2007-07-28

Sampling Methods in Ray-Based Global Illumination

David Cline
Brigham Young University - Provo

SAMPLING METHODS IN RAY-BASED GLOBAL ILLUMINATION

by

David Cline

A dissertation submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Brigham Young University

December 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

David Cline

This dissertation has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Parris K. Egbert, Chair
Date	Dan R. Olsen Jr.
Date	Thomas W. Sederberg
Date	Dan Ventura
Date	Michael D. Jones

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of David Cline in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Parris K. Egbert Chair, Graduate Committee
Accepted for the Department	Parris K. Egbert Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

SAMPLING METHODS IN RAY-BASED GLOBAL ILLUMINATION

David Cline

Department of Computer Science

Doctor of Philosophy

In computer graphics, algorithms that attempt to create photographic images by simulating light transport are collectively known as *Global Illumination* methods. The most versatile of these are based on ray tracing (following ray paths through a scene), and numerical integration using random or quasi-random sampling. While ray tracing and sampling methods in global illumination have progressed much in the last two decades, the goal of fast and accurate simulation of light transport remains elusive. This dissertation presents a number of new sampling methods that attempt to address some of the shortcomings of existing global illumination algorithms.

The first part of the dissertation concentrates on memory issues related to ray tracing of large scenes. In this part, we present memory-efficient *lightweight bounding volumes* as a data structure that can substantially reduce the memory overhead of a ray tracer, allowing more complicated scenes to be ray traced without complicated caching schemes.

Part two of the dissertation concerns itself with sampling algorithms related to direct lighting, an important subset of global illumination. In this part, we develop *two stage*

importance sampling to sample the product of the BRDF function and a large light source such as an environment map. We then extend this method to include all three terms of the direct lighting equation, sampling the *triple product* of the BRDF, lighting and visibility. We show that the new sampling methods have a number of advantages over existing direct lighting algorithms, including comparatively low memory overhead, little precomputation, and the ability to sample all three terms of the direct lighting equation.

Finally, the third part of the dissertation discusses sampling algorithms in the context of general global illumination. In this part, we develop two new algorithms that attempt to improve the sampling distribution over existing techniques by exploiting information gained during the course of sampling. The first of these methods, *energy redistribution path tracing*, works by using path mutation to spread energy, and thus share sampling information, between pixels. The second method, *sample swarming*, shares information gained during sampling by keeping importance maps for each pixel in the rendered image. Whenever a new pixel is to be rendered, the maps from neighboring pixels are averaged, propagating importance information through the scene. We demonstrate that both of these methods can perform substantially better than existing global illumination algorithms in a number of common rendering contexts.

ACKNOWLEDGMENTS

No large body of work can be completed without a good deal of intellectual, financial and moral support, and this dissertation is no exception.

To start, I would like to thank my advisor, Parris Egbert, for his constant support and tolerance for my many side projects and sarcastic remarks. The other members of my committee also provided invaluable feedback related to the scope and content of the dissertation. Dan Olsen's insistance that I fill out a "5 questions" report about my research before doing a proposal was particularly helpful.

In addition to my committee I want to thank the co-authors of the papers that make up the dissertation, including Dr. Egbert, Kevin Steele, Justin Talbot, David Cardon, Kenric White and Daniel Adams. Also, I wish to acknowledge the many people who read early drafts of the papers and other dissertation chapters, including the anonymous reviewers of the papers, Phillip Slusallek for his helpful comments on the Metropolis Light Transport tutorial, and all of the people in the "Siggraph support group" who read truly rough drafts of the ERPT paper. Jon Dinerstein comes to mind as always being willing to read drafts of my work, and Justin Talbot provided a number of valuable critiques of the harder math and statistics.

Special thanks to Peter Shirley, who allowed me to audit his advanced rendering course several times. His classes provided the inspiration for several of the research topics in this dissertation. I also want to thank Dr. Shirley for inviting me to give a dry run of my SIGGRAPH presentation to his rendering class.

A number of people and organizations contributed to the scenes that were used in the dissertation. Many of the 3D objects were downloaded from the Stanford 3D scanning repository, and one of the dragon models was scanned by XYZ RGB, Inc. Most of the high dynamic range environment maps that I used were created and posted on the web by Paul Debevec. David Cardon modeled the coin scene. Kate Kuttler modeled the iguana model, working at T-Splines, LLC. Daniel Adams modeled the blender scene and provided other valuable modeling assistance. Many of the images in the dissertation were rendered using the PBRT rendering system, created by Matt Pharr and Greg Humphries.

I also wish to express my deep gratitude and love to my parents, Ernest and Margaret Ann Cline, for their unwavering support of me and my educational goals. My father passed away just a month before my defense, and while I feel his loss deeply, I recognize that none of my graduate work would have been possible without his many years of love and self sacrifice while I was growing up.

Finally, I would like to dedicate this dissertation to Dr. Douglass Campbell, who tragically died as I was beginning my doctoral studies. Dr. Campbell's challenging teaching style and continual striving for excellence were my first inspiration to pursue graduate school.

Contents

1 Introduction **1**

1.1 Global Illumination 1

1.2 Ray-Based Global Illumination Methods 3

1.3 Dissertation Overview and Scope 4

I Ray Casting **7**

2 Ray Casting Methods **9**

2.1 Ray Intersection with Scene Primitives 9

2.2 Auxiliary Data Structures 10

2.2.1 Voxel Grids 10

2.2.2 Axis-Aligned BSP Trees 11

2.2.3 Bounding Volume Hierarchies 12

2.3 The Need for Memory-Efficient Data Structures 14

3 Lightweight Bounding Volumes for Ray Tracing **15**

3.1 Introduction 16

3.2 Memory Usage in Typical Ray Tracers 17

3.2.1 Standard Geometry 17

3.2.2 Lightweight Geometry 17

3.2.3 Auxiliary Data Structures 18

3.3 Lightweight Bounding Volumes 20

3.3.1 The LBVH Data Structure 20
3.3.2 LBVH Initialization in the General Case 21
3.3.3 LBVH Initialization for Regular Tessellations 24
3.3.4 LBVH Traversal 26
3.4 Comparison to Other Acceleration Methods 28
3.4.1 Memory Usage 28
3.4.2 Render Speed 29
3.4.3 Number of Primitives per Leaf Node 30

II Sampling Methods for Direct Lighting 31

4 Direct Lighting Methods 33

4.1 The Direct Lighting Equation 34
4.2 Monte Carlo Integration 35
4.2.1 Monte Carlo Integration in 1D 35
4.2.2 Coordinate Systems and MCI in Higher Dimensions 38
4.3 Applying MCI to the Direct Lighting Equation 39
4.4 Importance Sampling Techniques for Direct Lighting 41
4.4.1 BRDF Sampling 41
4.4.2 Light Source Sampling 42
4.4.3 Multiple Importance Sampling 47
4.4.4 Resampled Importance Sampling 49
4.5 Sampling Patterns 52
4.5.1 The Trouble with Random Sampling 52
4.5.2 Stratified sampling 53
4.5.3 Poisson disk sampling. 54
4.5.4 Latin Hypercube and Multi-Jittered Sampling 55

4.5.5 Low Discrepancy Sampling Methods . . . 56
4.5.6 Correlation Between Pixels . . . 57
4.6 Measuring the Convergence of a Rendering Algorithm . . . 58
4.6.1 Color Difference . . . 58
4.6.2 Image Difference . . . 59
4.6.3 Measuring Convergence . . . 60

5 Two Stage Importance Sampling for Direct Lighting 63
5.1 Introduction . . . 64
5.1.1 Importance Sampling for Direct Lighting . . . 64
5.1.2 Two Stage Importance Sampling . . . 67
5.2 Partitioning the Light Source . . . 68
5.2.1 Environment Map Encoding . . . 68
5.2.2 Partitioning the Environment Map . . . 68
5.3 Hierarchical Warping (revisited) . . . 75
5.4 Convergence to the Product Distribution . . . 78
5.5 Variable Samples . . . 81
5.6 Results . . . 84
5.7 Conclusions and Future Work . . . 86

6 Towards Triple Product Sampling in Direct Lighting 93
6.1 Introduction . . . 94
6.2 Adding a Visibility Term to Product Sampling . . . 95
6.2.1 Visibility in Preimage Space . . . 95
6.2.2 Visibility in World Space . . . 97
6.2.3 Creating the Visibility Maps . . . 98
6.3 Unbiased Rendering . . . 99
6.4 Results . . . 101

6.5 Conclusion 103

III Sampling Methods for Global Illumination 107

7 Ray Based Global Illumination Methods 109

7.1 Introduction 109

7.2 An Excursion: Brute Force Global Illumination 110

7.3 Mathematical Framework for Global Illumination 113

7.3.1 What does Global Illumination Measure? 113

7.3.2 The Rendering Equation 114

7.3.3 Pixel Integrals and the Measurement Equation 117

7.4 Path Tracing 119

7.4.1 Sampling Light Paths in a Path Tracer 119

7.4.2 Evaluating Light Paths 121

7.4.3 Evaluating Implicit Light Paths 121

7.4.4 Evaluating Explicit Light Paths 123

7.4.5 Light Paths in a Working Path Tracer 123

7.4.6 Typical Path Tracing Results 125

7.5 Irradiance and Radiance Caching 126

7.5.1 Irradiance and Radiance 126

7.5.2 Computing and Reusing Irradiance Samples 127

7.5.3 Limitations of Irradiance Caching 127

7.5.4 Radiance Caching 128

7.6 Light Tracing 129

7.6.1 Light Paths in Light Tracing 129

7.6.2 Projecting Points onto the Image Plane 129

7.6.3 Evaluating Light Paths in Light Tracing 130

7.6.4 Benefits and Drawbacks of Starting at the Light Sources 132
7.7 Photon Mapping 132
7.7.1 Creating the Photon Map 133
7.7.2 Estimating Radiance with the Photon Map 134
7.7.3 Direct Viewing of the Photon Map and Final Gather 134
7.8 Instant Radiosity 136
7.8.1 Creating the Point Lights 137
7.8.2 Rendering based on Point Lights 137
7.8.3 Reducing the number of Point Light Evaluations 138
7.9 Bidirectional Path Tracing 139
7.9.1 Evaluating Bidirectional Light Paths 140
7.9.2 Implementation Concerns in Bidirectional Path Tracing 140
7.9.3 Results of Bidirectional Path Tracing 142
7.10 The Measurement Contribution Function 142
7.11 Metropolis Light Transport 145
7.11.1 Metropolis Sampling 146
7.11.2 Creating a Sampling Distribution 146
7.11.3 Color Images 148
7.12 From Metropolis Sampling to MLT 151
7.12.1 MLT Mutation Strategies 151
7.12.2 MLT results 158
7.13 A Simple and Robust Formulation of MLT 158

8 Energy Redistribution Path Tracing 163

8.1 Introduction 164
8.2 Sampling Issues 166
8.2.1 Monte Carlo Integration 166
8.2.2 Correlated Integrals 167

8.2.3 Energy Flow . . . 168
8.2.4 Review of Metropolis Sampling . . . 171
8.3 Energy Redistribution Sampling . . . 173
8.3.1 The Detailed Balance Flow Rule . . . 174
8.3.2 The Equal Deposition Flow Rule . . . 174
8.4 Energy Redistribution Path Tracing . . . 177
8.4.1 Ray Paths and Monte Carlo Path Density . . . 178
8.4.2 Mutation and Changes in Path Density . . . 180
8.4.3 Mutation Strategies . . . 183
8.4.4 Noise Filtering . . . 184
8.5 Results . . . 186
8.6 Conclusion and Ideas for Future Study . . . 190

9 Sample Swarming **197**
9.1 Introduction . . . 198
9.1.1 Dimensionality of the Rendering Problem . . . 198
9.1.2 Monte Carlo, World Space and Preimage Space . . . 198
9.1.3 Evaluating a Ray Path Sample . . . 199
9.1.4 Importance Sampling in Global Illumination . . . 199
9.2 Sample Swarming . . . 201
9.2.1 World Space and Preimage Space Maps . . . 202
9.2.2 Maps Used in our System . . . 203
9.2.3 Map Storage and Importance Propagation . . . 208
9.3 Results . . . 211
9.4 Discussion . . . 212
9.5 Conclusion . . . 214

10 Conclusion **219**

10.1 Ray Casting 219
10.2 Sampling Methods in Direct Lighting 220
10.3 Sampling Methods in Global Illumination 221

List of Figures

1.1 Ad-hoc lighting vs. global illumination. 1

1.2 A global illumination pixel. 2

1.3 A Monte Carlo ray path sample 3

2.1 Voxel grids. 10

2.2 Axis-aligned BSP trees. 12

2.3 Bounding volume hierarchies. 13

3.1 LBVH initialization for a list of objects. 22

3.2 A lightweight bounding volume hierarchy. 23

3.3 Regular tessellations. 25

3.4 Ray-LBVH intersection. 27

3.5 Scenes used to test LBVH. 27

4.1 The direct lighting term. 33

4.2 The left endpoint rule and MCI. 36

4.3 Area measures. 38

4.4 Naive MCI for direct lighting. 40

4.7 Light probe image. 44

4.9 Benefits of multiple importance sampling. 48

4.10 Importance sampling methods for direct lighting. 51

4.11 Random points. 52

4.12 Stratified pattern. 53

4.13 Poisson disk pattern. 54
4.14 Latin hypercube and multi-jittered point sets. 56
4.15 (0,2)-sequence, Halton sequence and Hammersley point set. 57
4.16 Fixed vs. randomized sampling. 58
4.17 Images with different amounts of noise. 60

5.1 Two stage importance sampling algorithm. 67
5.2 Example environment map region. 69
5.3 Initial partition of the environment map 70
5.4 The specular lobe of the Ashikhmin model. 71
5.5 Cascading splits into neighbor regions. 73
5.6 Environment map partitioning algorithm and example partitions 76
5.7 A step of the sample warping algorithm. 77
5.8 Convergence of two stage importance sampling vs. MIS. 79
5.9 Fixed vs. variable sampling rates. 82
5.10 A scene with a spatially varying specular exponent. 85
5.11 Convergence of two stage importance sampling in terms of render time. . . 88
5.12 Visual comparison. 89
5.13 Warping algorithm for two stage importance sampling (part 1). 90
5.14 Warping algorithm for two stage importance sampling (part 2) 91

6.1 Triple product sampling with preimage visibility maps. 96
6.2 Triple product sampling with world space visibility maps. 97
6.3 Preimage vs. world space visibility maps. 98
6.4 Unbiased triple product sampling using preimage visibility maps. 99
6.5 Close-ups of the bunny scene. 101
6.6 Close-ups of the sponza scene. 103
6.7 Bunny scene error plots. 104

6.8 Sponza sceen error plots. 105

7.1 Example scene for global illumination. 110
7.2 Brute force global illumination algorithm. 111
7.3 Brute force global illumination. 112
7.4 Explicit and implicit light paths in a typical path tracer. 120
7.5 An implicit light path. 122
7.6 An explicit light path. 124
7.7 Path tracing (64 samples per pixel). 125
7.8 Irradiance caching. 128
7.9 A light path created by connecting directly to the eye point. 131
7.10 Importance in light tracing. 131
7.11 Light tracing. 132
7.12 Photon power. 134
7.13 Radiance estimate based on the photon map. 135
7.14 Photon mapping image. 136
7.15 Instant radiosity radiance estimate. 137
7.16 Instant radiosity. 138
7.17 Connecting paths in a bidirectional path tracer. 140
7.18 A bidirectional light path. 141
7.19 Bidirectional path tracing. 142
7.20 The measurement contribution function. 144
7.21 Detailed balance. 147
7.22 The copyImage function. 149
7.23 Accumulating in color. 150
7.24 Pseudocode for MLT. 152
7.25 Exponentially distributed pixel offset. 154
7.26 Exponentially distributed angular offset. 154

7.27 Multi-chain perturbation. 157
7.28 A caustic perturbation. 158
7.29 Bidirectional path tracing and MLT. 159
7.30 MLT using the simple and robust formulation. 161

8.1 Correlated integrals in path tracing. 168
8.2 Energy flow. 169
8.3 General and detailed balance of energy flow. 170
8.4 Metropolis transport. 171
8.5 The energy redistribution sampling algorithm. 173
8.6 The equal deposition flow rule. 175
8.7 The Energy Redistribution Path Tracing algorithm. 177
8.8 A ray path. 179
8.9 Proposed mutation noise filtering. 185
8.10 The effect of different ERPT flow rules. 187
8.11 Sample chain length in ERPT. 187
8.12 Stratification of ERPT vs. MLT. 187
8.13 A scene with strong indirect lighting. 189
8.14 A scene with difficult caustic lighting. 191
8.15 Convergence of ERPT compared to path tracing and MLT. 192

9.1 World space swarming maps. 204
9.2 Preimage space swarming maps. 204
9.3 Time map. 205
9.4 Lens map. 205
9.5 Scatter direction map. 206
9.6 Light selection map. 206
9.7 Bifurcation map. 206

9.8 Scatter distance map. 207
9.9 Point on light map. 207
9.10 Importance propagation. 210
9.11 Hare Noir, slight return. 215
9.12 Paperweight. 216
9.13 Blender. 217
9.14 Blurry iguana. 218

List of Tables

3.1 Memory usage of different bounding volume schemes. 29

3.2 Render times for different bounding volume schemes. 29

5.1 Error of different sampling methods for diffuse and glossy scenes. 86

9.1 Map table. 205

9.2 Error table. 211

Chapter 1

Introduction

1.1 Global Illumination

Today, a good number of rendering programs exist that can produce output virtually indistinguishable from a photograph. Methods used to create such "synthetic photographs" are collectively known as *global illumination* algorithms. The distinguishing characteristic of a global illumination algorithm, as opposed to an ad-hoc lighting algorithm, is the goal of accounting for all light scattering events that lead to the creation of a photograph. In a very real sense, the process of global illumination is a physical simulation in which light is followed through a virtual scene and recorded on a virtual film plane.

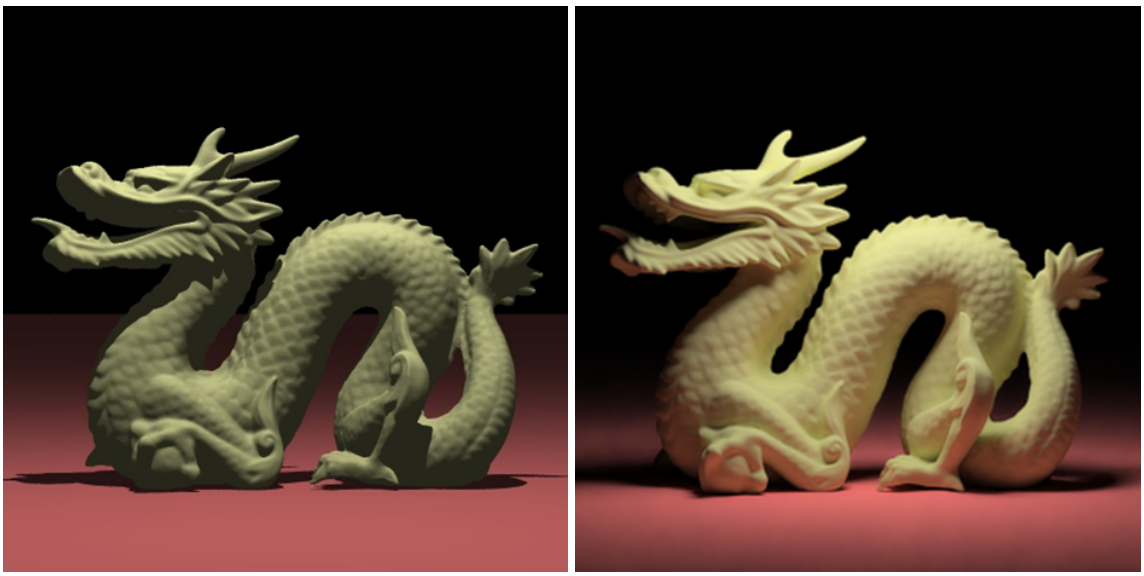
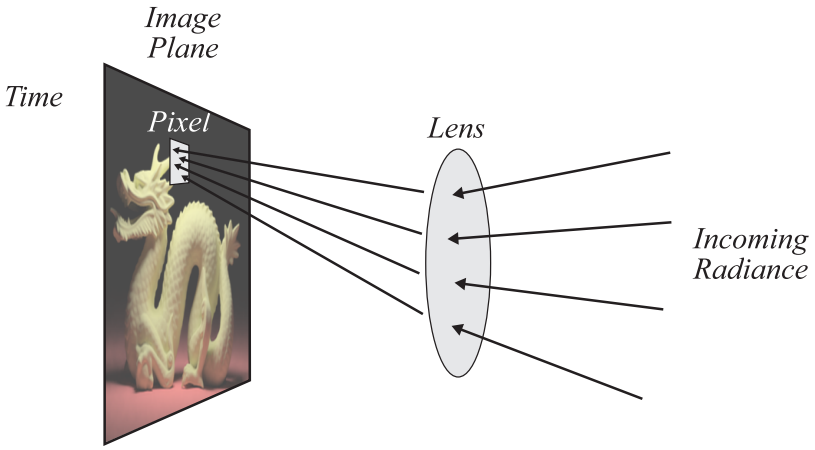


Figure 1.1: Ad-hoc lighting vs. global illumination.

Images rendered with global illumination can have a very compelling quality because they mimic subtle lighting effects that are present in real photographs. For example, the pair of images in figure 1.1 have the same geometry and a similar lighting setup, but the left image was rendered with an ad-hoc lighting algorithm, and the right image was rendered using global illumination. Note the photographic quality of the image rendered with global illumination. By contrast, the image created with ad-hoc lighting looks more like an illustration than a photograph, and is perceptually much flatter. Beyond mere appearances, however, the global illumination image is an attempt to accurately simulate a photograph of the computer-modeled scene. Since global illumination can create photographic images of objects that do not or cannot exist, it has applications in a number of industries, including lighting design, product prototyping and special effects.

To get a sense for the difficulty facing a global illumination renderer, let us consider the effort needed to calculate the value of just a single pixel in a global illumination image. The pixel itself can be considered to be the output of a light sensor on the image plane such as a CCD or tiny section of film. The job of the renderer is to compute the response of



$$Pixel\ Value = \int_{Time \times Lens \times Pixel} Incoming\ Radiance$$

Figure 1.2: The value of a pixel in a global illumination image is a high dimensional integral of the radiance striking a pixel sensor.

this sensor during the camera exposure. If we allow for photographic effects like motion blur, depth of field and antialiasing over the film plane, the output of the pixel sensor must be expressed as a five dimensional integral of the incoming radiance striking the sensor, with one dimension for time, two for lens area and two for the area of the pixel sensor, as shown in figure 1.2. Beyond these five dimensions, the incoming radiance that strikes the lens must be evaluated using the *rendering equation*, another high dimensional integral that accounts for light scattering in the scene.

1.2 Ray-Based Global Illumination Methods

Because of the difficulty of the integrals involved, the most versatile global illumination algorithms currently available tend to be based on some kind of numerical integration that samples the integrand by means of ray tracing. Kajiya [1986] was the first to publish a global illumination algorithm of this type. He defined the rendering equation and suggested *Monte Carlo integration* as a way to solve it, calling his algorithm "path tracing". Each Monte Carlo sample in path tracing consisted of a ray path that connects a point on a pixel sensor to a light source through a number of scattering events (reflections or transmissions). The value assigned to the ray path by the Monte Carlo sampler forms an unbiased

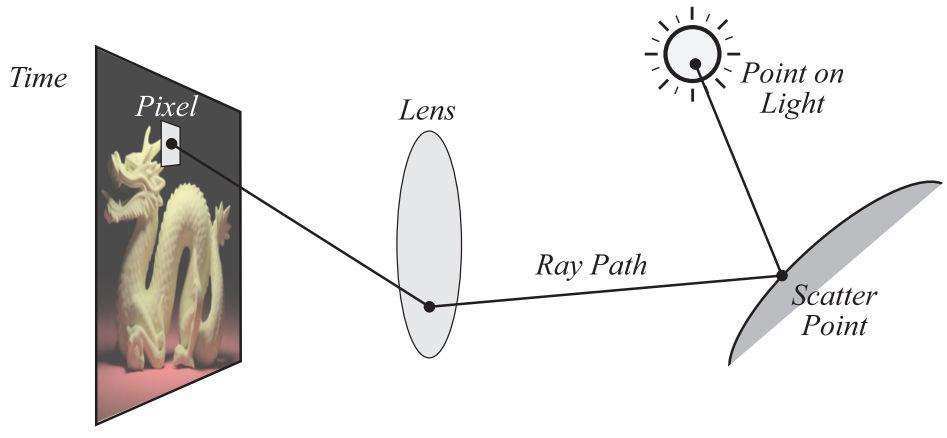


Figure 1.3: A Monte Carlo sample in a path tracer consists of a ray path that connects a point on a pixel sensor to a light source through a number of scattering events.

estimate of the true pixel value (it is correct on average, but individual samples may have a high error). The error can be reduced by averaging multiple ray paths per pixel; however, in its original form path tracing took many samples per pixel to produce high quality images. Even simple scenes would sometimes require many thousands of samples per pixel to converge, making the algorithm impractical.

While the past two decades have seen major progress in ray-based global illumination methods, the goal of fast and accurate simulation of light transport through ray sampling remains elusive. This dissertation presents a number of new algorithms that attempt to address some of the shortcomings of existing methods. The new algorithms range from very low level ray tracing optimizations to global sampling algorithms that help control the distribution of ray path samples used to create an image. The main thrust of the new sampling algorithms is to improve the sampling distribution using information gained during the course of sampling—information that cannot be provided a priori by current local sampling methods.

1.3 Dissertation Overview and Scope

This dissertation is divided into three parts, each of which explores a different topic related to sampling in ray based global illumination. Each part begins with a chapter that discusses existing work and then goes on to present one or more new algorithms in successive chapters.

Part I of the dissertation examines ray casting, the fundamental sampling operation used by all ray based renderers. The research in part I describes the use of low memory footprint *lightweight bounding volumes* as auxiliary data structures for ray tracing. Although bounding volumes are a much studied topic, most discussions concentrate on speed alone, and do not rigorously discuss memory costs. This is important since the size of scene descriptions that people want to render has more than kept pace with increases in system memory.

In part II, we discuss new sampling methods for direct lighting, a subset of global illumination. The research in this part presents a new sampling method called *two stage importance sampling*, which can quickly generate samples approximately distributed according to the product of a BRDF (the light scattering function on a surface) and an environment map or other large light source. We then extend this method to include an approximate shadow term, thus sampling the *triple product* of the BRDF, lighting and visibility.

Finally, part III presents sampling algorithms that are directly applicable to the general global illumination problem. The original work in part III consists of two algorithms that exploit coherence in path space through path reuse or mutation. The first of these methods, *energy redistribution path tracing*, works by using path mutation to spread energy, and thus share sampling information, between pixels. The second method, *sample swarming*, shares information gained during sampling by keeping importance maps for each pixel in the rendered image. Whenever a new pixel is to be rendered, the maps from neighboring pixels are averaged, propagating importance information through the scene.

Part I

Ray Casting

Chapter 2

Ray Casting Methods

Ray casting is the fundamental low level sampling operation required by all ray-based renderers. Given a geometric scene description and a ray in space, ray casting determines either the first object in the scene that the ray intersects, or whether the ray intersects an object within a predetermined distance.

2.1 Ray Intersection with Scene Primitives

Ultimately, a geometric scene description suitable for ray intersection must be composed of base level, or "primitive" objects for which direct ray intersection routines exist. Consequently, an important line of research in computer graphics has been the development and optimization of intersection algorithms for different types of object primitives. Examples of primitives for which ray intersection routines exist include spheres, triangles, polygons, quadrics, 3D curves, generalized cylinders, bicubic patches, NURBS and implicit surfaces, to name a few. Haines and Hanrahan, in [Glassner 1989], provide good surveys of the state of ray-object intersection techniques as of the late 1980s. Despite the large number of available objects, however, most ray tracers usually support only a few object types. This is true because most objects can be approximated well with polygon meshes, so it is often hard to justify the expense of implementing other object types.

2.2 Auxiliary Data Structures

Probably a more important question than what type of primitives a ray tracer should offer is how to efficiently cast rays in scenes that contain many primitives. Typical modern scenes may contain millions of primitives, and it is unfeasable to check rays against each primitive. Thus, nearly all ray tracers employ some kind of auxiliary data structure to speed up ray queries. Arvo and Kirk [Glassner 1989] and Chang [2001] have published surveys of work in this area. Here I will not provide a complete survey, but I will describe the three most popular acceleration data structures in use today: voxel grids, axis-aligned binary space partitioning (BSP) trees and bounding volume hierarchies (BVHs).

2.2.1 Voxel Grids

A voxel grid is an auxiliary data structure that subdivides space into a 3D array of rectangular cells called voxels, each of which references those scene primitives that it partially or fully encloses. To speed up ray casting, a voxel-based ray intersection routine traverses through the voxels in the order that they are pierced by the ray, only performing intersection tests with the primitives referenced by those voxels. Figure 2.1 shows an example voxel grid that encloses seven objects, in 2D rather than 3D for simplicity.

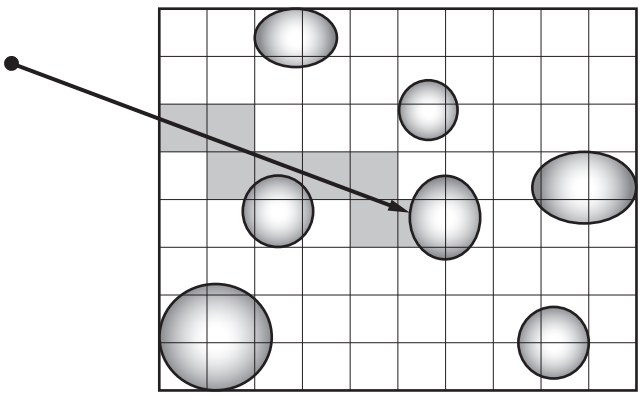


Figure 2.1: Voxel grids subdivide space into uniform cells that can be quickly traversed by a ray intersection routine. Speed is attained because the the ray only needs to be intersected with those scene primitives that fall within voxels pierced by the ray.

Fujimoto *et al.* [1986] first popularized voxel grids as an acceleration data structure, showing that regular grids can be quite efficient for some scenes, due to the simplicity of traversing the voxel data structure. The performance of regular voxel grids as an acceleration data structure is poor for some scenes, however. Scenes containing small but highly tessellated objects within a much larger scene are particularly difficult for regular voxel grids. This is commonly referred to as the "teapot in the stadium" problem. Adaptive grids, such as octrees [Glassner, 1984] can handle non-uniformly packed objects better than regular grids by adapting to local changes in scene object density. This flexibility comes at the expense of a slower and more complicated traversal routine, however. Consequently, adaptive grids have fallen out of favor as an acceleration data structure. An alternative to using a single adaptive grid is to nest regular grids. In the nested grid scheme, a top level voxel grid encloses secondary grids, which in turn either bound primitive objects or yet more grids, and so on. The effect of the nesting is to allow adaptive spatial subdivision in densly populated regions of a scene while keeping the fast, simple traversal methods used by regular voxel grids. In an early application of grid nesting, Snyder and Barr [1987] used hand generated hierarchies of grids and object lists to ray trace scenes containing hundreds of billions of instanced primitives. Later, Klimaszewski and Sederberg [1997] addressed automatic generation of nested voxel grid hierarchies. Pharr and Humphreys [2004] describe an implementation of voxel grids, and provide sample code for the method.

2.2.2 Axis-Aligned BSP Trees

Rather than partitioning space into a grid of regular cells, an axis-aligned BSP tree (also called a Kd-tree) subdivides space into a hierarchy of non-uniform rectangular regions with axis-aligned splitting planes. Leaf regions in the hierarchy reference those scene objects that they partially or fully enclose (see figure 2.2). To speed up ray casting, the intersection routine traverses the BSP tree recursively, only intersecting the ray against scene primitives referenced by the leaf nodes that the ray passes through.

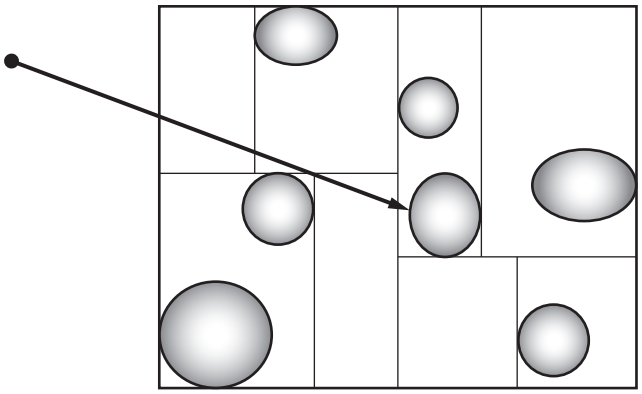


Figure 2.2: An axis-aligned BSP tree subdivides space with splitting planes aligned to the X, Y or Z axis, creating a hierarchy of rectangular regions. The BSP tree can accelerate ray casting since the intersection routine only needs to intersect the ray with scene objects referenced by those BSP leaf nodes that the ray passes through.

Kaplan [1985] was the first to describe the use of axis-aligned BSP trees to accelerate ray casting. Kaplan billed his method as an alternative way to traverse octree grids. Later, Fussell and Subramanian [1988] gave a more general description of axis aligned BSP trees as an acceleration data structure. MacDonald and Booth [1990] made a number of important observations about optimal creation of BSP trees as acceleration data structures. Wald *et al.* [2001] described a compact implementation of axis-aligned BSP trees that only requires 8 bytes per node, and made a number of important observations about cache coherence related to traversing the BVH hierarchy. More recently, Haines [2004] wrote an in-depth discussion of the surface area heuristic for determining the splitting planes in a BSP tree. Sung and Shirley [1992] and Pharr and Humphreys [2004] have published implementations of axis-aligned BSP trees.

2.2.3 Bounding Volume Hierarchies

Another type of data structure commonly used to accelerate ray casting is the bounding volume hierarchy, or BVH. The idea behind a BVH is to enclose successively smaller lists of objects at each level of the hierarchy. For example, consider the BVH shown in figure 2.3. The root node of this hierarchy encloses all seven objects in the scene, and

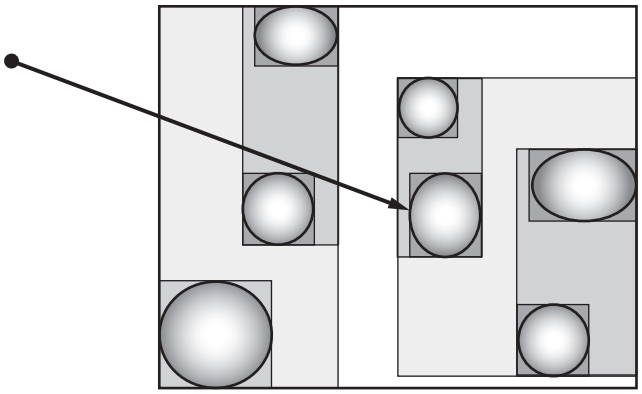


Figure 2.3: A bounding volume hierarchy, or BVH, encloses a group of objects in a hierarchy of nested bounding volumes, such as spheres or boxes. The root node of the BVH encloses the entire list of objects, and children and grandchildren enclose succesively smaller sublists. BVHs can accelerate ray casting because the intersection routine only traverses BVH nodes that the ray intersects.

the two children of the root enclose three and four objects, respectively. The children of these nodes each bound either one or two objects. Finally, leaf nodes in the hierarchy reference the scene primitives that they contain. Like BSP trees, ray intersection with a BVH proceeds in a recursive fashion. The ray is tested against the root bounding volume, and if it intersects the root the ray is tested against the root's children, and so on. Efficiency is gained because the intersection routine only has to traverse those BVH nodes that the ray intersects.

Bounding volume hierarchies were one of the first acceleration data structures introduced in the graphics literature. In an early implementation, Rubin and Whitted [1980] accelerated ray casting with hand-generated BVHs. Later work showed that automatically generated BVHs could be as good or better than those created by hand. The two main algorithms used to generate automatic hierarchies are "median split" [Kay and Kajiya, 1986] and the Goldsmith-Salmon algorithm [Goldsmith and Salmon, 1987]. Median split builds a BVH as follows: The bounding box of all the objects that will be in the hierarchy is calculated, and a list of these objects is sorted along the longest axis of the bounding box. The list is then partitioned into two sublists with equal numbers of objects. This process

is repeated on the two sublists, and so on, until some stopping criterion is met, such as a maximum tree depth or minimum number of objects in a leaf node. A variant of the median split algorithm subdivides lists of objects based on the spatial median rather than creating sublists with equal numbers of objects. The Goldsmith-Salmon algorithm is a more sophisticated technique that builds a BVH based on minimizing the surface area of the hierarchy. Objects are added to the hierarchy one by one, and a search is conducted to dermine where in the hierarchy each new object should be placed so as to minimize the increase in surface area. Smits [1998] discusses practical issues related to BVH implementation and traversal, and Shirley and Morley [2003] provide code for a BVH acceleration data structure.

