and sampler class interfaces are pretty much self-contained, but they need to be embedded in a suitable working environment, of course. For convenience, the library also contains a number of additional classes providing such an environment. Unlike the path node class interface, it is likely that some tuning will be needed in order to integrate these support classes into your global illumination application.

A.3.1 A Pinhole Camera Virtual Screen Abstraction: The `ScreenBuffer` Class

`EyeNode` class objects correspond to pixels on a virtual screen. Their implementation requires an abstraction of a virtual screen buffer. The library provides a `ScreenBuffer` class for this purpose. The `ScreenBuffer` class represents the virtual screen of a pinhole camera. It offers member functions `getDirection()` and `getPixelCoord()` for mapping pixel coordinates to the corresponding primary ray direction and vice versa. A member function `setView()` initializes the current view point, focus point, direction point upwards, and field of view angle in the same way as the `gluLookAt()` function in OpenGL. The `getPixelCoord()` function returns whether or not a primary ray direction points towards the screen. It is used in light tracing and bidirectional path tracing in order to splat path contributions to the screen, as shown in the examples (Section A.4.1).

The `ScreenBuffer` class also maintains two arrays of pixel color values: one usual set of low dynamic range RGB triplets plus transparency

that can be displayed efficiently using, for instance, the `glDrawPixels()` OpenGL function; and one set that contains high dynamic range color values in 32-bit packed RGBE format [220]. The `ScreenBuffer` class offers member functions `clear()`, `clearRGBA()`, `clearHDR()`, `setPixel()`, `getRGBAPixel()`, `getHDRPixel()`, `addPixel()`, etc., for clearing, querying, and modifying low and high dynamic range pixel color values.

A.3.2 Converting High to Low Dynamic Range Color Values: The `ToneMapper` Classes

A global illumination algorithm computes and stores high dynamic range pixel color values in the `ScreenBuffer`. A `ToneMapper` object will map the high dynamic range pixels to RGB color triplets for display as explained in Section 8.2. Different tone mapping algorithms are implemented in subclasses of a base `ToneMapper` class. Such classes maintain their own set of required parameters, such as the world adaptation luminance in the current view. The `ScreenBuffer` class provides a member function `adaptation_luminance()` for computing the world adaptation luminance as the exponentiated mean logarithmic luminance of the virtual screen high dynamic range pixel color values. The main member function provided by the `ToneMapper` classes is a `map()` function that does everything to convert the high dynamic range color values in a given `ScreenBuffer` object into low dynamic range color values for display.

A.3.3 Integration into an Application: The `Browser` and `Tracer` Classes

The library described here comes with an application in which several global illumination algorithms have been implemented. We describe here two additional classes that integrate the path node and sampler classes into this application.

The `Browser` Classes

We implemented a `Browser` base class to group and maintain the whole software environment in which the `PathNode` and `EmissionSampler` classes operate:

- The scene graph. In our implementation, the scene graph is a VRML97 scene graph with numerous extension nodes for representing physically based appearance and high dynamic range backgrounds as well as color calibration parameters of the computer monitor on which a model has been designed.

- The interface to a *ray-tracing engine* needed for finding ray-object intersections and for performing visibility queries.
- One instance of an `EmissionSampler`, containing a `WeightedSurfaceEmissionSampler` and a `BackgroundEmissionSampler`, as well as a reference unweighted `SurfaceEmissionSampler`.
- A `ScreenBuffer` and a `ToneMapper` object.

The `Browser` base class does not support a graphical user interface, and neither does it perform any global illumination computations itself. It needs to be augmented with such capabilities by means of inheritance. The `Browser` base class provides a virtual `trace()` member function, which needs to be implemented in a child class in order to:

- Initialize the `ScreenBuffer` for the current view.
- Perform the real global illumination computations for the view.
- Call the `ToneMapper` in order to map computed high dynamic range pixel colors into low dynamic range RGB color triplets for display.
- Display the results on a computer screen, or save them into a file.

The `Tracer` Classes

Rather than implementing each global illumination algorithm as a separate `Browser` subclass, we introduced yet another class, called `Tracer`, providing a common software interface for global illumination algorithms. Algorithms such as path tracing and light tracing (Chapter 5), bidirectional path tracing (Section 7.3), a ray-traced version of the instant radiosity algorithm (Section 7.7), and photon mapping (Sectoin 7.6) are implemented in `PathTracer`, `LightTracer`, `BiDirTracer`, `InstantRadiosity`, and `PhotonMapper` child classes of the `Tracer` base class. The main functions implemented by these classes are:

- An `init()` function performs initializations such as storage allocation of large arrays for each frame to be rendered.
- A `trace()` function computes an image for the current view.
- A `tonemap()` function properly rescales `ScreenBuffer` high dynamic range pixels and uses the current `Browser`'s `ToneMapper` object in order to convert to displayable RGB color triplets.

Our `Browser` subclass object creates an appropriate `Tracer` object according to the desires of a user and calls the above listed `Tracer` functions in its `Browser::trace()` handler.

In addition to the above functions, our `Tracer` classes also provide member function for distributed computations, for instance, indicating how to separate an image into several subimages to be computed on different network clients, and how to merge the resulting pixel values computed by each client afterwards.

A.4 Example Code Fragments

In this section, we provide some example code fragments, illustrating how global illumination algorithms can be implemented on top of the path node and sampler classes described previously.

A.4.1 A Light Tracer

We first present the core part of our `LightTracer` class, implementing light particle tracing (see Section 5.7):

```
// scrn is pointer to the current ScreenBuffer object
// class Vec3 and class Spectrum represent 3D vectors and spectra
// lightsampler is pointer to current EmissionSampler object
int nrparticles;   // nr of particles to trace

// splats particle on the screen
inline void LightTracer::splat(class PathNode *n)
{
  float dist;                      // distance between eye and n
  const Vec3 eyedir = scrn->eye.dirto(n->pos(), &dist);  // direction
  if (n->at_infinity()) dist = 1.; // don't divide by square distance

  float i, j;                      // compute pixel coordinates (i,j)
  if (scrn->getPixelCoord(eyedir, &i, &j)) {
    class EyeNode e(i, j);         // eye node corresponding to pixel
    if (visible(&e, n)) {          // n is not occluded from the eye
      float ncos, ecos;            // cosine factors at the eye
                        // and at n
      scrn->addPixel(i, j, e.scatter(eyedir, &ecos)
                 * n->scatter(-eyedir, &ncos)
                 * (ncos * ecos / (dist*dist * (float)nrparticles)));
    }
  }
}

inline void LightTracer::traceparticle(class PathNode *l)
{
```

```
  splat(l);                          // splat particle on screen
  class PathNode *n = l->expand();   // expand path
  if (n) traceparticle(n);           // recurse
  delete n;
}

void LightTracer::trace(void)
{
  for (int i=0; i<nrparticles; i++) {
    class EmissionNode *l = lightsampler->sample(); // sample lights
    if (l) traceparticle(l);             // trace light path
    delete l;
  }
}
```

In order to implement photon mapping, the `splat()` function shall be modified in order to store `SurfaceNode` hit points `n->pos()`, incident direction `n->indir`, and flux `n->value/n->pdf` in a photon map data structure. A ready-to-use implementation of a photon map data structure can be found in Jensen's book [83].

A.4.2 A Path Tracer

The implementation of a path tracer below is only slightly more complicated, in order to avoid dynamic storage allocation and to obtain easy checking of path node types returned by the `PathNode::expand()` and `EmissionSampler::sample()` functions.

```
// Again, scrn and lightsampler are the current ScreenBuffer
// and EmissionSampler.

// Array of SurfaceNodes in order to avoid the need for
// dynamic storage allocation in PathNode::expand().
// Storage is allocated in setup(), and freed in cleanup().
class SurfaceNode* PathTracer::sbuf =0;

// nr of light samples (shadow rays) at each path surface hit
int PathTracer::nrlightsamples = 1;

// Compute score associated with path landing on a light source.
inline const Spectrum PathTracer::source(class ScatteringNode* s)
{
  class Spectrum score(0.);
  if (s->depth() <= 1 || nrlightsamples == 0) {
    // Source contribution computed exclusively by means of
    // scattering.
    score = s->source_radiance() * s->value / s->pdf;
  } else {
    // Source contribution computed exclusively by means of
    // light source sampling.
```

```
  }
  return score;
}

// Light source sampling for computing direct illumination at
// the SurfaceNode s.
inline const Spectrum PathTracer::tracelight(class SurfaceNode* s)
{
  // Avoid dynamic storage allocation
  static class SurfaceEmissionNode sl;
  static class BackgroundEmissionNode bl;
  class EmissionNode *l = lightsampler->sample(&sl, &bl);
  if (l) {
    // cosine/distance at the light and at the surface
    float lcos, scos, dist;
    // dir/dist surface to light
    const Vec3 dir = s->dirto(l, &dist);
    // compute cosine at the light
    l->eval(-dir, 0, 0, 0, &lcos);
    // surface behind light or occluded
    if (lcos <= 0 || !visible(s, l))
      return Spectrum(0.);
    else
      return s->scatter(dir, &scos) * l->scatter(-dir)
             * (scos * lcos / (dist * dist));
  }
  return Spectrum(0.);
}

// Light source sampling at surface scattering node s.
inline const Spectrum PathTracer::tracelights(class SurfaceNode* s)
{
  class Spectrum score(0.);
  if (nrlightsamples > 0) {
    for (int i=0; i<nrlightsamples; i++) {  // shoot shadow rays
      score += tracelight(s);
    }
    score /= (float)nrlightsamples;
  }
  return score;
}

// Traces a path through the pixel represented by the EyeNode e
inline const Spectrum PathTracer::tracepixel(class EyeNode* e)
{
  static class BackgroundNode b;  // avoid dynamic storage allocation
  class SurfaceNode *s = sbuf;
  // sample + shoot eye ray
  class ScatteringNode *n = e->expand(s, &b);
  class Spectrum score(0.);
  while (n) {
    score += source(n);      // self-emitted illumination
```

```
    if (n == s)              // direct illumination: only surface nodes
      score += tracelights(s);
    n = n->expand(++s, &b);  // indirect illumination: expand path
  }

  return score;
}

void PathTracer::setup(void)
{
  sbuf = new SurfaceNode [PathNode::max_eye_path_depth];
}

void PathTracer::cleanup(void)
{
  delete [] sbuf;
}

// computes image for current view
void PathTracer::trace(void)
{
  setup();
  for (int j=0; j<scrn->height; j++) {
    for (int i=0; i<scrn->width; i++) {
      class EyeNode e(i, j);
      scrn->addPixel(i, j, tracepixel(&e));
    }
  }
  cleanup();
}
```

A.4.3 Multiple Importance Light Source Sampling

When light reflection at a surface hit by a path is highly specular, it is usually much better to compute direct illumination by means of a scattered ray rather than by light source sampling. We show here the modifications to the path tracer implementation above, in order to calculate direct illumination at path nodes by means of multiple importance sampling [201]. These modifications illustrate the use of the `PathNode::eval()` functions in cases where the higher-level `PathNode::scatter()` functions fall short. Some example results are shown in Figure A.2 on page 332.

```
// flag indicating whether or not to use bidirectional weighting
// for source contributions.
bool PathTracer::bidir_weighting = true;

// Compute score associated with path landing on a light source.
inline const Spectrum PathTracer::source(class ScatteringNode* s)
{
```

```
  class Spectrum score(0.);
  if (s->depth() <= 1 || nrlightsamples == 0) {
    // Source contributions computed exclusively by means of
    // scattering.
    score = s->source_radiance() * s->value / s->pdf;
  } else if (bidir_weighting) {
    // Source contributions computed by means of both scattering
    // and light source sampling.
    // Attenuate source radiance taking into account the probability
    // that s would have been obtained by light source sampling
    // rather than scattering.
    float w_scattering = s->pdf / s->parent()->pdf;
    float w_lsampling  = s->source_pdf() * (float)nrlightsamples;
    float w = w_scattering / (w_scattering + w_lsampling);
    score = s->source_radiance() * s->value * (w / s->pdf);
  } else {
    // Source contributions computed exclusively by means of
    // light source sampling.
  }
  return score;
}

// Light source sampling for computing direct illumination at
// the SurfaceNode s.
inline const Spectrum PathTracer::tracelight(class SurfaceNode* s)
{
  // Avoid dynamic storage allocation
  static class SurfaceEmissionNode sl;
  static class BackgroundEmissionNode bl;
  class EmissionNode *l = lightsampler->sample(&sl, &bl);
  if (l) {
    // cosine/distance at the light and at the surface
    float lcos, scos, dist;
    const Vec3 dir = s->dirto(l, &dist);
    // compute cosine at the light
    l->eval(-dir, 0, 0, 0, &lcos);
    // surface behind light or occluded
    if (lcos <= 0 || !visible(s, l))
      return Spectrum(0.);

    if (!bidir_weighting) {
      // source() doesn't pick up source radiance at hit surfaces
      return s->scatter(dir, &scos) * l->scatter(-dir)
             * (scos * lcos / (dist * dist));
    }

    else {
      // Attenuate direct illumination taking into account the
      // probability that the light source could have been hit
      // by a scattered ray.
      float survpdf, scatpdf;   // survival and scattering pdf
      class Spectrum fr, Le;    // BRDF at s and EDF at l
```

```
      s->eval( dir, &fr, &survpdf, &scatpdf, &scos);
      l->eval(-dir, &Le, 0, 0, 0);
      float g = lcos / (dist*dist);  // transition factor
      float w_scattering = survpdf * scatpdf * g; // scatt. weight
      float w_lsampling = l->pdf * (float)nrlightsamples;
      float w = w_lsampling / (w_lsampling + w_scattering);
      float G = scos * g;
      return (s->value * fr * Le) * (G * w / (s->pdf * l->pdf));
    }
  }
  return Spectrum(0.);
}

// The tracelights(), tracepixel() and trace() functions
// are the same as in the previous section.
```