2.3 The Need for Memory-Efficient Data Structures

Speed is always an issue with ray tracing, but equally critical is the need for memory efficiency. Modern scenes can contain millions of primitives, and performance suffers when the space requirements of a scene overrun physical memory. Chapter 3 discusses how to eliminate some of the momory overhead of a ray tracer through the use of low memory footprint *lightweight bounding volumes*.

Chapter 3

Lightweight Bounding Volumes for Ray Tracing

A version of this chapter was published as:

David Cline, Kevin Steele and Parris K. Egbert. "Lightweight Bounding Volumes for Ray Tracing." *Journal of Graphics Tools*. vol. 11, no. 4, pages 61-71, 2007.

Abstract. To speed up ray casting, ray tracers generally employ some kind of spatial subdivision scheme such as a voxel grid or bounding volume hierarchy. Much work has been done to optimize these data structures for speed, but less work has been published to optimize them in terms of space. This paper presents a memory-efficient auxiliary data structure for ray tracing called a *lightweight bounding volume hierarchy*, or LBVH. We show that LBVHs can be nearly as effective as standard bounding volumes in terms of speed while using significantly less memory. In addition, we show that LBVHs are particularly well suited as bounding volumes for lightweight geometry descriptions such as geometry images and tessellated NURBS surfaces.

3.1 Introduction

Managing scene complexity in ray tracing remains a challenge even when gigabytes of memory are available. Scanline renderers handle complexity by treating geometry descriptions as data that can be streamed through memory. Ray-based rendering algorithms such as path tracing, however, require random access to the entire scene description, making the stream data model impractical. The memory management model used by these algorithms has historically been to load the entire scene description into memory and render the scene from that description.

For scenes that will not fit in memory, two main types of techniques have been developed to allow random access to the scene description. The first of these techniques reduces scene memory requirements by using more efficient scene descriptions. For example, Snyder and Barr [1987] use object instancing to include similar objects in the same scene multiple times. Other work in this category is focused on direct ray tracing of complex primitives. Martin *et al.* [2000] ray trace NURBS surfaces directly, and Logie and Patterson [1995] and Smits *et al.* [2000] ray trace some types of displaced surfaces without the need to store bulky tessellations and bounding volumes. While these systems can produce impressive results for some scenes, they do not form a general framework for ray tracing all object types.

Experience has shown that, despite the memory hit, it is usually better and more flexible to tessellate complex primitives instead of trying to render them directly. To this end, a set of techniques has been developed that manage scene complexity by using geometry caching. Pharr and Hanrahan [1996] describe a ray tracing system that caches tessellations of displaced surfaces. Later, Pharr *et al.* [1997] introduce the idea of memory coherent ray tracing, in which both rays and geometry are cached to increase rendering efficiency. More recently, Christensen *et al.* [2003] describe a geometry cache that uses ray differentials to control tessellation rates.

Besides the geometry itself, nearly all ray tracers use some kind of acceleration data structure to speed ray casting. Since modern scenes often contain millions of primitives, an ideal acceleration data structure should be fast to traverse for large numbers of objects. It should also have a small memory footprint, and be quickly regenerated so that it can be used effectively in conjunction with geometry caching. In this paper we present a data structure called a *lightweight bounding volume hierarchy* or LBVH that meets these goals.

3.2 Memory Usage in Typical Ray Tracers

This section discusses the geometry memory requirements of typical ray tracers. Throughout this section we have attempted to be reasonably accurate, yet conservative, in our estimates of memory usage. We will assume that pointers and floating point numbers use 4 bytes of memory.

3.2.1 Standard Geometry

Geometry descriptions are one of the principal memory users in a ray tracer. Perhaps the most commonly rendered of these is the triangle mesh. A typical minimal configuration for a triangle mesh is to have each triangle in the mesh point to three vertex structures (12 bytes in pointers), with each vertex being used by about 6 triangles on average, so that each triangle bears one half the cost of a vertex [Shirley and Morley, 2003]. The vertex structures themselves must contain at least a position (12 bytes), but they likely include a vertex normal (12 bytes) and texture coordinates (8 bytes) as well. This yields a total memory usage per triangle of around 28 bytes.

3.2.2 Lightweight Geometry

To decrease memory usage, some scene descriptions include *lightweight geometry*, object descriptions that have a very small memory footprint per geometric primitive. Examples of lightweight geometry include height fields, geometry images, and tessellations of NURBS

or displaced surfaces. Lightweight geometry descriptions often take only a few bytes per primitive to store. For example, a geometry image that is stored as a 2D array of points uses slightly more than 6 bytes per triangle, and a height field that is represented as an array of floats consumes just over 2 bytes per triangle.

3.2.3 Auxiliary Data Structures

In order for ray tracing to be efficient, an auxiliary data structure such as a bounding volume hierarchy (BVH) or voxel grid must be used to speed intersection tests. This auxiliary data structure can use a significant amount of memory. To illustrate, consider the following reasonably efficient implementation of a BVH node similar to the one described in [Shirley and Morley, 2003]:

```
struct BVHnode {
    float minBounds[3];
    float maxBounds[3];
    BVHnode *left, *right;
};
```

Each BVH node contains six floating point numbers and two pointers, yielding 32 bytes. Smits [1998] shows how to eliminate one of the pointers in the BVH node structure by storing the nodes in an array in traversal order and using "skip" pointers instead of child pointers. Our technique takes this idea a step further, eliminating all pointers from the acceleration data structure.

Memory requirements can also be decreased by using reduced precision numbers. Terdiman [2001] describes the use of "quantized trees" to lower memory costs in a collision detection context. Mahovsky [2005] presents reduced precision hierarchies in a ray tracing context, using unsigned bytes instead of floating point numbers in the BVH nodes. To maintain precision, Mahovsky employs a hierarchical encoding scheme, which increases the BVH traversal time, however. Our algorithm also uses reduced precision numbers to

lower memory requirements, but we use a simpler, fixed encoding scheme, and maintain precision by nesting multiple bounding volume hierarchies.

A third way to decrease the amount of memory taken by a bounding volume hierarchy is to increase the branching factor (number of children per node). If we consider a hierarchy's "bounding ability" to be proportional to the number of leaf nodes, increasing the branching factor decreases the memory cost for the same bounding ability. We refer to the amount of memory that a hierarchy uses per leaf node as the *amortized cost* of leaf nodes. For a constant branching factor of k, the amortized cost of leaf nodes can be calculated as $k/(k-1)$ times the cost of a single BVH node. Hence, if the BVH nodes defined at the beginning of this section are arranged into a binary tree, the amortized cost of the leaf nodes is twice that of a single node, or 64 bytes. Bounding volume methods with a higher branching factor, such as the Goldsmith-Salmon algorithm [Goldsmith and Salmon, 1987] have a lower amortized cost per leaf node.

Acceleration data structures other than BVHs, such as the voxel grids [Fujimoto *et al.*, 1986; Klimaszewski and Sederberg, 1997] and axis-aligned BSP trees [Wald *et al.*, 2001], may fare a little better than standard bounding volumes in terms of memory footprint, but they too can require substantial amounts of memory. For voxel grids, each voxel typically contains a pointer (4 bytes), and non-empty voxels have a list of objects that intersect the voxel bounds. To achieve maximum speed, the number of voxels is often set to several times the number of objects enclosed in the grid. This causes a large percentage of the objects to span multiple voxels, which in turn increases the number of object pointers that must be stored. BSP nodes can be described compactly, using only one float and one pointer (8 bytes), so the amortized cost of leaf nodes, not including any object pointers in the leaves, is 16 bytes. However, as with voxel grids, objects may span multiple spatial partitions, so it is likely that several pointers will need to be stored for each object enclosed by the hierarchy.

3.3 Lightweight Bounding Volumes

In this section we show that in the case of bounding volume hierarchies, implicit indexing can be used to create auxiliary data structures that have no pointers at all. We then show that by combining implicit indexing with limited precision numbers, bounding volume hierarchies can be created that are easy to initialize, efficient to traverse, and require only a few bytes per enclosed object to store.

3.3.1 The LBVH Data Structure

A lightweight bounding volume hierarchy (LBVH) is an essentially complete k-ary tree that is stored in an array, and indexed like a heap. Node zero in the array is designated as the root node, and for any node q, its parent is node $\lfloor (q-1)/k \rfloor$, and its children are nodes $(qk+1)$ to $(qk+k)$. The data members of the LBVH node use limited precision numbers, typically two-byte shorts or unsigned bytes, to conserve space. One possible LBVH node is defined as follows:

```
struct LBVHnode {
    short   minBounds[3];
    short   maxBounds[3];
};
```

This structure only requires 12 bytes. Our implementation is based on a branching factor of four, so the amortized cost of leaf nodes is 4/3 the cost of a single node, or 16 bytes. We found four to be a good tradeoff between the size of the hierarchy and traversal speed. In our experiments, using a branching factor of four decreased speed by about 1.5% while saving a third of the memory cost of the hierarchy. By contrast, using a branching factor of 8 only saved an additional 9.5% of the memory cost, while being 14% slower.

As mentioned, child pointers are not needed since the hierarchy is indexed like a heap. Explicit object lists at the leaf nodes are also not needed. Instead, a simple function, which will be defined in section 3.3.4, determines which objects are enclosed by a given leaf.

In addition to the array of limited precision nodes, we store the bounding box of the whole hierarchy, B, at full precision in world space. The limited precision nodes are encoded as fractions of this world space bounding box. When the hierarchy is traversed, the bounding box will be used to transform rays from world space to the *range* of the limited precision numbers used by the LBVH (i.e. *range* = 255 for unsigned bytes and $2^{15}-1$ for shorts). The transformation is a translation followed by a scale:

$$translate = (-B_{xmin}, -B_{ymin}, -B_{zmin}) \tag{3.1}$$

$$scale = \left(\frac{range-1}{B_{xmax}-B_{xmin}}, \frac{range-1}{B_{ymax}-B_{ymin}}, \frac{range-1}{B_{zmax}-B_{zmin}} \right) \tag{3.2}$$

Note that we subtract 1 from *range* in equation 3.2 to avoid overflow, which can happen because of numerical imprecision.

To maintain tight bounds, we do not use a single LBVH for the entire scene. Instead, we enclose each polygonal mesh in its own LBVH, and then enclose these in a second level hierarchy that encompasses the whole scene. Without this step, large structures such as walls or ground planes could steal precision from finely tessellated parts of the scene.

3.3.2 LBVH Initialization in the General Case

This section discusses how to initialize an LBVH for an unorganized list of objects. Our implementation uses a variant of median split adapted to handle a branching factor of four rather than two. Instead of splitting a node once to create two children as in the median split algorithm, we split twice to create four children. In addition, we do not split object lists

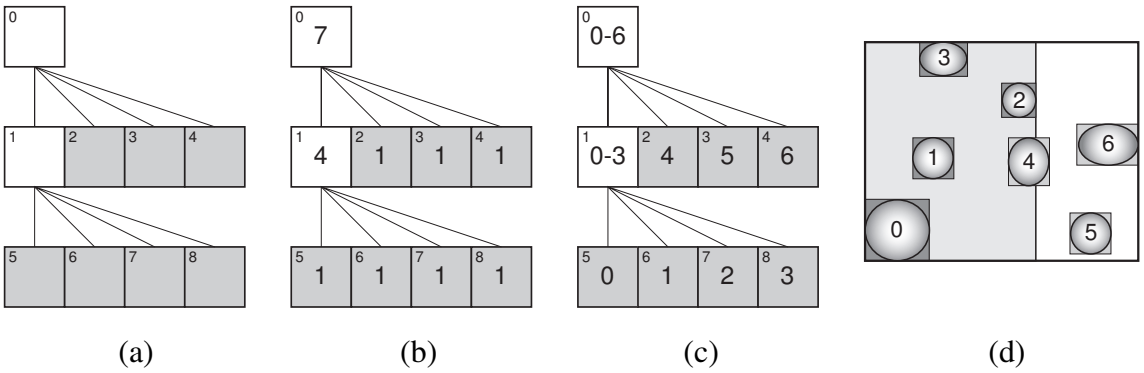
LBVH Initialization:

1. Calculate the bounding box of the n objects, B.
2. Calculate *translate* and *scale* using equations 3.1 and 3.2.
3. Calculate the number of nodes in the hierarchy, m.
   3a. $m = \lfloor 4n/3 \rfloor$.
   3b. Increment m by 1 until m % 4 equals 1.
4. Calculate how many objects will go into each node.
   4a. Assign one object to each leaf node.
   4b. Find number of objects in non-leaf nodes (sum number in children).
5. Recursively partition the objects, starting at the root node.
   5a. Store the bounding box of objects enclosed by the current node in the node.
   5b. If the current node is a leaf, return.
   5c. Sort the objects on the longest axis (in world space).
   5d. Divide the objects into two intermediate partitions.
      Let a, b, c and d be the number of objects that will be assigned to the children of the current node.
      Place $a+b$ objects in the first intermediate partition, and $c+d$ objects in the second intermediate partition.
   5e. Compute the bounding box of each intermediate partition.
   5f. Sort the objects in the intermediate partitions along their longest axes.
   5g. Place a, b, c and d objects in the four children of the current node.
   5h. Recursively partition each of the four child nodes.

Figure 3.1: Algorithm to initialize an LBVH for a list of n objects.

at the exact median. Rather, we determine how many objects should go into each node in the tree to make sure that all leaf nodes contain the same number of objects. Constructing the tree in this way guarantees that the BVH will be fully balanced, which is required for implicit indexing. Figure 3.1 gives pseudocode for initializing an LBVH. The remainder of this section describes the initialization steps in detail.

Determining the size of the LBVH. As an example of LBVH initialization, consider the problem of bounding 7 objects in an LBVH with one object in each leaf node (refer to figure 3.2). In step 1 of the algorithm, the bounding box of the seven objects is calculated. Step 2 calculates the translate and scale vectors associated with the bounding box (equations 3.1 and 3.2). In step 3, the number of nodes that will be in the hierarchy is determined



(a) (b) (c) (d)

Figure 3.2: A lightweight bounding volume hierarchy. (a) The LBVH is stored as an array of nodes. In this case, 9 total nodes are in the array, with 7 leaf nodes, shown in gray (step 3 of figure 3.1). (b) Before an LBVH can be initialized, the number of objects enclosed by each node in the hierarchy must be calculated. First, a predetermined number of objects is assigned to each leaf node, in this case one object per leaf. Then, the number of objects enclosed by internal nodes is calculated by summing the number of objects contained by their children (step 4 of figure 3.1). Finally, the LBVH is initialized based on repeated sorting of the array of objects (step 5 of figure 3.1). (c) shows the indices of the objects that are ultimately enclosed by the LBVH nodes. Note that the first leaf node on the bottom row of the hierarchy always encloses object zero. (d) provides a graphical depiction of the hierarchy, labeling the objects according to their final positions in the object array.

using the amortized cost of leaf nodes. We use a branching factor of four, so the hierarchy will contain 4/3 as many nodes as leaves, or $\lfloor 7 \times (4/3) \rfloor = 9$ total nodes. In step 3b, we increase this value slightly if necessary to make sure that all interior nodes in the hierarchy have four children. This eliminates an array bounds check during traversal.

Calculating the number of objects enclosed by LBVH nodes. Step 4 of the algorithm determines the number of objects that will be enclosed by each node in the hierarchy. We start by temporarily assigning one object (or some small fixed number) to each leaf node, and then calculate the number of objects in internal nodes by summing the number of objects in their children. Figure 3.2(a) shows the hierarchy for our example, and figure 3.2(b) shows the number of objects enclosed by LBVH nodes.

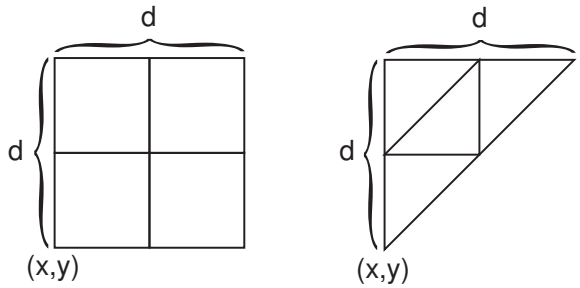
Initializing the bounding volume hierarchy. Once the number of objects assigned to the nodes in the LBVH has been established, the objects can be partitioned among the

nodes (step 5). Consider again the example of enclosing seven objects with an LBVH. To initialize the hierarchy, we transform the bounding box of the objects using equations 3.1 and 3.2, and store the result in the root node of the LBVH. Next, the objects are sorted along the longest axis of the bounding box (in world space, not limited precision space). The sorted list of objects is then partitioned so that the left partition has five objects (the sum of the number of objects enclosed by the first two children of the root, 4+1), and the right partition has two objects (the sum of the number of objects enclosed by the second two children of the root, 1+1). These partitions are then divided again by computing their bounding boxes and sorting the objects based on the corresponding longest axes. The objects are then distributed to the 4 child nodes, and the process repeats recursively for all internal nodes in the hierarchy, resulting in the configuration shown in figure 3.2(c).

3.3.3 LBVH Initialization for Regular Tessellations

This section discusses how to initialize lightweight bounding volumes for two common types of lightweight geometries, those based on rectangular and triangular subdivision. Notable examples of these include geometry images height fields. Unlike the algorithm shown in figure 3.1, an LBVH can be initialized for regular tessellations without the need for repeated sorting, making the initialization much faster and easier.

Region encoding. In both rectangular and triangular subdivision, the geometry is stored as an array of points rather than a list of triangles. We make the assumption that the x and y dimensions of this array are equal and a power of two plus one. This is the same assumption used by Christensen *et al.* [2003] in their geometry cache. Making this assumption allows us to encode array regions using a triple of position and extent. For example, the triple (x, y, d) encodes the rectangular and triangular regions shown in figures 3.3(a) and 3.3(b).



(a) (b)

Figure 3.3: Two common types of regular tessellations. Regions in both rectangular (a) and triangular (b) subdivision schemes can be parameterized using a triple of position and extent, (x, y, d).

Child regions. The encoded regions of child nodes in both the rectangular and triangular cases can easily be derived from their parents. For example, the child regions of the node encoded by the triple (x, y, d) are:

Rectangular	**Triangular**
$(x,\ y,\ d/2)$	$(x,\ y,\ d/2)$
$(x+d/2,\ y,\ d/2)$	$(x,\ y+d/2,\ d/2)$
$(x,\ y+d/2,\ d/2)$	$(x+d/2,\ y+d,\ -d/2)$
$(x+d/2,\ y+d/2,\ d/2)$	$(x+d/2,\ y+d/2,\ d/2).$

Array indexing for triangular subdivision. In the case of triangular subdivision, the array of points that defines the geometry is arranged in a triangle shape rather than a square. Consequently, a standard 2D array mapping function will not work. The mapping function that we use instead is $i = y(y-1)/2 + x$ where i is the index of point (x, y) in the array.

Initialization. As in the general case, the algorithm we use to initialize an LBVH for a regular tessellation starts by finding the bounding box of the objects and a transformation from world space to limited precision space. Next we calculate the number of nodes in the hierarchy, assuming two triangles will be enclosed by each leaf node for rectangular subdivision, and one or four will be enclosed for triangular subdivison. The LBVH is

then initialized in a bottom-up fashion, determining the bounding boxes of the leaves, and then calculating the bounding box for parents by combining the bounding boxes of their children.

3.3.4 LBVH Traversal

A ray can be intersected with an LBVH in the same way as with any other bounding volume hierarchy, except that the ray must be transformed into the coordinate system of the LBVH, and the indices of enclosed objects must be calculated by the traversal routine. During traversal, the object index corresponding to leaf j is found using the equation

$$i = \begin{cases} (j - l_b)\, n & \text{if } j \leq l_0 \\ (j - l_b + m - 1)\, n & \text{otherwise} \end{cases} \tag{3.3}$$

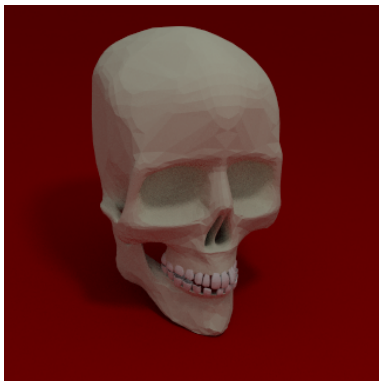
where i is the beginning index of objects enclosed by leaf node j, l_b is the node index of the first node on the bottom row of the hierarchy (5 for the hierarchy shown in figure 3.2), m is the number of leaf nodes in the hierarchy, and n is the number of objects enclosed by each leaf node. Figure 3.4 gives pseudocode for LBVH traversal. In the case of general object lists, only the node indices need to be pushed onto the stack. For regular tessellations, however, the triple describing the region covered by a node is pushed onto the stack along with the node index.

The pseudocode in figure 3.4 indicates the optimization of early traversal exit for shadow rays. Other common optimizations could be tried in the traversal as well, such as checking to see if bounding box intersections are further than the best intersection so far, implemented in the demo program, or sorting child nodes by distance along the ray, not implemented in the demo program.

LBVH Traversal

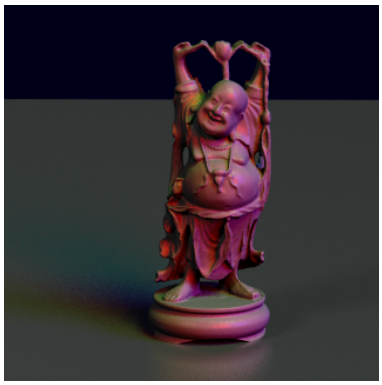
```
Transform the ray to LBVH space (translate and scale).
Push the root node onto the stack.
While the stack is not empty
    Pop a node off the stack.
    If the transformed ray intersects the node
        If the node is a leaf
            Intersect un-transformed ray with objects in the leaf.
            If the ray is a shadow ray and the ray intersected an object
                Return the intersection point.
            Else if the ray intersected an object
                Update the closest intersection point if needed.
        Else
            Push the children of the node onto the stack.
Return the closest intersection found or no intersection.
```

Figure 3.4: Algorithm to intersect a ray with an LBVH.

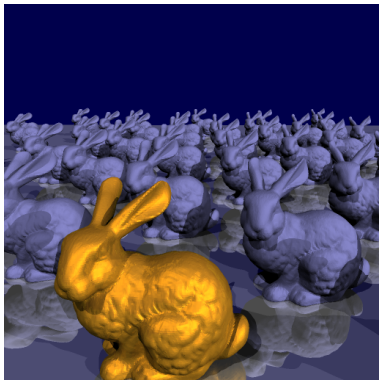


Skull (22 k tris.)

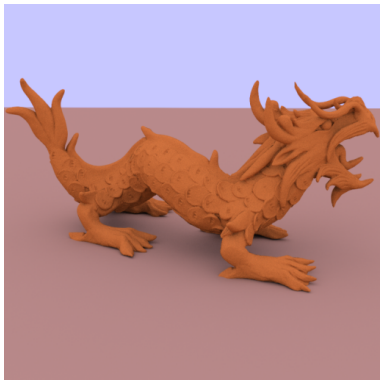
Bunny (68 k)



Buddha (1062 k)



Bunnies (3.6 M)



Dragon (6.9 M)

Angel (27 M)

Figure 3.5: Scenes used to test LBVH performance, showing the number of triangles in each scene.

3.4 Comparison to Other Acceleration Methods

This section compares the performance of lightweight bounding volumes to median split BVHs and voxel grids. To make the comparison fair, all of the methods are built as two level hierarchies, with a separate bounding structure used for each polygon mesh. We rendered several scenes of varying complexity using each of the different acceleration schemes. The test scenes are shown in figure 3.5.

3.4.1 Memory Usage

Table 3.1 compares the memory usage of different bounding volume schemes to the "typical" memory usage for the geometry itself. Voxel grids were initialized to have the same number of voxels as the number of enclosed objects. Bounding volumes were initialized to enclose one primitive in each leaf node. Standard BVH memory statistics are based on the BVHnode structure given in section 9.1. Memory usage for LBVHs using short (2 byte) integers and unsigned bytes are both listed. Triangle memory usage statistics are based on 28 bytes per triangle, assuming shared vertices. Of course, some scenes may not require as much memory per triangle, but decreasing the memory requirements of the triangles actually makes our technique more attractive because it increases the proportion of memory taken by the acceleration data structure.

As can be seen from table 3.1, standard bounding volumes with 1 primitive per leaf take more than twice as much memory as the geometry they enclose, whereas lightweight bounding volumes take only about 30 or 60 percent as much memory as the geometry (1/4 or 1/8 the memory cost of standard BVHs). Most of the voxel grid examples in the table are on par with LBVHs in terms of size, but the only example that is comparable in terms of speed, the Skull, requires 60 percent more memory than the LBVH implementation based on short integers. We believe this to be a general trend. Voxel grid methods gain speed by increasing the number of voxels in the grid, which in turn increases the number of object pointers that must be stored.

	Skull	Bunny	Buddha	Bunnies	Dragon	Angel
Geometry	0.61	1.85	29.0	100.1	192.8	749.1
BVH	1.40	4.24	66.4	228.9	440.6	-
Voxel grids	0.56	0.94	13.3	44.4	64.8	252.2
LBVH short	0.35	1.06	16.6	57.2	110.2	428.1
LBVH byte	0.17	0.53	8.3	28.6	55.1	214.0

Table 3.1: Memory usage of different bounding volume schemes compared with typical memory usage for the geometry itself. All table entries are given in megabytes. No statistics are included for the Angel scene with standard BVHs since we were unable to load the scene in this case.

	Skull	Bunny	Buddha	Bunnies	Dragon	Angel
BVH	18.5	6.6	9.2	15.3	8.0	-
Voxel grids	19.4	13.0	62.7	43.2	45.3	63.6
LBVH short	20.3	7.8	9.6	18.7	7.9	9.5
LBVH byte	21.1	9.5	16.9	23.3	37.7	57.4

Table 3.2: Comparison of render times for different bounding volume schemes. In all cases the images were rendered at 1024×1024 pixels with a single primary ray per pixel. Table entries are given in seconds.

3.4.2 Render Speed

Table 3.2 compares render times for standard and lightweight bounding volumes as well as voxel grids. As expected, LBVH times are slightly, but not significantly slower than BVHs. In most cases LBVHs using short integers perform within about 10 percent of standard BVHs, with the worst cases being about 20 percent slower. However, we were unable to load the largest test model (the angel) using standard bounding volumes on a machine with 2 GB of memory. Voxel grids slightly outperformed LBVHs in the "Skull" test, but as mentioned, they required significantly more memory than LBVHs in this case. In the other tests, voxel grids used less memory, but ran much slower.

Except for the smallest scenes, the one byte version of LBVHs ran much more slowly than the two byte version. There appears to be a direct relationship between the number of primitives in an LBVH and the performance gained by using shorts instead of

bytes. Thus, a promising strategy might be to divide large polygon meshes into smaller pieces that have only a few thousand polygons each, and then encode each of these in a separate byte-based LBVH.

3.4.3 Number of Primitives per Leaf Node

For the majority of the cases that we tried, placing a single primitive in each leaf node resulted in the fastest render times, but only by a small margin. This was the case for both standard and lightweight bounding volumes. Render time for our implementation tended to degrade gracefully with the addition of more primitives to leaf nodes, with an increase in render time of about 10% at 4 objects per leaf, and 30% at 8 objects per leaf. Based on these results, we suggest 4 objects per leaf as a default setting. That way, the memory needs of the hierarchy reduce to roughly 4 bytes per primitive, which is the same amount used up just pointing to the objects with an explicit indexing scheme.

One thing we noticed while profiling our demo program was that increasing the number of primitives has the effect of transferring some of the ray tracing bottleneck from LBVH traversal to triangle intersection. For example, in one scene that we tested triangle intersection only took about 4% of the ray tracing time when each triangle was enclosed by a leaf node, but it increased to 18% at 4 triangles per leaf and 37% at 8 triangles per leaf.

Part II

Sampling Methods for Direct Lighting

Chapter 4

Direct Lighting Methods

This chapter discusses methods to solve the direct lighting problem. Direct lighting is an important subset of global illumination that accounts for light that has scattered at most once on its way from the light source to the eye point, which I will refer to variously as the *direct lighting term* or the *single bounce radiance*. In a ray tracing context, direct lighting solutions must estimate the single bounce radiance of light flowing back towards the eye point along each primary ray. In many scenes the direct lighting term dominates the other, indirect lighting terms needed for full global illumination. Figure 4.1 demonstrates direct lighting along with full global illumination for a simple scene.

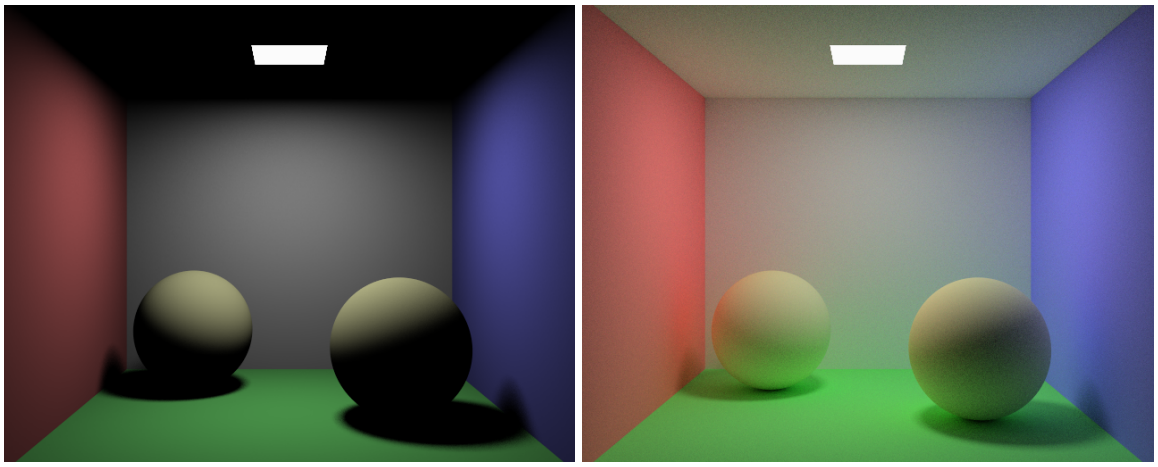
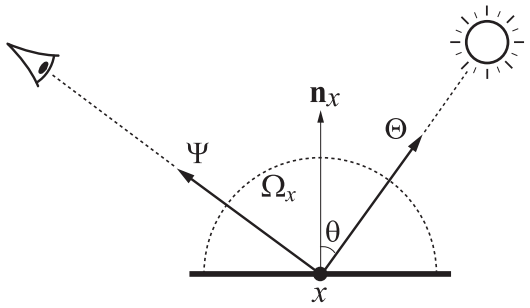


Figure 4.1: Direct lighting (left) vs. full global illumination (right). The direct lighting term is a physical simulation of the light that has bounced at most once, rather than an ad-hoc lighting solution, as represented in figure 1.1. Although the indirect lighting terms are needed to provide a photographic quality to the solution, the direct lighting term is by far the most important in this scene.

4.1 The Direct Lighting Equation

Mathematically, direct lighting can be expressed as a two dimensional integral that accounts for the single bounce radiance from point x:



$$L_d(x \rightarrow \Psi) = L_e(x \rightarrow \Psi) + \int_{\Omega_x} L_e(x \leftarrow -\Theta)\ f_r(\Psi \leftrightarrow x \leftrightarrow \Theta)\ |\cos\theta|\ d\Theta. \qquad (4.1)$$

The terms of the direct lighting equation are defined based on the local surface geometry of the point struck by an eye ray. The terms are:

x	The point struck by an eye ray.
Ψ	The direction from x back to the eye point.
$\mathbf{n}_x$	The surface normal at point x.
Ω_x	The hemisphere above point x and centered on $\mathbf{n}_x$.
Θ	A direction vector in Ω_x.
θ	The angle between Θ and $\mathbf{n}_x$.
$L_d(x \rightarrow \Psi)$	The direct lighting term, or single bounce radiance.
$L_e(x \rightarrow \Psi)$	The radiance emitted from point x in direction Ψ.
$L_e(x \leftarrow -\Theta)$	The emitted radiance that strikes point x from direction $-\Theta$.
$f_r(\Psi \leftrightarrow x \leftrightarrow \Theta)$	The BRDF function which defines the light scattering properties of the surface at point x.

Arrows in the terms of equation 4.1 indicate the direction of light flow. For example, the arrow in $L_d(x \rightarrow \Psi)$ indicates that L_d defines the radiance leaving point x, and the

arrow in $L_e(x \leftarrow -\Theta)$ indicates that this term describes the radiance arriving at point x from direction $-\Theta$. The arrows in the BRDF function, $f_r(\Psi \leftrightarrow x \leftrightarrow \Theta)$, indicate its reciprocal nature–that is, Ψ and Θ can be interchanged without affecting the value.

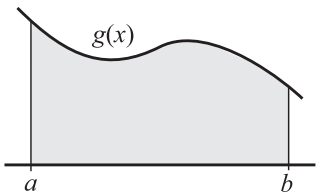
Although not explicitly stated, the integral in equation 4.1 is defined over units of *solid angle*. This distinction will become important in the context of Monte Carlo solutions to direct lighting.

4.2 Monte Carlo Integration

In practice, the direct lighting equation can almost never be evaulated symbolically. Instead, some form of numerical integration must be employed to approximate the integral value. One could of course use a fixed quadrature such as the trapezoidal rule or Simpson's rule, but it turns out that deterministic integration schemes do not work well for high dimensional integrals or for integrals of discontinuous functions, both of which apply to the direct lighting term. A better solution solution than a fixed quadrature is to use a randomized algorithm such as Monte Carlo Integration (MCI). In fact, MCI forms the basis of a large number of rendering algorithms, so this section will describe it in detail.

4.2.1 Monte Carlo Integration in 1D

As a concrete example of how Monte Carlo Integration works, consider the problem of integrating the 1D function $g(x)$ over some interval $[a,b]$:



$$\int_a^b g(x)\,dx$$

Suppose that we were to choose a random number x_i uniformly between a and b, and evaluate $g(x_i)$. In statistical terms, a randomized procedure like this is called a *random variable*. It shouldn't be hard to see that the average value of this random variable is the average of g over the interval $[a,b]$, or in other words $\frac{1}{b-a}\int_a^b g(x)\ dx$. This expression comes close to the integral we want, and in fact, we can make the expression equal the integral by dividing by the probability that point x_i was chosen, with respect to unit length. In our example, the probability with respect to unit length that any point between a and b was chosen has the constant value of $\frac{1}{b-a}$, so dividing by the probability yields the desired integral value, $\int_a^b g(x)dx$ (on average).

Monte Carlo integration works precisely in this manner. A random variable $\mathbf{X}_g$ is constructed that has expected value equal to the desired integral as follows:

1. A point x_i in the domain of the integral is chosen by some random procedure.
2. The integrand, g, is evaluated at x_i.
3. The resulting value is divided by the probability that x_i was chosen.

Intuitively, one can see why this procedure works, at least for uniform probability distributions, by comparing it to a deterministic integration rule such as the left endpoint rule. The left endpoint rule approximates the integral $\int_a^b g(x)dx$ as the area of a rectangle of width $b-a$ and height $g(a)$. MCI approximates the integral as a rectangle of height $g(x_i)$ and width $1/p_{length}(x_i)$, which for a uniform probability distribution will always equal $b-a$. Figure 4.2 shows this comparison graphically.

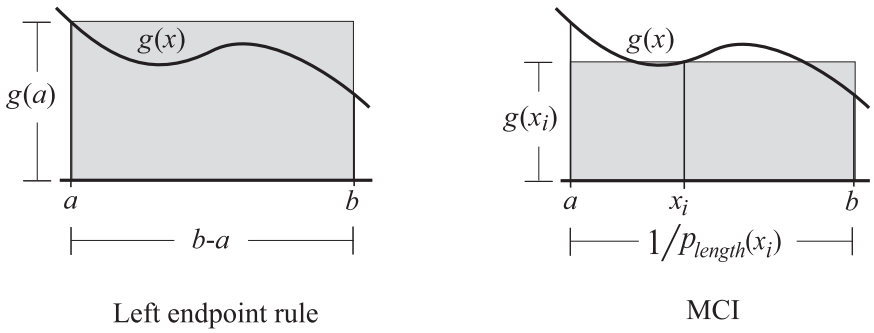


Figure 4.2: The left endpoint rule and Monte Carlo integration.

This analogy is not as straightforward for non-uniform probability distributions, but it can be shown that X_g will always produce the value of the integral "on average" as long as the probability p_{length} is non-zero wherever g is non-zero. Formally, the average value returned by a random variable such as X_g is called the random variable's *mean* or *expected value*. We write the expected value as an E with square brackets. Thus

$$\mathbf{X}_g \equiv \frac{g(x_i)}{p_{length}(x_i)} \qquad \text{and} \qquad E[\mathbf{X}_g] = \int_a^b g(x)\,dx, \tag{4.2}$$

where x_i is a value chosen by the random number generator and p_{length} is the probability that x_i was chosen, with respect to length. Other common expressions for the mean or expected value include angle brackets, $\langle \mathbf{X}_g \rangle$, the greek letter mu ($\mu$) to refer to the mean, and an overbar ($\overline{\mathbf{X}}_g$) to represent to the *sample mean*, which is the average of a population of samples drawn from $\mathbf{X}_g$.