A.4.4 A Bidirectional Path Tracer

Here is our code for a bidirectional path tracer:

```
// The purpose of the following arrays is to prevent dynamic
// storage allocation in PathNode::expand() and to allow
// efficient PathNode child class checking by comparing pointers.
class EyeNode eyenode;              // head of eye path
class SurfaceEmissionNode senode;   // surface emisison node
class BackgroundEmissionNode benode;// background emission node
// head of light path: pointer to senode or benode:
class EmissionNode *lightnode;
// surface scattering nodes
class SurfaceNode *eyesurfnodes, *lightsurfnodes;
class BackgroundNode eyebkgnode, lightbkgnode; // background nodes
// pointers surface or background scattering nodes:
class ScatteringNode **eyescatnodes, **lightscatnodes;

int eyepathlen, lightpathlen;              // eye/light path length
class PathNode **eyepath, **lightpath; // pointers to path nodes
float *erdpdf, *lrdpdf;  // reverse dir. selection  probabilities
float *erspdf, *lrspdf;  // survival prob. in reverse path direction
float *erhpdf, *lrhpdf;  // hit densities in reverse path direction
float nrparticles;       // nr of light particles traced

// for avoiding dynamic storage allocation when converting
// scattering nodes to emission nodes.
class BackgroundEmissionNode eeb;
class SurfaceEmissionNode ees;

// minimum and maximum light/eye/combined path length
static int min_light_path_length=2,
    max_light_path_length=7,
    min_eye_path_length=2,
```

```
    max_eye_path_length=7,
    max_combined_path_length=7;

// trace an eye path by expanding the eye node e.
// . a pointer to the eye node goes into eyepath[0]
// . the surface scattering nodes come into eyesurfnodes[1] etc... and
// a pointer to them in eyepath[1] and eyescatnode[1], etc...
// . the final background node goes into eyebkgnode and a pointer to
// it in eyepath[.] and eyescatnode[.] as well.
// Returns length of the eye path (nr of segments = nr of nodes - 1)
int BiDirTracer::trace_eye_path(class EyeNode* e)
{
  eyepath[0] = e;                    // store pointer to head of path

  int i=1;
  class ScatteringNode *n = e->expand(&eyesurfnodes[i], &eyebkgnode);
  while (n) {
    eyescatnodes[i] = n;              // store ScatteringNode pointer
    eyepath[i] = n;                   // store PathNode pointer
    i++;                              // expand the path
    n = n->expand(&eyesurfnodes[i], &eyebkgnode);
  }

  return i-1;                          // path length (nr of segments)
}

// Same as trace_eye_path, but for light path starting at the
// emission node l. Results go into lightpath[.], lightscatnodes[.],
// lightsurfnodes[.], and lightbkgnode.
// Returns length of light path.
int BiDirTracer::trace_light_path(class EmissionNode* l)
{
  lightpath[0] = l;

  int i=1;
  class ScatteringNode *n = l->expand(&lightsurfnodes[i], &lightbkgnode);
  while (n) {
    lightscatnodes[i] = n;
    lightpath[i] = n;
    i++;
    n = n->expand(&lightsurfnodes[i], &lightbkgnode);
  }

  return i-1;
}

// Computes the probabilities of sampling the eye path in reverse direction,
// that is: with incident and outgoing direction at the nodes exchanged.
// Result goes into:
// . erdpdf[i]: _D_irection sampling pdf for reverse directions at node i
// . erspdf[i]: unconstrained _S_urvival probability at node i (that is:
//   not taking into account minimum and maximum required path length)
```

```
// . erhpdf[i]: cos / distance squared from node i to node i-1 (_H_it pdf)
// The leading 'e' in the names of the arrays stands for _E_ye path. The
// 'r' for _R_everse.
void BiDirTracer::compute_reverse_eyepath_probs(void)
{
  erdpdf[0] = erspdf[0] = erhpdf[0] = 0.; // no reverse tracing at the eye
  if (eyepathlen == 0)
    return;
  class ScatteringNode* next = eyescatnodes[1];
  erhpdf[1] = 0.;  // chance of hitting eye point is 0 for pinhole camera
  for (int i=1; i<eyepathlen; i++) {
    class ScatteringNode* cur = next;
    next = eyescatnodes[i+1];
    class Vec3 toprevdir(cur->indir);
    class Vec3 tonextdir(-next->indir);
    erspdf[i] = cur->unconstrained_survival_probability(tonextdir);
    cur->eval(tonextdir, toprevdir, 0, 0, &erdpdf[i], 0);

    cur->eval(tonextdir, 0, 0, 0, &erhpdf[i+1]);
    if (!next->at_infinity())
      erhpdf[i+1] /= cur->position.sqdistance(next->position);
  }
  erspdf[eyepathlen] = erdpdf[eyepathlen] = 1.;   // not needed
}
// Same for the light path.
void BiDirTracer::compute_reverse_lightpath_probs(void)
{
  // no reverse tracing at the light source
 lrdpdf[0] = lrspdf[0] = lrhpdf[0] = 0.;
  if (lightpathlen == 0)
    return;
  class ScatteringNode* next = lightscatnodes[1];
  lightnode->eval(-next->indir, 0, 0, 0, &lrhpdf[1]);
  if (!lightnode->at_infinity() && !next->at_infinity())
    lrhpdf[1] /= lightnode->pos().sqdistance(next->position);
  for (int i=1; i<lightpathlen; i++) {
    class ScatteringNode* cur = next;
    next = lightscatnodes[i+1];
    class Vec3 toprevdir(cur->indir);
    class Vec3 tonextdir(-next->indir);
    lrspdf[i] = cur->unconstrained_survival_probability(tonextdir);
    cur->eval(tonextdir, toprevdir, 0, 0, &lrdpdf[i], 0);

    cur->eval(tonextdir, 0, 0, 0, &lrhpdf[i+1]);
    if (!next->at_infinity())
      lrhpdf[i+1] /= cur->position.sqdistance(next->position);
  }
  lrspdf[lightpathlen] = lrdpdf[lightpathlen] = 1.;   // not needed
}

// #define WEIGHT(w) (w)     // balance heuristic
#define WEIGHT(w) (w*w)   // power 2 heuristic
```

```
// Computes weight associated with the combined eye sub path up to
// eyepath[e] and light sub path up to lightpath[l].
// Requires that e>=0 and l>=0. Weighting for e==-1 or l==-1
// (empty sub-path) is special because there is no connecting path
// segment (nor visibility test), see eyepath_on_light() and
// lightpath_on_camera().
float BiDirTracer::weight(int e, int l,
        const Vec3& ltoedir, float ltoepdf, float etolpdf)
{
  class PathNode* en = eyepath[e];
  class PathNode* ln = lightpath[l];

  // weight of "this" strategy is proportional to product of
  // the pdfs of sampling the connected eye and light sub paths.
  // If e<=0, we are dealing with pure light path tracing (see
  // join_lightpath_with_eye() and an additional multiplication by
  // the total nr of light paths being traced is needed (= nr of pixels
  // since we trace one light path per pixel).
  double lpdf = ln->pdf * (e<=0 ? nrparticles : 1.);
  double epdf = en->pdf;
  double thisw = WEIGHT(lpdf * epdf);

  // compute sum of weights associated with all possible combinations
  // of shorter/longer eye/light sub-paths leading to the same path between
  // lightpath[0] and eyepath[0].
  double sumw = thisw;     // sum of weights
  int i, j;

  // shorter eye sub-paths / longer light sub-paths
  i = e; j = l;
  lpdf = ln->pdf;         // prolonged light sub-path pdf
  while (i>=0 && j<PathNode::max_light_path_depth) {
    double lxpdf = 0.;    // light path transition pdf
    if (j == l) {
      // transition probability for light path at lightpath[l]
      // going towards eyepath[e]. Probability is given as an
      // argument to this function.
      // i == e
      lxpdf = ltoepdf;
    } else if (j == l+1) {
      // evaluate transition probability for light path arriving at
      // eyepath[e] from lightpath[l] and going towards eyepath[e-1].
      // i == e-1
      class ScatteringNode* escat = eyescatnodes[e];
      float spdf = j < PathNode::min_light_path_depth   // survival pdf
        ? 1.
        : escat->unconstrained_survival_probability(-ltoedir);
      float dpdf;                             // direction selection pdf
      escat->eval(-ltoedir, escat->indir, 0, 0, &dpdf, 0);
      lxpdf = spdf * dpdf * erhpdf[e];   // third factor is cosine/dist^2
    } else {
      // transition probability for light path at eyepath[i+1]
```

```
    // from eyepath[i+2] and going towards eyepath[i]. Use
    // precomputed probabilities for reverse eye path).
    // i<e-1
    float spdf = j < PathNode::min_light_path_depth
      ? 1.
      : erspdf[i+1];
    lxpdf = spdf * erdpdf[i+1] * erhpdf[i+1];
  }
  lpdf *= lxpdf;
  // The light sub-path now ends at eyepath[i]. Consider connection
  // with eye sub-path ending at eyepath[i-1].
  i--; j++;
  double w = (i>=0) ? eyepath[i]->pdf * lpdf : lpdf;
  if (i<=0) w *= nrparticles;   // pure light path tracing case
  sumw += WEIGHT(w);
}

// shorter light sub-paths / longer eye sub-paths
i = e; j = l;
epdf = en->pdf;         // prolonged eye sub-path pdf
while (j>=0 && i<PathNode::max_eye_path_depth) {
  double expdf = 0.;    // eye path transition pdf
  if (i == e) {
    // transition probability for eye path at eyepath[e]
    // going towards lightpath[l]
    // j == l
    expdf = etolpdf;
  } else if (i == e+1) {
    // evaluate transition probability for eye path arriving at
    // lightpath[l] from eyepath[e] and going towards lightpath[l-1]
    // j == l-1
    class ScatteringNode* lscat = lightscatnodes[l];
    float spdf = i < PathNode::min_eye_path_depth
      ? 1.
      : lscat->unconstrained_survival_probability(ltoedir);
    float dpdf;
    lscat->eval(ltoedir, lscat->indir, 0, 0, &dpdf, 0);
    expdf = spdf * dpdf * lrhpdf[l];
  } else {
    // transition probability for eye path at lightpath[j+1]
    // from lightpath[j+2] and going towards lightpath[j]. Use
    // precomputed probabilities for reverse light path.
    // j < l-1
    float spdf = i < PathNode::min_eye_path_depth
      ? 1.
      : lrspdf[j+1];
    expdf = spdf * lrdpdf[j+1] * lrhpdf[j+1];
  }
  epdf *= expdf;
  // The eye sub-path now ends at lightpath[j]. Consider connection
  // with light sub-path ending at lightpath[j-1].
  j--; i++;
```

```
    double w = (j>=0) ? lightpath[j]->pdf * epdf : epdf;
    sumw += WEIGHT(w);
  }

  return thisw / sumw;
}

// e==0 and l==0: join eye node with light source node