Biased and Unbiased Monte Carlo estimators. The random variable $\mathbf{X}_g$ is said to be an *unbiased* estimator of $\int_a^b g(x)dx$ because its expected value is exactly equal to the integral. This fact has the important consequence that the error of the estimate can be reduced arbitrarily by averaging enough samples from $\mathbf{X}_g$. Because of this property, an important design goal for rendering algorithms is that they be unbiased. *Biased* estimators, on the other hand, do not give the desired answer on average. Instead, they trade off ultimate accuracy for more predictable results, which may be desirable depending on the application. The tension between biased and unbiased estimators is played out in a number of different rendering algorithms, and we will revisit the issue throughout this dissertation.

Variance in Monte Carlo estimates. While the expected value of a random variable such as $\mathbf{X}_g$ measures the average value drawn from it, the *variance* describes how far away from the average a particular draw is likely to be, or in other words, the expected error. Formally, variance is defined as the average squared distance of the random variable from

its expected value, and it is usually written as $var(\cdot)$ or σ^2. Thus

$$\sigma^2 = var(\mathbf{X}_g) = E\left[(\mathbf{X}_g - E[\mathbf{X}_g])^2\right]. \tag{4.3}$$

σ^2 is used to denote the variance because it is the mean *squared* error. The square root of the variance, σ, is called the *standard deviation*.

Equation 4.3 can be rearranged to separate $\mathbf{X}_g$ from its expected value, resulting in an alternate formula for the variance that is sometimes easier to evaluate:

$$var(\mathbf{X}_g) = E[\mathbf{X}_g^2] - E[\mathbf{X}_g]^2. \tag{4.4}$$

In later sections we will see that a large percentage of Monte Carlo rendering algorithms have been developed with no other goal in mind than decreasing the variance over existing solutions.

4.2.2 Coordinate Systems and MCI in Higher Dimensions

Monte Carlo Integration can be extended to higher dimensions by replacing p_{length} from equation 4.2 with probability with respect to area *measure* in the coordinate system of the integrand. The term measure, as used in this context, is the definition of a unit of the integration domain. A good way to think about this concept is to imagine that you are integrating the function $c(x) = 1$. For this constant function, a unit of area measure will always integrate to 1. Figure 4.3 shows two coordinate systems in $\mathbb{R}^2$ that have different area measures, Cartesian and polar coordinates.

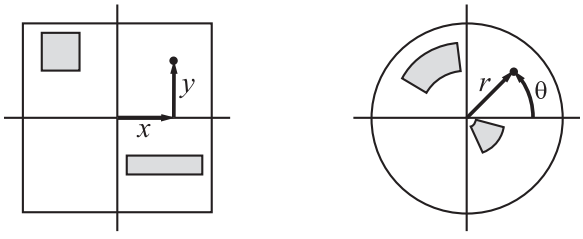


Figure 4.3: Area measures for Cartesian and polar coordinates. The gray regions indicate areas of equal measure. Note that regions with equal measure in polar coordinates do not necessarily cover the same amount of area on the plane.

Change of variables and the Jacobian. It is sometimes convenient to change the coordinate system of an integral to make it easier to evaluate. This is commonly referred to as a *change of variables*, and it involves rewriting the integrand in terms of the new coordinate system and multiplying by the *Jacobian*, which converts area measures from the original coordinate system to area measures in the new coordinate system. For example, the Jacobian for the change of variables from Cartesian to polar coordinates is the distance from the origin, r.

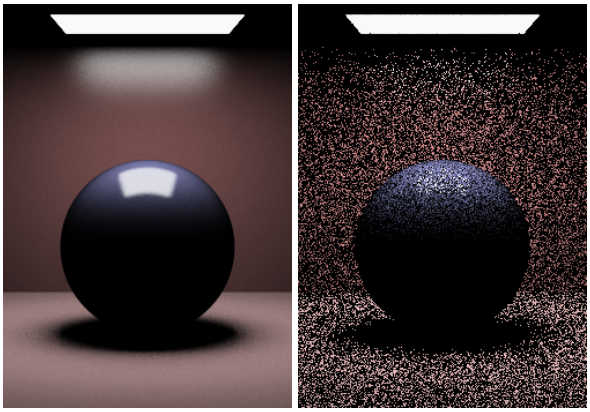
4.3 Applying MCI to the Direct Lighting Equation

Since the direct lighting equation (eq. 4.1) is just an integral over the hemisphere Ω_x, it is straightforward to evaluate it with MCI in a ray tracer. For each primary ray, the direct lighting term is estimated with N Monte Carlo samples as follows:

1. Evaluate $L_e(x \rightarrow \Psi)$, and add this value to the radiance estimate.
2. Choose a number of random directions in Ω_x.
3. For each chosen direction, Θ_j, cast a ray from point x in direction Θ_j, and evaluate $L_e(x \leftarrow -\Theta_j)$.
4. Calculate $L_e(x \leftarrow -\Theta_j) f_r(\Psi \leftrightarrow x \leftrightarrow \Theta_j) |\cos\theta_j|$ for each Θ_j and divide by the probability with respect to solid angle that the direction was chosen, $p_{angle}(\Theta_j)$, since the integral is defined in terms of solid angle.
5. Average the samples from step 4 together and add to the radiance estimate, resulting in the expression

$$L_d(x \rightarrow \Psi) \approx L_e(x \rightarrow \Psi) + \frac{1}{N} \sum_{j=1}^{N} \frac{L_e(x \leftarrow -\Theta_j)\ f_r(\Psi \leftrightarrow x \leftrightarrow \Theta_j)\ |\cos\theta_j|}{p_{angle}\ (\Theta_j)}. \tag{4.5}$$

The obvious choice for a sampling distribution over the Ω_x is to select sampling directions uniformly over the hemisphere (with respect to solid angle). Figure 4.4 shows an image generated in this manner with 16 Monte Carlo samples per pixel, along with the *accepted* image for comparison (an image made with many times more samples to show the desired outcome).



Accepted Image Naive MCI (16 spp)

Figure 4.4: Naive MCI for direct lighting, based on a uniform sampling of the integration domain. The accepted image is reproduced with 16 Monte Carlo samples per pixel.

Notice how poorly the uniform sampling distribution in figure 4.4 reproduces the ideal image, despite the very simple lighting setup. Many of the pixels vary greatly from their ideal values, and some features in the image such as the highlight on the sphere and the back wall are barely discernible. In Monte Carlo rendering, this kind of error is often called "noise", and it occurs because the variance of the Monte Carlo pixel estimates is high. Of course, the quality of a Monte Carlo rendering could be improved by drawing more ray samples to estimate the pixel values, but this would result in a corresponding increase in render time. What is needed is a way to reduce the error while taking the same number of samples. The next section describes a family of *importance sampling* techniques for direct lighting that decrease error by judiciously choosing the sampling distribution to reduce the variance of the Monte Carlo estimate.

4.4 Importance Sampling Techniques for Direct Lighting

Importance sampling attacks the problem of variance in Monte Carlo integration by choosing a sampling distribution that is as proportional as possible to the function being integrated. To see how such a strategy can reduce variance, consider the MCI definition of $\mathbf{X}_g$ from equation 4.2. If the sampling distribution p_{length} is chosen so that it is proportional to g, then

$$p_{length}(x_i) = \frac{g(x_i)}{\int_a^b g(x)dx} \qquad \text{and} \qquad \mathbf{X}_g = \frac{g(x_i)}{\frac{g(x_i)}{\int_a^b g(x)dx}} = \int_a^b g(x)dx,$$

or in other words, $\mathbf{X}_g$ returns a constant value that is equal to the desired integral, and the variance is zero. Unfortunately this ideal situation is difficult to achieve in practice for a variety of reasons. Primarily, just evaluating p_{length} in the ideal case requires knowledge of the desired integral, and even if p_{length} can be evaluated, we must still be able to draw samples from it, which may be difficult or impossible, depending on g.

Although it may not be feasable to sample proportionally to a particular integrand, if the integrand is the product of multiple functions, creating a sampling distribution that is proportional to one or more of the terms of the product may still be worthwhile. For direct lighting, a number of importance sampling techniques exist that create sampling distributions based on the BRDF or the incident light terms in equation 4.1.

4.4.1 BRDF Sampling

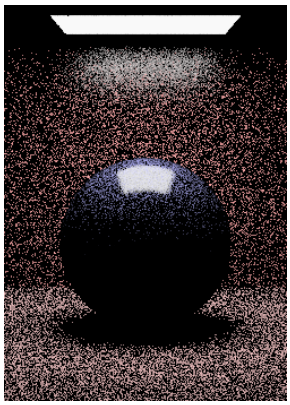


Figure 4.5:
BRDF Sampling.

Perhaps the most important property needed for a successful BRDF model besides the ability to mimic real world surfaces is the ability to perform importance sampling based on the BRDF function, or better yet, the product of the BRDF and the cosine terms in equation 4.1. That way, the term $f_r(\Psi \leftrightarrow x \leftrightarrow \Theta_j)|\cos\theta_j|$ will cancel out of the equation, hopefully reducing variance. As a consequence of this goal, most

BRDF models employed by Monte Carlo renderers have been designed to facilitate importance sampling. For example, a number of analytic BRDF models, including the Phong model [Phong, 1975], the Blinn model [Blinn, 1977], the Lafortune model [Lafortune *et al.*, 1997], and the Ward and Ashikhmin anisotropic models [Ward, 1992; Ashikhmin and Shirley, 2000] can be importance sampled directly. This is accomplished by analytically inverting the cumulative distribution function (CDF) of the BRDF. The inverted CDF can then be used to transform a uniform distribution of points to the distribution of the BRDF. Sampled BRDF models, such as the one described by Lawrence *et al.* [2004], can perform the same operation by searching a tabular representation of the CDF. The rendering in figure 4.5 was produced with 16 BRDF-based importance samples per pixel. Compare this to the right image in figure 4.4. Although most regions of the image are still very noisy, The highlights on the sphere and back wall are much better represented than without importance sampling.

4.4.2 Light Source Sampling

Importance sampling based on the BRDF function can reduce the error of a direct lighting estimate, but there are many situations in which the incident light term is the main source of variance. For example, if the light sources in a scene are small, only a small portion of Ω_x will have any incident illumination, so most directions chosen based on the BRDF will miss the light sources entirely, and thus have a zero contribution. In these situations, it would be better to send rays directly towards points on the light sources, ensuring that $L_e(x \leftarrow -\Theta_j)$ will be non-zero unless the light source is blocked by an intervening occluder.

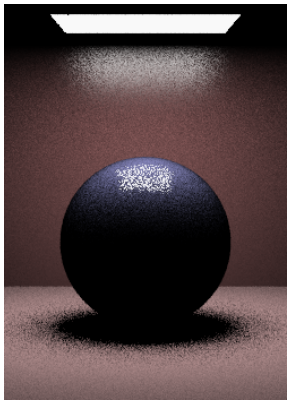
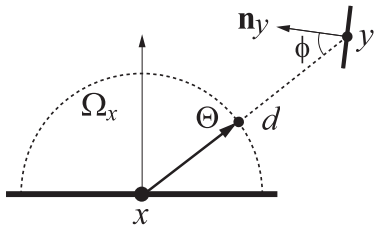


Figure 4.6:
Light Sampling.

Since lights are generally specified as surfaces in a scene rather than directions, it is usually more convenient to select points on the light sources and send rays towards those

points rather than trying to directly select directions in Ω_x that will hit the lights. Of course, this means that the samples are selected with respect to surface area rather than solid angle which is required by equation 4.5. To resolve this problem, we multiply the probability with respect to area by the Jacobian that converts to probability with respect to solid angle. Equation 4.6 gives this conversion:



$$p_{angle}(\Theta) = p_{area}(y)\ \frac{d^2}{cos\phi}. \tag{4.6}$$

In the equation, point y was chosen on a light source with probability $p_{area}(y)$, ϕ is the angle between the normal at point y and the direction from y to point x, and d is the distance between x and y.

In their text on ray tracing, Shirley and Morley [2003] describe techniques to generate samples on triangles, disks and spheres uniformly with respect to surface area. In earlier, but more sophisticated work, Arvo [1995] described a method to distribute samples uniformly on spherical triangles, and Shirley *et al.* [1996] presented algorithms to generate samples non-uniformly on a number of different light source shapes to decrease variance. Fig. 4.6 demonstrates the quality achieved by sampling the area of the light source instead of the BRDF for our simple test scene. Again, 16 ray samples per pixel were used. Note that the image quality is improved almost everywhere, but the highlight on the sphere is not as well represented as with BRDF sampling.

Environment map lighting. One popular type of light source geometry, that will play prominently in the next chapter, is the environment map [Debevec, 1998]. Environment

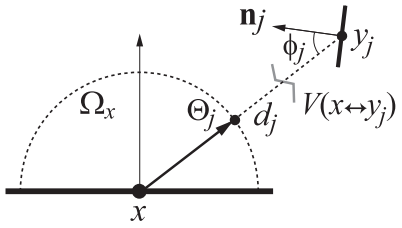
map lighting makes the assumption that the light source is an infinite sphere surrounding the scene, so that the emitted radiance in a particular direction is constant regardless of position. Despite this approximation, renderings made with environment map lighting can have a very realistic appearance, especially if the environment map was captured from real world photography. The most common way to capture an environment map in a real scene is to take photographs of a mirror ball, called a "light probe". The light probe images can then be used directly or warped into some other format such as latitude-longitude for rendering. Figure 4.7 shows an environment map captured in St. Peter's Basilica in Rome.

Figure 4.7: Light probe image captured at St. Peter's Basilica in Rome. The left image shows the original light probe image, and the right image shows a latitude-longitude projection of the same environment.

A number of techniques exist to perform importance sampling based on the emission profile of an environment map. Burke [2004] describes two algorithms to distribute samples according to the environment map brightness, one based on standard CDF inversion and the other based on an alternate inversion method called the alias method. Other work has tackled direct lighting from environment maps by approximating the illumination with a fixed set of point lights (sampling directions). Using a fixed set of lights instead of random samples gets rid of inter-pixel sampling noise because the same directions are used to evaluate L_d for each primary ray. The regularity of the sampling can create distracting shadow boundary artifacts, however. Kollig and Keller [2003] presented a relaxation technique that positions point lights so as to be both well spaced and approximately distributed according to the environment map brightness. Agarwal *et al.* [2003] subdivided an envi-

ronment map into regions, representing each region with a point light. The regions were created using an importance function that takes into account both brightness and angular extent. To eliminate shadow artifacts, the light source locations could be jittered during rendering. The number of ray samples required for a given quality was also reduced by sorting the regions based on potential contribution. Shadow rays would only be sent to the regions with the highest potential contribution. More recent work has concentrated on quickly defining high quality point light sets to represent environment map lighting [Ostomoukhov *et al.*, 2004; Debevec, 2005].

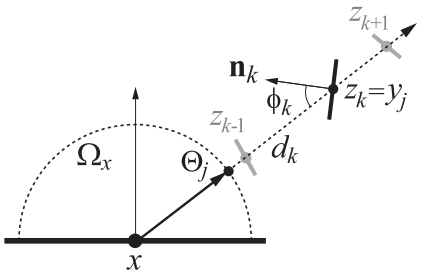
Sampling in the presence of occlusions and multiple light sources. When a renderer sends rays towards points on light sources to evaluate the direct lighting equation, it has several choices of how to treat the ray samples, each of which has particular consequences for the Monte Carlo reconstruction. One school of thought holds that points on surfaces are the entity being sampled. Within this framework, the renderer only needs to know if point x and point y are mutually visible, and the MCI formula (equation 4.5) is rewritten as follows:



$$L_d(x \rightarrow \Psi) \approx L_e(x \rightarrow \Psi) + \frac{1}{N} \sum_{j=1}^{N} \frac{L_e(x \leftarrow y_j)\ f_r(\Psi \leftrightarrow x \leftrightarrow \Theta_j)\ V(x \leftrightarrow y)\ |\cos\theta_j\ \cos\phi_j|}{p_{area}(y_j)\ \ d_j^2}. \qquad (4.7)$$

where $L_e(x \leftarrow y_j)$ is the emitted radiance from point y_j towards point x, and $V(x \leftrightarrow y_j)$ is the visibility between points x and y_j (defined as 1 if x and y_j are mutually visible and 0 otherwise). In essence, the visibility term treats occluded points as though they are outside the integration domain, assigning them a contribution of zero.

A second school of thought holds that choosing points on surfaces is really just a way to pick sample directions. Within this second framework, the renderer does not merely evaluate the visibility between points x and y. Rather, the renderer casts a ray from point x towards point y looking for the closest intersection point, and then calculates $L_e(x \leftarrow -\Theta_j)$ based on that intersection. Equation 4.5 can still be used directly to compute the radiance estimate, but p_{angle} turns into a summation over all light sources that the ray from x in direction Θ_j intersects:



$$p_{angle}(\Theta_j) = \sum_{k=1}^{M} p_{area}(z_k) \frac{d_k^2}{cos\phi_k}. \tag{4.8}$$

In the equation, M is the number of lights intersected by the ray. The benefit of this approach is that samples sent towards occluded points are not wasted. The drawback is that a special ray casting routine is needed to determine all of the light sources that are intersected by a particular ray.

Rendering scenes with large numbers of lights poses a difficult engineering problem for direct lighting. Traditionally, a secondary ray would be sent to each light source for each primary ray, but this is impractical in scenes with hundreds or thousands of lights. Ward [1991] addressed this problem by evaluating the potential contribution of samples on all lights without visibility. He then sorted the light sources based on potential contribution and sent shadow rays to the lights with the highest potential contribution. A fraction of the potential contribution of untested lights was also included to complete the estimate. Since Ward's solution only approximates the contributions of untested lights, it is biased, however. Shirley *et al.* [1996] gave an unbiased variant of Ward's algorithm that probabilis-

tically selects light sources to sample. Another way to reduce calculations in the presence of many lights is to cluster light sources into groups. Fernandez *et al.* [2002] take this approach, calling each light source cluster a "local illumination environment", or LIE. Each LIE caches information about light visibility and irradiance that is used to determine which shadow rays should be sent.

4.4.3 Multiple Importance Sampling

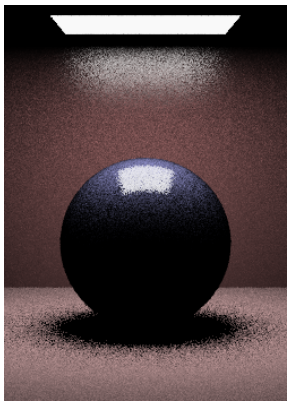


Figure 4.8: MIS.

We have seen that for direct lighting, BRDF sampling is sometimes superior to light source sampling, and vice-versa. A reasonable goal would be to design a sampling strategy based on both BRDF and light source sampling that preserves the good qualities of both methods. One could simply average Monte Carlo estimates based on BRDF and light source sampling, but this approach tends to preserve the bad qualities of both sampling methods as well as the good. A better way is to take some samples from each sampling distribution, but then use the resulting probability distribution to weight the samples. This key idea is the basis of Multiple Importance Sampling, or MIS [Veach and Guibas, 1995]. As an example of how MIS works, suppose that you want to estimate the direct lighting integral by choosing half of the samples based on the BRDF and half based on sampling the light sources. For a given sample direction, Θ_j, let the probability (with respect to solid angle) that it was generated by BRDF sampling be $p_1(\Theta_j)$, and let the probability (again with respect to solid angle) that is was generated by light source sampling be $p_2(\Theta_j)$. The actual probability p_{angle} that Θ_j was generated by either technique is then $\frac{1}{2}p_1(\Theta_j) + \frac{1}{2}p_2(\Theta_j)$. This value can be used directly as the probability in equation 4.5. Figure 4.8 was generated with this method. Note how the MIS rendering preserves the good qualities of both BRDF and light source sampling, achieving a high quality reconstruction on the highlight of the sphere as well as the back wall and floor.

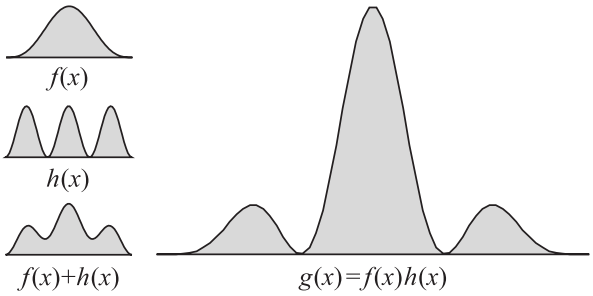


Figure 4.9: The benefits of multiple importance sampling. Function g is the product of two terms, f and h. If importance sampling methods exist for both f and h, samples from both methods can be combined into a distribution proportional to $f+h$ which is more proportional to g than f or h alone.

Figure 4.9 demonstrates the benefits of MIS graphically. In the figure, function g is the product of two terms, f and h. There may not be a straightforward importance sampling method for g, but if importance sampling methods exist for both f and h, MIS can be employed to create a sampling distribution proportional to $f+h$. This sum distribution fits g much better than f or h alone, leading to reduced variance in the Monte Carlo estimate.

The down side of using MIS is that it is more complicated, and can be slower than using a single importance function. To make MIS work, a sampler must be able to generate samples according to each sampling distribution, and determine with what probability an arbitrary sample would be chosen for each sampling method. This expense is justified, however, because of the decrease in variance that MIS offers.

The variant of multiple importance sampling just described is called the "balance heuristic" because the value returned for direction Θ_j is the same no matter what distribution the sample was drawn from. Surprisingly, however, this is not the only possible unbiased weighting scheme. Veach and Guibas described several other unbiased weighting heuristics that sometimes decrease variance more than the balance heuristic, the most popular being called the "power heuristic", because it weights samples according to a power of the probability that they were drawn.

4.4.4 Resampled Importance Sampling

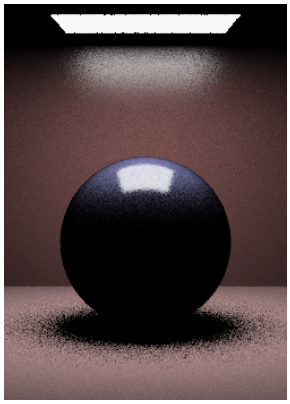


Figure 4.9: RIS.

MIS can sample the BRDF and incident light terms of equation 4.1 simultaneously, but the resulting distribution is essentially proportional to the sum of the terms rather than the product. An ideal importance sampling method for direct lighting should be able to generate samples proportional to the product of the BRDF and incident light rather than just the sum. One way to achieve this goal is to draw a large number of tentative samples from one distribuition, and then discarding most of them, leaving a distribution more like the desired one. Of course, this is only useful if the tentative samples can be evaluated more quickly than complete samples (otherwise information is just being thrown away). In direct lighting, ray casting is the most expensive operation, so the typical approach is to evaluate a large number of samples without visibility, discard most of these samples, and then evaluate the visibility term for the remainder of the samples. Burke *et al.* [2005] described a technique called Bidirectional Importance Sampling (BIS) that calculates the direct lighting equation in this manner. First, a large number of direct lighting samples are drawn from an existing importance sampling distribution, and evaluated without visibility. In a second step, some of the samples are discarded, either by *rejection sampling* or *resampling*. The algorithm then evaluates the visibility term for the remaining samples and weights them to form an unbiased estimate of the direct lighting integral. The resampling version of BIS is generally to be preferred over the rejection sampling version because the number of tentative samples per accepted sample can be fixed a priori. Talbot *et al.* [2005] showed that the resampling variant of bidirectional importance sampling fits into the more general category of Resampled Importance Sampling, or RIS. They also demonstrated several useful extensions to the basic algorithm, for instance showing how to select the number of tentative samples that should be drawn per accepted sample to achieve near optimal variance reduction for a given render time.

As an example of how RIS for direct lighting works, consider the case in which four tentative samples are taken, and we would like to retain only one of them. RIS begins by selecting four tentative sample directions $\Theta_1 \dots \Theta_4$ through some importance sampling method. Let the probabilities that the four samples were chosen (with respect to solid angle) be $p_1 \dots p_4$. Next, the integrand $L_e(x \leftarrow -\Theta) f_r(\Psi \leftrightarrow x \leftrightarrow \Theta) |\cos\theta|$ is evaluted without visibility for the four directions, producing values $f_1 \dots f_4$. By "evaluated without visibility" we mean that the first light source intersected by each sample direction is determined, to calculate L_e, but no ray is cast to see if the light source is occluded by other scene objects. At this point, we could cast shadow rays to determine the visibility of the four samples, $V_1 \dots V_4$, and construct the standard Monte Carlo estimate:

$$\frac{1}{4}\left(\frac{f_1 V_1}{p_1} + \frac{f_2 V_2}{p_2} + \frac{f_3 V_3}{p_3} + \frac{f_4 V_4}{p_4}\right).$$

However, we would like to avoid calculating all but one of the visibility terms, since ray casting is the most expensive operation. To avoid casting three of the shadow rays, the resampling step of RIS selects one of the samples to keep. For the sake of argument, suppose that RIS chooses Θ_1, with probability q_1. Because of the resampling, the actual probability that Θ_1 was selected by the RIS procedure changes by the factor $4q_1$. At this point, the algorithm casts a shadow ray in direction Θ_1 to calculate V_1, and the resulting RIS estimate becomes

$$\frac{f_1 V_1}{4 p_1 q_1}. \tag{4.9}$$

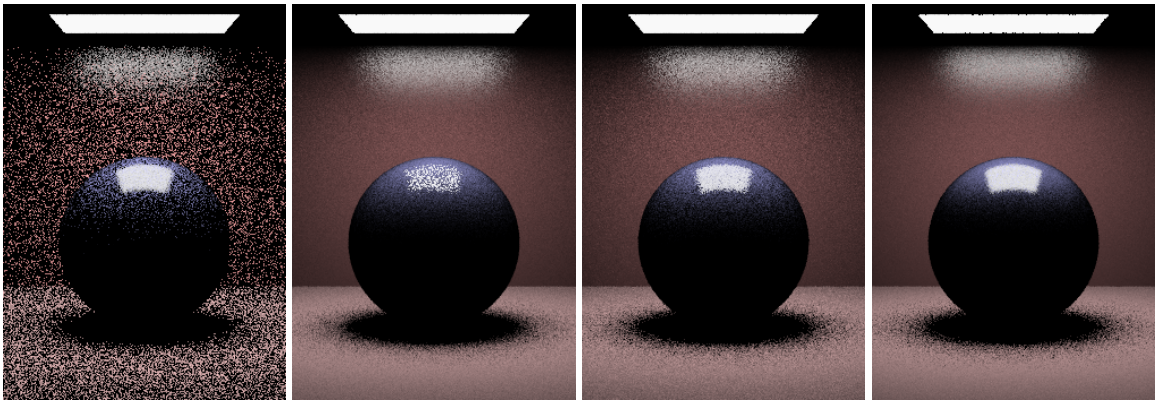
Since the goal of importance sampling is to reduce variance, we would like to choose $q_1 \dots q_4$ in such a way that the resulting estimate will have the same value no matter which Θ is chosen (ignoring visibility). This leads to the following set of equations:

$$q_1 + q_2 + q_3 + q_4 = 1 \quad \text{and} \quad \frac{f_1}{p_1 q_1} = \frac{f_2}{p_2 q_2} = \frac{f_3}{p_3 q_3} = \frac{f_4}{p_4 q_4}.$$

Solving this set of equations yields the optimal probability for the q values:

$$q_i = \frac{p_i/f_i}{\sum_{j=1}^{4} p_j/f_j}. \tag{4.10}$$

Note that even though assigning the q values as just described makes all of the samples have equal value, as a whole, the RIS estimator still has non-zero variance. This is because the sum in the denominator of equation 4.10 is not constant between different runs of the algorithm. Nevertheless, RIS can reduce variance substantially in many circumstances. Figure 4.9 was rendered with resampled importance sampling, using 64 tentative samples, and 8 accepted samples per pixel. Figure 4.10 gives a side-by-side comparison of the four importance sampling techniques that have been discussed. Not surprisingly, RIS performs better than the other importance sampling techniques in our test scene, even with half as many final samples.



BRDF (16 spp) Light (16 spp) MIS (16 spp) RIS (8 spp)

Figure 4.10: Importance sampling methods for direct lighting. BRDF sampling tends to capture glossy surfaces well, whereas light source sampling captures matte surfaces better. Multiple importance sampling (MIS) can combine the good qualities of BRDF and light source sampling, and resampled importance sampling (RIS) approximates the product distribution of BRDF and lighting by taking a large number of tentative samples without visibility and then probabilistically discarding most of the tentative samples.

4.5 Sampling Patterns

For most applications a single Monte Carlo sample will not suffice; multiple samples are needed so that the error in the integral estimate falls into acceptable limits. Generally, Monte Carlo Integration begins with uniformly-distributed point samples in some primary sample space, such as $[0,1)^n$. These samples are then warped to conform to the domain of the integrand and the importance function used by the integrator. The manner in which the initial samples are chosen can have a big impact on the variance of the final estimator. This section discusses methods to generate good sample patterns to reduce the variance of multi-sample Monte Carlo estimates.

4.5.1 The Trouble with Random Sampling

The trouble with drawing samples randomly for Monte Carlo integration is that samples tend not to be distributed as evenly as one would think. For example, consider figure 4.11, showing 64 samples drawn from a uniform distribution in $[0,1)^2$:

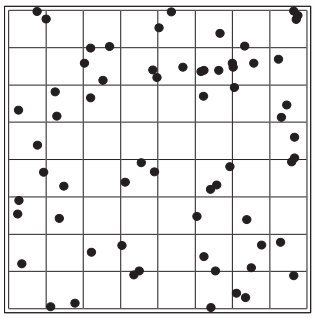


Figure 4.11: Random points.

As can be seen, samples tend to clump together in bunches rather than being evenly distributed. For comparison purposes, we have superimposed an 8×8 grid on the $[0,1)^2$ square. Ideally, each small square should contain one of the 64 samples, but less than half (only 23) have this ideal number. The unevenness caused by random sample placement leads to a decreased convergence rate for Monte Carlo integration with multiple samples.

A classic result from statistics states that the variance of a Monte Carlo estimate with N randomly drawn samples will decrease at a rate proportional to $1/N$. A convergence rate of $1/N$ may not seem that bad until one realizes that the variance represents the square of the error rather than the error itself. The actual expected error decreases according to the square root of the number of samples, $1/\sqrt{N}$, meaning that to reduce the error by half, the number of samples must be increased four fold.

4.5.2 Stratified sampling

One way to counteract the clumping that occurs with random sampling is to divide the integral into a number of pieces and then sample each sub-domain separately. Each sub-domain of the original integral is called a "stratum", and the resulting sample pattern is said to be "stratified", or "jittered". Figure 4.12 shows a stratified sampling pattern, once again with 64 samples:

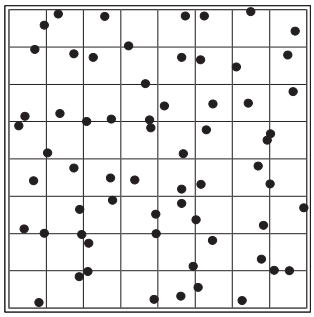


Figure 4.12: Stratified pattern.

Mitchell [1996] conducted a thorough analysis of the convergence properties of stratified MCI in common graphics situations. Mitchell's results showed that stratified sampling is never worse than random sampling, but that the benefits of stratification quickly disappear as the dimensionality of the integral increases. Another important result was that the variance reduction offered by stratification depends on the integrand. In his studies of 2D integrals, Mitchell identified three common integral types with three different

convergence rates. Integrals with very smooth integrands received the most benefit from stratification, with variance reducing at a rate proportional to $1/N^2$. Moderately complex integrands with a few discontinuities cutting through them benefited less, converging according to $1/N^{1.5}$. Finally, very complex integrands did not benefit at all from stratification and converged at the familiar $1/N$ rate. For this case, the variability of the integrand within each stratum does not appreciably reduce over the variability of the integrand as a whole, so convergence does not improve.

4.5.3 Poisson disk sampling.

While stratification improves the evenness of samples compared to random sampling, some clumping can still occur in stratified patterns because individual samples do not account for how close samples in neighboring stata might be. An alternative to stratification called *Poisson disk* sampling fixes the clumping problem by requiring all samples to be separated by some minimum distance. Consequently, samples can never clump together. Interestingly, this is how photoreceptors are spaced within the human eye. Rods and cones pack evenly next to each other, but not in a regular pattern. Figure 4.13 shows a Poisson disk sampling pattern. Notice how much more evenly the points are distributed than with stratified samples.

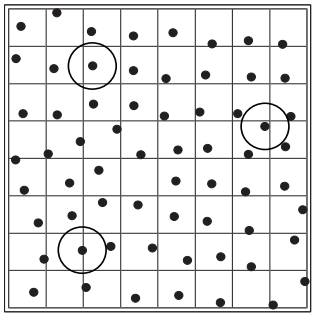


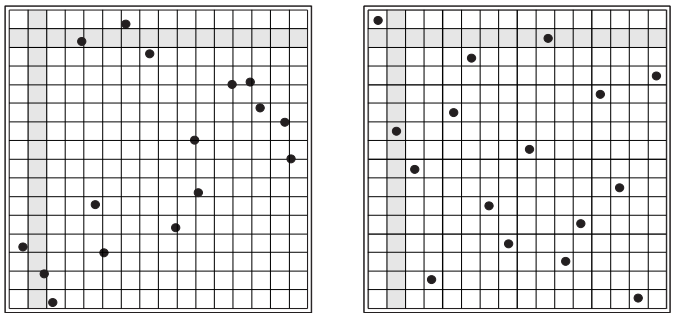
Figure 4.13: Poisson disk pattern.

The simplest way to generate a Poisson disk pattern is by a process of "dart throwing". Samples are generated at random, but if a new sample falls too close to an existing one it is discarded. Dart throwing can be time consuming, however, so a number of researchers have developed methods to more quickly generate Poisson disk-like patterns. Mitchell [1987] created point sets with Poisson disk properties using an error diffusion process similar to Floyd-Steinberg dithering [Floyd and Steinberg, 1976]. Later, Mitchell showed how to generate well spaced sample sets based on a modified dart throwing technique. Instead of simply discarding samples that are too close to an existing point, Mitchell's "best candidate" algorithm [Mitchell, 1991] generates n candidates for each new sample, keeping the one that is furthest away from any existing sample. Poisson disk-like patterns can also be produced by applying Lloyd relaxation [Lloyd, 1983] to an initial random or stratified point set. Lloyd relaxation works by creating a voronoi diagram of the point set. The points are then moved to the centers of their respective voronoi cells, and the process repeats. After a few iterations, the points are quite evenly spaced, and closely resemble a Poisson disk pattern. Because of their importance in Monte Carlo methods, Poisson disk point sets remain an active area of research. Recent work in this area includes faster generation methods [Ostomoukhov *et al.*, 2004; Dunbar and Humphreys, 2006; Jones, 2006], and methods that tile Poisson disk point sets aperiodically [Cohen *et al.*, 2003; Kopf *et al.*, 2006].

4.5.4 Latin Hypercube and Multi-Jittered Sampling

Stratified or Poisson disk point sets may be impractical for high dimensional integrals because of the number of samples involved. To illustrate, a stratification grid with 8 divisions per dimension results in 8 samples in 1D and 64 in 2D, but extending the pattern to 3D requires 512 samples, and in 4D a whopping 4096 samples are needed. Rather than resorting back to random sampling, however, it may be reasonable to stratify each dimension separately and then randomly associate the dimensions. This strategy is called *latin hypercube* or *N-rooks* sampling [Shirley, 1991]. Following this work, Chiu *et al.* [1994] showed

how to generate point sets that are both N-rooks and stratified patterns at the same time. They called the method *multi-jittered sampling*. Figure 4.14 shows latin hypercube and multi-jittered point sets with 16 samples each.

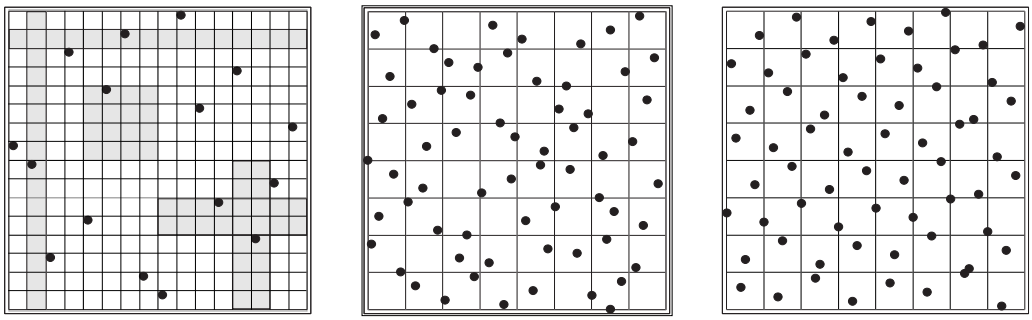


Latin Hypercube Multi Jittered

Figure 4.14: Latin hypercube and multi-jittered point sets.

4.5.5 Low Discrepancy Sampling Methods

Parallel to the theory of Monte Carlo integration, which relies on random samples, is the field of Quasi-Monte Carlo integration, which replaces random samples with deterministic, *low discrepancy* (LD) sampling patterns. The term *discrepancy* refers to the idea that the number of samples within a particular region should be proportional to the size of the region. For example, an LD sampling pattern called the (0,2)-sequence is shown in figure 4.15. With 16 samples, the (0,2)-sequence places a single sample in five different sub-region types of the unit square simultaneously, shown in gray. A number of other low discrepancy sampling patterns exist, two of which are included in the figure: the Halton sequence, and the Hammersley point set. Pharr and Humphreys [2004] give a good introduction to low discrepancy sampling in computer graphics, along with detailed explanations of how several of the low discrepancy sample sets are created. Kollig and Keller [2002] provide implentations and variance analysis for various LD sampling patterns.

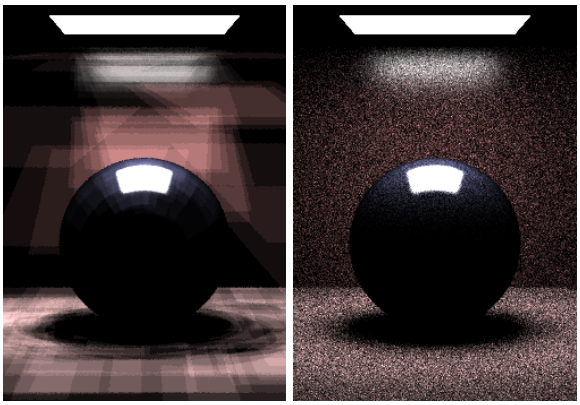


(0, 2) Halton Hammersley

Figure 4.15: Low discrepancy sampling patterns, (0,2)-sequence, Halton sequence and Hammersley point set.

4.5.6 Correlation Between Pixels

Reusing the same set of samples to evaluate direct lighting over the whole image plane can lead to distracting artifacts. This is true whether the samples were generated by a random or deterministic process. The left image in figure 4.16 shows this effect. Correlation between pixels leads to correlated errors in the pixel integral estimates, which can be distracting to the human eye. The standard fix for this problem is to use a different set of samples for each pixel. With random sampling one can simply create a new sample set for each primary ray, relying on the randomness inherent in the sampling method to break up correlation artifacts. Low discrepancy patterns, on the other hand, do not have any inherent randomness, so alternative methods are needed to add variety to them. A simple idea that works fairly well in practice is to shift the sample pattern by some random amount (modulo 1) each time it is used. This is called a *Cranley Patterson rotation* [Cranley and Patterson, 1976]. A more sophisticated method called *Owen scrambling* randomizes a sample set by swapping different subsets of the $[0,1)$ interval in each dimension [Owen, 1995]. The right image in figure 4.16 shows the result of randomizing the sampling pattern used by the left image. Randomization does not reduce statistical error (variance), but perceptual error is reduced because the error is scattered into high frequency noise that is less distracting to the eye.



Fixed Randomized

Figure 4.16: Fixed vs. randomized sampling patterns.

4.6 Measuring the Convergence of a Rendering Algorithm

This section describes the error metrics that will be used to measure the convergence properties of the algorithms in this dissertation. Since the dissertation concentrates on unbiased rendering, I have chosen to use an objective rather than a perceptual metric as the primary validation tool.

4.6.1 Color Difference

The first step in defining the difference between two images is to define the difference between color pixel values. There are a number of methods to calculate the "difference" or "distance" between two colors, and many of these difference measures would be suitable for the purposes of this work. After experimenting with several measures, we chose to use a weighted L1 norm where the weights are proportional to the luminance contributions of the different color components. In the RGB color space, the luminance of a color $C = (r, g, b)$ is defined $|C|_c = 0.299r + 0.587g + 0.114b$. Similarly, we define the difference between two colors, $C_1 = (r_1, g_1, b_1)$ and $C_2 = (r_2, g_2, b_2)$, as follows:

$$|C_2 - C_1|_c \;=\; 0.299\,|r_2 - r_1| \;+\; 0.587\,|g_2 - g_1| \;+\; 0.114\,|b_2 - b_1| \tag{4.11}$$

where $|\cdot|_c$ denotes the difference between two color values. We use the L1 norm instead of the Euclidian norm to be consistent with the definition of luminance.

4.6.2 Image Difference

A standard error metric for the difference between two images is the mean squared error (MSE), or variance (σ^2). For two images **A** and **B** with the same pixel dimensions, the variance is defined as:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} |\mathbf{A}_i - \mathbf{B}_i|_c^2 \tag{4.12}$$

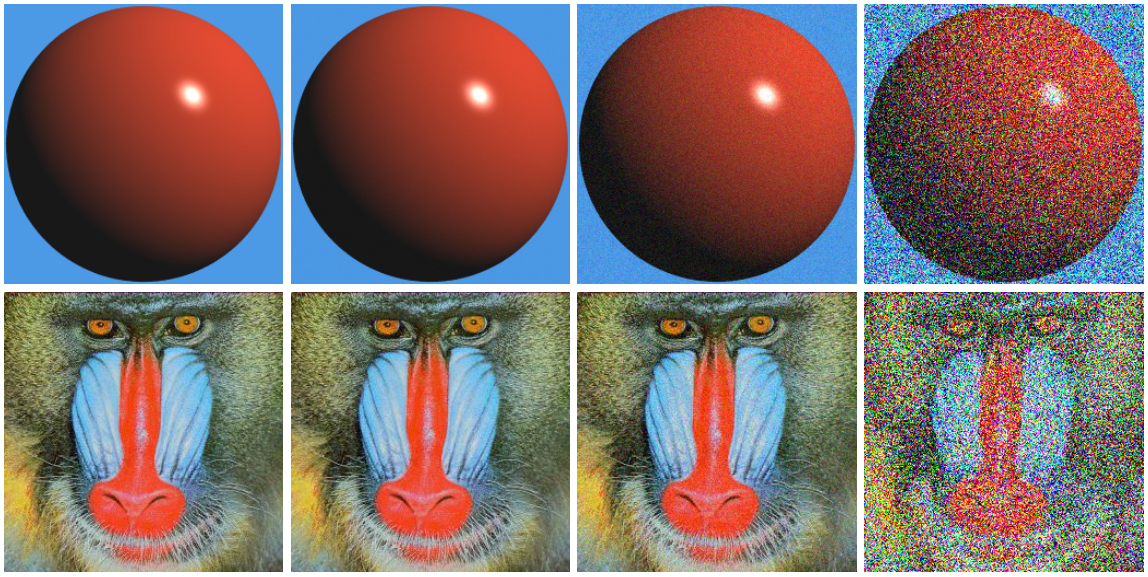
where σ^2 is the variance, N is the number of pixels in the two images, and $|\mathbf{A}_i - \mathbf{B}_i|_c$ is the color difference between pixel i in image **A** and pixel i in image **B**. Note how similar this definition is to the variance described in equation 4.3. It is particularly appropriate to call equation 4.12 a variance measure if one of the images is the "expected image", and the other is an approximation made by a Monte Carlo rendering algorithm.

Although σ^2 is a valid statistical metric by itself, we would prefer a metric that describes the relative difference between two images rather than the absolute difference. This would allow rough comparisons between different rendering algorithms without the need to render all the same scenes with each method. Consequently, we will report image difference in terms of the *coefficient of variation*, which is the standard deviation divided by the mean image lumance: (σ/μ). More formally, we define σ/μ for images **A** and **B** as follows:

$$\sigma/\mu \;=\; \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N} |\mathbf{A}_i - \mathbf{B}_i|_c^2}}{\frac{1}{N}\sum_{j=1}^{N} |\mathbf{A}_j|_c}. \tag{4.13}$$

σ/μ is the reciprocal to another statistical measure of image difference called the *signal to noise ratio*. In this dissertation, we use the coefficient of variation because smaller image differences map to smaller values, which is an advantage in convergence plots.

Visual interpretation of σ/μ. Figure 4.17 gives examples of images that have been corrupted with different amounts of gaussian-distributed noise. With a σ/μ of 0.01, the corrupted images are practically indistinguishable from the originals, even under close inspection. On the other hand, with $\sigma/\mu = 1.0$, there is essentially as much noise as signal, and the image is probably unusable for most applications.



original $\sigma/\mu = 0.01$ $\sigma/\mu = 0.1$ $\sigma/\mu = 1.0$

Figure 4.17: Images corrupted with different amounts of gaussian-distributed noise. At $\sigma/\mu = 0.01$ the images are virtually noise free visually, even under close scrutiny. At $\sigma/\mu = 0.1$, the images contain roughly the same amount of noise that can be seen in a high quality movie or television broadcast. Finally, at $\sigma/\mu = 1.0$, image features are barely visible through the noise.

4.6.3 Measuring Convergence

Generally, in a Monte Carlo rendering context convergence is measured by comparing a rendered image to an "accepted" image rendered with many more samples per pixel. In some cases it may be impractical to render a complete accepted image, however. A simple way to get around this difficulty is to only render a random subset of the image pixels as an accepted image. Another way to get around the need for an accepted image is to compare

two images created with the same number of samples per pixel. This will work if the noise produced by the rendering algorithm has zero mean and will not be correlated between different program runs. In this case, we can invoke the *variance sum law*, which states that the variance of a sum or a difference of two independent random variables, **X** and **Y**, is equal to the sum of the variances of the two random variables:

$$\sigma^2_{\mathbf{X}\pm\mathbf{Y}} = \sigma^2_{\mathbf{X}} + \sigma^2_{\mathbf{Y}}. \tag{4.14}$$

In the case of the two images, the two variances are equal (the two images were rendered with the same number of samples per pixel), so $\sigma_{\mathbf{X}}$ can be substituted for $\sigma_{\mathbf{Y}}$. Making this substitution, and rearranging a bit yields

$$\frac{\sigma_{\mathbf{X}}}{\mu_{\mathbf{X}}} = \frac{\sigma_{\mathbf{X}\pm\mathbf{Y}}}{\mu_{\mathbf{X}}\sqrt{2}}. \tag{4.15}$$

Finally, since the expected value of the difference of the two images is zero everywhere, we can estimate the noise level of the rendered images by computing σ/μ between the images, and then dividing this value by $\sqrt{2}$.

Chapter 5

Two Stage Importance Sampling for Direct Lighting

A version of this chapter was published as:

David Cline, Parris K. Egbert, Justin F. Talbot and David L. Cardon. Two Stage Importance Sampling for Direct Lighting. *Eurographics Symposium on Rendering*, 2006.

Abstract. We describe an importance sampling method to generate samples based on the product of a BRDF and an environment map or large light source. The method works by creating a hierarchical partition of the light source based on the BRDF function for each primary (eye) ray in a ray tracer. This partition, along with a summed area table of the light source, form an approximation to the product function that is suitable for importance sampling. The partition is used to guide a sample warping algorithm to transform a uniform distribution of points so that they approximate the product distribution. The technique is unbiased, requires little precomputation, and we demonstrate that it works well for a variety of BRDF types. Further, we present an adaptive method which allocates varying numbers of samples to different image pixels to reduce shadow artifacts.

5.1 Introduction

Increasingly, image based lighting is being used for rendering. Image based lighting offers a number of advantages over simple lighting techniques such as directional or point lights. Spatially varying image based lighting provides a more realistic lighting environment, so images rendered with it often have a more realistic appearance. Using light probes as lighting allows virtual objects to be rendered as if they were actually located in the imaged location. This technique is employed often in movies, where rendered objects need to be seamlessly integrated into live action shots.

5.1.1 Importance Sampling for Direct Lighting

The problem that this paper addresses is how to solve the direct lighting equation. In a ray tracing context, to solve for direct lighting each primary ray must evaluate the integral:

$$L(x \rightarrow \Psi) = \int_{\Omega_x} L_e(x \leftarrow -\Theta) f_r(\Psi \leftrightarrow x \leftrightarrow \Theta) |\cos\theta| d\Theta. \tag{5.1}$$

In this equation, x is the point hit by the primary ray, L_e represents the incoming radiance at point x, f_r is the BRDF function, and θ is the angle between the surface normal at x and the outgoing direction, Θ.

Monte Carlo Integration. A common approach to evaluate the direct lighting equation is to use Monte Carlo integration, which replaces the continuous integral with the average of N Monte Carlo samples:

$$L(x \rightarrow \Psi) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{L_e(x \leftarrow -\Theta_j)\ f_r(\Psi \leftrightarrow x \leftrightarrow \Theta_j) |\cos\theta_j|}{p\ (\Theta_j)}. \tag{5.2}$$

Importance sampling attempts to minimize the variance of the above expression by choosing a sampling distribution to make the terms of the sum as constant as possible.

Light source and BRDF sampling. Many importance sampling techniques for direct lighting concentrate on either sampling the light source or the BRDF. For example, Burke [2004] described two methods to distribute samples according to the brightness of an environment map, based on cdf inversion and the alias method. Other research [Kollig and Keller, 2003; Agarwal *et al.*, 2003; Ostomoukhov *et al.*, 2004] has tackled direct lighting from environment maps by approximating the illumination with a set of point lights (sampling directions). More recently, Debevec [2005] presented a simple technique to generate point lights to approximate environment lighting using a summed area table of the environment map. Our algorithm also uses a summed area table, but we are able to approximate the product distribution, not just the incident light term.

Methods that generate samples based solely on illumination do not work well for highly specular surfaces. In many situations it is more efficient to sample according to the BRDF function rather than the incident illumination. Some analytical BRDFs can be directly importance sampled, including the Blinn model [Blinn, 1977], the Ward model [Ward, 1992], the Lafortune model [Lafortune *et al.*, 1997], and the Ashikhmin model [Ashikhmin and Shirley, 2000]. Besides analytical BRDFs, importance sampling can be employed with sampled BRDFs. Lawrence, Rusinkiewicz and Ramamoorthi [2004] described a factored, tabular representation for BRDFs that is both compact and amenable to importance sampling.

Sampling a product distribution. Sampling according to one of the terms of the lighting equation can reduce variance, but it is more advantageous to generate samples based on all of the terms, rather than just one. Multiple importance sampling (MIS) [Veach and Guibas, 1995] can sample the BRDF and lighting simultaneously, but the resulting distribution is more akin to the average of the terms rather than the product. Ideally, an importance sampling algorithm should be able to generate samples according to the product distribution.

Burke, Ghosh and Heidrich [2005] described a technique called bidirectional importance sampling (BIS) that can sample the product of an environment map and the BRDF, based on rejection sampling. The rejection sampling can be costly, however, and requires an unknown number of tries to produce samples from the product distribution. A second form of BIS that can produce samples in a deterministic amount of time replaces rejection sampling with resampling, but the resulting samples are only approximately distributed according to the product. Talbot, Cline and Egbert [2005] generalized this second form of BIS, placing it into the more general category of resampled importance sampling (RIS). Resampling methods can also be costly, however, since they rely on taking a large number of tentative samples, most of which will be discarded.

Recently, Clarberg et al. [2005] presented an algorithm called Wavelet Importance Sampling (WaIS) that samples products of wavelet functions. Their algorithm uses a property of wavelets that allows a wavelet product to be evaluated in a top-down fashion, and they introduced a warping technique that transforms a uniform distribution of points to the product distribution using the wavelet product as a guide. WaIS produces very impressive results, but has a number of shortcomings that make it impractical in some situations. WaIS requires all BRDFs in the scene to be resampled as wavelets, which may be impractical for scenes with large numbers of BRDFs. Also, WaIS requires the wavelet functions to share a common coordinate system. In the case of environment maps, they accomplish this by storing a separate wavelet decomposition for each possible orientation of the environment map. Our two stage importance sampling algorithm can be thought of as a variant of WaIS that does not require a wavelet product to drive the sample warping. Instead, we use a hierarchical partitioning of the environment map similar to the probability trees described by McCool and Harwood [1997].

5.1.2 Two Stage Importance Sampling

This section gives an overview of our new algorithm to perform importance sampling for direct lighting. The algorithm proceeds in two stages, so we call it *two stage importance sampling*, or just *two stage sampling*. The first stage creates an approximation to the product of the BRDF and the environment map suitable for importance sampling. The approximate product consists of (1) a summed area table of the environment map, times the cosine of the angle of inclination to compensate for foreshortening at the poles, along with (2) a hierarchical partition of the environment map annotated with BRDF values at the region corners. The second stage of the algorithm uses the summed area table and environment map partition to warp a set of uniformly distributed points so that they approximate the product distribution (BRDF $\times$ incident light). Both the hierarchical partitioning of the environment map and the sample warping are performed for each primary ray in a ray tracer. Figure 5.1 gives high level pseudocode for two stage importance sampling.

Two Stage Importance Sampling

Create a summed area table of the light source.
For each primary ray
1. Create an approximation to the product of the BRDF and the light source.
2. Warp a set of uniformly-distributed samples to approximate the product distribution.

Figure 5.1: Two stage importance sampling algorithm.

Benefits of the new approach. Two stage importance sampling offers a number of benefits over existing techniques to sample according to the product distribution, such as RIS and WaIS. It requires very little precomputation, just a summed area table of the environment map. Furthermore, two stage sampling can work with both sampled and analytical BRDFs, and since BRDFs do not need any preprocessing, it can handle scenes with many BRDFs. Two stage sampling does not even require the ability to importance sample the

BRDF function. All that is needed are the BRDF peaks. Since two stage sampling does not use rejection sampling or resampling, it preserves the stratification of an input sampling pattern better than algorithms that discard some of their samples, such as RIS. Finally, two stage sampling does not require the terms in the product to have the same coordinate system, so our algorithm could easily be adapted for other large light sources besides spherical environment maps.

5.2 Partitioning the Light Source

This section describes how to create a hierarchical partitioning of an environment map based on the BRDF function.

5.2.1 Environment Map Encoding

As a preprocessing step, we create a copy of the environment map as a summed area table [Crow, 1984]. This allows rectangular regions within the map to be summed with just four table look-ups. Creating the summed area table typically requires less than 1 second. In our implementation, the summed area table stores the luminance of all of the static terms in the product (terms that are constant within the coordinate system of the environment map). For a spherical environment map in latitude, longitude format, we store the value in the environment map times the cosine of the angle of inclination, which compensates for the foreshortening that occurs near the map poles. This is the same encoding suggested by Debevec [2005]. To maintain precision, the summed area table is stored as 64 bit floating point values.

5.2.2 Partitioning the Environment Map

Two stage importance sampling relies on a piece-wise linear approximation of the BRDF and all of the terms of the product that are not included in the summed area table. This approximation is created independently for each primary ray. In the case of a spherical

environment map, the terms not included in the summed area table are the BRDF and cosine from equation 5.1: $f_r(\Psi \leftrightarrow x \leftrightarrow \Theta)\ |\cos\theta|$. For brevity, we will refer to this value simply as f.

Our algorithm approximates the product function by partitioning the environment map into disjoint regions using an axis-aligned BSP tree. Nodes in the BSP hierarchy represent rectangular regions in the environment map, and each node stores the value of f at its four corners. Consequently, the approximation to the product becomes the product of the values stored in the summed area table and the bilinear interpolation of f from the leaf nodes of the BSP tree. Figure 5.2 shows one of these leaf nodes. Note that although the values of f are interpolated across the region, the values stored in the summed area table are represented exactly.

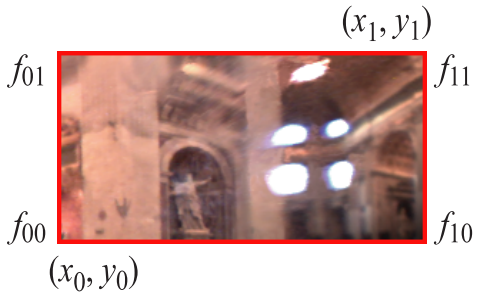


Figure 5.2: Example region R, labeled with min and max extents along with f values at the corners.

Creating the root node. The first step in partitioning the light source is to create a root node that will encompass all non-zero parts of the product function. One obvious choice would be to include the entire environment map as the root. However, we can easily find a smaller rectangle that encompasses all parts of the environment map that are in the same hemisphere as the surface normal. Assuming that the pixel coordinates of the normal in the environment map are (x_n, y_n), and w and h are the width and height of the environment map in pixels, the bounds of this rectangle are $(0,\ max(0,\ y_n - h/2))$ and $(w,\ min(h,\ y_n + h/2))$.

After creating the root node, we make an initial partition by subdividing at two locations. First, we subdivide the root node at the normal location, (x_n, y_n), and half the width of the environment map away from the normal, $((x_n + w/2)\%w,\ y_n)$. Subdividing at $((x_n + w/2)\%w,\ y_n)$ as well as the normal has the benefit that the $cos\theta$ term of the product is monotonic in all of the resulting regions. Since the hierarchy is a binary tree, we split at a particular location by first horizontally splitting the leaf node containing the position, and then vertically splitting both of the new nodes created by the first split. Figure 5.3 shows the initial partition.

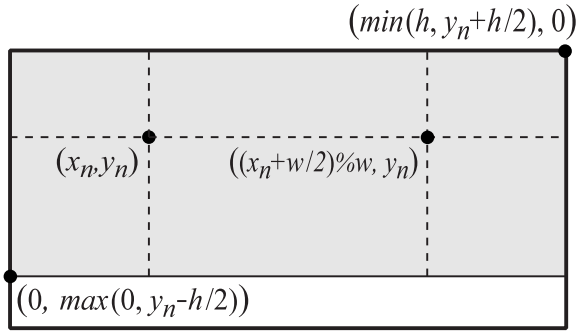


Figure 5.3: Initial partition of the environment map. Two initial splits divide the root node into six regions, and include the peak of the $\cos\theta$ term in f as one of the region corners.

Subdividing at the BRDF peaks. After the initial partition of the environment map has been made, we subdivide the leaf nodes at the peaks of the BRDF. The peak locations that we use for different BRDF types are as follows:

- **The Lambertian model** does not need any other peaks besides the normal direction.
- **The Oren-Nayar model** [1994] was designed to simulate retro-reflection, so we add the incident direction as a peak for this model.
- **The Phong model** [1975] and **Blinn microfacet model** [1977] have a glossy lobe centered around the reflection vector, so we add the reflection vector as a peak for these models.

- **The Lafortune model** [1997] consists of a number of generalized cosine lobes, each of which has a well-defined peak location. The peaks of the lobes are found by transforming the incident direction to the local shading coordinate system. The transformed incident direction can then be directly scaled based on the lobe parameters to produce the lobe peaks.

- **The Ashikhmin anisotropic model** [2000] produces a long, thin specular lobe that traces out a cone of directions, rather than being centered around a single point. For this BRDF, we first add the reflection vector as a peak, since this is the brightest point on the BRDF. In addition, we add nine peaks along the cone of directions traced out by the specular lobe, if the anisotropy is large enough. In our implementation, we add the extra nine peaks of if n_u/n_v or n_v/n_u is greater than 3, where n_u and n_v are the anisotropy parameters of the BRDF. Figure 5.2.2 shows the local surface geometry for the Ashikhmin model where $n_u < n_v$. In the case where $n_u < n_v$, the specular lobe of the BRDF stretches out into a cone of directions aligned with the **u** axis of the local coordinate system, which includes the reflection vector as the brightest point. We define nine "peak directions" to represent this conical lobe as follows: As in figure 5.2.2, let $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ be the local coordinate system of the surface, and let (r_u, r_v, r_n) be the reflection vector defined in the local coordinate system. If $n_u < n_v$,

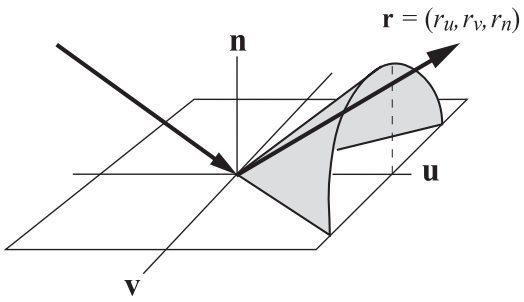


Figure 5.4: The specular lobe of the Ashikhmin model stretches over a cone of directions rather than being centered around a single peak direction.

the nine peaks, $peak_0 \ldots peak_8$, are defined by

$$peak_i = \left(r_u,\ \ cos\frac{\pi i}{8}\sqrt{r_v^2 + r_n^2},\ \ sin\frac{\pi i}{8}\sqrt{r_v^2 + r_n^2} \right),$$

If $n_v < n_u$, the conical specular lobe is aligned with the **v** axis rather than the **u**, and the nine peaks are defined by

$$peak_i = \left(cos\frac{\pi i}{8}\sqrt{r_u^2 + r_n^2},\ \ r_v,\ \ sin\frac{\pi i}{8}\sqrt{r_u^2 + r_n^2} \right).$$

Peaks for more general reflection models. For multi-component reflection models, we use all of the peaks defined by the components. For unknown or measured BRDFs, we start with the reflection and retro-reflection vectors and then importance sample the BRDF to find additional "peaks". In our implementation, we sample the BRDF $N/4$ times, where N is the number of samples to be taken. Each of the BRDF samples is then added as a new peak if the value of f at the sample location is greater than the average of f at the corners of the leaf node that the sample falls in.

Splitting region neighbors. Whenever a region is split, we check the neighboring regions that border the split location to see if they should be split as well. In particular, if the two corners of the neighbor region that are adjacent to the split have f values which sum to less than f_{split}, then we cascade the split into the neighbor region. For example, suppose that the region directly below region R in Figure 5.5 was split horizontally at location x_{split}, with $x_0 < x_{split} < x_1$. Then if the value of f at (x_{split}, y_0) is greater than $f_{00} + f_{10}$, we split region R horizontally at location x_{split} as well. Note that in a spherical environment map the x axis wraps around, so that regions on the leftmost part of the environment map are neighbors of regions on the rightmost part. This extra splitting helps in the case of specular BRDF lobes that are not axis-aligned.

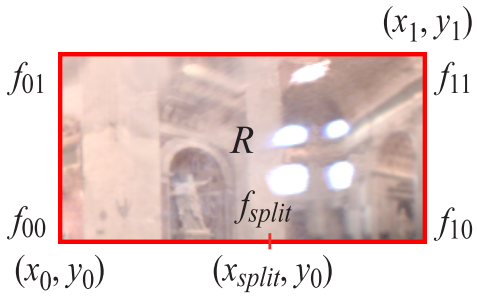


Figure 5.5: Cascading region splits. In the diagram, the region below region R has just been split horizontally at location x_{split}. The region splitting algorithm cascades this split into region R if $f_{split} > f_{00} + f_{10}$.

Subdividing based on split potential. At this point, we have a hierarchy that includes the peaks of the BRDF as region corners, but it still may not be a good approximation of the product. We can improve the approximation by splitting some of the regions that do not fit the product well.

For a region R with corners (x_0, y_0) and (x_1, y_1), and f values $f_{00} \ldots f_{11}$ (refer to Figure 5.5), we define a heuristic called the "split potential" that determines which region should be split to improve the product approximation the most. We base our heuristic on the idea that large and bright regions, and regions in which f varies a lot should be split first. More formally, we define the split potential, $p_{split}(R)$, as

$$p_{split}(R) = \sigma_f(R)\ sum(R)\ area(R), \tag{5.3}$$

where $sum(R)$ is the sum of the environment map over region R, $area(R)$ is the area of the region, and $\sigma_f(R)$ is the standard deviation of the f values at the corners of R:

$$\sigma_f(R) = \frac{1}{2}\sqrt{(f_{00} - f_{ave})^2 + (f_{10} - f_{ave})^2 + (f_{01} - f_{ave})^2 + (f_{11} - f_{ave})^2}.$$

Our strategy is to perform a fixed number of node splits after creating the initial hierarchy, the same as the number of samples that will be taken, always splitting the leaf node with

the highest split potential. This has the effect that each additional sample improves the sampling distribution, so that the convergence rate increases as more samples are added. Once again, whenever a region is split, its neighbors are checked to see if they should be split as well.

Split direction. In addition to deciding which region to split, the algorithm must decide whether to split the region along the x or y axis. One possibility would be to always split the region along the longest dimension, but we have found it better to choose the split axis based on the values of f as well as the axis lengths. Referring to Figure 5.5, a region will be split in the x direction if

$$\left((f_{10}-f_{00})^2+(f_{11}-f_{01})^2\right)(x_1-x_0) > \left((f_{01}-f_{00})^2+(f_{11}-f_{10})^2\right)(y_1-y_0). \quad (5.4)$$

Otherwise, the region will be split in the y direction.

Calculating region weights. As part of the sample warping algorithm, each region in the hierarchy must have an approximation to its contribution to the total product integral, which we call the region weight. For a leaf region L the weight is given by

$$weight(L) = sum(L)\ (f_{00}+f_{10}+f_{01}+f_{11})/4. \quad (5.5)$$

For a non-leaf region R, the weight is obtained by summing the weights of its children. As a convenience for the warping algorithm, we store two other values in non-leaf nodes in the hierarchy – the probability of choosing child A, $prob(A,R)$, and the fraction of the area that is taken up by child A, $areaFraction(A,R)$:

$$prob(A,R) = weight(A)/weight(R) \quad (5.6)$$

$$areaFraction(A,R) = area(A)/area(R). \quad (5.7)$$

Figure 5.6 gives pseudocode for the partitioning algorithm and shows example partitions for different BRDF types.

5.3 Hierarchical Warping (revisited)

This section describes the warping algorithm that forms the second stage of two stage importance sampling. Our warping algorithm is based on the technique presented by Clarberg et al. [2005] to transform a uniform distribution of points to the product distribution. The main difference between our algorithm and the one presented by Clarberg et al. is that in our algorithm warping is controlled by the environment map partition described in section 5.2 rather than a wavelet product. Of course, this means that our algorithm only approximates the product distribution, but it also means that our algorithm requires minimal precomputation and storage (just a summed area table of the environment map). Also, since two stage importance sampling does not rely on a discrete sampling of the BRDF, it can work directly with analytical or sampled BRDFs in their native formats. Other differences include the fact that we warp one sample at a time rather than groups of samples, and the fact that regions do not always divide exactly in half in the environment map partition.

As an illustration of how the sample warping works, consider figure 5.7. The figure depicts a region, R, of the environment map that has two children. The left child of the region, A, shown in gray, contains 70% of the area of R, and the right child, B, shown in white, contains 30% of the area of R. Now, suppose that the algorithm wants to create a distribution that puts 20% of the samples in child A, and 80% of the samples in child B. The warping starts with a uniform distribution of samples, as shown in the left image. It then stretches the leftmost 20% of the samples to cover 70% of the area of R, and squeezes the remaining 80% of the samples to fit into 30% of the area, as shown in the right image. This process can then be repeated on A and B, and so on, to create any desired sampling distribution.

Environment map partitioning algorithm

For each primary ray
    Create the root node (Fig. 5.3).
    Split the root at (x_n, y_n) and $((x_n + w/2)\%w,\ y_n)$.
    Split leaves at BRDF peaks (and neighbors as needed).
    For $i = 1$ to # of samples per primary ray
        Find the leaf node, R, that maximizes p_{split} (eq. 5.3).
        Split R along the axis specified by equation 5.4.
        Split the neighbors of R as needed (Fig. 5.5).
    Calculate sampling weights for all regions (eq. 5.5, 5.6, 5.7).

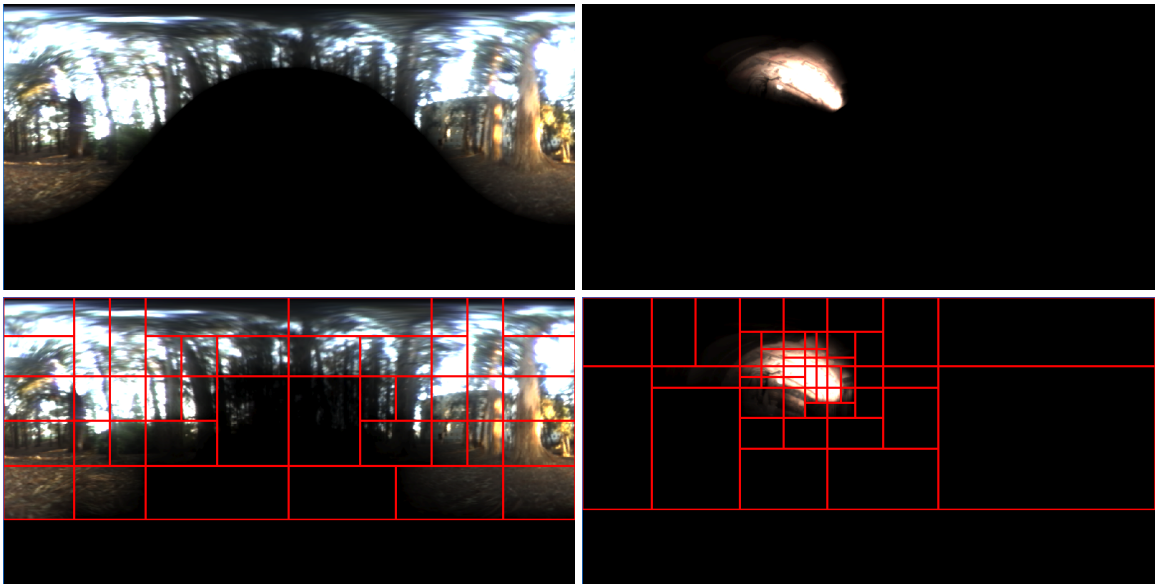


Figure 5.6: Environment map partitioning algorithm and example partitions. The left images show the product of a highly diffuse Oren-Nayar BRDF in the Eucalyptus Grove environment (top), and the approximation made by the partitioning algorithm (bottom). The right images show a more specular BRDF that uses the Blinn microfacet model in Galileo's Tomb. In both cases, the partitioning algorithm produces a good approximation to the product. Note particularly how the split potential heuristic "homes in" on the specular lobe of the Blinn BRDF.

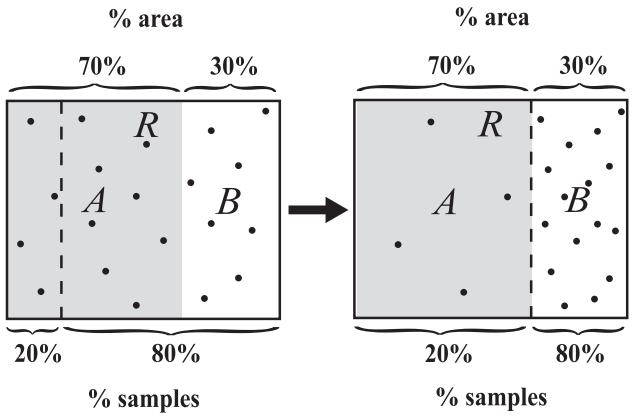


Figure 5.7: One step of the sample warping algorithm. Here, an environment map region R has two children, with the left child, A, shown in gray, and the right child, B, shown in white. The samples allocated to R are originally distributed evenly (left image), but the sample warping stretches 20% of the samples to cover the child A, and the remaining 80% of the samples are squeezed to fit into child B (right image).

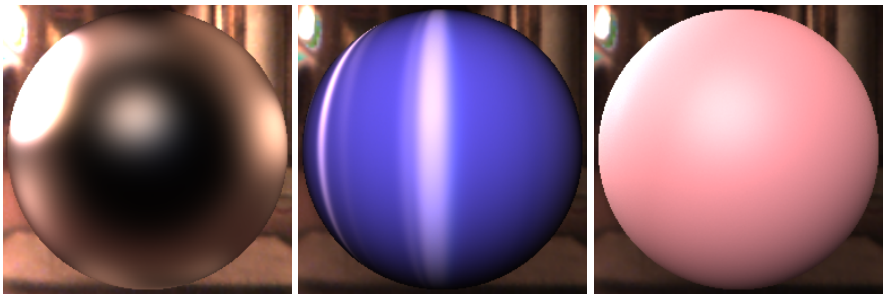
As previously stated, our warping algorithm guides the sample warping with the environment map partition described in section 5.2. The warping starts at the root node of the environment map partition with a sample (s,t) that is uniformly distributed in $[0,1)^2$ and a probability that is set to the area of the environment map divided by the area of the root node. The algorithm next decides which child of the root that the sample belongs to based on the value of $prob(A,R)$ stored in the root node. It then warps s or t depending on the split axis, and updates the probability for the sample. For simplicity, assume that the root is split on the x axis, so that s should be warped. If $s \leq prob(A,R)$, then s is warped by dividing by $prob(A,R)$, transforming it into the local coordinate system of child region A. If $s > prob(A,R)$, s is set to $(s - prob(A,R))/(1 - prob(A,R))$, transforming it into the local coordinate system of child B. The sample probability is updated in a similar manner. If $s \leq prob(A,R)$, the sample probability is multiplied by $prob(A,R)/areaFraction(A,R)$, and if $s > prob(A,R)$, the sample probability is multiplied by $(1 - prob(A,R))/(1 - areaFraction(A,R))$, reflecting the change in probability density in the two child regions.

This warping process repeats on the chosen child node, and so on. Once the leaf level of the hierarchy is reached, a second warping routine takes over, warping the sample down to the pixel level in like manner, constructing virtual environment map regions below the leaf level to do the warping. Figures 5.13 and 5.14 in appendix 5A give code for the two sample warping routines. After a sample has been warped, it can be used directly to define a Monte Carlo sample for equation 5.2. Since the warping routine keeps track of the probability with which the sample direction was generated, Monte Carlo estimates made in this manner are unbiased.