// (adds self-emitted radiance from a light source node to the
// image). This is handled by eyepath_on_light() for eye nodes
// of depth 1.
const Spectrum BiDirTracer::join_light_eye(void)
{
  return Spectrum(0.);
}

const EmissionNode* BiDirTracer::convert_to_lightnode(int e)
{
  if (eyescatnodes[e] == &eyebkgnode) {
    // scattering node is background node
   // convert to background emission node
    eeb = BackgroundEmissionNode(eyebkgnode);
    return &eeb;
  } else {
    // scattering node is surface node
    // convert to surface emission node
    ees = SurfaceEmissionNode(eyesurfnodes[e]);
    return &ees;
  }
}

// e>0 && l==-1: eye path arriving on a light source (that is:
// we check for every surface hit, whether it is a light source
// or not and take its self-emitted radiance into the incident
// direction into account if it is a light source.)
const Spectrum BiDirTracer::eyepath_on_light(const int e)
{
  class ScatteringNode* es = eyescatnodes[e];

  if (!es->on_light_source()) {
    return Spectrum(0.);
  }

  if (e==1) {
    // this is the complementary strategy of join_light_eye(), but
    // join_light_eye() does nothing, so this strategy gets full weight.
    return es->source_radiance() * es->value / es->pdf;
  }

  // Convert the scattering node into a corresponding emission node
  const EmissionNode* ee = convert_to_lightnode(e);
  class Spectrum Le;      // self-emitted radiance
```

```
  float spdf, dpdf;       // light path survival and direction s. pdf
  ee->eval(es->indir, &Le, &spdf, &dpdf, 0);

  // Compute weight of this strategy
  double thisw = WEIGHT(es->pdf);   // pdf of the eye path

  // Compute sum of weights of all equivalent strategies. This is
  // different from the other cases, because there is no connecting
  // path segment here (for the same reason, there's no
  // additional visibility test for this strategy.)
  double sumw = thisw;
  int i=e, j=0;
  double lpdf = ee->pdf; // pdf of the same position using emission sampling
  while (i>=0 && j<PathNode::max_light_path_depth) {
    double lxpdf = 0.;
    if (j==0) {
      lxpdf = spdf * dpdf * erhpdf[e];
    } else {
      double spdf = j<PathNode::min_light_path_depth
        ? 1.
        : erspdf[i];
      lxpdf = spdf * erdpdf[i] * erhpdf[i];
    }
    i--; j++;
    double w = (i>=0) ? eyepath[i]->pdf * lpdf : lpdf;
    if (i<=0) w *= nrparticles;
    sumw += WEIGHT(w);
    lpdf *= lxpdf;
  }

  return Le * es->value * (thisw / (es->pdf * sumw));
}

// e>0, l==0: join eye path vertex e>0 with light source node
// = standard path tracing
const Spectrum BiDirTracer::join_eyepath_with_light(const int e)
{
  if (eyescatnodes[e] == &eyebkgnode ||
      !visible(eyescatnodes[e], lightnode))
    return Spectrum(0.);

  class SurfaceNode *en = &eyesurfnodes[e];
  class EmissionNode *ln = lightnode;
  float ecos, lcos, espdf, lspdf, edpdf, ldpdf, dist;
  class Spectrum efr, Le;
  const Vec3 ltoedir = ln->dirto(en, &dist);
  en->eval(-ltoedir, &efr, &espdf, &edpdf, &ecos);
  ln->eval( ltoedir, &Le,  &lspdf, &ldpdf, &lcos);
  double invdist2 = 1. / (dist * dist);
  float etolpdf = espdf * edpdf * lcos * invdist2;
  float ltoepdf = lspdf * ldpdf * ecos * invdist2;
  double G = ecos * lcos * invdist2;
```

```
  float w = weight(e, 0, ltoedir, ltoepdf, etolpdf);
  return en->value * efr * Le * (G / (en->pdf * ln->pdf) * w);
}

// e==-1, l>0: corresponds with a light path node arriving on the
// surface of the camera. Since we are using a pinhole camera,
// this can not happen.
const Spectrum BiDirTracer::lightpath_on_camera(const int l)
{
  return Spectrum(0.);
}

// e==0, l>0: Join light path vertex with eye node
// = standard light particle tracing
// Score contributes to different pixel than the one through
// which the eye path was traced. Therefore we add the score
// directly to the screen buffer and we return a null spectrum here.
const Spectrum BiDirTracer::join_lightpath_with_eye(const int l)
{
  if (lightscatnodes[l] == &lightbkgnode)
    return Spectrum(0.);

  // find pixel through which the light path node is visible.
  class SurfaceNode* ln = &lightsurfnodes[l];
  double dist;
  class Vec3 ltoedir = ln->position.dirto(scrn->eye, &dist);

  float i, j;
  if (!scrn->getPixelCoord(-ltoedir, &i, &j) ||
      !visible(&eyenode, ln))
    return Spectrum(0.);

  class EyeNode e(i, j);          // EyeNode for pixel
  class Spectrum We;              // pixel measurement value
  float espdf, edpdf, ecos, lcos; // path survival/dir.sel. pdf and cos.
  e.eval(-ltoedir, &We, &espdf, &edpdf, &ecos);
  class Spectrum score = ln->scatter(ltoedir, &lcos);
  float invdist2 = 1./(dist*dist); // inverse square distance
  score *= We * (ecos * lcos * invdist2);
  float etolpdf = espdf * edpdf * lcos * invdist2;
  float ltoepdf = 0.;   // no chance of hitting eye point (pinhole cam)
  float w = weight(0, l, ltoedir, ltoepdf, etolpdf);

  scrn->addPixel(i, j, score * (w / nrparticles));
  return Spectrum(0.);
}

// e>0, l>0: join eye and light sub-path at intermediate nodes
const Spectrum BiDirTracer::join_intermediate(const int e, const int l)
{
  if (eyescatnodes[e] == &eyebkgnode ||
      lightscatnodes[l] == &lightbkgnode ||
```

```
      !visible(eyescatnodes[e], lightscatnodes[l]))
    return Spectrum(0.);

  class SurfaceNode *en = &eyesurfnodes[e];    // eye sub-path end
  class SurfaceNode *ln = &lightsurfnodes[l];  // light sub-path end
  double dist;                                 // dist. and dir. between en/ln
  class Vec3 ltoedir = (en->position - ln->position).normalized(&dist);
  float ecos, lcos, espdf, lspdf, edpdf, ldpdf;// cos., surv,pdf, dir.sel.pdf
  class Spectrum efr, lfr;                     // BSDF at eye/light node
  en->eval(en->indir, -ltoedir, &efr, &espdf, &edpdf, &ecos);
  ln->eval(ln->indir,  ltoedir, &lfr, &lspdf, &ldpdf, &lcos);
  float invdist2 = 1. / (dist * dist);         // inverse square distance
  float G = ecos * lcos * invdist2;            // geometric factor
  float etolpdf = espdf * edpdf * lcos * invdist2;  // transition pdf en->ln
  float ltoepdf = lspdf * ldpdf * ecos * invdist2;  // transition pdf ln->en

  float w = weight(e, l, ltoedir, ltoepdf, etolpdf);
  return en->value * efr * lfr * ln->value * (G / (en->pdf * ln->pdf) * w);
}

// joins the eye sub-path vertex of depth e with light sub-path
// vertex of depth l. e or l equal to -1 means empty sub-path.
const Spectrum BiDirTracer::joinat(const int e, const int l)
{
  class Spectrum score(0.);
  if (e==0 && l==0)
    score = join_light_eye();
  else if (e<=0 && l<=0)     // eye point on light or light node on camera
    score = Spectrum(0.);    // or both sub-paths empty: can't happen
  else if (e==-1)
    score = lightpath_on_camera(l);
  else if (l==-1)
    score = eyepath_on_light(e);
  else if (e==0)
    score = join_lightpath_with_eye(l);
  else if (l==0)
    score = join_eyepath_with_light(e);
  else
    score = join_intermediate(e, l);
  return score;
}

const Spectrum BiDirTracer::join(void)
{
  // pre-calculate probabilities of sampling the eye and light path
  // in reverse direction.
  compute_reverse_eyepath_probs();
  compute_reverse_lightpath_probs();

  class Spectrum score(0.);

  // t is total combined path length: length of the eye sub-path +
```

```
  // length of the light sub-path + 1 for the connecting segment.
  // A length of '-1' indicates an empty path (no nodes)
  for (int t=1; t<=eyepathlen+lightpathlen+1
            && t<=max_combined_path_length; t++) {
    for (int e=-1; e<=eyepathlen; e++) {  // e is eye sub-path length
      int l=t-e-1;                        // l is light sub-path length
      if (l>=-1 && l<=lightpathlen)
        score += joinat(e, l);
    }
  }

  return score;
}

const Spectrum BiDirTracer::tracepixel(const int i, const int j)
{
  // trace eye path
  eyenode = EyeNode(i,j);
  eyepathlen = trace_eye_path(&eyenode);

  // sample light sources and trace light path
  lightnode = 0;
  while (!lightnode) lightnode = lightsampler->sample(&senode, &benode);
  lightpathlen = trace_light_path(lightnode);

  // join eye and light paths
  return join();
}

// pre-calculates constants and allocates memory for arrays needed
// for rendering a frame.
void BiDirTracer::setup(int orgi, int orgj, int di, int dj)
{
  PathNode::min_light_path_depth = min_light_path_length;
  PathNode::max_light_path_depth = max_light_path_length;
  PathNode::min_eye_path_depth = min_eye_path_length;
  PathNode::max_eye_path_depth = max_eye_path_length;

  eyesurfnodes = new class SurfaceNode [PathNode::max_eye_path_depth+1];
  lightsurfnodes = new class SurfaceNode [PathNode::max_light_path_depth+1];
  eyescatnodes = new class ScatteringNode* [PathNode::max_eye_path_depth+1];
  lightscatnodes = new class ScatteringNode* [PathNode::max_light_path_depth+1];
  lightpath = new class PathNode* [PathNode::max_light_path_depth+1];
  eyepath = new class PathNode* [PathNode::max_eye_path_depth+1];
  erdpdf = new float [PathNode::max_eye_path_depth+1];
  lrdpdf = new float [PathNode::max_light_path_depth+1];
  erspdf = new float [PathNode::max_eye_path_depth+1];
  lrspdf = new float [PathNode::max_light_path_depth+1];
  erhpdf = new float [PathNode::max_eye_path_depth+1];
  lrhpdf = new float [PathNode::max_light_path_depth+1];

  int npixx = scrn->width;
```

```
  int npixy = scrn->height;
  nrparticles = npixx * npixy;
}

// undoes the effects of setup().
void BiDirTracer::cleanup(void)
{
  delete [] eyesurfnodes;
  delete [] lightsurfnodes;
  delete [] eyescatnodes;
  delete [] lightscatnodes;
  delete [] lightpath;
  delete [] eyepath;
  delete [] erdpdf;
  delete [] lrdpdf;
  delete [] erspdf;
  delete [] lrspdf;
  delete [] erhpdf;
  delete [] lrhpdf;
}

void BiDirTracer::trace(void)
{
  setup();
  for (int j=0; j<scrn->height; j++) {
    for (int i=0; i<scrn->width; i++) {
      scrn->addPixel(i, j, tracepixel(i, j));
    }
  }
  cleanup();
}
```