5.4 Convergence to the Product Distribution

A note about our error metric. To test convergence we are using an error metric called the "coefficient of variation", which is defined as the standard deviation divided by the mean, σ/μ (RMSE / mean pixel value). Since this value is a scalar multiple of σ, it is just as valid a statistical measure as using σ or σ^2 directly. However, it also provides an intuitive metric for how much visual noise exists within an image. Based on our observations, a good rule of thumb seems to be that an image (with Gaussian noise spread evenly over the image plane) will be nearly flawless visually if σ/μ is less than about 0.01.

Pseudo-random number sequences. For all sampling methods in all of the experiments in the paper, we used Hammersley point sets that were randomly shifted to avoid correlation artifacts (called a Cranley-Patterson rotation [Cranley and Patterson, 1976; Kollig and Keller, 2002]). The Hammersley point set has several properties that make it well suited for our application. First, it can create point sets of arbitrary size, not just powers of 2 or $n \times m$. Second, it consistently produced the lowest error of the sampling methods that we tried, including random and jittered point sets, and scrambled (0,2)-sequences.



Blinn Ashikhmin Lafortune

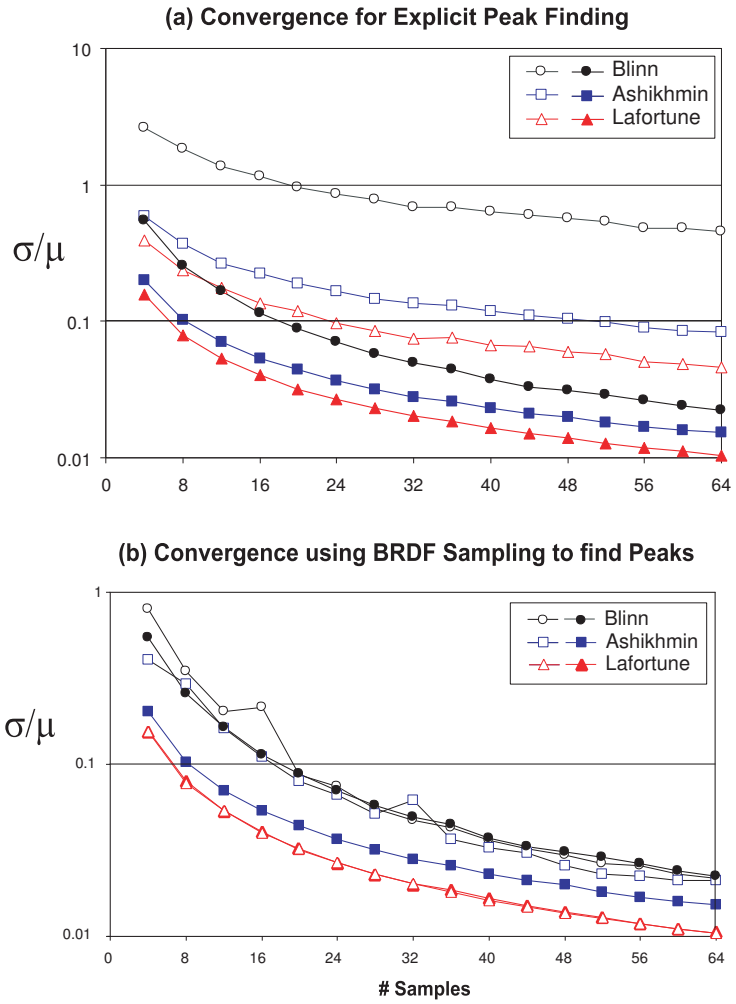


Figure 5.8: **(a)** Convergence of two stage sampling (solid markers) vs. multiple importance sampling (hollow markers) for different BRDF types. The "Blinn" scene uses the Blinn microfacet model with a roughness of 0.02. "Ashikhmin" uses the Ashikhmin anisotropic model with u and v roughnesses of 0.001 and 1.0, respectively. "Lafortune" is a fit of the Lafortune model to measured skin reflectance. **(b)** Convergence for different peak finding methods: sampling the BRDF distribution to find peak directions (hollow markers), and explicit enumeration of BRDF peaks (solid markers).

Convergence of two stage sampling. To get a feel for how quickly two stage sampling converges to the product distribution, we rendered a number of spheres with different BRDFs using two stage importance sampling and multiple importance sampling in the Galileo's Tomb environment. Figure 5.8(a) shows the results of these experiments for three different BRDFs. Not only does our algorithm have a lower error than MIS for these examples, but the rate of convergence is much better as well. For example, two stage importance sampling converges at a rate of $N^{-1.17}$ for the Blinn scene, $N^{-0.93}$ for Ashikhmin, and $N^{-0.98}$ for Lafortune. Compare this to MIS, which converges according to $N^{-0.64}$, $N^{-0.70}$ and $N^{-0.77}$ for the same scenes, a much slower rate. Our algorithm is able to achieve such high convergence rates because increasing the number of samples actually improves the sampling distribution.

Convergence for different peak finding methods. Figure 5.8 demonstrates the convergence of two stage sampling when the BRDF peak directions are known. However, peak directions may not be easily computed for some reflection models, such as sampled BRDFs. In section 5.2.2, we advocated BRDF importance sampling as an alternative method to find peak directions when they are not easily computed. To test the effectiveness of this alternate procedure, we re-rendered the scenes from figure 5.8 using BRDF sampling to find the peak directions. To make the comparison fair, we did not include the reflection or retro-reflection vectors as initial peaks, since the Blinn and Ashikhmin model both have strong peaks around the reflection vector. Figure 5.8(b) plots the results of using BRDF sampling to find peaks (hollow markers) alongside the results from figure 5.8 for comparison. As can be seen in the figure, the Lafortune scene performs almost identically for both peak finding methods. The Blinn scene starts out a little worse with the alternate peak finding method, but quickly overtakes the performance of the standard peak finding method. The Ashikhmin scene, on the other hand, does not perform as well with the alternate peak finding method, likely because of the complexity of the conical specular lobe, but it still

outperforms MIS by a wide margin. These results suggest that BRDF importance sampling can be an effective general peak finding method for many BRDF types.

5.5 Variable Samples

Section 5.4 describes the performance of two stage importance sampling in the absence of shadows. However, most scenes contain shadows and other features that pose difficulties for a Monte Carlo sampler. Penumbra regions are particularly prone to noise. A desirable goal would be to assign different numbers of samples to different parts of the image to distribute noise more or less evenly over the entire image plane.

We achieve this goal by allocating more samples to pixels that are likely to have a high variance, and less samples to pixels likely to have a low variance. To do this, we need an estimate of the standard deviation of samples for the current pixel, σ_p, and the average standard deviation over all pixels, σ_{ave}. We obtain σ_{ave} in a preprocessing step, computing the average standard deviation for a small random subset of the pixels in the image (1024 in our implementation). σ_p is computed as a running average over previous samples. It's value is set to zero initially, and after a pixel (primary ray) has been processed, the value is updated using the formula

$$\sigma_p^{new} = 0.75\sigma_p + 0.25\sigma_{pixel} \tag{5.8}$$

where σ_{pixel} is the sample variance of the pixel just processed. This process is similar to the variance tracking method used for efficiency optimized Russian roulette in bidirectional path tracing [Veach and Guibas, 1994], except that we are using the value for a different purpose.

At this point, we need to determine a number of samples to use for the next pixel. To make the sample allocation robust, the number of samples for a pixel is always set to at least half of what it would be in the fixed case. For the other half of the samples,

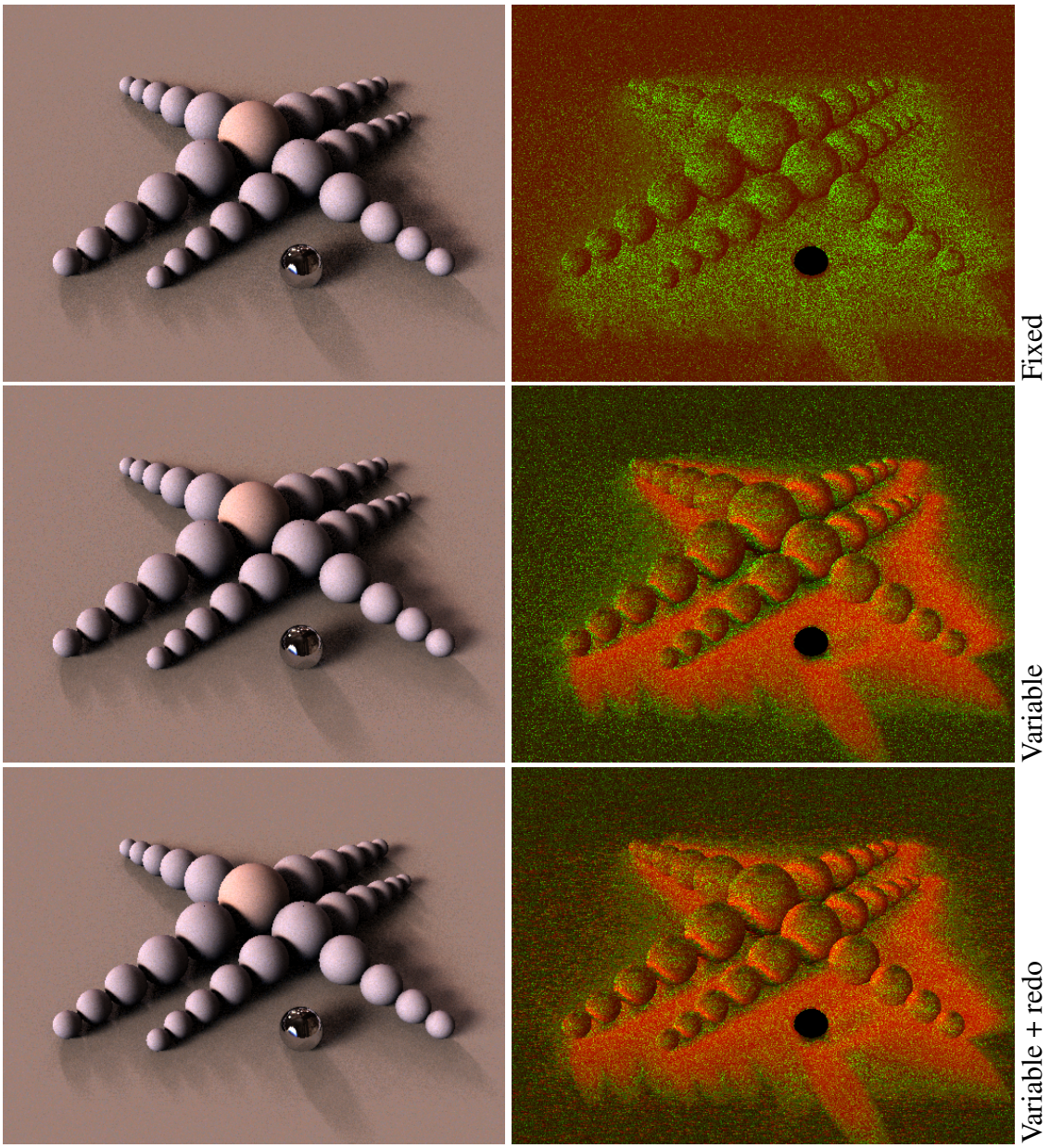


Figure 5.9: Fixed and variable sampling rates. The left column of images shows renderings made with different sampling strategies in the Galileo's Tomb environment, and the right column shows error in green and the number of samples allocated to different pixels in red. All renderings use about 16 samples per pixel on average. Fixed rate sampling concentrates errors in the penumbra regions of the image ($\sigma/\mu = 0.0446$, 25.5 sec.). Variable sampling spreads the error more evenly, and overall error is reduced ($\sigma/\mu = 0.0409$, 25.9 sec.). Note how samples concentrate in the high error regions. Finally, re-rendering the few pixels that significantly overshoot their variance estimates gets rid of most of the highest error pixels ($\sigma/\mu = 0.0366$, 27.3 sec.).

the algorithm must decide how many to allocate to each pixel based on σ_p. We could use the variance analysis results from Mitchell [1996] or Kollig and Keller [2002] as a guide to determine how many samples to allocate to different pixels, but this would lead to a different expression for each input sequence type. Instead, we make the rather crude assumption that the standard deviation will decrease as N^{-1}, and allocate the second half of the samples accordingly. Thus, the number of samples that we allocate to a pixel is

$$M = \frac{N}{2}\left(1 + \frac{\sigma_p}{\sigma_{ave}}\right), \tag{5.9}$$

where N is the average number of samples that should be allocated to each pixel. Reallocating samples in this way tends to decrease statistical error slightly, but perceptual error is reduced much more, since the sample reallocation spreads the error more or less evenly over the image plane. Allocating variable samples as just described does not introduce bias since information from the current pixel does not influence the variance estimate, σ_p [Kirk and Arvo, 1991].

A problem that can occur when using variable samples is that the variance estimate, σ_p, may not be accurate. This can happen at object edges, for instance. Our solution is to re-render those few pixels (about 5%) in which the actual variance within the pixel is much greater than the estimate. Specifically, if $\sigma_{pixel} > \sigma_p + \sigma_{ave}$, we reset σ_p to σ_{pixel}, and render the pixel again. Although this step adds a slight bias to the solution, it is quite adept at eliminating most of the highest error pixels. Figure 5.9 shows a scene rendered with fixed and variable numbers of samples.

5.6 Results

We have implemented two stage importance sampling as an extension to the PBRT rendering system [Pharr and Humphreys, 2004]. PBRT supports a variety of analytical BRDF models which we used to validate our algorithm, including the Blinn, Ashikhmin, Lafortune and Oren-Nayar models.

Comparison with other robust sampling algorithms. Figure 5.12 compares our two stage sampling algorithm to multiple importance sampling (MIS) and resampled importance sampling (RIS). For RIS, we use multiple importance sampling to generate 2 tentative samples per accepted sample. The number of tentative samples per accepted sample in our study, 2, was chosen by trial and error to maximize image quality vs. render time. Burke, Ghosh and Heidrich [2005] were able to quickly generate dozens of tentative samples per accepted sample using the Phong model. We have found the expense of generating samples for arbitrary BRDFs and evaluating them without visibility in PBRT to be comparatively large, about $1/3$ the cost of shadow queries. This is more in line with the results reported by Talbot, Cline and Egbert [2005].

We do not presently have an implementation of wavelet importance sampling to directly compare with our algorithm, but we can simulate WaIS using our algorithm by greatly increasing the number of region splits in the environment map. The resulting probability distribution comes very close to the actual product distribution, and thus mimics WaIS quite well. WaIS render times were approximated simply by copying the timing results from MIS. Note that the timing results do not include the substantial startup costs of WaIS. As can be seen in the table of images, two stage importance sampling displays a lot of noise at very low sample counts, but it quickly converges. Although we cannot expect to surpass the convergence rate of WaIS, our algorithm does start to become competitive with it in terms of quality per number of samples after just a few dozen samples per pixel, without the long preprocessing times needed for WaIS.

Scenes with large numbers of BRDFs. Figure 5.10 shows a scene in which the specular roughness parameter is controlled by a texture map. This scene would be impractical for WaIS because each surface point potentially has a BRDF with a different shape. By contrast, two stage importance sampling has no difficulties with the scene because it does not need to preprocess BRDFs.

Figure 5.10: Scene with a spatially varying specular exponent. This scene literally contains hundreds of different BRDF shapes. The scene was rendered with 4 primary rays per pixel and 32 two stage samples per primary ray at a resolution of 1024×512. Render time was 895 seconds.

BRDF and light source sampling alone. For some scenes, BRDF or light source sampling alone may perform better than MIS. However, these methods are not robust to changes in BRDF and lighting. For example, light source sampling performs well on the scene in figure 5.12 ($\sigma/\mu = 0.038$ vs. 0.030 for approximately equal time using our algorithm), but the performance of light source sampling drops quite dramatically if the diffuse floor is changed to a glossy Blinn model with a roughness of 0.1. In the changed scene, both light source and BRDF sampling perform poorly, while our algorithm remains robust (σ/μ = 0.243, 0.206 and 0.041 for BRDF sampling, light source sampling, and our algorithm, respectively). Table 5.1 summarizes these results.

	 σ/μ (Diffuse)	 σ/μ (Glossy)
BRDF (64 spp)	0.300	0.243
Light Source (64 spp)	0.038	0.206
Two Stage (32 spp)	0.030	0.041

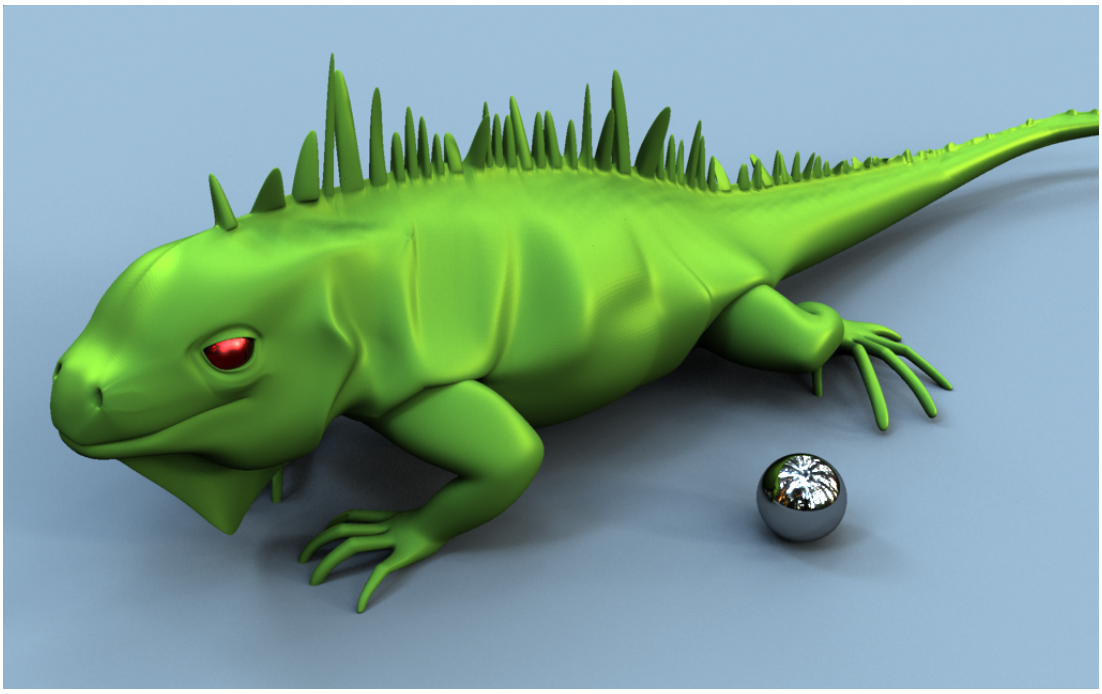
Table 5.1: Overall error of BRDF, light source and two stage importance sampling for predominantly diffuse and glossy scenes.

5.7 Conclusions and Future Work

In this paper we have presented a new importance sampling technique for direct lighting called *two stage importance sampling*. The technique is unbiased, and we showed that it works well for scenes with complex BRDFs and spatially varying environment map lighting. We showed that two stage importance sampling outperforms multiple importance sampling (MIS) and resampled importance sampling (RIS) in a number of rendering contexts, and that the new algorithm compares favorably with wavelet importance sampling in terms of quality per number of samples at fairly low sample densities, while requiring substantially less precomputation. In addition, we described a novel technique to allocate variable numbers of samples to different pixels in a rendered image to reduce noise in shadow areas, and showed that the new sample allocation technique can decrease statistical and visual error in rendered images.

There is a fairly large overhead associated with partitioning the light source, and this limits the number of primary rays that can be cast per pixel. It would be interesting to cast a larger number of primary rays, and then use RIS to limit the number of product approximations that must be evaluated. Another idea would be to reuse partitions between primary rays when the BRDF setup is similar enough. Currently, our product approximation does not account for color information because the f values and the summed area table

only store luminances. We could incorporate color into our algorithm by storing a color summed area table and color f values. Our current implementation uses a very simple rule to determine the number of regions to split when partitioning the environment map (the number of splits equals the number of samples). There may be better heuristics for the number of splits in terms of image quality vs. time. Finally, we note that while our algorithm to allocate different numbers of samples to different pixels is useful, it does not take directional information into account. As the algorithm now stands, shadow regions account for the majority of the noise at sample rates above a few dozen per pixel, so improved anti-aliasing for shadow regions would be of great benefit.



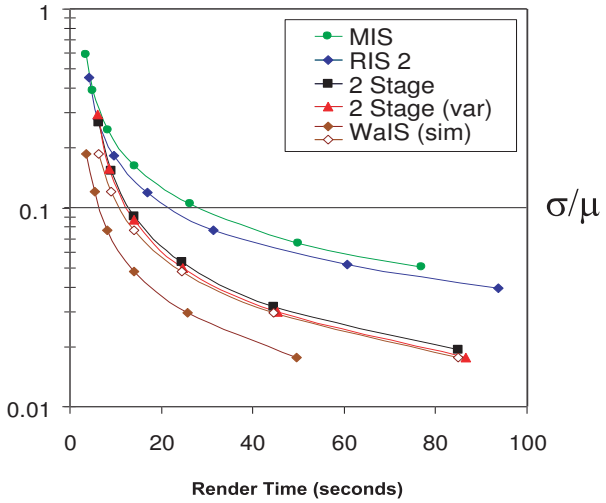


Figure 5.11: Convergence of two stage importance sampling in terms of render time. The chart plots image quality against render time for multiple importance sampling (MIS), resampled importance sampling with two tentative samples per accepted sample (RIS 2), our two stage importance sampling algorithm (2 Stage), and a simulated version of wavelet importance sampling (WaIS). For comparison, we have plotted WaIS against estimated render time (solid diamonds), and using the same timing as two stage sampling (hollow diamonds). Although WaIS beats our algorithm in terms of quality per unit time (if we don't count startup time) this is mainly because WaIS can generate samples more quickly. When we compare using number of samples instead of render time (hollow diamonds), our algorithm starts out worse, but quickly converges to within a few percent of the noise level of WaIS. Note also that render times for WaIS do not include preprocessing. A major advantage of our algorithm over WaIS is that we have extremely low startup time (less than a second), whereas Clarberg et al. [2005] reported startup times of more than an hour for some scenes.

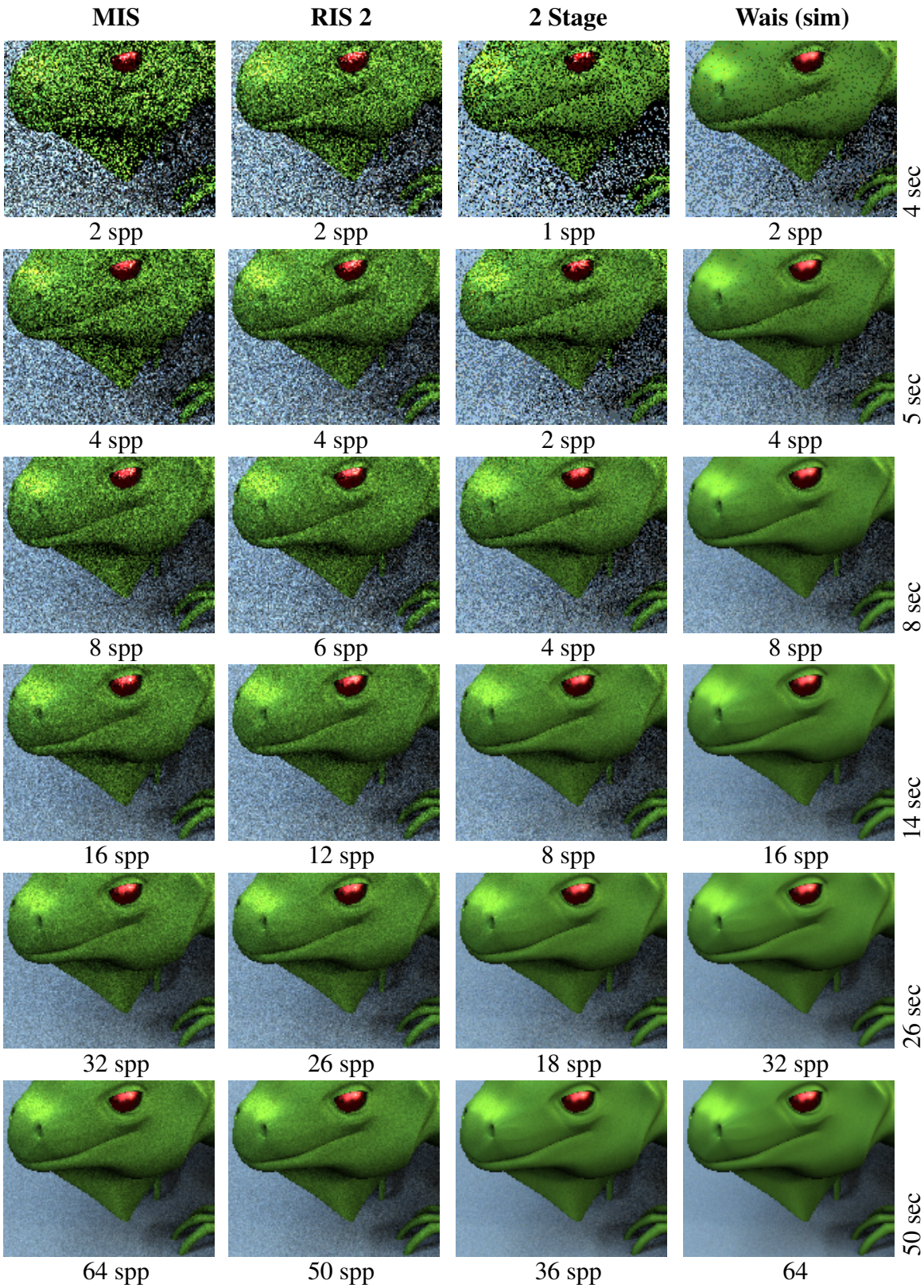


Figure 5.12: The table of images gives a quality comparison for approximately equal render times for two stage importance sampling and other sampling methods.

Appendix 5A: Sample warping routines.

```
void warpSample1(float &s, float &t, float &prob, Region *R,
        SummedAreaTable &sumTable)
{
    prob = sumTable.width * sumTable.height / (float)R->area();
    while (R->hasChildren()) {
        if (R->splitAxis == X_AXIS) {   // SPLIT ON X AXIS
            if (s < R->probA) {   // S IS ON MIN X SIDE
                s /= R->probA;
                prob *= R->probA / R->areaFractionA;
                region = R->childA;
            } else {   // S IS ON MAX X SIDE
                s = (s − R->probA) / (1 − R->probA);
                prob *= (1 − R->probA) / (1 − R->areaFractionA);
                R = R->childB;
            }
        } else {   // SPLIT ON Y AXIS
            .
            .   // Y CASE OMITTED FOR BREVITY
            .
        }
    }
    warpSample2(s, t, prob, R->x0, R->y0, R->x1, R->y1,
            R->f00, R->f10, R->f01, R->f11, sumTable);
}
```

Figure 5.13: Warping Algorithm (part 1). The *WarpSample1* routine is called with s and t uniformly distributed in $[0,1)^2$, and R set to the root node in the hierarchy. When *warpSample1* returns, s and t are still in $[0,1)^2$, but are now approximately distributed according to the product distribution, and *prob* contains the probability that the resulting point was chosen in terms of solid angle.

```
void warpSample2(float &s, float &t, float &prob,
                 int x0, int y0, int x1, int y1,
                 float f00, float f10, float f10, float f11,
                 SummedAreaTable &sumTable)
{
    float f0mid, f1mid, fmid0, fmid1;
    float faveA, faveB, areaFractionA;
    float sumA, sumB, probA, cosPhi;
    int xmid, ymid;

    while ((x1−x0) > 1 || (y1−y0) > 1) {
        if (x1−x0 > y1−y0) {   // SPLIT ON X AXIS
            .
            .   // X CASE OMITTED FOR BREVITY
            .
        } else {   // SPLIT ON Y AXIS
            ymid = (y0+y1)/2;
            areaFractionA = (ymid−y0) / (float)(y1−y0);
            f0mid = f00 + areaFractionA * (f01−f00);
            f1mid = f10 + areaFractionA * (f11−f10);
            faveA = f00 + f10 + f0mid + f1mid;
            faveB = f0mid + f1mid + f01 + f11;
            sumA = sumTable.sum(x0, y0, x1, ymid) * faveA;
            sumB = sumTable.sum(x0, ymid, x1, y1) * faveB;
            probA = sumA / (sumA+sumB);
            if (t < pA) {   // T IS ON MIN Y SIDE
                t = t / probA;
                prob *= probA / areaFractionA;
                y1 = ymid;
                f01 = f0mid;
                f11 = f1mid;
            } else {   // T IS ON MAX Y SIDE
                t = (t − probA) / (1 − probA);
                prob *= (1 − probA) / (1 − areaFractionA);
                y0 = ymid;
                f00 = f0mid;
                f10 = f1mid;
            }
        }
    }
    // CONVERT S AND T TO GLOBAL COORDINATES
    s = (s+x0) / sumTable.width;
    t = (t+y0) / sumTable.height;
    // CONVERT TO PROBABILITY OVER SOLID ANGLE
    cosPhi = cos( PI * (0.5 − t) );
    prob *= cosPhi / (2.0 * PI * PI);
}
```

Figure 5.14: Warping Algorithm (part 2). The warpSample2 warps (s,t) down to the pixel level. After the warping is done, s and t are converted to the global coordinate system of the environment map. *prob* is also converted from probability over the area of the environment map to be in terms of solid angle, multiplying by $cos\phi/2\pi^2$, where ϕ is the angle of inclination to point (s,t) in the environment map.

Chapter 6

Towards Triple Product Sampling in Direct Lighting

A version of this chapter was published as:

David Cline, Kenric B. White and Parris K. Egbert. "Towards Triple Product Sampling in Direct Lighting." *2006 IEEE Symposium on Interactive Ray Tracing, Poster Compendium*. pages 11-12, 2006.

Abstract. State of the art sampling methods for direct lighting from environment maps generate samples according to the product of the environment map and BRDF without shadows. In this paper we describe techniques that augment existing product sampling strategies to include an approximate shadow term with little additional overhead. We compare these augmented techniques to a recently described product sampling method, two stage importance sampling, showing substantial quality improvements in rendered shadow regions.

6.1 Introduction

Lighting from environment maps has become popular in recent years due to the high visual quality of results that the technique can achieve. The main difficulty with environment map lighting lies in the complexity of the lighting equation. Even a direct lighting solution requires the evaluation of the direct lighting integral for each visible surface point:

$$L_d(x \rightarrow \Psi) = \int_{\Omega_x} L_e(x \leftarrow -\Theta) f_r(\Psi \leftrightarrow x \leftrightarrow \Theta) |\cos\theta| V(x \leftarrow -\Theta) d\Theta,$$

In the equation, the direct lighting from a surface point x in direction Ψ, $L_d(x \rightarrow \Psi)$, is the integral of three terms: the incident light from the environment map, $L_e(x \leftarrow -\Theta)$, the BRDF term, $f_r(\Psi \leftrightarrow x \leftrightarrow \Theta)|\cos\theta|$, and the environment map visibility in direction Θ, $V(x \leftarrow -\Theta)$.

Current state of the art importance sampling methods can generate samples according to the product of the incident light and BRDF terms of the direct lighting equation. Burke, Ghosh and Heidrich [2005] and Talbot, Cline and Egbert [2005] construct the product distribution (or approximation) starting with samples drawn from a simpler distribution. Most of these samples are then discarded, leaving samples that are distributed according to the product of the BRDF and incident light. Other techniques achieve the product distribution by warping an initial set of uniform samples. Clarberg et al. [2005] guide the sample warping with a wavelet product, and Cline et al. [2006] build a hierarchical partition of the light source for each primary ray that directs the sample warping. Still other work by Ghosh, Doucet and Heidrich [2006] uses sequential importance sampling to propagate the product distribution through time in the presence of animated objects or lighting.

Unfortunately, none the above mentioned importance sampling methods includes a visibility term in its resulting sample distribution. Visibility is determined after the expensive product sampling step by ray casting. The ideal importance sampling method for direct lighting should generate samples according to the *triple product* of the incident light,

BRDF and visibility rather than just the double product of the BRDF and incident light terms. This paper presents two methods that accomplish this goal by augmenting existing product sampling algorithms with either a pre- or post-processing step to account for the visibility of the light source. Our algorithms are similar in spirit to the work by Ben-Artzi, Ramamoorthi and Agrawala [2006], but we concentrate on direct sampling of environment maps rather than sampling sets of approximating point lights.

6.2 Adding a Visibility Term to Product Sampling

This section describes two methods that augment existing importance sampling algorithms to include a visibility term, either as a pre-process or a post-process. Both of the methods rely on low resolution visibility maps to define the visibility importance term, but we will show that even very coarse maps can appreciably reduce the error in penumbra regions.

6.2.1 Visibility in Preimage Space

Most of the product sampling algorithms mentioned in section 6.1 begin with a set of uniformly-distributed points in $[0,1)^2$ which are transformed to the product distribution. Ray samples are then created from the transformed points, and light source visibility is tested for these rays. Conceptually, however, there is no reason why the shadow rays cannot be replaced by a visibility map defined over $[0,1)^2$. We call this kind of a visibility map a *preimage visibility map* because it is defined over the domain of input numbers rather than the range of output directions. The benefit of defining a visibility map in this way is that occluded points can be culled before they go through an expensive transformation. When points that survive this preimage space visibility culling procedure are fed into a product distribution sampler such as two stage importance sampling, the resulting samples will be distributed according to the triple product of the visibility, BRDF, and environment map. The combined procedure thus forms a *triple product sampler* for direct lighting.

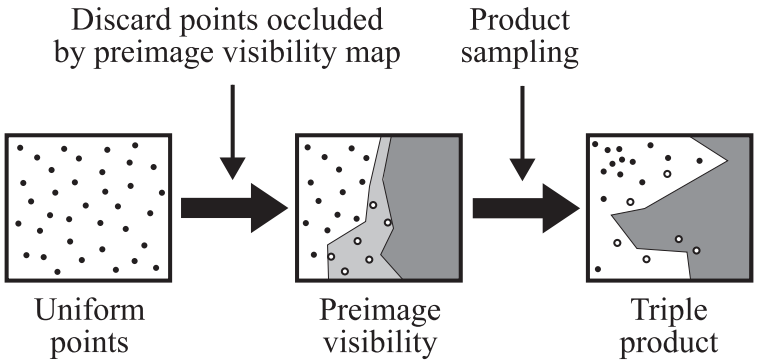


Figure 6.1: Triple product sampling based on preimage visibility maps. Uniform input samples in $[0,1)^2$ are generated (left), and samples within occluded regions of the random number space are discarded (middle). The remaining samples are fed into an existing product sampling algorithm. Samples in the high confidence region for visibility are assumed to be visible, and samples from the uncertain region (hollow points) cast shadow rays to test visibility.

In practice, of course, it is not feasible to produce an accurate visibility map for each primary ray, but it is possible to create approximate visibility maps without too much trouble. We will describe how to create these maps in section 6.2.3. For our purposes we define three zones within a visibility map: (1) areas of high confidence for occlusion, (2) areas of high confidence for visibility, and (3) uncertain areas. Our triple product sampling algorithm proceeds as follows: First, we must determine how many uniform points in $[0,1)^2$ to generate. To maintain consistent quality, one of our design goals is for each pixel to have roughly the same number of visible samples, so the number of samples that we create, m, is

$$m = n/v,$$

where n is the desired number of visible samples and v is the fraction of the visibility map that is visible or uncertain. Once the points have been generated, we discard zone 1 points immediately. The remaining points are then transformed to the product distribution. Finally, visibility is assumed for samples from zone 2, and shadow rays are cast to determine visibility for zone 3 samples. Figure 6.1 shows the steps of the algorithm graphically.

6.2.2 Visibility in World Space

A second variant of our algorithm relies on visibility maps defined in world space instead of preimage space. The second algorithm works as a post-process to product sampling rather than a preprocess. This has the consequence that occluded samples must be transformed to the product distribution before culling, but the world space maps offer geometric coherence that may not be present in the preimage maps. The algorithm proceeds as follows: first, uniform points are sent through a product sampling algorithm as usual. The resulting samples are then checked against a world space visibility map. Once again, samples in the occluded regions of the map are discarded, samples within visible regions are assumed to be visible, and samples within uncertain regions must cast shadow rays to determine visibility. Because of the overhead of transforming samples to the product distribution before they are discarded, we use a fixed number of samples per pixel in this variant of the algorithm. Figure 6.2 shows the second variant of the algorithm graphically.