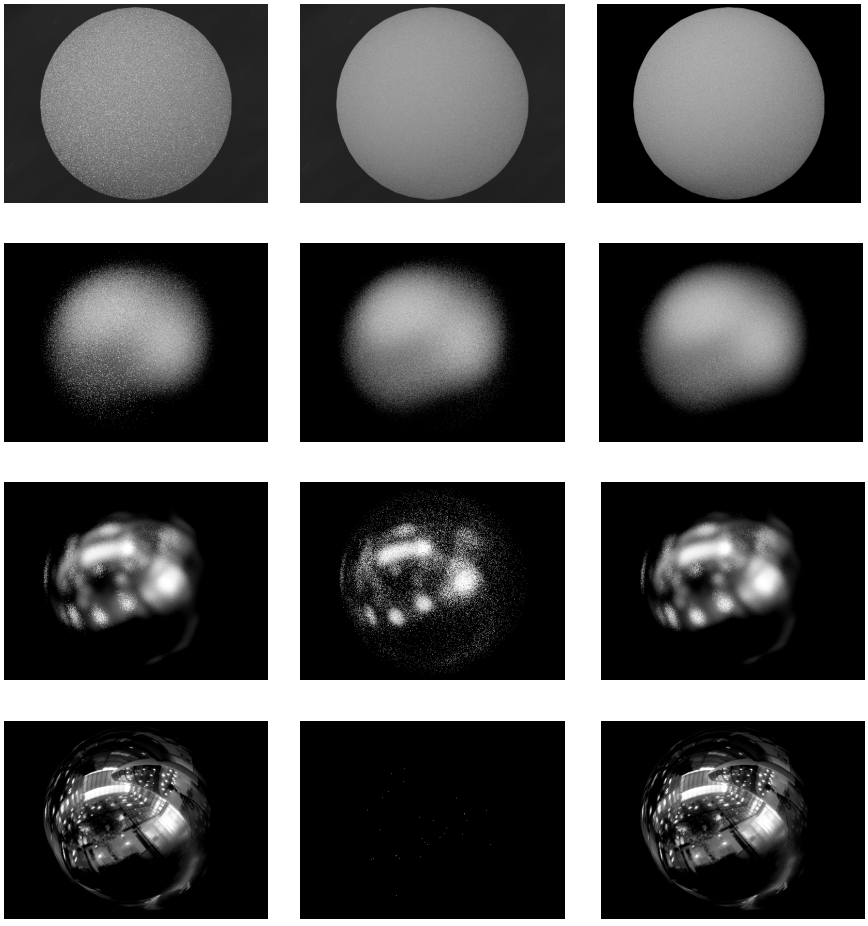

Figure A.2. Multiple importance light source sampling results obtained with the implementation shown in this section: The spheres on top are diffuse. They become more and more mirror-like towards the bottom. The left column of pictures was generated using BSDF sampling only. BSDF sampling works well for specular-like surfaces. The middle column shows results obtained with light sampling only. Light sampling is at its best for diffuse surfaces. The right column shows that combining BSDF sampling and light sampling using multiple importance sampling [201] yields better results overall. (See Plate XVII.)

B

Hemispherical Coordinates

B.1 Hemispherical Coordinates

In photorealistic rendering, one often wants to work with functions defined over a hemisphere (one-half of a sphere), centered around a surface point. A hemisphere consists of all the directions in which one can look when standing at the surface point: one can look from the horizon all the way up to the zenith and all around. A hemisphere is therefore a two-dimensional space, in which each point on the hemisphere defines a direction. Spherical coordinates are a useful way of parameterizing the hemisphere.

In the spherical coordinate system, each direction is characterized by two angles (Figure B.1). The first angle, φ, represents the azimuth and is measured with regard to an arbitrary axis located in the tangent plane at x; the second angle, θ, gives the elevation, measured from the normal vector N_x at surface point x. Writing directions using capital Greek letters, we can express direction Θ as the pair (φ, θ).

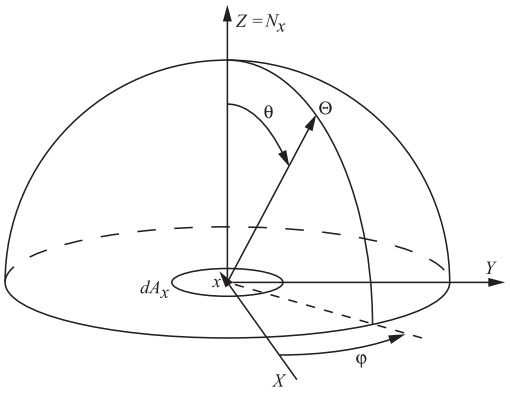

Figure B.1. Hemispherical coordinates.

The values for the angles φ and θ belong to the intervals

$$\varphi \in [0, 2\pi],$$
$$\theta \in [0, \pi/2].$$

So far, we have defined directions (or points) on the hemisphere. If we want to specify every three-dimensional point in space (not only points on the hemisphere), a distance r along the direction Θ is added. Any three-dimensional point is then defined by three coordinates (φ, θ, r). The transformation between Cartesian coordinates and spherical coordinates (place x at the origin, N_x is parallel to the Z-axis, and at the X-axis the angle $\varphi = 0$) is straightforward using some elementary trigonometry:

$$x = r \cos\varphi \sin\theta,$$
$$y = r \sin\varphi \sin\theta,$$
$$z = r \cos\theta,$$

or also

$$r = \sqrt{x^2 + y^2 + z^2},$$
$$\tan\varphi = y/x,$$
$$\tan\theta = \frac{\sqrt{x^2 + y^2}}{z}.$$

In most rendering algorithms, usually only hemispherical coordinates without the distance parameter r are used. This is because we are interested in integrating functions that are defined over directions incident at a given surface point rather than in expressing functions in three-dimensional space in full spherical coordinates.

B.2 Solid Angle

In order to integrate functions over the hemisphere, a measure on the hemisphere is needed. That measure is the solid angle.

A finite solid angle Ω subtended by an area on the hemisphere is defined as the total area divided by the squared radius of the hemisphere (Figure B.2):

$$\Omega = \frac{A}{r^2}.$$

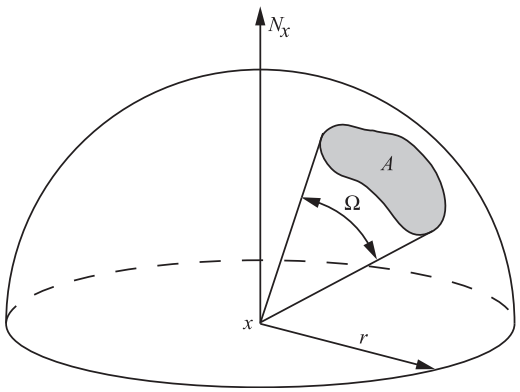

Figure B.2. Solid angle.

If the radius $r = 1$, the solid angle is simply the area on the hemisphere. Since the area of the hemisphere equals $2\pi r^2$, the solid angle covered by the entire hemisphere equals 2π; the solid angle covered by a complete sphere equals 4π. Solid angles are dimensionless but are expressed in *steradians* (sr). Note that the solid angle is not dependent on the shape of surface A, but is only dependent on the total area.

To compute the solid angle subtended by an arbitrary surface or object in space, we first project the surface or object on the hemisphere and compute the solid angle of the projection (Figure B.3). Note that two objects different in shape can still subtend the same solid angle.

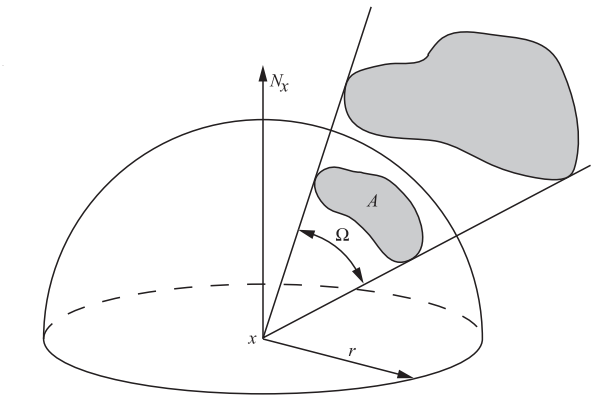

Figure B.3. Solid angle subtended by an arbitrary object.

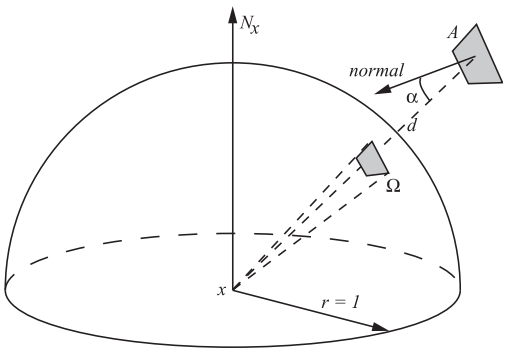

Figure B.4. Solid angle for small surfaces.

For small surfaces, the following approximation can be used to compute the solid angle subtended by a surface or object (Figure B.4):

$$\Omega = \frac{A \cos \alpha}{d^2}.$$

$A \cos \alpha$ is an approximation for the projected surface area.

B.3 Integrating over the Hemisphere

Just as we can define differential surface areas or differential volumes to integrate functions in Cartesian XY or XYZ space, we can define differential solid angles to integrate functions in hemispherical space. Compared to Cartesian spaces, there is a difference: the "area" on the hemisphere "swept" out by a differential $d\Theta$ is larger near the horizon than near the pole. The differential solid angle takes this into account by using a $\sin(\theta)$ factor (this factor can easily be deduced from the Jacobian when applying a coordinate transform from Cartesian to hemispherical coordinates).

A differential solid angle, centered around direction Θ, is then written as

$$d\omega_\Theta = \sin \theta d\theta d\varphi.$$

Integrating a function $f(\Theta) = f(\varphi, \theta)$ over the hemisphere is then expressed as

$$\int_\Omega f(\Theta) d\omega_\Theta = \int_0^{2\pi} \int_0^{\pi/2} f(\varphi, \theta) \sin \theta d\theta d\varphi.$$

Example 1 (Computing the area of the hemisphere.) Computing the area of the hemisphere can be achieved by simply integrating the differential solid angle over the entire integration domain:

$$\begin{aligned}\int_\Omega d\omega_\Theta &= \int_0^{2\pi} d\varphi \int_0^{\pi/2} \sin\theta d\theta \\ &= \int_0^{2\pi} d\varphi [-\cos\theta]_0^{\pi/2} \\ &= \int_0^{2\pi} 1 \cdot d\varphi \\ &= 2\pi.\end{aligned}$$

Example 2 (Integrating a cosine lobe.) Integrating a cosine lobe over the hemisphere is useful when working with certain BRDF models that use cosine lobes as their fundamental building blocks (e.g., the Phong or Lafortune models). A cosine lobe, centered around N_x, to the power N, can be integrated in a straightforward manner:

$$\begin{aligned}\int_\Omega \cos^N(\Theta, N_x) d\omega_\Theta &= \int_0^{2\pi} d\varphi \int_0^{\pi/2} \cos^N\theta \sin\theta d\theta \\ &= \int_0^{2\pi} d\varphi [-\frac{\cos^{N+1}\theta}{N+1}]_0^{\pi/2} \\ &= \int_0^{2\pi} \frac{1}{N+1} \cdot d\varphi \\ &= \frac{2\pi}{N+1}.\end{aligned}$$

B.4 Hemisphere-Area Transformation

In rendering algorithms, it is sometimes more convenient to express an integral over the hemisphere as an integral over visible surfaces seen from x. For example, if we want to compute all incident light at a point due to a distant light source, we can integrate over all directions within the solid angle subtended by the light source, or we can integrate over the actual area of the light source. To transform a hemispherical integral into an area integral, the relationship between a differential surface and a differential solid angle must be used:

$$d\omega_\Theta = \frac{\cos\theta_y dA_y}{r_{xy}^2}.$$

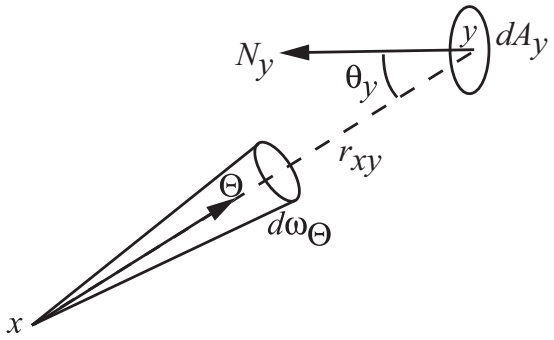

Figure B.5. Area to solid angle conversion.

The differential solid angle $d\omega_\Theta$ around direction Θ is transformed to a differential surface dA_y at surface point y (Figure B.5). Therefore, any integral over the hemisphere can also be written as an integral over each visible differential surface dA_y in each direction Θ:

$$\int_\Omega f(\Theta) d\omega_\Theta = \int_A f(y) \frac{\cos\theta_y}{r_{xy}^2} dA_y.$$

C

Theoretical Analysis of Stochastic Relaxation Radiosity

In this appendix, we show how the variance of the incremental shooting iterative algorithm of Section 6.3 can be analyzed, and demonstrate how a number of practical results can be derived from it. The analysis of the other algorithms is very similar and is a recommended exercise for the interested reader.

We start with the derivation of the variance of the incremental shooting iterative algorithm. The first thing to point out is that the resulting radiosities are obtained as the sum of increments computed in several iteration steps. We first derive the variance of a single iteration and next show how the variance on the converged results is composed from the single-iteration variances.

Variance of a single incremental shooting iteration. The variance of a single incremental shooting iteration can be derived by straightforward application of the definition of Monte Carlo summation variance:

$$S = \sum_{i=1}^{n} a_i \qquad \text{sum to be computed } (n \text{ terms}),$$

$$S \approx \frac{a_{i_s}}{p_{i_s}} \qquad \text{single-sample estimate},$$

$$V[\hat{S}] = \sum_{i=1}^{n} \frac{a_i^2}{p_i} - S^2 \qquad \text{single-sample variance}.$$

For N samples, the variance is $V[\hat{S}]/N$.

The sum to be estimated here is given in Equation 6.11. The probabilities p for picking terms from the sum are in Equation 6.12. The resulting single-sample variance of the kth incremental shooting iteration is

$$V[\hat{\Delta P}_i^{(k+1)}] = \rho_i \Delta P_T^{(k)} \Delta P_i^{(k+1)} - \left(\Delta P_i^{(k+1)}\right)^2. \qquad \text{(C.1)}$$

The latter term is usually negligible compared to the former ($\Delta P_i^{(k+1)} \ll \Delta P_T^{(k)}$).

Variance of a sequence of incremental shooting iterations until convergence. The solution P_i is eventually obtained as a sum of increments $\Delta P_i^{(k)}$ computed in each iteration step. The single-sample variance on each increment $\Delta P_i^{(k)}$ is given above in Equation C.1. Assuming that subsequent iterations are independent (which is to good approximation true in practice), and that N_k independent samples are used in the kth iteration, the variance on the result of K iterations will be

$$V[\hat{P}_i] = \sum_{k=1}^{K} \frac{1}{N_k} V[\Delta \hat{P}_i^{(k)}].$$

Optimal allocation of $N = \sum_{k=1}^{K} N_k$ samples over the individual iterations is obtained if $1/N_k$ is inversely proportional to $V[\Delta \hat{P}_i^{(k)}]$ (Section 3.6.5). For all patches i, $V[\Delta \hat{P}_i^{(k)}]$ (Equation C.1) is approximately proportional to $P_T^{(k-1)}$, suggesting that we choose the number of samples in the kth iteration proportional to the total unshot power $\Delta P_T^{(k-1)}$ to be propagated in that iteration:

$$N_k \approx N \frac{\Delta P_T^{(k-1)}}{P_T}.$$

When N_k drops below a small threshold, convergence has been reached. Combining all above results, it can be shown that the variance on the radiosity B_i after convergence is *to good approximation* given by

$$V[\hat{B}_i] \approx \frac{P_T}{N} \frac{\rho_i (B_i - B_i^e)}{A_i}. \tag{C.2}$$

Time complexity. We now turn to the question of how the number of samples N needs to be varied as a function of the number of patches n in order to compute all radiosities B_i to prescribed accuracy ε with 99.7% confidence. According to the central limit theorem (Section 3.4.4), the number of samples N shall be chosen so that

$$3\sqrt{\frac{V[\hat{B}_i]}{N}} \leq \varepsilon$$

for all i. Filling in Equation C.2 then yields

$$N \geq \frac{9 P_T}{\varepsilon^2} \cdot \max_i \frac{\rho_i (B_i - B_i^e)}{A_i}. \tag{C.3}$$

This formula allows us to examine how the number of rays to be shot must be increased as a scene to be rendered is "made larger." There are, however, many possible scenarios of how a scene can be "made larger." For instance, new objects can be added, or one can switch to a finer tessellation of the surfaces in the scene without adding new objects. If all patches in a scene are split in two, the required number of rays in order to obtain a given accuracy will need to be doubled, as dividing the patches (asymptotically) has no effect on reflectivities and radiosities. The cost of shooting a ray is often assumed to be logarithmic in the number of polygons. Although the truth is much more complicated, it is often stated that Monte Carlo radiosity algorithms have log-linear complexity. In any case, their complexity is much lower than quadratic. This result is not only valid for incremental stochastic shooting of power but also for other Monte Carlo radiosity algorithms based on shooting [169, 162, 15].

A heuristic for choosing the number of samples N. We have demonstrated that the number of samples in each incremental shooting iteration shall be chosen proportional to the amount of power to be distributed in that iteration. In other words, each ray to be shot shall propagate the same "quantum" of light energy. We have not yet answered the question of how large the quanta should be, however, or equivalently, how many rays N to choose for a complete sequence of incremental shooting iterations to convergence. That's the point of this paragraph.

Equation C.3 allows us to derive the answer. Suppose one wants to choose N so that with 99.7% confidence, the error ε on any patch i will be less than the average radiosity $B^{\text{av}} = P_T/A_T$ in the scene. The total power P_T in Equation C.3 can then be replaced by $A_T\varepsilon$. Typically, $B_i^e = 0$ for most patches in the scene. Approximating $B_i - B_i^e$ by the average radiosity, and thus by ε, then yields

$$N \approx 9 \cdot \max_i \frac{\rho_i A_T}{A_i}. \tag{C.4}$$

In practice, it makes a lot of sense to skip, for instance, the 10% of patches in a scene with the largest ratio ρ_i/A_i. Note that a rough heuristic for N suffices: a higher accuracy can always be obtained by averaging the result of several independent runs of the algorithm.

Bibliography

[1] Sameer Agarwal, Ravi Ramamoorthi, Serge Belongie, and Henrik Wann Jensen. "Structured Importance Sampling of Environment Maps." *ACM Transactions on Graphics* 22:3 (2003), 605–612.

[2] L. Alonso, F. Cuny, S. Petit Jean, J.-C. Paul, S. Lazard, and E. Wies. "The Virtual Mesh: A Geometric Abstraction for Efficiently Computing Radiosity." *ACM Transactions on Graphics* 3:20 (2001), 169–201.

[3] Thomas Annen, Jan Kautz, Frdo Durand, and Hans-Peter Seidel. "Spherical Harmonic Gradients for Mid-Range Illumination." In *Rendering Techniques 2004 Eurographics Symposium on Rendering*, pp. 331–336, 2004.

[4] A. Appel. "Some Techniques for Shading Machine Renderings of Solids." In *AFIPS 1968 Spring Joint Computer Conference*, 32, 32, 1968.

[5] J. Arvo. "Backward Ray Tracing." In *SIGGRAPH 1986 Developments in Ray Tracing course notes*, 1986.

[6] J. Arvo. "Stratified Sampling of Spherical Triangles." In *Computer Graphics Proceedings, Annual Conference Series, 1995 (ACM SIGGRAPH '95 Proceedings)*, pp. 437–438, 1995.

[7] Kavita Bala, Julie Dorsey, and Seth Teller. "Interactive Ray-Traced Scene Editing Using Ray Segment Trees." In *Tenth Eurographics Workshop on Rendering*, pp. 39–52, 1999.

[8] Kavita Bala, Julie Dorsey, and Seth Teller. "Radiance Interpolants for Accelerated Bounded-Error Ray Tracing." *ACM Transactions on Graphics* 18:3 (1999), 213–256.

[9] Kavita Bala, Bruce Walter, and Donald P. Greenberg. "Combining Edges and Points for High-Quality Interactive Rendering." *ACM Transactions on Graphics* 23:3 (SIGGRAPH 2003), 631–640.

[10] Kavita Bala. "Radiance Interpolants for Interactive Scene Editing and Ray Tracing." Ph.D. thesis, Massachusetts Institute of Technology, 1999.

[11] Ph. Bekaert and H.-P. Seidel. "A Theoretical Comparison of Monte Carlo Radiosity Algorithms." In *Proc. 6th Fall Workshop on Vision, Modeling and Visualisation 2001 (VMV01), Stuttgart, Germany*, pp. 257–264, 2001.

[12] Ph. Bekaert, L. Neumann, A. Neumann, M. Sbert, and Y. D. Willems. "Hierarchical Monte Carlo Radiosity." In *Proceedings of the 9th. Eurographics Workshop on Rendering, Vienna, Austria*, 1998.

[13] Ph. Bekaert, M. Sbert, and Y. Willems. "The Computation of Higher-Order Radiosity Approximations with a Stochastic Jacobi Iterative Method." In *16th Spring Conference on Computer Graphics, Comenius University, Bratislava, Slovakia*, 2000. 212–221.

[14] Ph. Bekaert, M. Sbert, and Y. Willems. "Weighted Importance Sampling Techniques for Monte Carlo Radiosity." In *Rendering Techniques '2000 (Proceedings of the 11th Eurographics Workshop on Rendering, Brno, Czech Rep.)*, p. 35–46. Springer Computer Science, 2000.

[15] Ph. Bekaert. "Hierarchical and Stochastic Algorithms for Radiosity." Ph.D. thesis, K. U. Leuven, Department of Computer Science, 1999.

[16] Gary Bishop, Henry Fuchs, Leonard McMillan, and Ellen J. Scher Zagier. "Frameless Rendering: Double Buffering Considered Harmful." *Computer Graphics* 28: Annual Conference Series (1994), 175–176.

[17] Ph. Blasi, B. Le Saëc, and C. Schlick. "A Rendering Algorithm for Discrete Volume Density Objects." *Computer Graphics Forum* 12:3 (1993), 201–210.

[18] K. Bouatouch, S. N. Pattanaik, and E. Zeghers. "Computation of Higher Order Illumination with a Non-Deterministic Approach." *Computer Graphics Forum* 15:3 (1996), 327–338.

[19] P. Bratley, B. L. Fox, and H. Niederreiter. "Implementation and Tests of Low-Discrepancy Sequences." *ACM Transactions on Modelling and Computer Simulation* 2:3 (1992), 195–213.

[20] M. Bunnell. "Dynamic Ambient Occlusion and Indirect Lighting." In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, edited by M. Pharr, pp. 223–233. Reading, MA: Addison-Wesley, 2005.

[21] David Burke, Abhijeet Ghosh, and Wolfgang Heidrich. "Bidirectional Importance Sampling for Direct Illumination." In *Rendering Techniques 2005: 16th Eurographics Workshop on Rendering*, pp. 147–156, 2005.

[22] S. Chandrasekhar. *Radiative Transfer*. Oxford: Oxford University Press, 1950.

[23] S. Chattopadhyay and A. Fujimoto. "Bi-directional Ray Tracing." In *Computer Graphics 1987 (Proceedings of CG International 1987)*, edited by Tosiyasu Kunii, pp. 335–43. Tokyo: Springer-Verlag, 1987.

[24] S. E. Chen, H. E. Rushmeier, G. Miller, and D. Turner. "A Progressive Multi-Pass Method for Global Illumination." In *Computer Graphics (SIGGRAPH '91 Proceedings)*, pp. 165–174, 1991.

[25] P. H. Christensen, D. H. Salesin, and T. D. DeRose. "A Continuous Adjoint Formulation for Radiance Transport." In *Fourth Eurographics Workshop on Rendering*, pp. 95–104, 1993.

[26] P. H. Christensen, E. J. Stollnitz, and D. H. Salesin. "Global Illumination of Glossy Environments Using Wavelets and Importance." *ACM Transactions on Graphics* 15:1 (1996), 37–71.

[27] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. "Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions." *ACM Transactions on Graphics* 24:3 (2005), 1166–1175.

[28] M. F. Cohen and D. P. Greenberg. "The Hemi-Cube: A Radiosity Solution for Complex Environments." *Computer Graphics (SIGGRAPH '85 Proceedings)* 19:3 (1985), 31–40.

[29] M. F. Cohen and J. R. Wallace. *Radiosity and Realistic Image Synthesis*. Boston, MA: Academic Press Professional, 1993.

[30] M. F. Cohen, D. P. Greenberg, D. S. Immel, and P. J. Brock. "An Efficient Radiosity Approach for Realistic Image Synthesis." *IEEE Computer Graphics and Applications* 6:3 (1986), 26–35.

[31] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg. "A Progressive Refinement Approach to Fast Radiosity Image Generation." In *Computer Graphics (SIGGRAPH '88 Proceedings)*, pp. 75–84, 1988.

[32] S. Collins. "Adaptive Splatting for Specular to Diffuse Light Transport." In *Fifth Eurographics Workshop on Rendering*, pp. 119–135, 1994.

[33] R. Cook and K. Torrance. "A Reflectance Model for Computer Graphics." *ACM Transactions on Graphics* 1:1 (1982), 7–24.