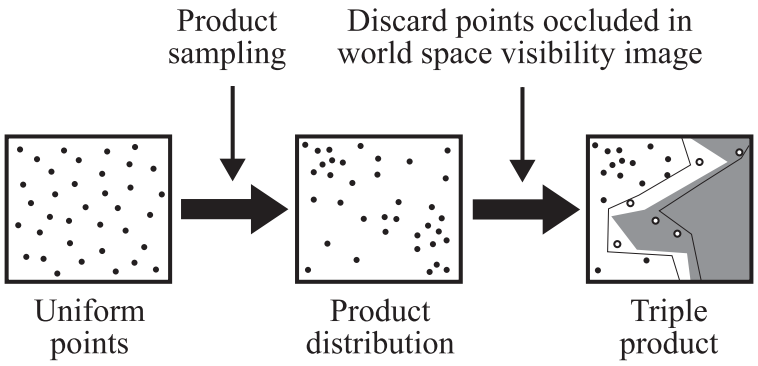


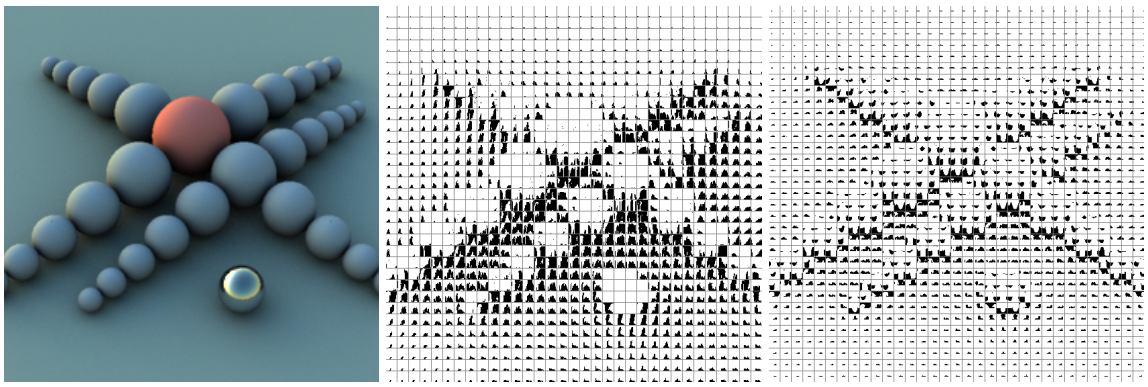
Figure 6.2: Approximate triple product sampling based on world space visibility maps. Uniform input samples (left) are transformed to the product distribution (middle). The transformed samples are then checked against a world space visibility map. As with preimage visibility maps, samples in "occluded" map regions are discarded, samples in "visible" regions are considered to be visible, and samples in "uncertain" regions (hollow points) test visibility by casting shadow rays.

6.2.3 Creating the Visibility Maps

This section describes how to create the visibility maps needed for our triple product samplers. We create visibility maps at a sparse grid of locations over the image plane. The size of the visibility maps is kept on the order of the pixel spacing between them so that the total cost of the visibility maps is only a few rays per pixel. For example, a grid of 16×16 visibility maps with a spacing of 8 pixels can be computed using just 4 ray samples per pixel. After the visibility maps are created, we mark pixels on the border between visible and non-visible regions as uncertain.

Figure 6.3 shows a simple scene along with its preimage and world space visibility maps. Note that the world space maps essentially form an ambient occlusion field of the scene, whereas the preimage maps include BRDF and lighting effects, more accurately reflecting the amount of shadowing present in the rendering.

To obtain a visibility map for a particular pixel location, we combine the four closest maps in the grid of maps. Texels that agree between all four neighbor maps are copied to the new map, and all other texels are set to "uncertain".



(a) (b) (c)

Figure 6.3: (a) An example scene with (b) the preimage visibility maps and (c) the world space visibility maps generated for it.

6.3 Unbiased Rendering

The triple product sampling algorithms described in the previous section are biased unless the visibility maps are completely accurate, which is almost never the case in practice. This section describes an unbiased version of our triple product sampler based on preimage visibility maps. An unbiased algorithm based on world space maps is also possible, but less useful because samples cannot be culled before they are fed through the expensive product distribution sampler.

Observe that the algorithms in section 6.2 are biased precisely because the visibility maps are not accurate. Thus, an unbiased algorithm can use the visibility maps as a guide, but not for final visibility queries. Additionally, an unbiased algorithm cannot discard all of the samples in the "occluded" regions of the visibility map, since they may not actually be occluded. Based on these observations, our unbiased triple product algorithm works by moving some fraction of the samples from the occluded regions of the visibility maps to the visible and uncertain regions. Typically we reallocate 50% of the samples from occluded regions, since reallocating more would lead to very bright samples in locations wrongly guessed as occluded by the visibility map. Note, however, that these gleaned samples

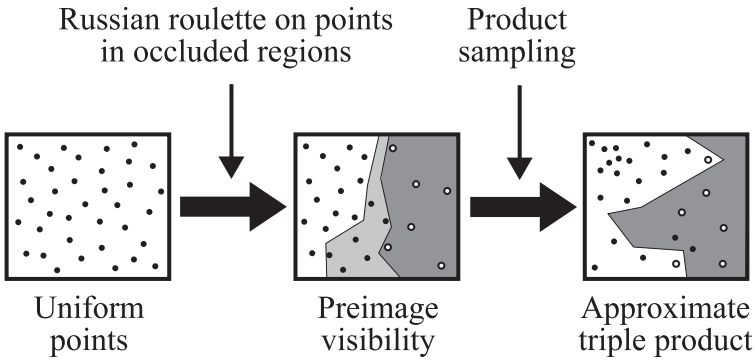


Figure 6.4: Unbiased triple product sampling based on preimage visibility maps. Uniform points (left) are generated, and Russian roulette is used to discard some of the points in occluded regions, leaving a higher density of samples in visible and uncertain areas (middle). The remaining points undergo product sampling, resulting in a distribution that is closer to the triple product than the double product alone.

can greatly increase the sample density in visible regions depending on the percentage of the map that is deemed uncertain or visible. Finally, to make the algorithm unbiased, we must cast rays to determine visibility. Figure 6.4 shows our unbiased algorithm based on preimage visibility maps.

Generating samples. Our unbiased triple product sampler accomplishes the sample reallocation by generating enough uniform points in $[0,1)^2$ to account for the samples that will be needed in visible regions, thinning out samples in occluded regions of the map by Russian roulette. The number of uniform points that the algorithm must generate, m, is given by

$$m = n\,\frac{v+(1-v)(1-a)}{v},$$

where n is the desired number of final samples per pixel, v is the fraction of the environment map that is uncertain or visible, and a is the relative density of points in occluded regions compared to standard sampling. The probability of keeping points in occluded regions of the visibilty map, p, then becomes

$$p = \frac{av}{v+(1-v)(1-a)}.$$

Correcting the visibility maps. During the course of rendering, a ray cast into an "occluded" region of the visibilitiy map may actually find the light source to be visible. When this happens, the algorithm corrects the error by setting the corresponding entry and its immediate neighbors in the four visibility maps nearest the current pixel to "uncertain".

6.4 Results

We have implemented our triple product samplers as an extension to PBRT [Pharr and Humphreys, 2004], using two stage importance sampling [Cline *et al.*, 2006] as the product sampler. This section compares standard two stage importance sampling to our augmented triple product samplers.

Light shining through a small hole. The scene in figure 6.7 consists of a bunny enclosed in a box with a small hole in the top to let light through. Most of the light from the environment map (grace cathedral) is blocked, so that the product distribution does not form a good importance function for the scene. The graphs to the left of the image plot image quality for our biased and unbiased triple product algorithms along with standard two stage importance sampling for comparison. Note that in both cases the preimage visibility maps outperform the world space maps.

For the bunny scene, our unbiased algorithm based on preimage visibility maps outperforms two stage importance sampling alone by about two times as measured by RMSE, but the quality difference in low light regions is actually much more dramatic. The images in figure 6.5 demonstrate this. The images have been tone mapped to bring out detail in

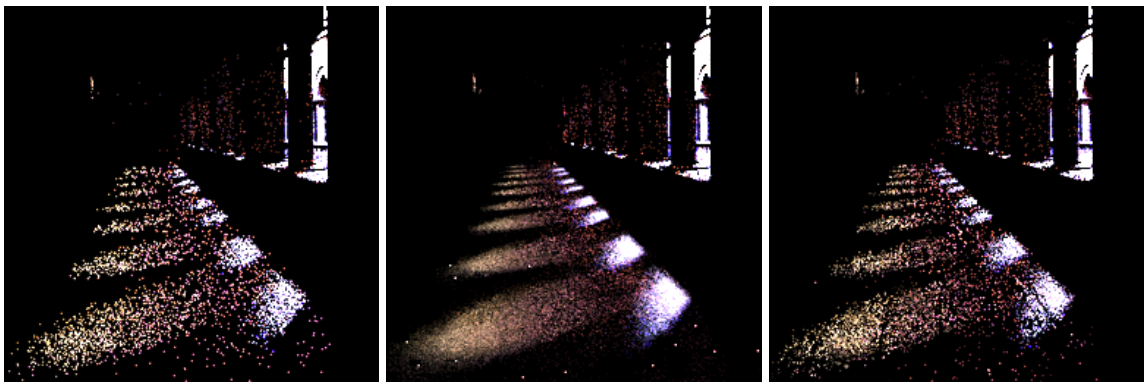
(a) (b) (c)

Figure 6.5: Insets of the bunny scene tone mapped to bring out details in dark regions of the image. (a) Two stage importance sampling, (b) biased triple product sampling and (c) unbiased triple product sampling.

dark regions. Two stage importance sampling (a) maintains high quality in bright regions near the bunny, but the dark parts of the image are extremely noisy. The biased version of our triple product sampler (b) is smoother everywhere, but some of the dark areas in the image are completely black because of errors in the visibility maps. Finally, the unbiased version of our algorithm (c) is smooth almost everywhere, except for a few pixels in the darkest regions. These bright pixels correspond to samples that were erroneously marked as occluded in the visibility maps. If some slight bias can be tolerated, the bright spots could be eliminated by using the same probability for these samples that is used for unoccluded ones in the Monte Carlo Integration formula.

Sponza atrium scene. Figure 6.8 shows results for a second test case of our technique, the Sponza atrium, once again in the Grace cathedral environment. For this test, we used the unbiased triple product sampler based on preimage visibility maps. We tested this scene with and without bump maps. In the first test, we compared two stage importance sampling to our triple product sampler without using bump maps in the scene. As with the bunny example, the render quality improved somewhat overall, but much more in low light areas. In a second test, we added bump maps to the scene. Since the bump maps drastically changed the orientation of scene normals, the modified scene violated the assumption that preimage space coordinates correlate to similar world space directions within pixel neighborhoods. Consequently, the triple product sampler essentially reverted to standard product sampling, and quality remained approximately the same as two stage importance sampling. Figure 6.6 shows close-ups of the sponza scene rendered using the same number of samples with two stage importance sampling, and triple product sampling without and with bump maps.

Lazy evaluation of the visibility maps. With the unbiased algorithms it is possible to evaluate the visibility maps lazily rather than as a preprocess, avoiding the startup cost needed for the visibility maps. To do this, we initialize all of the maps to "occluded", and update the maps as described in section 6.3 for all visible samples instead of just visible



(a) (b) (c)

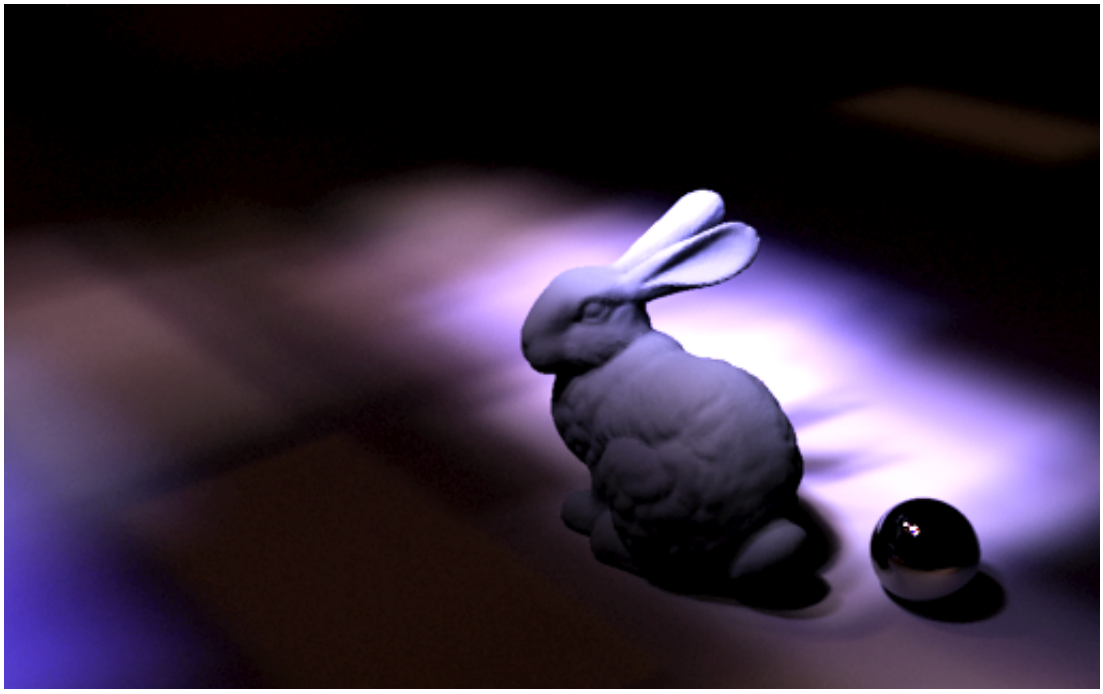
Figure 6.6: Close-ups of the lower left hand corner of the sponza scene showing light coming through the colonade. (a) Two stage importance sampling is quite noisy in low light regions. (b) Unbiased triple product sampling produces a much smoother result, but adding bump maps to the scene (c) destroys the assumption of correlation between preimage space points and world space directions, reducing the quality. Note, however, that the triple product sampler does not fail terribly in this case. Instead, its performance degrades gracefully to approximately that of two stage importance sampling.

samples from occluded regions. We implemented this technique, and found that quality remained about the same for similar render time (time gained by avoiding preprocessing was approximately offset by a corresponding reduction in image quality). However, one unexpected result was that almost all of the bright speckles from visibility errors were missing. We believe that this is the consequence of updating the maps for each visible sample.

6.5 Conclusion

This paper introduced several triple product samplers for direct lighting that augment existing product sampling methods with an approximate visibility term. We presented the idea of preimage space visibility maps, which operate in the space of input numbers rather than world space, allowing occluded samples to be culled without the need to transform them into world space. Finally, we compared our triple product samplers to an existing product distribution sampler (two stage importance sampling), and demonstrated quality improvements in shadowed regions, both in terms of objective error and visual quality.

“Hare Noir” Bunny Scene



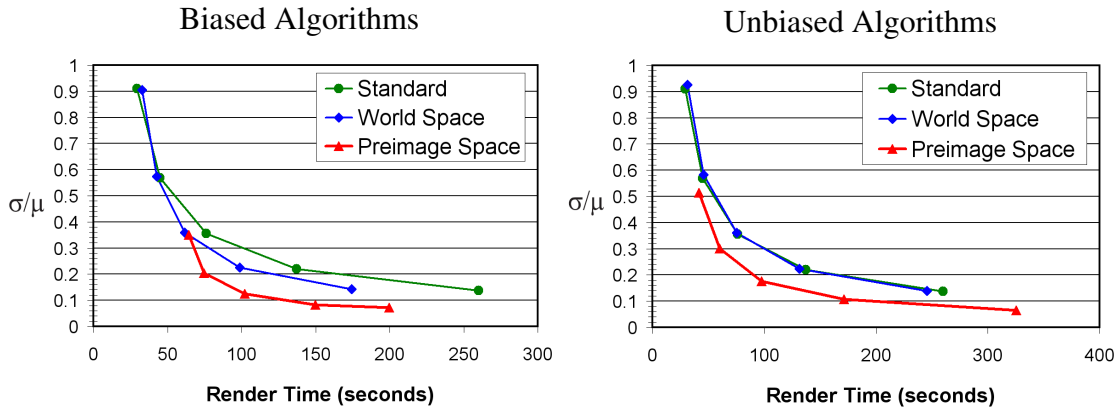


Figure 6.7: In the scene above, the bunny is enclosed in a box that has a small hole in the top to let light through. The left graph plots image error against render time for standard two stage importance sampling vs. the biased variants of our triple product samplers. “World Space” uses world space visibility maps and “Preimage Space” uses preimage visibility maps. The graph to the right compares our unbiased triple product samplers to two stage importance sampling.

Sponza Atrium

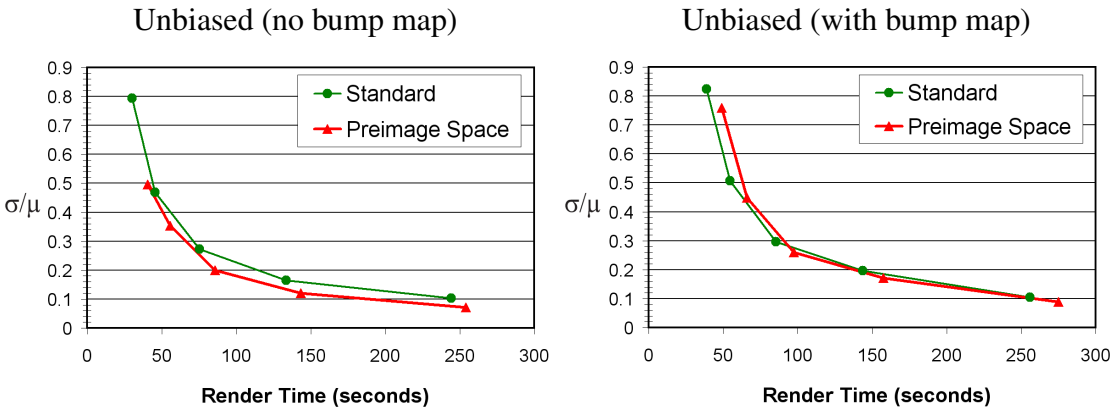


Figure 6.8: The Sponza atrium scene in the Grace cathedral environment. The left graph plots error against render time for two stage importance sampling and the unbiased version of our preimage space algorithm when bump maps are not used in the scene. Without bump maps, a good quality improvement is achieved. When bump maps are added to the scene (right graph), the assumption of correlation between preimage space coordinates and world space directions is broken. In this case, the algorithm receives little benefit from the visibility maps, and essentially reverts to product sampling. Even when this happens, however, the triple product sampler does no worse than the product sampler by itself.

Part III

Sampling Methods for Global Illumination

Chapter 7

Ray Based Global Illumination Methods

Much of the content in this chapter was adapted from the technical report:

David Cline and Parris K. Egbert. A Practical Introduction to Metropolis Light Transport. *Technical Report, Brigham Young University*, 2005.

7.1 Introduction

This chapter is intended as a primer for ray based global illumination as well as a survey of previous work in the area. Several of the algorithms covered will be presented in tutorial form, with an eye on bringing the reader up to speed with the issues related to the original research that will be presented in chapters 8 and 9. This chapter will not cover view-independent or finite element techniques such as traditional radiosity or precomputed radiance transfer.

The goal. The goal of global illumination is a physically accurate simulation of the light transport leading to a photograph. In other words, given a scene description and the emission characteristics of light sources, global illumination seeks to snap a "synthetic photograph", following light paths through the scene, and recording the intensity of light striking a virtual film plane.

In practice, global illumination renderers must simplify the physics of light transport somewhat to make computations practical. The particle model of light is assumed, so

wave effects like diffraction and interference cannot be simulated. In addition, all visible wavelengths are often conglomerated into just three, *red*, *green* and *blue*. Other simplifictions include the assumptions that the speed of light is infinite and that all light is randomly polarized, as well as the assumptions that time delay of photon re-emission (phosphorescence) and frequency change during light scattering (flourescence) do not occur. Despite these compromises, global illumination can produce images that are visually difficult to distinguish from photographs and predictive of real world illumination.

An example scene. Throughout this chapter we will use the "three spheres" example scene shown in figure 7.1 to demonstrate the kinds of results achieved by different rendering algorithms. The scene contains diffuse, specular and transparent surfaces, and is designed to highlight artifacts that are typical of the different rendering algorithms.

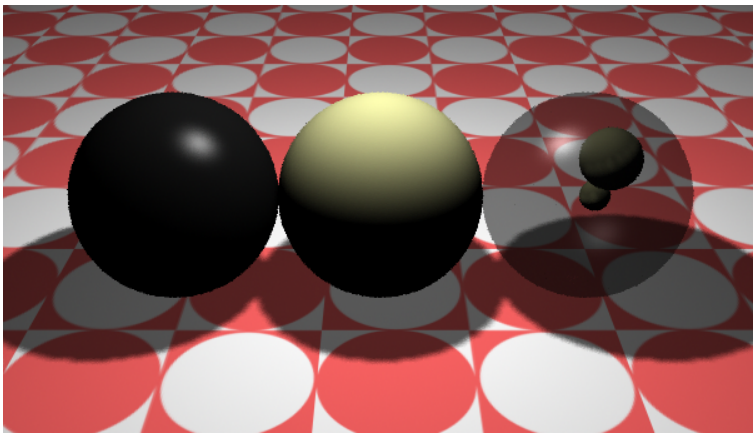


Figure 7.1: "Three spheres" example scene. The left sphere is a mirror ball, the middle sphere is diffuse and the right sphere is made of glass. Inside the right sphere are two smaller, diffuse spheres. The rendering above was not made using global illumination, so global effects such as correct shadows, reflection, refraction and indirect lighting are not present.

7.2 An Excursion: Brute Force Global Illumination

Since global illumination is a physical simulation of light transport, perhaps the simplest and most obvious global illumination algorithm is the direct simulation of individual photons. The idea behind this scheme is to generate all photons that are emitted from the

Brute Force Global Illumination

```
Clear the image.
For i = 1 to the number of photons emitted while the camera shutter is open
    Create a photon.
        Pick a wavelength, emission point and initial direction for the photon.
    While the photon is not absorbed
        Find the intersection of the photon with the closest surface.
        If the intersection lies on the camera lens
            Project the photon through the lens onto the image plane.
            Increment the photon counter for the resulting pixel.
            Absorb the photon.
        Else
            Determine whether to absorb the photon based on the surface reflectance (BSDF).
            If the photon was not absorbed
                Determine a new direction for the photon using the surface reflectance.
```

Figure 7.2: Brute force global illumination algorithm by direct simulation of photons.

light sources while the camera shutter is open, and follow their trajectories until they either strike the camera lens or are absorbed by the environment. Photons that do hit the lens are projected onto a virtual image plane, contributing to the resulting image. The appeal of this method is its simplicity. The connection to the physical world is obvious, and all mathematical concerns reduce to ray casting and the simulation of light scattering events. Figure 7.2 gives pseudocode for global illumination by direct photon simulation.

Despite being simple and theoretically complete, direct simulation of photons for global illumination is impractical because of the large number of photons involved. A simple back of the envelope calculation will verify this fact. Consider the number of visible light photons emitted by a 100 watt light bulb in a typical camera exposure of $1/60^{th}$ of a second. A watt is a Joule per second, so the light bulb uses 100 Joules of energy per second. An incandescent bulb is only about 10% efficient, however, so the amount of visible light emitted by a 100 watt bulb in a sixtieth of a second exposure is roughly:

$$100\frac{Joules\ power\ input}{second} \times 0.1\frac{visible\ light}{power\ input} \times 0.0167\ \frac{seconds}{exposure} = 0.167\frac{Joules\ visible\ light}{exposure}.$$

Now we need to determine how many visible photons are needed to make up 0.167 Joules of energy. Planck determined that the energy of a photon is equal to its frequency times Planck's constant. Planck's constant is 6.262×10^{-34} *Joule seconds*, and a typical photon in the middle of the visible spectrum has a frequency of $\nu = 6 \times 10^{14}/second$, so a single visible photon contains about

$$6.262 \times 10^{-34} Joule\ seconds \times \frac{6 \times 10^{14}}{second} = 3.76 \times 10^{-19} Joules$$

of energy, and the hypothetical 100 watt light bulb emits

$$\frac{0.0167\ Joules}{3.76 \times 10^{-19}\ Joules} = 4.44 \times 10^{17}$$

photons per exposure. A typical modern PC can simulate on the order of a few billion photons per day, so a single 100 watt light bulb emits roughly a hundred million times more photons during a single shutter click than can be simulated on a PC in a 24 hour period.

Of course, the same image can be achieved on average by simulating fewer photons with higher energy. Figure 7.3 shows an image produced by simulating 1 billion photons

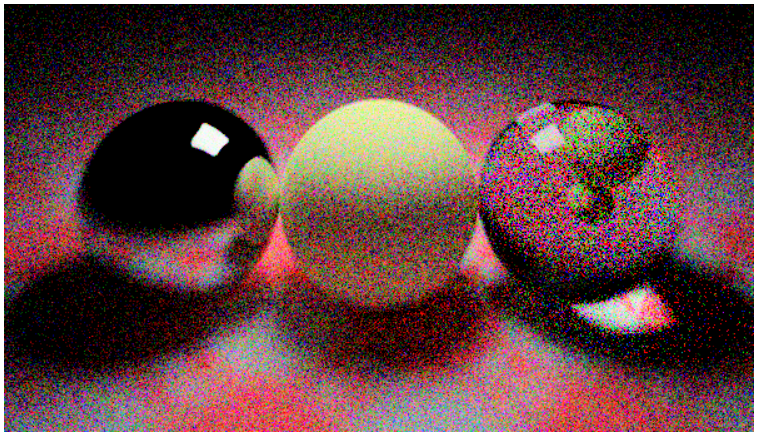


Figure 7.3: Brute force global illumination by direct simulation of photons.

in the three spheres example scene. Note the extremely shallow depth of field. This is a consequence of the large lens aperature needed to "catch" enough photons to make a coherent image. Even with the large lens size, less than half a percent of the photons hit the lens, and fewer still project to the within the bounds of the image. Besides the narrow depth of field, the image suffers from a large amount of color noise that results from simulating different wavelengths separately. This is consistent with the kinds of artifacts seen in real photographs taken under very low light conditions. On the plus side, however, all of the major global illumination effects are trivially achieved, and the image displays constant quality throughout, although at least a hundred times as many samples would likely be needed to produce an artifact free image.

7.3 Mathematical Framework for Global Illumination

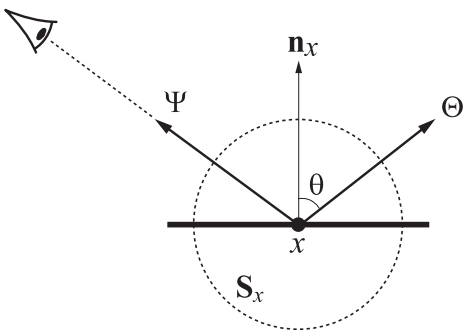
7.3.1 What does Global Illumination Measure?

Because of the computational expense of simulating individual photons, almost all global illumination algorithms treat light as a continuous energy flow rather than a set of discrete photons. In this setting, the task of global illumination becomes to calculate the flow rate of light energy through each pixel on the virtual image plane. The specific radiometric quantity that is measured is called *radiance*, which is defined formally as the flow of light energy per unit time per unit projected area per unit solid angle. Intuitively, radiance can be thought of as the amount of light energy arriving at a particular point from a given direction per unit time. Most rendering texts include extensive derivations of radiance and other radiometric quantities related to global illumination (e.g. [Jensen, 2001; Dutré *et al.*, 2003; Pharr and Humphreys, 2004]). Luckily, however, we can ignore these issues for the most part–the emission characteristics of the light sources are specified in units of radiance, and these units propagate through most global illumination algorithms unchanged.

The conservation of radiance. An important property of radiance is that it remains constant along lines of sight through empty space. This fact may seem counter-intuitive at first, since, for example, the amount of light reaching a point from the sun varies as the inverse square of the distance. However, the reduction in incoming light is actually due to a reduction in the solid angle that the sun covers in the sky rather than a change in the radiance (the sun looks just as bright, but smaller as you move away from it). The upshot of this "conservation of radiance" along sight lines is that global illumination algorithms only need to calculate the radiance at surface points visible to the camera, rather than at all points in space.

7.3.2 The Rendering Equation

The equation that describes the radiance leaving a surface point in a particular direction is called the "rendering equation". Simultaneously introduced to the graphics community by [Kajiya] and [Immel *et al.*] in 1986, the rendering equation relates the total radiance leaving a point to the emission characteristics of the point, the incident radiance at the point, and the reflectance properties of the surface, called the BSDF. Equation 7.1 gives one common form of the rendering equation:



$$L(x \rightarrow \Psi) = L_e(x \rightarrow \Psi) + \int_{\mathbf{S}_x} L(x \leftarrow -\Theta)\ f_s(\Psi \leftrightarrow x \leftrightarrow \Theta)\ |\cos\theta|\ d\Theta. \qquad (7.1)$$

Note the similarity between equation 7.1 and the direct lighting equation defined in chapter 4. The terms of the rendering equation are defined similarly as well:

x	A point on a surface.
Ψ	The direction for which we want to compute the radiance.
$\mathbf{n}_x$	The surface normal at point x.
$\mathbf{S}_x$	The sphere of directions surrounding point x.
Θ	A direction vector in $\mathbf{S}_x$.
θ	The angle between Θ and $\mathbf{n}_x$.
$L(x \rightarrow \Psi)$	The radiance that leaves point x in direction Ψ.
$L_e(x \rightarrow \Psi)$	The radiance emitted from point x in direction Ψ.
$L(x \leftarrow -\Theta)$	The incoming radiance that strikes point x from direction $-\Theta$.
$f_s(\Psi \leftrightarrow x \leftrightarrow \Theta)$	The BSDF function which defines the light scattering properties of the surface at point x.

The most obvious difference between the rendering equation and the direct lighting equation is the use of the BSDF, f_s, rather than the BRDF, f_r. The BSDF, or *bidirectional scattering distribution function*, is a generalization of the BRDF that allows light scattering to occur by transmission as well as reflection. By using the BSDF, the rendering equation can account for transparent surfaces as well as opaque ones, but it requires the integration domain to be defined over the whole sphere around point x rather than just the hemisphere above it.

A more subtle, but problematic difference is that in the rendering equation the emitted radiance arriving at point x, $L_e(x \leftarrow -\Theta)$, is replaced by the total radiance, $L(x \leftarrow -\Theta)$. Consequently, L appears on both sides of the equation. Interpreted literally, the presence of L on both sides means that to determine the radiance leaving point x in a particular di-

rection, one must know the radiance arriving at x from all directions. These radiances, in turn, must be calculated, and so on, leading to an infinite regression of evaluations. Mathematically, this infinite regression is equivalent to a *Neumann expansion* of the rendering equation, which successively replaces L on the right hand side with what it is equal to, yielding an infinite series of terms, one for each bounce:

$$
\begin{aligned}
L(x \rightarrow \Psi) = {} & L_e(x \rightarrow \Psi) \\
& + \int_{\mathbf{S}_x} L_e(x \leftarrow -\Theta)\ f_s(\Psi \leftrightarrow x \leftrightarrow \Theta)\ |\cos\theta|\ d\Theta \\
& + \int_{\mathbf{S}_x} \int_{\mathbf{S}_{x'}} L_e(x' \leftarrow -\Theta') f_s(-\Theta \leftrightarrow x' \leftrightarrow \Theta') |\cos\theta'| d\Theta'\ \ f_s(\Psi \leftrightarrow x \leftrightarrow \Theta) |\cos\theta| d\Theta \\
& + \int_{\mathbf{S}_x} \int_{\mathbf{S}_{x'}} \int_{\mathbf{S}_{x''}} L_e(x'' \leftarrow -\Theta'') \cdots \\
& \vdots
\end{aligned}
$$

The first two terms of the Neumann expansion describe the radiance resulting from zero or one bounces from point x, or in other words, the direct lighting. The third term accounts for light that has bounced twice, and so on. Despite the fact that the Neumann expansion has an infinite number of terms, in practice only the first few are significant. The total light power diminishes with each reflection, and becomes insignificant after a few bounces, although the exact number of bounces that are significant depends on the scene and desired accuracy of the simulation.

7.3.3 Pixel Integrals and the Measurement Equation

Before delving into specific global illumination algorithms, we will formalize an idea that has been hinted at throughout the dissertation–namely, that pixel values in global illumination images are actually high dimensional integrals. To put it in more concrete terms, a pixel value in a global illumination image records the output of an idealized light sensor placed somewhere in the scene. The response of the pixel sensor varies over its surface and with respect to incoming direction, resulting in an integral called the *measurement equation*, which can be written as follows:

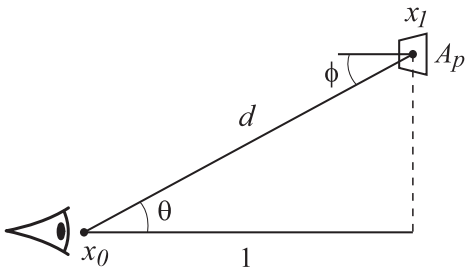
$$M_j = \int_A \int_{\mathbf{S}_{x_0}} W_e(x_0 \rightarrow \Theta)\, L(x_0 \leftarrow -\Theta)\, d\Theta\, dx_0, \tag{7.2}$$

where M_j is the output of the sensor, A is the area of the sensor, $\mathbf{S}_{x_0}$ is the sphere of directions, and $W_e(x_0 \rightarrow \Theta)$, called the *importance* or *weight emitted*, is the sensor response at position x_0 in direction Θ. W_e is usually normalized so that its integral is equal to one. That way, M_j becomes a weighted average of the radiance striking the sensor. M_j can also be integrated over time to capture time varying effects such as motion blur and lighting changes during a camera exposure.

The W_e term for a pinhole camera. Here we derive the W_e term for a pixel in a standard pinhole camera. As previously stated, we proceed from the assumption that W_e should integrate to 1 over the sensor domain. Since the sensor location is a single point, we can ignore the sensor area[1] and just worry about the angular extent sampled by the pixel. We will also assume a box filter, so that W_e will be constant over the pixel's area. Under these assumptions, the importance for a point x_1 in pixel P should be $1/\alpha$, where α is the solid

[1] The sensor response at the eye point is actually infinite (a delta function), but we can ignore this fact since the infinite value will cancel out when integrated over the zero surface area of the eye point.

angle subtended by the pixel. We can determine α by calculating the area of the pixel and then converting the area to solid angle. The figure below shows the geometric setup for the conversion:



Assuming that the image plane is located one unit away from the eye point, the area of pixel P, A_p, is given by:

$$A_p = \frac{4}{wh} \tan\frac{\theta_h}{2} \tan\frac{\theta_v}{2},$$

where w and h are the width and height of the image in pixels, and θ_h and θ_v are the horizontal and vertical field of view angles of the camera. Based on equation 4.6, we derive the formula $\alpha = A_p \cos\phi / d^2$, where, once again, α is the resulting solid angle, and other variables are as in the above figure. Applying this formula to W_e yields

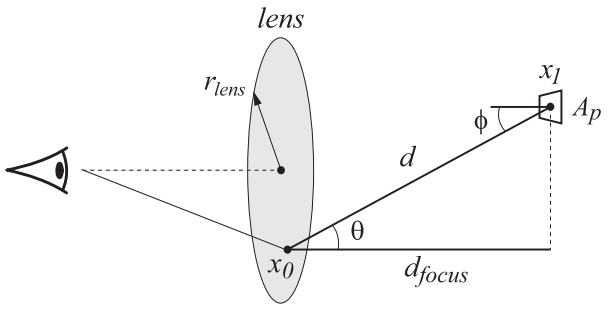
$$W_e(x_0 \rightarrow x_1) = \frac{1}{\alpha} = \frac{d^2}{A_p \cos\phi}.$$

Making two other substitutions: $d = 1/\cos\theta$, and $\phi = \theta$ gives us the final value:

$$W_e(x_0 \rightarrow x_1) = \frac{d^2}{A_p \cos\phi} = \frac{1}{A_p \cos^3\theta} = \frac{wh}{4 \tan\frac{\theta_h}{2} \tan\frac{\theta_v}{2} \cos^3\theta}. \tag{7.3}$$

The W_e term for a thin lens camera. The W_e term for a thin lens camera can be derived similar to the pinhole case. For this case, it is easiest to think of the lens itself as being the light sensor, with each pixel being sensitive to different incoming directions. We assume

that each part of the lens is equally light sensitive, so that one of the terms that make up W_e will be $1/A_{lens}$, where $A_{lens} = \pi r_{lens}^2$ is the lens area. This is shown in the figure below:



Following the same derivation for the direction term as in the pinhole case yields the importance for a point x_0 on the lens through point x_1 on the focus plane:

$$W_e(x_0 \rightarrow x_1) = \frac{wh}{4\, A_{lens}\, \tan\frac{\theta_h}{2}\, \tan\frac{\theta_v}{2}\, \cos^3\theta}. \tag{7.4}$$

7.4 Path Tracing

Path tracing [Kajiya, 1986] was the first ray based global illumination algorithm published in the graphics literature, and it is a direct application of Monte Carlo integration to the measurement equation. A path tracer samples the incoming radiance on a pixel by means of *light paths*. By light path, we mean a ray path that connects a light source to the eye point through a number of scattering events (reflections or transmissions).