[34] R. L. Cook, T. Porter, and L. Carpenter. "Distributed Ray Tracing." *Computer Graphics* 18:3 (1984), 137–145.

[35] S. Daly. "Engineering Observations from Spatio-Velocity and Spatiotemporal Visual Models." *IST/SPIE Conference on Human Vision and Electronic Imaging III, SPIE* 3299 (1998), 180–191.

[36] L. M. Delves and J. L. Mohamed. *Computational Methods for Integral Equations*. Cambridge, UK: Cambridge University Press, 1985.

[37] K. Devlin, A. Chalmers, A. Wilkie, and W. Purgathofer. "Tone Reproduction and Physically Based Spectral Rendering." In *Eurographics 2002: State of the Art Reports*, pp. 101–123. Aire-la-Ville, Switzerland: Eurographics Association, 2002.

[38] Kirill Dmitriev, Stefan Brabec, Karol Myszkowski, and Hans-Peter Seidel. "Interactive Global Illumination using Selective Photon Tracing." In *Thirteenth Eurographics Workshop on Rendering*, 2002.

[39] George Drettakis and Francois X. Sillion. "Interactive Update of Global Illumination Using a Line-Space Hierarchy." In *Computer Graphics (SIGGRAPH 1997 Proceedings)*, pp. 57–64, 1997.

[40] Reynald Dumont, Fabio Pellacini, and James A. Ferwerda. "Perceptually-Driven Decision Theory for Interactive Realistic Rendering." *ACM Transactions on Graphics* 22:2 (2003), 152–181.

[41] Ph. Dutré and Y. D. Willems. "Importance-Driven Monte Carlo Light Tracing." In *Fifth Eurographics Workshop on Rendering*, pp. 185–194. Darmstadt, Germany, 1994.

[42] Ph. Dutré and Y. D. Willems. "Potential-Driven Monte Carlo Particle Tracing for Diffuse Environments with Adaptive Probability Density Functions." In *Eurographics Rendering Workshop 1995*, 1995.

[43] S. M. Ermakow. *Die Monte-Carlo-Methode und verwandte Fragen*. Berlin: V.E.B. Deutscher Verlag der Wissenschaften, 1975.

[44] M. Feda. "A Monte Carlo Approach for Galerkin Radiosity." *The Visual Computer* 12:8 (1996), 390–405.

[45] S. Fernandez, K. Bala, and D. Greenberg. "Local Illumination Environments for Direct Lighting Acceleration." *Eurographics Workshop on Rendering 2002*, pp. 7–14, 2002.

[46] Randima Fernando. *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Reading, MA: Addison-Wesley Professional, 2004.

[47] J. Ferwerda, S. Pattanaik, P. Shirley, and D. Greenberg. "A model of Visual Adaptation for Realistic Image Synthesis." In *SIGGRAPH 96 Conference Proceedings*, pp. 249–258, 1996.

[48] James A. Ferwerda, Stephen H. Westin, Randall C. Smith, and Richard Pawlicki. "Effects of Rendering on Shape Perception in Automobile Design." In *APGV 2004*, pp. 107–114, 2004.

[49] R. P. Feynman. *QED*. Princeton: Princeton University Press, 1988.

[50] G. E. Forsythe and R. A. Leibler. "Matrix Inversion by a Monte Carlo Method." *Math. Tabl. Aids. Comput.* 4 (1950), 127 – 129.

[51] C. Piatko G. Ward, H. Rushmeier. "A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes." *IEEE Transactions on Visualization and Computer Graphics* 3:4 (1997), 291–306.

[52] A. S. Glassner, editor. *An Introduction to Ray Tracing*. London: Academic Press, 1989.

[53] A. S. Glassner. "A Model for Fluorescence and Phosphorescence." In *Proceedings of the Fifth Eurographics Workshop on Rendering*, pp. 57–68, 1994.

[54] A. S. Glassner. *Principles of Digital Image Synthesis*. San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1995.

[55] J. S. Gondek, G. W. Meyer, and J. G. Newman. "Wavelength Dependent Reflectance Functions." In *Proceedings of SIGGRAPH'94*, pp. 213–220, 1994.

[56] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. "Modeling the Interaction of Light Between Diffuse Surfaces." In *SIGGRAPH '84 Conference Proceedings*, pp. 213–222, 1984.

[57] Steven Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael Cohen. "The Lumigraph." In *Computer Graphics (SIGGRAPH 1996 Proceedings)*, pp. 43–54, 1996.

[58] H. Gouraud. "Continuous Shading of Curved Surfaces." *IEEE Transactions on Computers* 20:6 (1971), 623–629.

[59] D. Greenberg, K. Torrance, P. Shirley, J. Arvo, J. Ferwerda, S. Pattanaik, E. Lafortune, B. Walter, S. Foo, and B. Trumbore. "A Framework for Realistic Image Synthesis." In *Proceedings of ACM SIGGRAPH*, pp. 44–53, 1997.

[60] E. A. Haines and D. P. Greenberg. "The Light Buffer: a Shadow Testing Accelerator." *IEEE Computer Graphics & Applications* 6:9 (1986), 6–16.

[61] J. H. Halton. "A Restrospective and Prospective Survey of the Monte Carlo Method." *SIAM Review* 12:1 (1970), 1 – 63.

[62] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. London: Methuen/Chapman and Hall, 1964.

[63] P. Hanrahan and W. Krueger. "Reflection from Layered Surfaces Due to Subsurface Scattering." In *Proceedings of SIGGRAPH 93*, pp. 165–174, 1993.

[64] P. Hanrahan, D. Salzman, and L. Aupperle. "A Rapid Hierarchical Radiosity Algorithm." In *Computer Graphics (SIGGRAPH '91 Proceedings)*, pp. 197–206, 1991.

[65] D. Hart, Ph. Dutré, and D. P. Greenberg. "Direct Illumination With Lazy Visibility Evaluation." In *Proceedings of SIGGRAPH 99*, *Computer Graphics Proceedings, Annual Conference Series*, pp. 147–154, 1999.

[66] Milos Hasan, Fabio Pellacini, and Kavita Bala. "Direct-to-Indirect Transfer for Cinematic Relighting." To appear in *SIGGRAPH: ACM Trans. Graph.*, 2006.

[67] X. D. He, K. E. Torrance, F. X. Sillion, and D. P. Greenberg. "A Comprehensive Physical Model for Light Reflection." In *Computer Graphics (SIGGRAPH 1991 Proceedings)*, pp. 175–86, 1991.

[68] E. Hecht and A. Zajac. *Optics*. Reading, MA: Addison-Wesley Publishing Company, 1979.

[69] P. S. Heckbert and J. Winget. "Finite Element Methods for Global Illumination." Technical Report UCB/CSD 91/643, Computer Science Division (EECS), University of California, Berkeley, California, USA, 1991.

[70] P. S. Heckbert. "Adaptive Radiosity Textures for Bidirectional Ray Tracing." *Computer Graphics (SIGGRAPH '90 Proceedings)* 24:4 (1990), 145–154.

[71] P. S. Heckbert. "Discontinuity Meshing for Radiosity." *Third Eurographics Workshop on Rendering*, pp. 203–226.

[72] Wolfgang Heidrich and Hans-Peter Seidel. "Realistic, Hardware-Accelerated Shading and Lighting." In *Proceedings of SIGGRAPH 99*, *Computer Graphics Proceedings, Annual Conference Series*, pp. 171–178, 1999.

[73] D. Hockney. *Secret Knowledge*. London: Thames and Hudson, 2001.

[74] Piti Irawan, James A. Ferwerda, and Stephen R. Marschner. "Perceptually Based Tone Mapping of High Dynamic Range Image Streams." In *16th Eurographics Workshop on Rendering*, pp. 231–242, 2005.

[75] A. Ishimaru. *Wave Propagation and Scattering in Random Media, Volume 1: Single Scattering and Transport Theory*. New York: Academic Press, 1978.

[76] H. W. Jensen and J. Buhler. "A Rapid Hierarchical Rendering Technique for Translucent Materials." *ACM Transactions on Graphics* 21:3 (2002), 576–581.

[77] H. W. Jensen and N. J. Christensen. "Photon Maps in Bidirectional Monte Carlo Ray Tracing of Complex Objects." *Computers & Graphics* 19:2 (1995), 215–224.

[78] H. W. Jensen and P. H. Christensen. "Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps." In *Proceedings of SIGGRAPH'98*, pp. 311–320, 1998.

[79] H. W. Jensen, J. Arvo, M. Fajardo, P. Hanrahan, D. Mitchell, M. Pharr, and P. Shirley. "State of the Art in Monte Carlo Ray Tracing for Realistic Image Synthesis." In *SIGGRAPH 2001 Course Notes (Course 29)*, 2001.

[80] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan. "A Practical Model for Subsurface Light Transport." In *Proceedings of ACM SIGGRAPH 2001*, *Computer Graphics Proceedings, Annual Conference Series*, pp. 511–518, 2001.

[81] H. W. Jensen. "Global Illumination using Photon Maps." In *Eurographics Rendering Workshop 1996*, pp. 21–30. Eurographics, 1996.

[82] H. W. Jensen. "Rendering Caustics on Non-Lambertian Surfaces." In *Proceedings of Graphics Interface 1992*, pp. 116–121. Canadian Information Processing Society, 1996.

[83] H. W. Jensen. *Realistic Image Synthesis Using Photon Mapping*. Wellesley, MA: A K Peters, 2001.

[84] T. Tawara K. Myszkowski, P. Rokita. "Preceptually-Informed Accelrated Rendering of High Quality Walkthrough Sequences." *Proceedings of the 10th Eurographics Workshop on Rendering*, pp. 5–18.

[85] J. T. Kajiya. "The Rendering Equation." *Computer Graphics (SIGGRAPH '86 Proceedings)* 20:4 (1986), 143–150.

[86] M. H. Kalos and P. Whitlock. *The Monte Carlo method. Volume 1: Basics*. J. Wiley and Sons, 1986.

[87] Jan Kautz, Peter-Pike Sloan, and John Snyder. "Fast, Arbitrary BRDF Shading for Low-Frequency Lighting using Spherical Harmonics." In *Eurographgics Rendering Workshop '02*, pp. 291–296, 2002.

[88] Jan Kautz, J. Lehtinen, and T. Aila. "Hemispherical Rasterization for Self-Shadowing of Dynamic Objects." In *Eurographics Symposium on Rendering*, pp. 179–184, 2004.

[89] A. Keller. "The Fast Calculation of Form Factors Using Low Discrepancy Sequences." In *Proceedings of the Spring Conference on Computer Graphics (SCCG '96)*, pp. 195–204, 1996.

[90] A. Keller. "Quasi-Monte Carlo Radiosity." In *Eurographics Rendering Workshop 1996*, pp. 101–110, 1996.

[91] A. Keller. "Instant Radiosity." In *SIGGRAPH 97 Conference Proceedings*, pp. 49–56, 1997.

[92] A. Keller. "Quasi-Monte Carlo methods for photorealistic image synthesis." Ph.D. thesis, Universität Kaiserslautern, Germany, 1997.

[93] A. J. F. Kok and F. W. Jansen. "Sampling Pattern Coherence for Sampling Area Light Sources." In *Third Eurographics Workshop on Rendering*, p. 283, 1992.

[94] A. J. F. Kok. "Ray Tracing and Radiosity Algorithms for Photorealistic Images Synthesis." Ph.D. thesis, Technische Universiteit Delft, The Netherlands, 1994.

[95] C. Kolb, D. Mitchell, and P. Hanrahan. "A Realistic Camera Model for Computer Graphics." In *Computer Graphics Proceedings, Annual Conference Series, 1995 (SIGGRAPH 1995)*, pp. 317–324, 1995.

[96] T. Kollig and A. Keller. "Efficient Multidimensional Sampling." *Computer Graphics Forum* 21:3 (2002), 557–564.

[97] Thomas Kollig and Alexander Keller. "Efficient Illumination by High Dynamic Range Images." In *Eurographics Symposium on Rendering: 14th Eurographics Workshop on Rendering*, pp. 45–51, 2003.

[98] R. Kress. *Linear Integral Equations*. New York: Springer Verlag, 1989.

[99] Frank Suykens-De Laet. "On Robust Monte Carlo Algorithms for Multi-Pass Global Illumination." Ph.D. thesis, Dept. of Computer Science, Katholieke Universiteit Leuven, 2002.

[100] E. P. Lafortune and Y. D. Willems. "Bi-Directional Path Tracing." In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pp. 145–153, 1993.

[101] E. P. Lafortune and Y. D. Willems. "The Ambient Term as a Variance Reducing Technique for Monte Carlo Ray Tracing." In *Fifth Eurographics Workshop on Rendering*, pp. 163–171. New York: Springer Verlag, 1994.

[102] E. P. Lafortune and Y. D. Willems. "A Theoretical Framework for Physically Based Rendering." *Computer Graphics Forum* 13:2 (1994), 97–107.

[103] E. P. Lafortune and Y. D. Willems. "A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing." In *Rendering Techniques '95 (Proceedings of the Eurographics Workshop on Rendering, Dublin, Ireland*, pp. 11–20, 1995.

[104] E. P. Lafortune and Y. D. Willems. "Rendering Participating Media with Bidirectional Path Tracing." In *Eurographics Rendering Workshop 1996*, pp. 91–100, 1996.

[105] E. P. Lafortune, Sing-Choong Foo, K. Torrance, and D. Greenberg. "Non-Linear Approximation of Reflectance Functions." In *Computer Graphics (SIGGRAPH '97 Proceedings)*, *Annual Conference Series*, pp. 117–126, 1997.

[106] E. Languenou, K. Bouatouch, and P. Tellier. "An Adaptive Discretization Method for Radiosity." *Computer Graphics Forum* 11:3 (1992), C205–C216.

[107] Greg Ward Larson and Rob Shakespeare. *Rendering with Radiance: The Art and Science of Lighting Visualization*. San Fransisco, CA: Morgan Kaufmann Books, 1998.

[108] Patrick Ledda, Alan Chalmers, Tom Troscianko, and Helge Seetzen. "Evaluation of Tone Mapping Operators using a High Dynamic Range Display." *ACM Transactions on Graphics* 24:3 (2005), 640–648.

[109] H. P. A. Lensch, M. Goesele, Ph. Bekaert, J. Kautz, M. A. Magnor, J. Lang, and Hans-Peter Seidel. "Interactive Rendering of Translucent Objects." In *Proceedings of Pacific Graphics*, pp. 214–224, 2002.

[110] Mark Levoy and Pat Hanrahan. "Light Field Rendering." In *Computer Graphics (SIGGRAPH 1996 Proceedings)*, pp. 31–42, 1996.

[111] D. Lischinski, F. Tampieri, and D. P. Greenberg. "Discontinuity Meshing for Accurate Radiosity." *IEEE Computer Graphics and Applications* 12:6 (1992), 25–39.

[112] D. Lischinski, B. Smits, and D. P. Greenberg. "Bounds and Error Estimates for Radiosity." In *Proceedings of SIGGRAPH '94*, pp. 67–74, 1994.

[113] Xinguo Liu, Peter-Pike Sloan, Heung-Yeung Shum, and John Snyder. "All-Frequency Precomputed Radiance Transfer for Glossy Objects." In *Rendering Techniques 2004 Eurographics Symposium on Rendering*, 2004.

[114] Thurstone L.L. "The Method of Paired Comparisons for Social Values." *Journal of Abnormal and Social Psychology* :21 (1927), 384–400.

[115] G. Meyer M. Bolin. "A Perceptually Based Adaptive Sampling Algorithm." *SIGGRAPH 98 Conference Proceedings*, pp. 299–310.

[116] D. Greenberg M. Ramasubramanian, S. Pattanaik. "A Perceptually Based Physical Error Metric for Realistic Image Synthesis." *SIGGRAPH 99 Conference Proceedings*, pp. 73–82.

[117] Yang J.N. Maloney L.T. "Maximum Likelihood Difference Scaling." *Journal of Vision* 3:8 (2003), 573–585.

[118] Vincent Masselus. "A Practical Framework for Fixed Viewpoint Image-based Relighting." Ph.D. thesis, Dept. of Computer Science, Katholieke Universiteit Leuven, 2004.

[119] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, H. Teller, and E. Teller. "Equations of State Calculations by Fast Computing Machines." *Journal of Chemical Physics* 21:6 (1953), 1087–1092.

[120] Tomas Möller and Eric Haines. *Real-Time Rendering*. Natcik, MA: A K Peters, 1999.

[121] K. Myszkowski. "Lighting Reconstruction Using Fast and Adaptive Density Estimation Techniques." In *Eurographics Rendering Workshop 1997*, pp. 251–262, 1997.

[122] K. Myszkowski. "The Visible Differences Predictor: Applications to Global Illumination Problems." *Proceedings of the Ninth Eurographics Workshop on Rendering*, pp. 223–236.

[123] L. Neumann, M. Feda, M. Kopp, and W. Purgathofer. "A New Stochastic Radiosity Method for Highly Complex Scenes." In *Fifth Eurographics Workshop on Rendering*, pp. 195–206. Darmstadt, Germany, 1994.

[124] L. Neumann, W. Purgathofer, R. Tobler, A. Neumann, P. Elias, M. Feda, and X. Pueyo. "The Stochastic Ray Method for Radiosity." In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, 1995.

[125] L. Neumann, R. F. Tobler, and P. Elias. "The Constant Radiosity Step." In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pp. 336–344, 1995.

[126] A. Neumann, L. Neumann, Ph. Bekaert, Y. D. Willems, and W. Purgathofer. "Importance-Driven Stochastic Ray Radiosity." In *Eurographics Rendering Workshop 1996*, pp. 111–122, 1996.

[127] L. Neumann, A. Neumann, and Ph. Bekaert. "Radiosity with Well Distributed Ray Sets." *Computer Graphics Forum* 16:3.

[128] L. Neumann. "Monte Carlo Radiosity." *Computing* 55:1 (1995), 23–42.

[129] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. "All-Frequency Shadows using Non-Linear Wavelet Lighting Approximation." *ACM Transactions on Graphics* 22:3 (2003), 376–381.

[130] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. "Triple Product Wavelet Integrals for All-Frequency Relighting." *ACM Transactions on Graphics* 23:3 (2004), 477–487.

[131] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. "Geometric Considerations and Nomenclature for Reflectance." In *Monograph 161*. National Bureau of Standards (US), 1977.

[132] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, CBMS-NSF regional conference series in Appl. Math., 63. Philadelphia: SIAM, 1992.

[133] T. Nishita and E. Nakamae. "Continuous Tone Representation of 3-D Objects Taking Account of Shadows and Interreflection." *Computer Graphics (SIGGRAPH '85 Proceedings)* 19:3 (1985), 23–30.

[134] Marc Olano, John C. Hart, Wolfgang Heidrich, and Michael McCool. *Real-Time Shading*. Natick, MA: A K Peters, 2001.

[135] Guilford J. P. *Psychometric Methods*. New York: McGraw-Hill, 1954.

[136] E. Paquette, P. Poulin, and G. Drettakis. "A Light Hierarchy for Fast Rendering of Scenes with Many Lights." *Eurographics 98* 17:3 (1998), pp. 63–74.

[137] Steven Parker, William Martin, Peter-Pike Sloan, Peter Shirley, Brian Smits, and Chuck Hansen. "Interactive Ray Tracing." In *Interactive 3D Graphics (I3D)*, pp. 119–126, 1999.

[138] S. N. Pattanaik and S. P. Mudur. "Computation of Global Illumination by Monte Carlo Simulation of the Particle Model of Light." *Third Eurographics Workshop on Rendering*, pp. 71–83.

[139] S. N. Pattanaik and S. P. Mudur. "The Potential Equation and Importance in Illumination Computations." *Computer Graphics Forum* 12:2 (1993), 131–136.

[140] S. N. Pattanaik and S. P. Mudur. "Adjoint Equations and Random Walks for Illumination Computation." *ACM Transactions on Graphics* 14:1 (1995), 77–102.

[141] S. Pattanaik, J. Tumblin, H. Yee, and D. Greenberg. "Time-Dependent Visual Adaption for Fast Realistic Image Display." *Proceedings of SIGGRAPH 2000*, pp. 47–54.

[142] M. Pellegrini. "Monte Carlo Approximation of Form Factors with Error Bounded A Priori." In *Proc. of the 11th. annual symposium on Computational Geometry*, pp. 287 – 296. New York: ACM Press, 1995.

[143] F. Perez-Cazorla, X. Pueyo, and F. Sillion. "Global Illumination Techniques for the Simulation of Participating Media." In *Proceedings of the Eighth Eurographics Workshop on Rendering*. Saint Etienne, France, 1997.

[144] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Reading, MA: Addison-Wesley Professional, 2005.

[145] M. Pharr and P. M. Hanrahan. "Monte Carlo Evaluation Of Non-Linear Scattering Equations For Subsurface Reflection." In *Proceedings of ACM SIGGRAPH 2000*, *Computer Graphics Proceedings, Annual Conference Series*, pp. 75–84, 2000.

[146] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. "Rendering Complex Scenes with Memory-Coherent Ray Tracing." In *Computer Graphics (SIGGRAPH 1997 Proceedings)*, pp. 101–108, 1997.

[147] Bui-T. Phong and F. C. Crow. "Improved Rendition of Polygonal Models of Curved Surfaces." In *Proceedings of the 2nd USA-Japan Computer Conference*, 1975.

[148] A. J. Preetham, P. Shirley, and B. Smits. "A Practical Analytic Model for Daylight." In *SIGGRAPH 99 Conference Proceedings, Annual Conference Series*, pp. 91–100, 1999.

[149] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in* FORTRAN, Second edition. Cambridge, UK: Cambridge University Press, 1992.

[150] Paul Rademacher, Jed Lengyel, Ed Cutrell, and Turner Whitted. "Measuring the Perception of Visual Realism in Images." In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, pp. 235–248, 2001.

[151] Ravi Ramamoorthi and Pat Hanrahan. "An Efficient Representation for Irradiance Environment Maps." In *SIGGRAPH '01: Proceedings*

of the 28th annual conference on Computer graphics and interactive techniques, pp. 497–500, 2001.

[152] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. "Multi-Level Ray Tracing Algorithm." *SIGGRAPH: ACM Trans. Graph.* 24:3 (2005), 1176–1185.

[153] R. Y. Rubinstein. *Simulation and the Monte Carlo method.* New York: J. Wiley and Sons, 1981.

[154] H. E. Rushmeier and K. E. Torrance. "The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium." In *Computer Graphics (Proceedings of SIGGRAPH 87)*, pp. 293–302, 1987.

[155] L. Santaló. *Integral Geometry and Geometric Probability.* Reading, Mass: Addison-Welsey, 1976.

[156] Mirko Sattler, Ralf Sarlette, Thomas Mücken, and Reinhard Klein. "Exploitation of Human Shadow Perception for Fast Shadow Rendering." In *APGV 2005*, pp. 131–134, 2005.

[157] M. Sbert, X. Pueyo, L. Neumann, and W. Purgathofer. "Global Multipath Monte Carlo Algorithms for Radiosity." *The Visual Computer* 12:2 (1996), 47–61.

[158] M. Sbert, A. Brusi, R. Tobler, and W. Purgathofer. "Random Walk Radiosity with Generalized Transition Probabilities." Technical Report IIiA-98-07-RR, Institut d'Informàtica i Aplicacions, Universitat de Girona, 1998.

[159] M. Sbert, A. Brusi, and Ph. Bekaert. "Gathering for Free in Random Walk Radiosity." In *Rendering Techniques '99 (Proceedings of the 10th Eurographics Workshop on Rendering, Granada, Spain)*, pp. 97–102. Springer Computer Science, 1999.

[160] M. Sbert. "An Integral Geometry Based Method for Fast Form-Factor Computation." *Computer Graphics Forum* 12:3 (1993), C409–C420.

[161] M. Sbert. "The Use of Global Random Directions to Compute Radiosity—Global Monte Carlo Techniques." Ph.D. thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 1996.

[162] M. Sbert. "Error and Complexity of Random Walk Monte Carlo Radiosity." *IEEE Transactions on Visualization and Computer Graphics* 3:1 (1997), 23–38.

[163] M. Sbert. "Optimal Source Selection in Shooting Random Walk Monte Carlo Radiosity." *Computer Graphics Forum* 16:3 (1997), 301–308.

[164] P. Schröder. "Numerical Integration for Radiosity in the Presence of Singularities." In *4 th Eurographics Workshop on Rendering, Paris, France*, pp. 177–184, 1993.

[165] Peter Shirley and Kenneth Chiu. "A Low Distortion Map Between Disk and Square." *Journal of Graphics Tools* 2:3 (1997), 45–52.