7.4.1 Sampling Light Paths in a Path Tracer

A path tracer generates light paths by casting rays from the eye point into the scene. These *eye subpaths* (also called *lens subpaths*) are then allowed to bounce around in the scene according to some probability distibution (PDF). The path tracer can make a completed light path from an eye subpath in one of two ways. First, an eye subpath may just happen

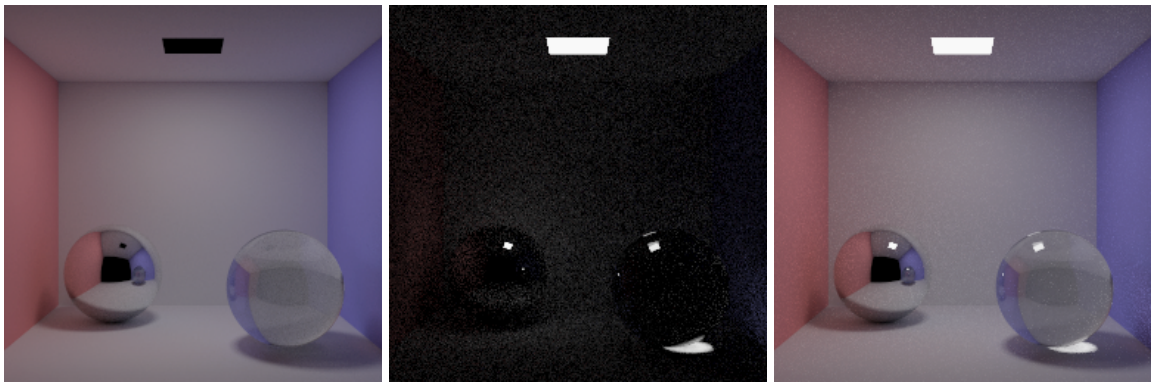


Figure 7.4: Explicit and implicit light paths in a typical path tracer. Explicit light paths (left) are used for direct lighting situations, while implicit light paths (middle) are able to sample caustics and reflections of light sources more efficiently. The final image (right) combines samples from both path types.

to hit a light source. We call this kind of light path an *implicit light path*. A second kind of light path is created by connecting the end of an eye subpath directly to a point on a light source. We call these *explicit light paths*. (See figures 7.5 and 7.6.) Although a theoretically complete path tracer can be made using just implicit or just explicit light paths, typical path tracers leverage the strengths of both these path types. For example, explicit light paths are often used to compute direct lighting, whereas implicit light paths are more suited to computing caustics and reflections of light sources. Figure 7.4 demonstrates the roles of implicit and explicit light paths in a typical path tracer.

Rather than explicitly dealing with the W_e term of the measurement equation, most path tracers simply average the radiance estimates from a number of light paths for each pixel. This works because the light paths are created starting at the camera, which allows the renderer to choose initial sampling directions proportionally to the W_e term. Later, we will see that the W_e term must be considered for algorithms such as light tracing that grow light paths from the light sources.

7.4.2 Evaluating Light Paths

When a path tracer evaluates a light path, what is actually being computed is a Monte Carlo estimate of the radiance flowing to the eye point along the first leg of the path, $L(x_1 \rightarrow x_0)$. This estimate, which we will denote $\widehat{L}(x_1 \rightarrow x_0)$, must stand as a proxy for the contributions of all light sources and possible scattering events, even though only one light source and one set of scattering events has been sampled. To put it in statistical terms, the *expected value* of a light path must be equal to the radiance along the first leg of the path:

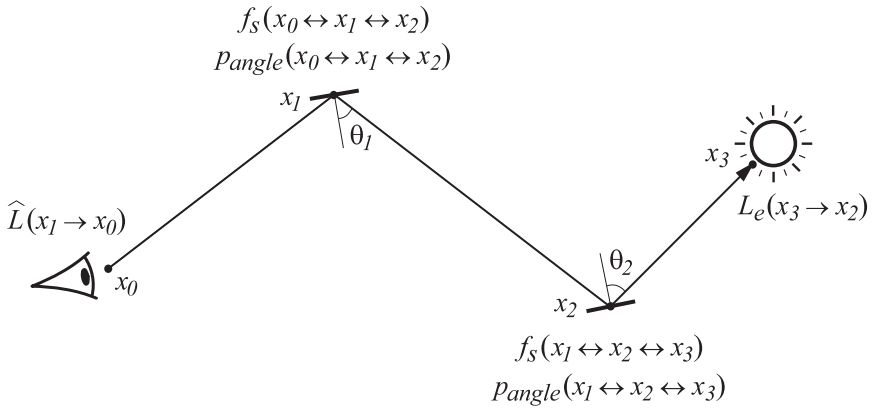
$$E[\,\widehat{L}(x_1 \rightarrow x_0)\,] \;=\; L(x_1 \rightarrow x_0).$$

A path tracer achieves this feat by cleverly combining the light scattering properties of surfaces in the scene (BSDFs) and the sampling distributions used to choose directions in path space (PDFs). To illustrate how this is done, we give examples for both implicit and explicit light paths.

7.4.3 Evaluating Implicit Light Paths

A path tracer evaluates an implicit light path by multiplying the radiance of the light source (L_e) by the product of the BSDF terms $(f_s \cos\theta)$ evaluated at the interior vertices of the path, and dividing by the probability with respect to solid angle (p_{angle}) that the ray directions in the path were chosen by the sampling procedure. To obtain a proper estimate, $\widehat{L}$ must also be divided by the probability that the sampling procedure chose to create a path of the given length, p_{length} (section 7.4.5 describes how to calculate p_{length}). Figure 7.5 shows the evaluation of an example implicit light path.

Choosing sampling distributions that cancel with the BSDF term. Often, it is possible to choose a sampling distribution such that p_{angle} cancels out the non-constant parts of the BSDF term, $f_s \cos\theta$. This is nothing more than importance sampling the BSDF function,



$$\widehat{L}(x_1 \rightarrow x_0) = \frac{f_s(x_0 \leftrightarrow x_1 \leftrightarrow x_2)\ \cos\theta_1}{p_{angle}(x_0 \leftrightarrow x_1 \leftrightarrow x_2)} \times \frac{f_s(x_1 \leftrightarrow x_2 \leftrightarrow x_3)\ \cos\theta_2}{p_{angle}(x_1 \leftrightarrow x_2 \leftrightarrow x_3)} \times \frac{L_e(x_3 \rightarrow x_2)}{p_{length}}$$

Figure 7.5: An *implicit light path* is created when the sampling procedure is lucky enough to hit a light source. The term p_{length} is the probability that the sampling procedure chose to create a path of the given length. This term is needed because $L(x_1 \rightarrow x_0)$ describes all the light flowing to the eye point along $x_1 \rightarrow x_0$ through any number of scattering events. Since the light path only samples one set of scattering events, its value must be "pumped up" by dividing by p_{length}.

which we have already seen in the context of direct lighting in chapter 4. As a concrete example of how the sampling works, consider the case of an ideal diffuse surface. The value of f_s for an ideal diffuse surface is ρ/π, where ρ is the surface reflectance (percentage of incoming light reflected from the surface). If sampling directions are chosen according to a cosine-weighted distribution about the surface normal, then the value of p_{angle} is $\cos\theta/\pi$, and the ratio $(f_s \cos\theta / p_{angle})$ becomes:

$$\frac{(\rho/\pi)\cos\theta}{\cos\theta/\pi} = \rho.$$

Cancelling can occur for an ideal specular surface (mirror) as well. In the ideal specular case, f_s is a delta function centered on the reflected direction divided by a cosine term: $f_s = \rho\ \delta_R/\cos\theta$, and the only sampling distribution that makes sense is another delta func-

tion centered on the reflected direction: $p_{angle} = \delta_R$. The delta functions cancel out in the ratio, once again leaving the surface reflectance:

$$\frac{(\rho\ \delta_R/\cos\theta)\ cos\theta}{\delta_R} = \rho.$$

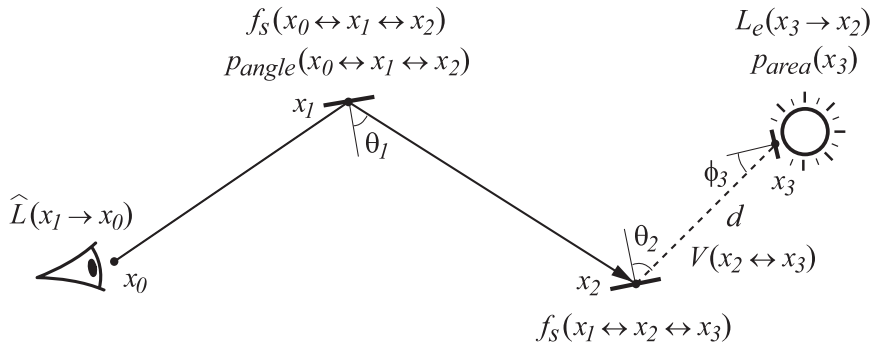
7.4.4 Evaluating Explicit Light Paths

In an explicit light path, the renderer connects the eye subpath directly to a point on a light source that the sampler has chosen probabilistically with respect to area, rather than solid angle. Consequently, the path evaluation must convert probability with respect to area, p_{area}, to probability with respect to angle, and account for the visibility of the chosen point from the last node in the eye subpath. Applying equation 4.6 to perform the conversion, we evaluate explicit light paths as shown in the example in figure 7.6.

An intuitive way to think about explicit light paths is to imagine that the light source has been probabilistically concentrated into a single point. This interpretation is particularly appealing when one considers points chosen uniformly with respect to area on the light source. When points are chosen uniformly, p_{area} becomes $1/A$, where A is the area of the light source, so dividing by the probability actually multiplies by the area, "squeezing" all of the radiance of the light source into a single point.

7.4.5 Light Paths in a Working Path Tracer

Terminating eye subpaths. A practical issue that comes up in a path tracer is how to terminate eye subpaths. Light can bounce infinitely, but the computer can only simulate a finite number of bounces. One idea is to cap eye subpaths at some maximum length. This works fairly well in practice, but it introduces bias into the solution because light that bounces more than the maximum number of times is never counted. A second solution that gets rid of the bias is to use *Russian roulette* to terminate eye subpaths. Russian roulette



$$\widehat{L}(x_1 \rightarrow x_0) = \frac{f_s(x_0 \leftrightarrow x_1 \leftrightarrow x_2)\ \cos\theta_1}{p_{angle}(x_0 \leftrightarrow x_1 \leftrightarrow x_2)} \times \frac{V(x_2 \leftrightarrow x_3)\ f_s(x_1 \leftrightarrow x_2 \leftrightarrow x_3)\ \cos\theta_2}{d^2 p_{area}(x_3)/\cos\phi_3} \times \frac{L_e(x_3 \rightarrow x_2)}{p_{length}}$$

Figure 7.6: An *explicit light path* is created by connecting the end of an eye subpath directly to a point on a light source. The term $V(x_2 \leftrightarrow x_3)$ is the visibility between points x_2 and x_3, d is the distance between x_2 and x_3, and ϕ_3 is the angle between the surface normal at x_3 and the direction $x_3 \rightarrow x_2$. Note that p_{area} must account for all of the light sources in the scene.

works by defining a probability of terminating the eye subpath at each bounce. In this case, p_{length} becomes the product of the Russian roulette probabilities for all bounces in the path. If the bounce directions are sampled according to the BSDF of the surface, Russian roulette can cancel out the surface reflectance term that results from the sampling by choosing the reflectance (ρ) as the probability of extending the path at each bounce.

Creating multiple paths from a single eye subpath. For efficiency reasons a path tracer usually makes multiple light paths from a single eye subpath. At each intersection point in the path, the sampling procedure decides whether the eye subpath will be considered an implicit light path, and whether explicit light paths should be created by connecting to light sources. A group of light paths created in this way serves as a better estimate of the radiance than a single light path could.

Multiple importance sampling of different light path types. Explicit and implicit light path types can be combined by multiple importance sampling. To do this, the path tracer treats the connection of the path to the light source as a direct lighting situation and evaluates p_{angle} as described in section 4.4.3. Combining the path types through multiple importance sampling has two benefits, it improves the PDFs used to evalute the light paths and simplifies the main loop of the path tracer, since implicit and explicit paths do not need to be treated as distinct entities.

7.4.6 Typical Path Tracing Results

Figure 7.7 shows the three spheres scene path traced with 64 samples per pixel. While this image is much better than the brute force image shown in figure 7.3, there are still a number of artifacts. The most obvious issue is the bright speckles that appear throughout the image. These speckles are caused by the infrequent inclusion of very bright radiance estimates that occur when the amount of light flowing along a path is large, but the probability that the sampler creates the path is small. This can occur, for example, when the light path reflects or refracts from a specular surface and directly hits a light source. Since the sampler does not know which directions will reflect to the light sources, it cannot sample them with a high probability, leading to variance.

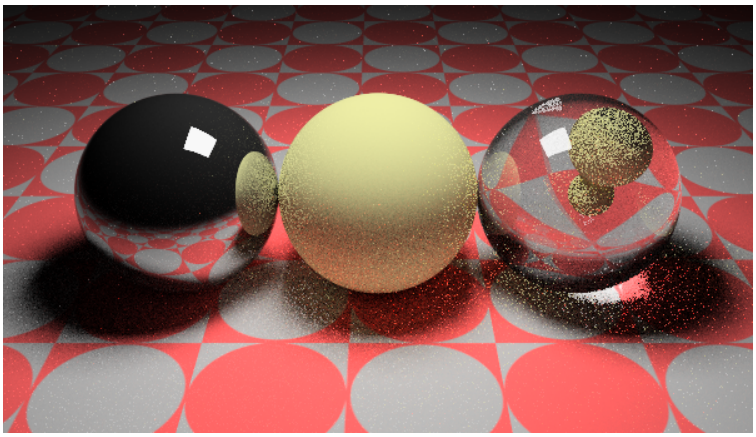


Figure 7.7: Path tracing.

7.5 Irradiance and Radiance Caching

From the outset it was clear that standard path tracing was too time consuming to be practical. A single rendering can require thousands of samples per pixel and days of CPU time to produce a noise free image. For this reason, a lot of research has focused on speeding up global illumination through caching and interpolation schemes. Ward's *Irradiance caching* algorithm [Ward *et al.*, 1988] was one of the earliest and most successful algorithms of this type.

As a motivation for his irradiance caching algorithm, Ward noted that most of the visible noise in path traced images stems from indirect lighting on diffuse surfaces, even though the true indirect lighting usually varies smoothly. Compared to the indirect lighting, the amount of visible noise in specular reflections and direct lighting is relatively small. Based on this observation, Ward factored the rendering equation into two pieces, with one term for direct lighting and specular reflection and a second term for indirect diffuse lighting. At render time, the specular reflection/direct lighting term is evaluated normally. The indirect diffuse term, on the other hand, is calculated using a large number of ray samples to reduce variance, but it is only evaluated at a few points in space and interpolated to save time.

7.5.1 Irradiance and Radiance

Instead of storing all of the ray samples used to calculate the indirect difuse term, irradiance caching takes advantage of a property of ideal diffuse surfaces that allows the total incident light on the surface (*irradiance*) to be cached rather than the angular distribution of incident light (*radiance*). To see how this works, let's look at the definition of irradiance. Irradiance, normally written as E, is defined as the total radiant flux arriving on a surface per unit area, and it can be written as an integral of the radiance over the hemisphere:

$$E(x) = \int_{\Omega_x} L(x \leftarrow \Theta) \cos\theta \, d\Theta, \tag{7.5}$$

where once again, x is the point for which we want to determine the irradiance and Ω_x is the hemisphere above x. Assuming an opaque surface, the outgoing radiance from x is also an integral defined over the hemisphere: $L(x \rightarrow \Psi) = \int_{\Omega_x} f_r(\Psi \leftrightarrow x \leftrightarrow \Theta) L(x \leftarrow \Theta) \cos\theta d\Theta$. For a diffuse surface, however, the BRDF term is the constant $\frac{\rho}{\pi}$, so it can be factored out of the integral, leaving the BRDF times the irradiance: $L(x \rightarrow \Psi) = \frac{\rho}{\pi} E(x)$. Thus, the irradiance provides enough information to calculate the outgoing radiance for a diffuse surface.

7.5.2 Computing and Reusing Irradiance Samples

At render time, the irradiance caching algorithm creates irradiance samples as-needed, and stores them in an octree data structure for easy spatial lookup. Before shading a point, the renderer searches for nearby irradiance samples in the octree. It then determines if the samples are suitable for interpolation based on a heuristic called "irradiance gradients" [Ward and Heckbert, 1992], which includes terms for scene geometry, surface normal and the distance from the point being shaded to the cached sample. If one or more suitable irradiance samples are found, the irradiance at the point being shaded is approximated with a weighted average of the existing samples. Otherwise, the algorithm computes a new irradiance sample at the point, uses it to shade the point, and stores it in the octree.

7.5.3 Limitations of Irradiance Caching

Irradiance caching can work well for scenes with mostly diffuse surfaces, but it breaks down in some situations because irradiance gradients do not account for abrupt changes in incoming light. For example, figure 7.8 demonstrates the results of applying irradiance caching to the three spheres test scene. The indirect diffuse lighting is smooth in most places due to the interpolation, but there are some obvious artifacts. First, because the algorithm evaluates the irradiance samples lazily, there are some shading discontinuities that appear when new samples are added to the cache. Another problem is the caustic

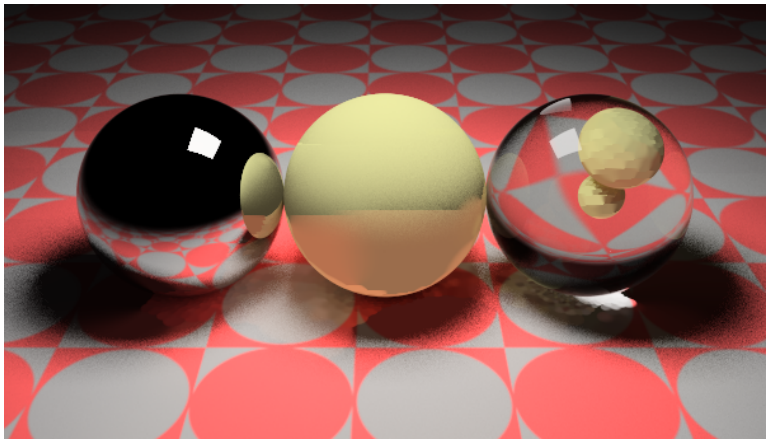


Figure 7.8: Irradiance caching.

lighting below the glass ball. Since the irradiance gradients heuristic does not account for specular interactions, the algorithm undersamples the caustic region, missing the right half of the caustic entirely.

7.5.4 Radiance Caching

As originally defined, irradiance caching could only render diffuse surfaces. Radiance caching, on the other hand, retains some directional information to allow for glossy surfaces. Tabellion and Lamorlette [2004] agregate the radiance into three incoming directions, one for red, green and blue, whereas Křivánek *et al.* [2006] take a more rigorous approach, storing the incoming radiance as spherical harmonics. In both cases, rendering proceeds in the same manner as irradiance caching; radiance values are evaluated as needed and interpolated whenever possible.

7.6 Light Tracing

So far we have only discussed algorithms that solve the rendering equation starting at the eye point. Along with these methods, there are a number of algorithms that estimate light transport by shooting out packets of energy called "particles" from the light sources. Particles are similar in many ways to photons, but usually a particle encapsulates a large number of photons that are treated as a unit for simulation purposes.

The most straightforward of the particle tracing methods is called *Light tracing* [Dutré and Willems, 1994]. In a light tracer, light paths are created by following the trajectories of particles from the light sources through the scene. In this sense, light tracing is very similar to the brute force global illumination algorithm presented earlier in the chapter, except that light tracing attempts to sample light flows rather than directly simulate photons.

7.6.1 Light Paths in Light Tracing

To make a light path, a light tracer emits a particle from a light source, choosing an emission point on a light source and an initial direction. The particle is then allowed to bounce a number of times in the scene, creating a ray path called the *light subpath*. After each bounce, a light tracer completes a light path by connecting the growing subpath to a point on the camera lens (or the eye point for a pinhole camera).

7.6.2 Projecting Points onto the Image Plane

When a point in the scene is connected directly to the camera lens or eye point to produce a light path, the renderer must determine which image pixel the light path contributes to. In other words, for a point in world space $P = [\ x\ y\ z\ 1\ ]^T$, the renderer must find its pixel

coordinates in screen space. If the camera uses a pinhole model, P can be projected directly onto the image plane with the following projection matrix:

$$M = \begin{pmatrix} \frac{w}{2\tan(\theta_h/2)} & 0 & -w/2 & 0 \\ 0 & \frac{h}{2\tan(\theta_v/2)} & -h/2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ -n_x & -n_y & -n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where

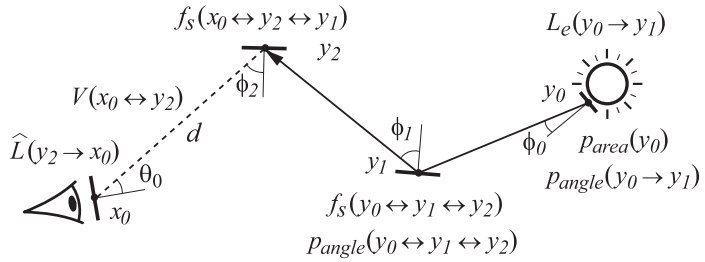
$C = [c_x\ c_y\ c_z\ 1]^T$ is the camera's center of projection (eye point),
$N = [n_x\ n_y\ n_z]^T$ is the direction that the camera is looking,
$U = [u_x\ u_y\ u_z]^T$ is the horizontal axis of the camera,
$V = [v_x\ v_y\ v_z]^T$ is the vertical axis or "view up" vector of the camera,
θ_h and θ_v are the horizontal and vertical field of view angles, and
w and h are the width and height of the image in pixels.

If the camera uses a lens model, the projected pixel location of point P also depends on the point on the lens though which the light passes. To find the pixel coordinates in the presence of a lens model, therefore, we must first choose a point Q on the surface of the lens. We can then solve for the intersection of the ray $Q \rightarrow P$ on the plane of perfect focus and project the resulting point, P_f, onto the image plane using matrix M.

7.6.3 Evaluating Light Paths in Light Tracing

To evaluate a light path, a light tracer must both determine the amount of light flowing along the path and convert the sampling rate in world space to a corresponding rate in image space using the importance term, W_e. Figure 7.9 gives an example of this calculation. Note that in light tracing, samples may contribute to any image pixel, so $\widehat{L}$ should be divided by the total

number of particles that are traced from the light sources rather than the number of samples "per pixel". To account for this effect, we divide W_e by the number of image pixels (wh) in figure 7.9. Figure 7.10 shows a typical edge darkening artifact that results from failing to convert the sampling rate from world to screen space.



$$\widehat{L}(y_2 \rightarrow x_0) = \frac{W_e(x_0 \rightarrow y_2)}{wh} \times \frac{f_s(x_0 \leftrightarrow y_2 \leftrightarrow y_1)\, V(x_0 \leftrightarrow y_2)\, \cos\phi_2}{d^2} \times \frac{f_s(y_0 \leftrightarrow y_1 \leftrightarrow y_2) \cos\phi_1}{p_{angle}(y_0 \leftrightarrow y_1 \leftrightarrow y_2)}$$

$$\times \frac{L_e(y_0 \rightarrow y_1) \cos\phi_0}{p_{length}\, p_{area}(y_0)\, p_{angle}(y_0 \rightarrow y_1)}$$

Figure 7.9: A light path created by connecting a light subpath directly to the eye point.

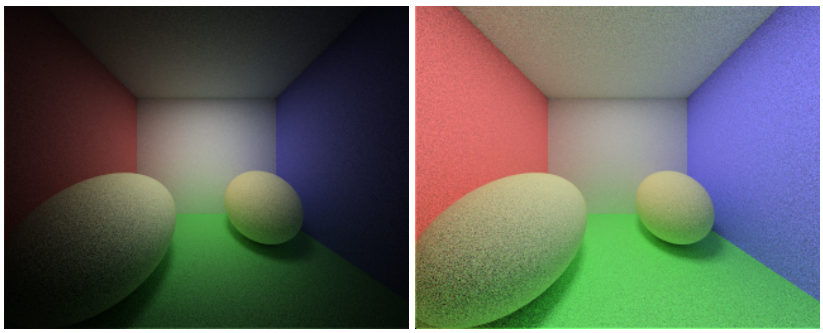


Figure 7.10: The importance term in light tracing. Failing to multiply by W_e (left) results in darkening away from the center of the image. Multiplying by W_e (right) compensates for lower sampling density near the image edges.

7.6.4 Benefits and Drawbacks of Starting at the Light Sources

There are several benefits to sampling from the light sources instead of the eye point. First, starting at the lights has the aesthetic appeal of following light in the direction that it travels. Additionally, some parts of path space are better sampled. For example, consider the light traced image of the three spheres scene in figure 7.11. The caustic below the right ball is much smoother than in the path tracing case. Most of the image has worse quality than path tracing, however. Furthermore, the reflections and refractions from the mirror and glass balls are missing entirely. This is a consequence of connecting the light subpaths directly to the eye point. Since the materials of these spheres are specular, the probability that the specular directions will be sampled by connecting to the eye point is zero. (The dark gray tint comes from a small diffuse term.)

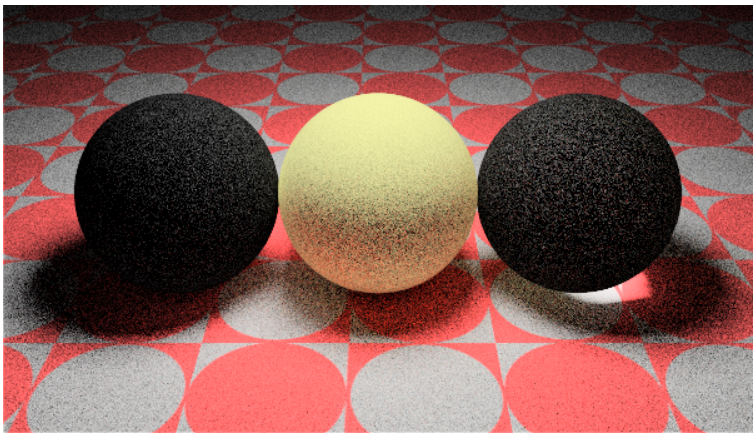


Figure 7.11: Light tracing.

7.7 Photon Mapping

Photon mapping [Jensen, 1996; Jensen, 2001] is probably the most successful global illumination algorithm to date. Although biased, the algorithm is able to produce near photographic images in a reasonable amount of time. It can also elegantly handle rendering tasks that were very difficult before it was introduced, such as subsurface scattering and the rendering of participating media.

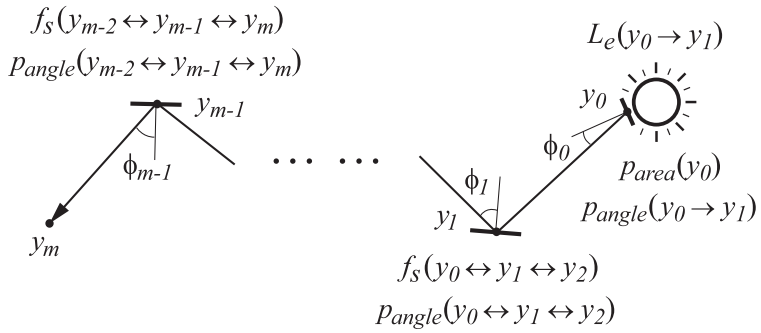
Photon mapping solves the global illumination problem in two passes. In the first pass, the renderer shoots out photons (particles) from the light sources and stores hit records in a *photon map* data structure where the photons strike objects in the scene. In the second pass, ray tracing from the eye is used to generate a displayable image. During the ray tracing pass, the renderer queries the photon map to estimate indirect illumination, making the process efficient.

7.7.1 Creating the Photon Map

As mentioned, the first stage of photon mapping traces photon particles from the light sources into the scene. Simulating a photon's trajectory is done by generating a light subpath in precisely the same manner as in light tracing. An emission point on a light source is chosen and a photon particle is shot out from the emission point in a randomly chosen direction. The photon then bounces around in the scene, and as it bounces photon hit records are generated for the (non-specular) intersection points along the growing path. The renderer stores the photon hit records in a k-d tree structure called a *photon map* to permit later spatial lookup.

Photon power. One interpretation of the photon map is that each photon represents some fraction of the power emitted by the light sources, and the photon hit records represent the incident illumination on surfaces in the scene. The power assigned to a given hit record, Φ_i, follows directly from the probability with which the photon's path was generated and the emissive characteristics of the light sources. Figure 7.12 gives the calculation of Φ_i in the general case.

Jensen [2001] points out that ideally, the photon hit records should all contain equal power. This can be achieved by (1) choosing an emission point and inital direction proportional to the emissive properties of the light sources, (2) choosing reflected directions proportionally to the the BSDF term and (3) setting $p_{roulette}$ proportional to the surface re-



$$\Phi_i = \frac{L_e(y_0 \rightarrow y_1)|\cos\phi_0|}{p_{area}(y_0)\ p_{angle}(y_0 \rightarrow y_1)} \times \prod_{j=1}^{m-1} \frac{f_s(y_{j-1} \leftrightarrow y_j \leftrightarrow y_{j+1})|\cos\phi_j|}{p_{angle}(y_{j-1} \leftrightarrow y_j \leftrightarrow y_{j+1})\ p_{roulette}(y_j)}$$

Figure 7.12: The power assigned to a photon hit, Φ_i, is calculated similar to the evaluation of a light subpath.

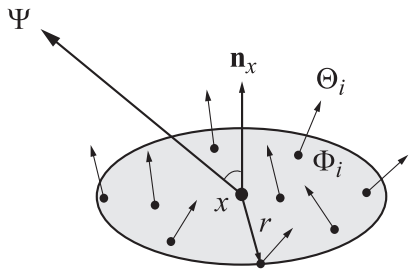
flectance. If these steps are followed, Φ_i reduces to Φ_{total}/n, where Φ_{total} is the total power emitted by the light sources and n is the number of photons emitted from the light sources.

7.7.2 Estimating Radiance with the Photon Map

Given a photon map of the scene, the renderer can estimate the reflected radiance leaving a surface point x using a process of k-nearest neighbor density estimation. The idea is to look up the k nearest hit records to x in the photon map and use them to reconstruct the incoming illumination at x. The kd-tree structure of the photon map makes finding the neighboring hit records efficient. Once the neighbors have been found, the reflected radiance $L_r(x \rightarrow \Psi)$ can be estimated as in figure 7.13.

7.7.3 Direct Viewing of the Photon Map and Final Gather

Direct viewing of the photon map. The second pass of the photon mapping algorithm creates a displayable image by tracing rays from the eye point. The most obvious use of the photon map would be to replace the reflected radiance at points hit by the eye rays with the estimate from figure 7.13. In practice, this does not work well, however, since the radiance



$$L_r(x \to \Psi) \approx \frac{1}{\pi r^2} \sum_{i=1}^{k} \Phi_i \, f_s(\Psi \leftrightarrow x \leftrightarrow \Theta_i)$$

Figure 7.13: Radiance estimate based on the photon map. The reflected radiance at x can be approximated as the sum of the contributions of the k nearest photon records, divided by the area in which the neighbors were found (πr^2 where r is the distance to the k^{th} nearest photon). Note that a cosine term is not needed since the density of photon hits on the surface is naturally proportional to the cosine of the incoming angle.

estimate is generally not accurate enough for direct viewing. The left side of the image in figure 7.14 demonstrates the typical blurring artifacts that result from directly viewing the photon map.

Final gather and the caustics map. Most of the blurring artifacts caused by directly viewing the photon map can be eliminated by performing a *final gather* step, which estimates radiance using the photon map on the second bounce rather than the first. In this scheme, the radiance at x is determined by sending out a number of probing rays, and estimating the radiance at the probed locations using the photon map. Although the final gather step is expensive, it is still much more efficient than standard path tracing for the same image quality in most situations.

One notable problem with final gather is that it will not work for very shiny or specular surfaces. Jensen's solution to this problem is to define a separate "caustics" photon map that only includes photon hits that have just bounced off of specular surfaces. The caustic lighting can then be estimated directly from the caustics map, while other lighting is computed in the final gather step. The caustics map idea works because photon hits naturally concentrate in caustic regions, leading to a more accurate radiance estimate. The

right half of the image in figure 7.14 shows the improvement in quality that results from rendering with a final gather step and separate caustics photon map. While the quality is much improved over path tracing with a similar number of samples, some blurring is evident in the caustic below the glass ball. Additionally, the diffuse spheres inside the glass ball look mottled because the sample density in the caustics photon map is low there.

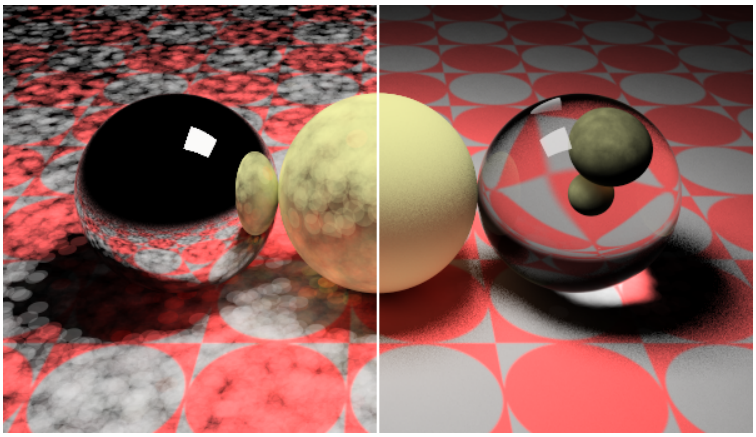
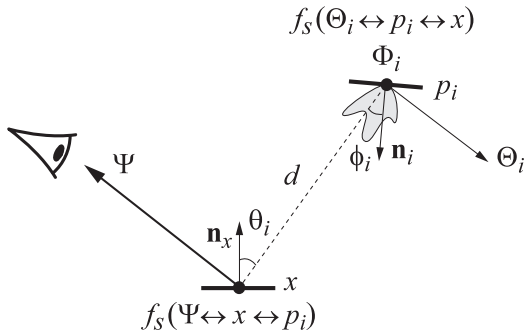


Figure 7.14: Direct viewing of the photon map (left) produces a very blurry image. Perfoming a final gather step (right) eliminates most of the image artifacts.

7.8 Instant Radiosity

While photon mapping uses particle hits to estimate the radiance arriving at different points in a scene, *instant radiosity* [Keller, 1997] turns the particle hits into *virtual point lights*. The hit records in a photon map represent the illumination *arriving* at surfaces in the scene. By converting the hit records into point lights that shine according to the BSDF term of the surface they are attached to, a representation of the light *leaving* scene surfaces is created. Thus, like photon mapping, instant radiosity is a two pass algorithm. In the first pass, a small number of particles (a few hundred or so) is traced through the scene. The particles are deposited on surfaces, and the hit locations are turned into point lights. The second pass of the algorithm renders an image using the virtual point lights in place of the scene illumination.



$$L_r(x \rightarrow \Psi) \approx \sum_{i=1}^{N} \Phi_i \, f_s(\Psi \leftrightarrow x \leftrightarrow p_i) |\cos \phi_i| \times \frac{f_s(x \leftrightarrow p_i \leftrightarrow \Theta_i) |\cos \theta_i| V(x \leftrightarrow p_i)}{d^2}$$

Figure 7.15: Instant radiosity evaluates the radiance as the sum of contributions from the virtual point lights scattered through the scene. (For virtual point lights created at the particle emission points on the light sources, the value $\Phi_i f_s(\Psi \leftrightarrow x \leftrightarrow p_i)$ is replaced by $L_e(p_i \rightarrow x)/p_{area}(p_i)$).

7.8.1 Creating the Point Lights

An instant radiosity renderer creates the virtual point lights it needs in an initial pass that is essentially the same process as building a photon map. The renderer shoots out particles from the light sources and creates point lights at surface intersections along the particle trajectories. Unlike in photon mapping, however, the renderer creates point lights at the particle emission points on the light sources as well as the intersection points. This has the effect of reducing all of the illumination in the scene, direct and indirect, to a uniform and simple representation (point lights).

7.8.2 Rendering based on Point Lights

Once the point lights have been created, the scene can be rendered. Figure 7.15 demonstrates this calculation. Rendering from virtual point lights has a number of advantages over other Monte Carlo methods such as path tracing and photon mapping with a final gather step. First, unlike photon mapping, which introduces bias into the reconstruction, the radiance estimate in instant radiosity (figure 7.15) is unbiased. Also, since the same

set of point lights is used to illuminate every point in the scene, instant radiosity renderings appear very smooth. Furthermore, reducing the illumination to point lights allows for efficient hardware rendering.

On the other hand, there are a number of drawbacks to representing the scene illumination as a set of point lights. One issue is that shadow boundaries will not appear smooth unless a large number of lights is used. For example, in figure 7.16 the shadows of individual point lights are clearly visible. Short range effects are another problem. Because the point lights are a discrete approximation to continuous scene lighting, areas near the point lights will appear too bright, leading to artifacts in concave corners. A number of these are visible in figure 7.16. The typical remedy for this problem is to clamp the distance d in the radiance calculation to some minimum value. This reduces spurious bright spots in the image, but at the price of darkening edges near concave corners in the scene. Finally, like the final gather step in photon mapping, instant radiosity cannot render caustic effects. This can be seen in figure 7.16, where the the caustic below the glass ball and the lighting on the diffuse spheres inside it are missing.