[166] P. Shirley, B. Wade, Ph. M. Hubbard, D. Zareski, B. Walter, and Donald P. Greenberg. "Global Illumination via Density Estimation." In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pp. 219–230, 1995.

[167] P. Shirley. "A Ray Tracing Method for Illumination Calculation in Diffuse–Specular Scenes." In *Graphics Interface '90*, pp. 205–212, 1990.

[168] P. Shirley. "Radiosity via Ray Tracing." In *Graphics Gems II*, edited by J. Arvo, pp. 306–310. Boston: Academic Press, 1991.

[169] P. Shirley. "Time Complexity of Monte Carlo Radiosity." In *Eurographics '91*, pp. 459–465, 1991.

[170] P. Shirley. *Realistic Ray Tracing*. Natick, MA: A K Peters, 2000.

[171] F. Sillion and C. Puech. "A General Two-Pass Method Integrating Specular and Diffuse Reflection." In *Computer Graphics (SIGGRAPH '89 Proceedings)*, pp. 335–344, 1989.

[172] F. Sillion and C. Puech. *Radiosity and Global Illumination*. San Francisco: Morgan Kaufmann, 1994.

[173] F. Sillion, J. Arvo, S. Westin, and D. Greenberg. "A Global Illumination Solution for General Reflectance Distributions." *Computer Graphics (SIGGRAPH '91 Proceedings)* 25:4 (1991), 187–196.

[174] F. Sillion. "A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters." *IEEE Transactions on Visualization and Computer Graphics* 1:3 (1995), 240–254.

[175] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. London: Chapman and Hall, 1986.

[176] Maryann Simmons and Carlo H. Séquin. "Tapestry: A Dynamic Mesh-based Display Representation for Interactive Rendering." In *Eleventh Eurographics Workshop on Rendering*, pp. 329–340, 2000.

[177] Peter-Pike Sloan, Jan Kautz, and John Snyder. "Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments." In *SIGGRAPH '02*, pp. 527–536, 2002.

[178] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. "Clustered Principal Components for Precomputed Radiance Transfer." *ACM Transactions on Graphics* 22:3 (2003), 382–391.

[179] Peter-Pike Sloan, Xinguo Liu, Heung-Yeung Shum, and John Snyder. "Bi-Scale Radiance Transfer." *ACM Trans. Graph.* 22:3 (2003), 370–375.

[180] Peter-Pike Sloan, Ben Luna, and John Snyder. "Local, Deformable Precomputed Radiance Transfer." *ACM Trans. Graph.* 24:3 (2005), 1216–1224.

[181] B. Smits, J. Arvo, and D. Salesin. "An Importance-Driven Radiosity Algorithm." In *Computer Graphics (SIGGRAPH '92 Proceedings)*, pp. 273–282, 1992.

[182] B. Smits, J. Arvo, and D. Greenberg. "A Clustering Algorithm for Radiosity in Complex Environments." In *SIGGRAPH '94 Proceedings*, pp. 435–442, 1994.

[183] J. Spanier and E. M. Gelbard. *Monte Carlo Principles and Neutron Transport Problems*. Reading, MA: Addison-Wesley, 1969.

[184] J. Stam and E. Languenou. "Ray Tracing in Non-Constant Media." In *Proceedings of the 7th Eurographics Workshop on Rendering*, pp. 225–234, 1996.

[185] J. Stam. "Multiple Scattering as a Diffusion Process." In *Proceedings of the 6th Eurographics Workshop on Rendering*, pp. 51–58, 1995.

[186] J. Stam. "Diffraction Shaders." In *SIGGRAPH 99 Conference Proceedings, Annual Conference Series*, pp. 101–110, 1999.

[187] William A. Stokes, James A. Ferwerda, Bruce Walter, and Donald P. Greenberg. "Perceptual Illumination Components: A New Approach to Efficient, High Quality Global Illumination Rendering." *ACM Transactions on Graphics* 23:3 (2004), 742–749.

[188] I. E. Sutherland. "Sketchpad–A Man-Machine Graphical Communication System." Technical Report 296, MIT Lincoln Laboratory, 1963.

[189] L. Szirmay-Kalos and W. Purgathofer. "Global Ray-bundle Tracing with Hardware Acceleration." In *Ninth Eurographics Workshop on Rendering*. Vienna, Austria, 1998.

[190] L. Szirmay-Kalos and W. Purgathofer. "Analysis of the Quasi-Monte Carlo Integration of the Rendering Equation." In *WSCG '99 (Seventh International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media)*, pp. 281–288, 1999.

[191] L. Szirmay-Kalos, T. Foris, L. Neumann, and C. Balasz. "An Analysis of Quasi-Monte Carlo Integration Applied to the Transillumination Radiosity Method." *Computer Graphics Forum (Eurographics '97 Proceedings)* 16:3. 271–281.

[192] L. Szirmay-Kalos, C. Balasz, and W. Purgathofer. "Importance-Driven Quasi-Random Walk Solution of the Rendering Equation." *Computers and Graphics* 23:2 (1999), 203–211.

[193] L. Szirmay-Kalos. "Stochastic Iteration for Non-Diffuse Global Illumination." *Computer Graphics Forum* 18:3 (1999), 233–244.

[194] Whitted T. "An Improved Illumination Model for Shaded Display." *Communications of the ACM* 23:6 (1980), 343–349.

[195] Justin Talbot, David Cline, and Parris Egbert. "Importance Resampling for Global Illumination." In *Rendering Techniques 2005: 16th Eurographics Workshop on Rendering*, pp. 139–146, 2005.

[196] Seth Teller, Kavita Bala, and Julie Dorsey. "Conservative Radiance Interpolants for Ray Tracing." In *Seventh Eurographics Workshop on Rendering*, pp. 258–269, 1996.

[197] R. Tobler, A. Wilkie, M. Feda, and W. Purgathofer. "A Hierarchical Subdivision Algorithm for Stochastic Radiosity Methods." In *Eurographics Rendering Workshop 1997*, pp. 193–204, 1997.

[198] Parag Tole, Fabio Pellacini, Bruce Walter, and Donald Greenberg. "Interactive Global Illumination." In *Computer Graphics (SIGGRAPH 2002 Proceedings)*, 2002.

[199] J. Tumblin and H. E. Rushmeier. "Tone Reproduction for Realistic Images." *IEEE Computer Graphics and Applications* 13:6 (1993), 42–48.

[200] E. Veach and L. J. Guibas. "Bidirectional Estimators for Light Transport." In *Fifth Eurographics Workshop on Rendering*, pp. 147–162. Darmstadt, Germany, 1994.

[201] E. Veach and L. J. Guibas. "Optimally Combining Sampling Techniques for Monte Carlo Rendering." In *SIGGRAPH 95 Conference Proceedings*, pp. 419–428, 1995.

[202] Eric Veach and Leonidas J. Guibas. "Metropolis Light Transport." In *Computer Graphics Proceedings, Annual Conference Series, 1997 (SIGGRAPH 1997)*, 1997.

[203] E. Veach. "Non-Symmetric Scattering in Light Transport Algorithms." In *Eurographics Rendering Workshop 1996*, pp. 81–90, 1996.

[204] E. Veach. "Robust Monte Carlo Methods for Light Transport Simulation." Ph.D. thesis, Stanford university, Department of Computer Science, 1997.

[205] Edgar Velzquez-Armendriz, Eugene Lee, Bruce Walter, and Kavita Bala. "Implementing the Render Cache and the Edge-and-Point Image on Graphics Hardware." *Graphics Interface*, 2006.

[206] V. Volevich, K. Myszkowski, A. Khodulev, and E. A. Kopylov. "Using the Visual Differences Predictor to Improve Performance of Progressive Global Illumination Computations." *ACM Transactions on Graphics* 19:2 (2000), 122–161.

[207] Ingo Wald and Philipp Slusallek. "State of the Art in Interactive Ray Tracing." In *State of the Art Reports, EUROGRAPHICS 2001*, pp. 21–42, 2001.

[208] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. "Interactive Rendering with Coherent Ray Tracing." In *Proc. of Eurographics*, pp. 153–164, 2001.

[209] J. R. Wallace, M. F. Cohen, and D. P. Greenberg. "A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods." *Computer Graphics (SIGGRAPH '87 Proceedings)* 21:4 (1987), 311–320.

[210] B. Walter, Ph. M. Hubbard, P. Shirley, and D. F. Greenberg. "Global Illumination Using Local Linear Density Estimation." *ACM Transactions on Graphics* 16:3 (1997), 217–259.

[211] Bruce Walter, George Drettakis, and Steven Parker. "Interactive Rendering using the Render Cache." In *Tenth Eurographics Workshop on Rendering*, pp. 19–30, 1999.

[212] Bruce Walter, George Drettakis, and Donald Greenberg. "Enhancing and Optimizing the Render Cache." In *Thirteenth Eurographics Workshop on Rendering*, pp. 37–42, 2002.

[213] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. "Lightcuts: A Scalable Approach to Illumination." *SIGGRAPH: ACM Trans. Graph.* 24:3 (2005), 1098–1107.

[214] Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. "Multidimensional Lightcuts." To appear in *SIGGRAPH: ACM Trans. Graph.*, 2006.

[215] Rui Wang, John Tran, and David Luebke. "All-Frequency Relighting of Non-Diffuse Objects Using Separable BRDF Approximation." In *Rendering Techniques 2004 Eurographics Symposium on Rendering*, pp. 345–354, 2004.

[216] Rui Wang, John Tran, and David Luebke. "All-Frequency Interactive Relighting of Translucent Objects with single and multiple scattering." *ACM Trans. Graph.* 24:3 (2005), 1202–1207.

[217] G. J. Ward and P. Heckbert. "Irradiance Gradients." In *Rendering in Computer Graphics (Proceedings of the Third Eurographics Workshop on Rendering)*, pp. 85–98, 1992.

[218] Greg Ward and Maryann Simmons. "The Holodeck Ray Cache: An Interactive Rendering System for Global Illumination." *ACM Transactions on Graphics* 18:4 (1999), 361–398.

[219] G. J. Ward, F. M. Rubinstein, and R. D. Clear. "A Ray Tracing Solution for Diffuse Interreflection." In *Computer Graphics (SIGGRAPH 1988 Proceedings)*, pp. 85–92, 1988.

[220] G. J. Ward. "Real Pixels." In *Graphics Gems II*, edited by James Arvo, pp. 80–83. Boston: Academic Press, 1991.

[221] G. J. Ward. "Measuring and modeling anisotropic reflection." In *Computer Graphics (SIGGRAPH 1992 Proceedings)*, pp. 265–272, 1992.

[222] G. Ward. "A Contrast-Based Scalefactor for Luminance Display." In*Graphics Gems 4*, edited by Paul S. Heckbert, pp. 415–421. Boston: Academic Press, 1994.

[223] G. J. Ward. "Adaptive Shadow Testing for Ray Tracing." In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, pp. 11–20, 1994.

[224] W. Wasow. "A Comment on the Inversion of Matrices by Random Walks." *Math. Tabl. Aids. Comput.* 6 (1952), 78–81.

[225] M. Watt. "Light-Water Interaction using Backward Beam Tracing." In *Computer Graphics (SIGGRAPH 1990 Proceedings)*, pp. 377–85, 1990.

[226] A Wilkie, R. Tobler, and W Purgathofer. "Combined Rendering of Polarization and Fluorescence Effects." In *Proceedings of Eurographics Workshop on Rendering 2001*, pp. 11–20, 2001.

[227] Hector Yee, Sumanta Pattanaik, and Donald P. Greenberg. "Spatiotemporal Sensitivity and Visual Attention for Efficient Rendering of Dynamic Environments." *ACM Transactions on Graphics* 20:1 (2001), pp. 39–65.

[228] H. R. Zatz. "Galerkin Radiosity: A Higher Order Solution Method for Global Illumination." In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pp. 213–220, 1993.

[229] Kun Zhou, Yaohua Hu, Stephen Lin, Baining Guo, and Heung-Yeung Shum. "Precomputed Shadow Fields for Dynamic Scenes." *ACM Trans. Graph.* 24:3 (2005), 1196–1201.

[230] K. Zimmerman and P. Shirley. "A Two-Pass Realistic Image Synthesis Method for Complex Scenes." In *Rendering Techniques 1995 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pp. 284–295. New York: Springer-Verlag, 1995.

Index