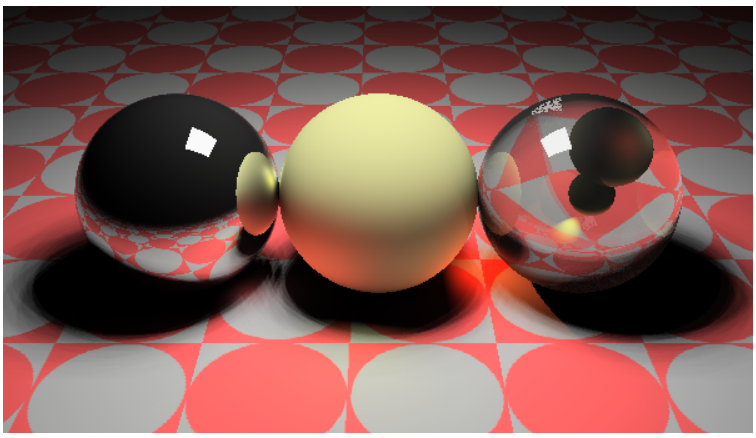


Figure 7.16: Instant radiosity, tracing 64 particles to create the virtual point lights.

7.8.3 Reducing the number of Point Light Evaluations

Probably the biggest concern with instant radiosity is that the number of point lights needed to create an accurate approximation to the scene illumination may be very high. This is

particularly true if the scene contains glossy surfaces. Wald *et al.* reduce the number of point light evaluations that are needed per pixel by interleaving the lights over a small neighborhood and interpolating the resulting illumination. A more recent technique, *lightcuts* [Walter *et al.*, 2005] reduces the number of evaluations by grouping the lights into hierarchical clusters, allowing a single point light within each cluster to act as a representative for the entire cluster. This work was followed on by *multidimensional lightcuts* [Walter *et al.*, 2006], which clusters eye rays within a pixel as well as virtual point lights. By choosing a representative from a group of eye rays and a representative from a group of point lights, the illumination between a group of eye rays and a group of point lights can be approximated with a single point light evaluation.

7.9 Bidirectional Path Tracing

In previous sections, we have seen that light paths can be created either from the eye point (path tracing) or from the light sources (light tracing), with each sampling direction having distinct advantages over the other. Furthermore, several of the particle tracing methods (photon mapping and instant radiosity) are "bidirectional" in the sense that lighting information propagates both from the eye point and the light sources. *Bidirectional path tracing*, developed nearly simultaneously by Lafortune and Willems [1993] and Veach and Guibas [1994], takes the bidirectional sampling concept a step further. A bidirectional path tracer samples the illumination in a scene by creating both an eye subpath and a light subpath. The vertices of the two subpaths are then connected together as shown in figure 7.17, resulting in a number of *bidirectional light paths*. The benefits of creating light paths in this manner are two-fold. First, starting from both the eye point and light sources allows bidirectional path tracing to retain the best qualities of both sampling strategies. Second, the average cost of creating light paths is reduced since most of the ray casting operations are reused by several light paths.

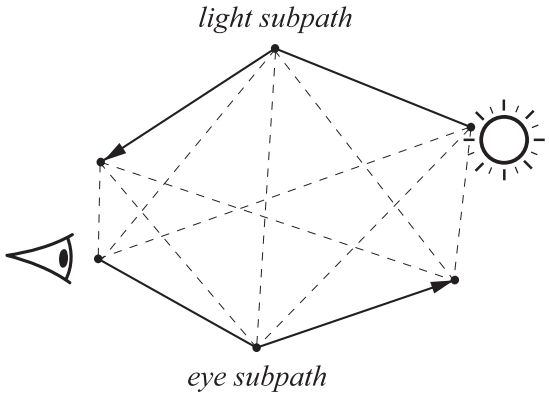


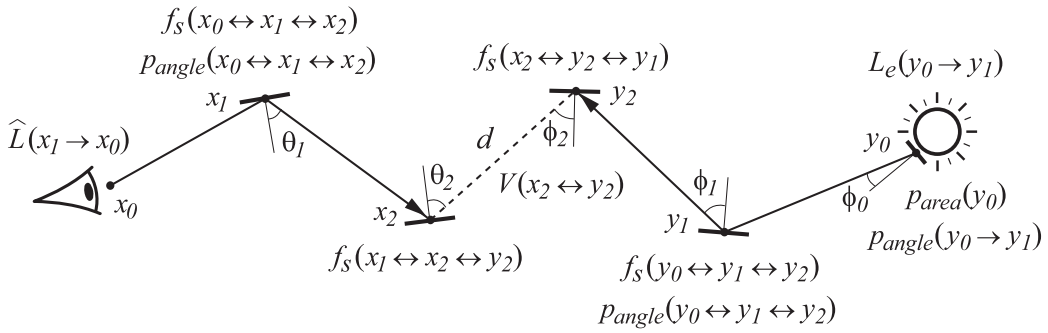
Figure 7.17: In a bidirectional path tracer, sampling begins with both an eye subpath and a light subpath. Complete light paths are created by connecting vertices from the eye subpath to vertices in the light subpath.

7.9.1 Evaluating Bidirectional Light Paths

To create a bidirectional light path, a bidirectional path tracer connects a vertex of an eye subpath to a vertex of a light subpath in a deterministic step similar to the connection made for an explicit light path. The connection is so similar that a bidirectional light path can be thought of as just an explicit light path in which the light source has been allowed to stochastically migrate through the scene. In fact, we have already seen this phenomenon with the virtual point lights created in instant radiosity. Figure 7.18 gives an example bidirectional path.

7.9.2 Implementation Concerns in Bidirectional Path Tracing

Distribution of paths lengths. An important consequence of connecting each vertex of the light subpath with each vertex of the eye subpath is that the number of light paths of different lengths is not uniform. For example, figure 7.17 shows the nine light paths created by connecting a light subpath of length 2 with an eye subpath of length 2. Of the nine paths, only one is of length 5, but three are of length 3, resulting in oversampling of that length. To compensate for this effect, the value of $\widehat{L}$ from figure 7.18 must be divided by the number of paths that will be created at that length.



$$\widehat{L}(x_1 \rightarrow x_0) = \frac{L_e(y_0 \rightarrow y_1)\ |\cos\phi_0|}{p_{area}(y_0)\,p_{angle}(y_0 \rightarrow y_1)} \times \frac{f_s(y_0 \leftrightarrow y_1 \leftrightarrow y_2)\ |\cos\phi_1|}{p_{angle}(y_0 \leftrightarrow y_1 \leftrightarrow y_2)\ p_{roulette}(y_1)} \times$$

$$\frac{f_s(x_0 \leftrightarrow x_1 \leftrightarrow x_2)\ |\cos\theta_1|}{p_{angle}(x_0 \leftrightarrow x_1 \leftrightarrow x_2)\ p_{roulette}(x_1)} \times \frac{f_s(x_1 \leftrightarrow x_2 \leftrightarrow y_2)|\cos\theta_2|\ f_s(x_2 \leftrightarrow y_2 \leftrightarrow y_1)|\cos\phi_2|\ V(x_2 \leftrightarrow y_2)}{d^2}$$

Figure 7.18: A *bidirectional light path* is created by connecting an eye subpath to a light subpath. The light subpath can be thought of as a stochastic migration of the light source. Note that the top line of the calculation is the same as the photon power (Φ_i) from photon mapping and instant radiosity.

Specular vertices. Specular vertices in the light or eye subpaths do not need to be connected, since the probability of sampling the specular direction by a direct connection will be zero.

Treating the eye subpath as an implicit light path. In addition to the light paths created by connecting the light and eye subpaths, the eye subpath itself may form implicit light paths. These implicit paths must be included because some lighting effects, such as caustics seen in reflection, cannot be sampled by explicitly connecting subpaths.

Russian roulette to avoid visibility tests. As the lengths of the eye and light subpaths increase, the number of possible connections increases dramatically (# connections = (eye subpath length + 1) $\times$ (light subpath length + 1)). Russian roulette can be used to eliminate the bulk of the visibility tests. First the contribution of each connection is evaluated without visibility. Then, Russian roulette is used to decide if the visibility test is worth the effort.

7.9.3 Results of Bidirectional Path Tracing

Figure 7.19 shows a rendering of the three spheres scene made with bidirectional path tracing. Compared to standard path tracing (figure 7.7), many parts of the image are much smoother, particularly the caustic below the glass ball. However, the lighting on the spheres inside the glass ball, and the middle sphere seen in reflection is just as noisy as in the path tracing case. This is a consequence of the fact that caustics seen through specular surfaces are only sampled by implicit light paths, so bidirectional path tracing offers no sampling advantage in these regions.

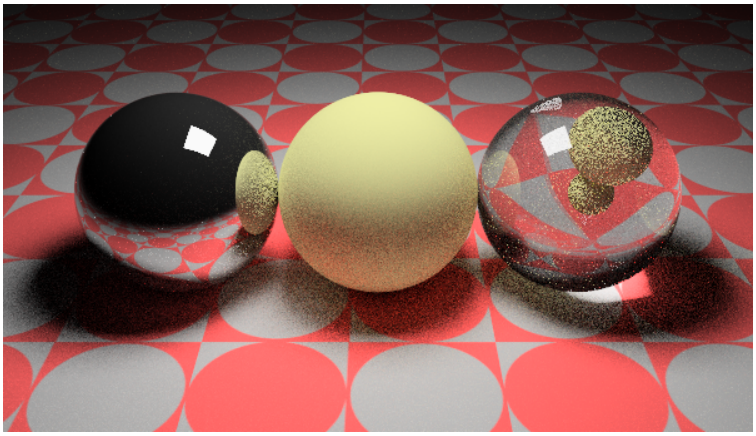
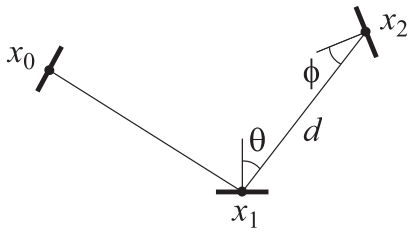


Figure 7.19: Bidirectional path tracing.

7.10 The Measurement Contribution Function

By now it should be apparent just how cumbersome light transport notation can be. To make matters worse, each new algorithm introduces new equations that are similar, but not identical, to those used by previous algorithms. In part to counter this problem, Veach [1997] showed that the notation for many ray based global illumination algorithms can be unified by rewriting the measurement equation in terms of area instead of solid angle. He started

by writing the rendering equation in terms of points in the scene rather than in terms of a point and incoming and outgoing directions as follows:



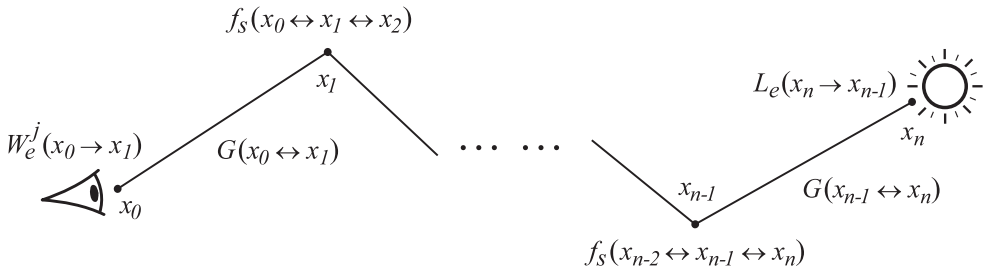
$$L(x_1 \rightarrow x_0) = L_e(x_1 \rightarrow x_0) + \int_M f_s(x_0 \leftrightarrow x_1 \leftrightarrow x_2)\, G(x_1 \leftrightarrow x_2)\, L(x_2 \rightarrow x_1)\, dx_2, \qquad (7.6)$$

where M is the union of all surfaces in the scene and $G(x_1 \leftrightarrow x_2)$, called the *geometry term*, converts area measure to projected solid angle measure:

$$G(x \leftrightarrow y) = V(x \leftrightarrow y)\frac{|\cos\theta\ \cos\phi|}{d^2}. \qquad (7.7)$$

Next, Veach performed a Neumann expansion on the new form of the rendering equation, resulting once again in an infinite set of terms, except that this time the terms were defined over areas on surfaces rather than directions on the sphere. We can think of the terms of this expansion as describing the differential, or derivative of, light flow along light paths of particular lengths, with respect to sampled surface area. By extracting the term for a particular light path and multiplying by the importance for pixel j, $W_e^j(x_0 \rightarrow x_1)$, Veach was able to determine the differential contribution of the light path to pixel j in the absence of probability. He called this the *measurement contribution function* (see figure 7.20).

The path probability. A main advantage of the measurement contribution function over other representations is that it describes the contribution of a light path to a pixel without regard to how the path was created. Based on similar arguments, we can also define



$$F_n^j(x_0 \ldots x_n) = W_e^j(x_0 \rightarrow x_1)\, G(x_0 \leftrightarrow x_1) \times \left[\prod_{i=1}^{n-1} f_s(x_{i-1} \leftrightarrow x_i \leftrightarrow x_{i+1})\, G(x_i \leftrightarrow x_{i+1})\right] \times L_e(x_n \rightarrow x_{n-1})$$

Figure 7.20: The measurement contribution function. For a light path with vertices $(x_0 \ldots x_n)$ contributing to pixel j, the measurement contribution function $F_n^j(x_0 \ldots x_n)$ determines the differential contribution from the light path to the measurement at pixel j.

the probability with which a light path was generated in a way that is independent of how it was generated. This *path creation probability*, p_{path}, is simply the product of the probabilites of choosing each point in the path with respect to area:

$$p_{path}(x_0 \ldots x_n) = \prod_{i=0}^{n} p_{area}(x_i). \tag{7.8}$$

By combining the measurement contribution function with the path creation probability, we can calculate the Monte Carlo contribution of a light path sample to pixel j as

$$\widehat{L}(x_1 \rightarrow x_0) = \frac{F_n^j(x_0 \ldots x_n)}{p_{path}(x_0 \ldots x_n)} \tag{7.9}$$

no matter how the path was created, provided that we can evaluate F_n^j and p_{path}. This derivation does not mean to suggest that the primary sampling method in a global illumination renderer should be choosing points on surfaces–quite the contrary, it is merely pointing out that by converting all probabilities to be in terms of surface area, a unified method of evaluation results.

Example conversions. Let's look at a couple of conversions to area probability. Often, when sampling a BSDF, the probability is expressed in terms of solid angle. We can convert solid angle probability to area probability based on equation 4.6:

$$p_{area} = \frac{p_{angle} \cos\phi}{d^2}. \tag{7.10}$$

As a second example, consider sampling points on the image plane in pixel coordinates. When a pixel coordinate is chosen, we would like to determine the probability that the surface hit by the ray passing through that coordinate was chosen, with respect to area. Instead of going through a long derivation, we simply observe that p_{area} in this case must cancel out the terms $W_e^j(x_0 \to x_1)G(x_0 \leftrightarrow x_1)$ from the measurement contribution function to make path tracing work properly. Thus the probability must equal

$$p_{area} = \frac{p_{pixel}}{W_e^j(x_0 \to x_1)\ G(x_0 \leftrightarrow x_1)}, \tag{7.11}$$

where p_{pixel} is the probability of choosing the pixel coordinates with respect to pixel area, W_e^j is the importance for a pinhole camera (equation 7.3), and G is the geometry term (equation 7.7).

7.11 Metropolis Light Transport

One of the most interesting global illumination algorithms to emerge in the past decade is Metropolis Light Transport, or MLT [Veach and Guibas, 1997]. Unlike the other ray based global illumination algorithms introduced in this chapter, MLT does not rely on Monte Carlo integration. Instead, it relies on a different statistical integration technique called Metropolis sampling [Metropolis *et al.*, 1953; Pharr, 2003]. The main strength of MLT lies

in its ability to explore local regions of the space of light paths in an unbiased way. This makes MLT particularly attractive for hard sampling problems in global illumination such as caustics and light shining through small aperatures.

7.11.1 Metropolis Sampling

Suppose that we want to approximate a function f over some domain, D. One way to do this is to produce a sampling distribution proportional to f and then make a histogram of samples taken from the distribution. The resulting histogram will be proportional to f, so it only needs to be scaled to approximate it. *Metropolis sampling* uses this method to approximate functions, and can be summarized as follows:

- Create a sampling distribution proportional to f.
- Make a histogram of samples taken from the sampling distribution.
- Scale the histogram to approximate f.

In the case of an image, f is defined on some subset of $\mathbb{R}^2$, and the histogram contains one bin for each pixel in the image. The scale factor, s, needed to make the histogram approximate f is the ratio of the average value of f in the sampling domain, f_{ave}, to the average number of samples per bin in the histogram, h_{ave}:

$$s = f_{ave}/h_{ave} \tag{7.12}$$

In practice, f_{ave} can be estimated by averaging a large number of samples selected at random from the sampling domain.

7.11.2 Creating a Sampling Distribution

Detailed balance. The Metropolis algorithm uses an idea called detailed balance to create a sampling distribution proportional to f. An intuitive way to think about detailed balance

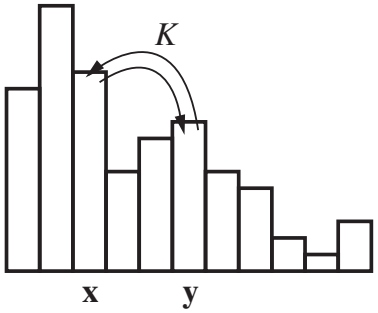


Figure 7.21: Detailed balance. The desired (stationary) distribution will be maintained as long as the number of samples flowing between any two bins **x** and **y** in the histogram is balanced. In other words $K(\mathbf{x} \rightarrow \mathbf{y}) = K(\mathbf{y} \rightarrow \mathbf{x})$.

is to imagine that a histogram proportional to f already exists. This distribution of samples is called the *stationary distribution*. Now imagine that a transition function exists that allows samples to flow between bins in the histogram. The stationary distribution will be maintained as long as the number of samples flowing from one bin in the histogram to another is the same as the number of samples flowing back. This property is called *detailed balance*. Let K denote a transition function that obeys the detailed balance property. (See figure 7.21.)

An important consequence of detailed balance is that if a single sample is allowed to migrate in the domain of f according to K, it will eventually trace out the stationary distribution (proportional to the function we want to approximate). The strategy adopted by Metropolis sampling is to create a suitable K function and then use it to migrate a single sample though the domain of f. As the sample moves, a histogram of its location is kept, and this histogram is used to approximate f.

Defining the transition function. The function K is defined by using a *tentative transition function* T, also known as a *mutation strategy*. $T(\mathbf{x} \rightarrow \mathbf{y})$ gives the probability of choosing point **y** as the proposed next sample location if **x** is the current sample location. To complete K, a tentative sample location **y** is chosen based on T, f is evaluated at **x** and

y, and the next sample location either migrates to **y** with probability $a(\mathbf{x} \rightarrow \mathbf{y})$, or remains at **x** with probability $1 - a(\mathbf{x} \rightarrow \mathbf{y})$, where

$$a(\mathbf{x} \rightarrow \mathbf{y}) = min\left\{1, \frac{f(\mathbf{y})T(\mathbf{y} \rightarrow \mathbf{x})}{f(\mathbf{x})T(\mathbf{x} \rightarrow \mathbf{y})}\right\} \tag{7.13}$$

The **copyImage** function in figure 7.22 uses Metropolis sampling to copy an image. This is not a very useful way to copy an image, but it does provide a good example of Metropolis sampling in action. CopyImage uses a very simple mutation strategy, namely choosing a random point on the image plane with a uniform probability, but a wide range of transition functions can be used. Later we will see that the power of Metropolis sampling lies in choosing good mutation strategies.

7.11.3 Color Images

The Metropolis sampling framework can easily be extended to handle color images by redefining the histogram to accumulate in color. The luminance of the color samples at points **x** and **y** is used to define $a(\mathbf{x} \rightarrow \mathbf{y})$, and colors added to the histogram are scaled to have a luminance of 1. Figure 7.23 shows how this might look in code. Note that any additive color space can be used for the histogram. In the common case of RGB, luminance is defined as $(0.299\ R + 0.587\ G + 0.114\ B)$.

```
void copyImage(float F[w][h], float histogram[w][h], int mutations)
{
    int i, x0, x1, y0, y1;
    float Fx, Fy, Txy, Tyx, Axy;

    // Create an initial sample point
    x0 = randomInteger(0, w-1);
    x1 = randomInteger(0, h-1);
    Fx = F[x0][x1];

    // In this example, the tentative transition function T simply chooses
    // a random pixel location, so T(x->y) and T(y->x) are always equal.
    Txy = 1.0 / (w * h);
    Tyx = 1.0 / (w * h);

    // Create a histogram of values using Metropolis sampling.
    for (i=0; i < mutations; i++) {
        // choose a tentative next sample according to T.
        y0 = randomInteger(0, w-1);
        y1 = randomInteger(0, h-1);
        Fy = F[y0][y1];
        Axy = MIN(1, (Fy * Tyx) / (Fx * Txy));   // equation 7.13.
        if (randomReal(0.0, 1.0) < Axy) {   // jump to y with probability a(x->y)
            x0 = y0;
            x1 = y1;
            Fx = Fy;
        }
        histogram[x0][x1] += 1;   // increment histogram
    }
}
```

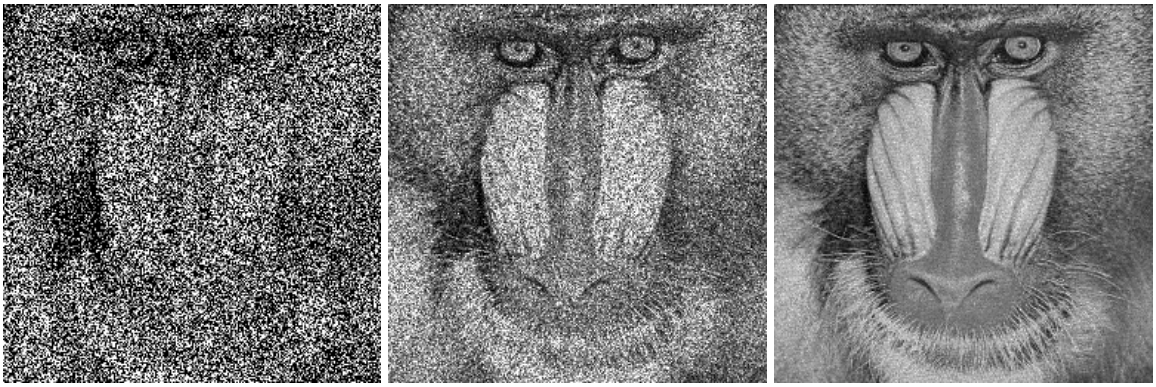


Figure 7.22: The **copyImage** function. This function uses Metropolis sampling to make a histogram from an image that is passed in the **F** array. It is assumed that the **histogram** array is initialized to be all zeros. After the function returns, the histogram can be scaled to approximate **F**. Below the code are several images created using **copyImage**. The original image is approximated using an average of 1 (left), 8 (middle) and 256 samples per pixel (right).

```
DomainLocation X,Y;
  :
  :
for (i=0; i < mutations; i++) {
        Y = mutateAccordingToT(X);
        Tyx = T(Y, X);
        Txy = T(X, Y);
        colorY = F[Y.xloc][Y.yloc];
        Fy = colorY.luminance();
        colorY /= Fy; // scale colorY to have luminance 1
        Axy = MIN(1, (Fy * Tyx) / (Fx * Txy));
        if (randomReal(0.0, 1.0) < Axy) {   // jump to y with probability a(x->y)
                X = Y;
                Fx = Fy;
                colorX = colorY;
        }
        histogram[X.xloc][X.yloc] += colorX;   // increment histogram
}
```

Figure 7.23: Accumulating in color. The image and histogram have both been converted to color arrays. **Fx** and **Fy** are redefined to be luminance values, and colors added to the histogram have a luminance of 1. In addition, the mutation strategy and pixel coordinates of each sample have been encapsulated. In the case of MLT, the **DomainLocation** structure will not only contain a pixel location, but will include an entire light path.

7.12 From Metropolis Sampling to MLT

Metropolis light transport is nothing more than a version of Metropolis sampling that evaluates light paths using the measurement contribution function (section 7.10) instead of directly evaluating image pixels. Since the measurement contribution function is defined in terms of surface area, the value of the transition function T must also be defined in terms of surface area, so we write it as T_{area}. Putting these two facts together, if T_{area} mutates path $\mathbf{x} = (x_0 \ldots x_n)$ to path $\mathbf{y} = (y_0 \ldots y_m)$, the acceptance probability $a(\mathbf{x} \rightarrow \mathbf{y})$ will be

$$a(\mathbf{x} \rightarrow \mathbf{y}) = min\left\{1,\ \frac{F_m^k(\mathbf{y})\ T_{area}(\mathbf{y} \rightarrow \mathbf{x})}{F_n^j(\mathbf{x})\ T_{area}(\mathbf{x} \rightarrow \mathbf{y})}\right\}, \tag{7.14}$$

where j is the pixel to which path $\mathbf{x}$ contributes, and k is the pixel to which path $\mathbf{y}$ contributes.[2] This is true no matter what mutation strategy is used. Of course, the hard part is determining T_{area}, so we will give some examples of this in the next section. Figure 7.24 gives pseudocode for the main loop of MLT.

7.12.1 MLT Mutation Strategies

Restrictions on mutation strategies. There are two main restrictions on the types of transition functions that can be used to form a mutation strategy. First, the transition probabilities $T_{area}(\mathbf{x} \rightarrow \mathbf{y})$ and $T_{area}(\mathbf{y} \rightarrow \mathbf{x})$ or their ratio must be computable. Second, every part of the space of light paths must be reachable from every other part. The second restriction ensures the so called "ergodicity" condition, which means that the distribution of samples will eventually converge to the stationary distribution.

[2] If a light path contributes to more than one pixel, the F values should be replaced by the sum of the contributions over all pixels to which the light path contributes.

```
LightPath X, Y;
Color colorX, colorY;
float Txy, Tyx, Fx, Fy, Axy;
  :
  :
for (i=0; i < mutations; i++) {
        Y = mutateAccordingToT(X);
        Tyx = Tarea(Y, X);
        Txy = Tarea(X, Y);
        colorY = evaluateMeasurementContribution(Y);
        Fy = colorY.luminance();
        colorY /= Fy;
        Axy = MIN(1, (Fy * Txy) / (Fx * Tyx));   // calculate a(x->y)
        if (randomReal(0.0, 1.0) < Axy) {   // jump to y with probability a(x->y)
                X = Y;
                Fx = Fy;
                colorX = colorY;
        }
        histogram[X.xloc][X.yloc] += colorX;   // increment histogram
}
```

Figure 7.24: Pseudocode for the main loop of MLT.

Combining transition functions. It may be difficult to design a single mutation strategy that efficiently samples all of the lighting in a particular scene. It is much easier to design transition functions around specific sampling problems, and then combine them to form a robust mutation strategy.

New Path Mutations: A First MLT Mutation Strategy

One obvious mutation strategy is to create a new random light path at a random pixel location using standard or bidirectional path tracing. We call this a *new path mutation*. For a new path mutation, T_{area} is not dependent on the starting state, so $T_{area}(\mathbf{x} \rightarrow \mathbf{y}) = p_{path}(\mathbf{y})$, and

$$a(\mathbf{x} \rightarrow \mathbf{y}) = min\left\{1, \frac{F_m^k(\mathbf{y})\ p_{path}(\mathbf{x})}{F_n^j(\mathbf{x})\ p_{path}(\mathbf{y})}\right\} = min\left\{1, \frac{\widehat{L}(\mathbf{y})}{\widehat{L}(\mathbf{x})}\right\},$$

or in other words, in a new path mutation $a(\mathbf{x} \rightarrow \mathbf{y})$ is the ratio of the Monte Carlo estimates of the mutated path and the original path.

In practice, new path mutations do not work well by themselves. However, they are useful in two ways. First, because they can reach any part of the space of light paths, new path mutations guarantee the ergodicity condition. Second, new path mutations can be used to estimate the average pixel brightness (just average the $\widehat{L}$ values from all the new paths).

Heckbert's Regular Expressions

Heckbert [1990] devised a regular expression notation to describe different types of light paths, which Veach and Guibas adopted for their MLT descriptions. In Heckbert's notation, L stands for a light source, D is a non-specular surface, S is a specular surface, and E is the eye point. These interaction letters can be combined in regular expressions to describe full light paths. For example, the light path LDS^*E begins at the light source, and propagates through one non-specular bounce and zero or more specular bounces before joining with the eye point.

Mutations Starting With the Eye Point

Veach and Guibas describe several mutations that attempt to move the current light path starting at the eye point. The basic idea behind these mutations, which are referred to as *lens perturbations* and *multi-chain perturbations*, is to create a new light path by perturbing the pixel coordinates of the current light path. This process can be summarized as follows:

- The mutation routine perturbs the pixel location of the current path by an exponentially distributed random amount as described in figure 7.25. Let x_1 be the point seen through the original pixel location, and let y_1 be the point seen through the mutated

```
void exponentialPixelOffset(float r1, float r2, float &x, float &y)
{
    float phi = randomReal(0.0, 1.0) * 2.0 * PI;
    float r = r2 * exp( -log(r2/r1) * randomReal(0.0, 1.0) );
    x += r * cos(phi);
    y += r * sin(phi);
}
```

Figure 7.25: Exponentially distributed pixel offset. The above function perturbs the pixel location (x, y) by an exponentially distributed random distance r between r_1 and r_2. Veach and Guibas suggested values of 0.1 pixels for r_1 and about 10% of the image width for r_2.

```
void exponentialAngularOffset(float theta1, float theta2, Point &N)
{
    // Make a UVN coordinate system from N
    Point U, V;
    if (fabs(N.x) < 0.5) U = N.cross(Point(1,0,0));
    else U = N.cross(Point(0,1,0));
    U.normalize();
    V = U.cross(N);

    // Determine offsets using the approximation θ ≈ sin θ
    float phi = randomReal(0.0, 1.0) * 2.0 * PI;
    float r = theta2 * exp( -log(theta2/theta1) * randomReal(0.0, 1.0) );

    // Calculate the new direction
    N = N + r*cos(phi)*U + r*sin(phi)*V;
    N.normalize();
}
```

Figure 7.26: Exponentially distributed angular offset. The above function perturbs the direction N by a random angle that is exponentially distributed between θ_1 and θ_2. We assume that N is normalized and θ_1 and θ_2 are small. Veach and Guibas suggested values of 0.0001 radians for θ_1 and 0.1 radians for θ_2.

pixel location. The transition probability for this operation, $T_{pixel}(x_1 \to y_1)$, is equal to the area probability for y_1, which by equation 7.11 is proportional to[3]

$$T_{pixel}(x_1 \to y_1) \propto \frac{1}{W_e^j(y_0 \to y_1)\, G(y_0 \leftrightarrow y_1)}.$$

- Starting at the eye point, the new subpath is propagated through the same number of specular bounces as the original path. The same reflection mode is used at each path vertex as was used by the corresponding vertex in the original path (reflection or refraction). It may not be obvious how to compute the transition probability for this operation, but we can think of specular propagation as sampling a direction with respect to solid angle, where p_{angle} is infinite. Consequently, we can use equation 7.10 to determine the transition probability:

$$T_{specular}(x_n \to y_n) \propto \frac{\cos\phi_n}{d^2}.$$

- For a lens perturbation, the first non-specular vertex (counting from the eye point) is connected directly to the next vertex of the original path, which must also be non-specular, or the light source. For example, in the path *LDSSE*, the suffix $\ldots DSSE$ is replaced by a new one of the same form, and the new subpath is connected directly to the point L from the original path.[4] This operation does not have a probability since no points in the path are changed, but the visibility of the connection must be checked.

- In a multi-chain perturbation, the outgoing direction from the first non-specular vertex (counting from the eye point) is perturbed by a random angle, as described in

[3] It may be difficult to calculate the exact transition probability between two points for a given operation, but as long as we can calculate something proportional to it, the ratio $T(x \to y)/T(y \to x)$ will be preserved, and the acceptance probability will be calculated correctly.

[4] In a path such as *LSSE*, no explicit connection is needed. The mutation simply stops when the light source is reached.

figure 7.26. As with the specular propagation, the transition probability for this operation is

$$T_{perturbAngle}(x_n \rightarrow y_n) \propto \frac{\cos\phi_n}{d^2}.$$

The new subpath is then propagated through the same number of specular bounces as the original path, arriving at the next non-specular surface, with each specular bounce having a transition probability of $T_{specular}$. If the vertex after this non-specular vertex in the path is also non-specular, the new subpath can be joined back onto the remainder of the old path (with no transition probability). Otherwise, the path must be propagated through another chain of specular bounces. This process repeats until the old path is exhausted, or a pair of non-specular vertices is found. For example, consider the light path $LDSSDSE$. A new suffix of the form $\ldots DSE$ is generated starting at the eye point. The direction of the outgoing ray from D is perturbed and the path is propagated through two specular bounces to form the subpath $\ldots DSSDSE$. Since the next vertex in the original path is non-specular (L), the new subpath can be connected directly to the next vertex of the original path, L. Figure 7.27 gives pseudocode for a muti-chain perturbation.

There are several reasons why a lens or multi-chain perturbation may fail to create a new light path. For instance, one of the specular vertices of the original path may migrate to a non-specular surface. Also, the new path may fail to hit a light source, or consecutive points along the mutated path may not be mutually visible. Finally, the mutated path may no longer lie within the pixel bounds of the image. If any of these situations occurs, the mutation is rejected.

If the mutation successfully creates a new light path $\mathbf{y}$ from path $\mathbf{x}$, the renderer can calculate the acceptance probability $a(\mathbf{x} \rightarrow \mathbf{y})$ based on equation 7.14. The values for $T_{area}(\mathbf{x} \rightarrow \mathbf{y})$ and $T_{area}(\mathbf{y} \rightarrow \mathbf{x})$ are calculated as the product of all of the transition events that went into the mutation. Infinite values in $F_n^j(\mathbf{x})$ and $F_m^k(\mathbf{y})$ that result from specular surfaces can be eliminated by replacing delta functions in the BSDFs with a value of 1.

Lens subpath or multi-chain perturbation

1. Perturb the pixel location. (Transition probability = T_{pixel})
2. While the next vertex is specular
   Propagate through the specular vertex. ($T_{specular}$)
3. If we are doing a lens subpath mutation, go to step 6.
4. Perturb the outgoing direction. ($T_{perturbAngle}$)
5. While the next vertex is specular
   Propagate through the specular vertex. ($T_{specular}$)
6. Connect to the next vertex if it is not specular, else go to step 4.

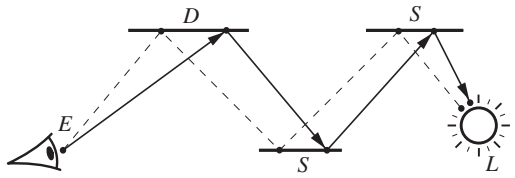


Figure 7.27: A *multi-chain perturbation* modifies the current path starting at the eye point.

Mutations Starting at the Light Source

Some lighting situations can be sampled better by mutations starting at the light source instead of the eye point. Veach and Guibas describe a mutation strategy called a *caustic perturbation* that moves the current path starting at the light source. They use this mutation type to sample paths of the form LS^*DE or $\ldots DS^*DE$.

Caustic perturbations are created in much the same way as lens perturbations, except that they start at the light source, or second diffuse vertex in the path from the eye point. The mutation routine perturbs the direction $L \rightarrow S$ (or $D \rightarrow S$) by a random angle as in figure 7.26. The transition probability of this operation is once again $T_{perturbAngle}$.

The new subpath is then propagated through the same number of specular bounces as the original path, creating the subpath $LS^*D\ldots$ or $\ldots DS^*D\ldots$. As with lens subpath mutations, the transition probability of propagating through a specular vertex is $T_{specular}$, but using θ_n instead of ϕ_n.

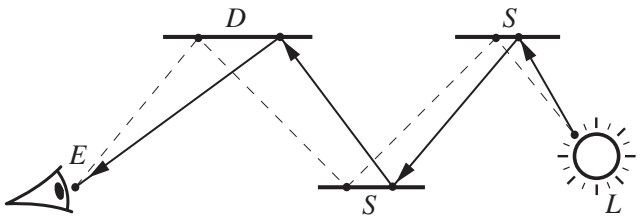


Figure 7.28: A *caustic perturbation* can efficiently sample paths of the form LS^*DE. The mutation routine replaces the subpath $LS^*D\ldots$ starting at the light source. The new subpath is then connected directly to the eye point.

Finally, the renderer connects the mutated subpath directly to the eye point. Note that since the direction $D \rightarrow E$ has changed, the pixel coordinates of the new light path will have changed as well, and must be calculated as in section 7.6.2. Figure 7.28 shows a caustic perturbation graphically.

7.12.2 MLT results

Figure 7.29 compares the results obtatined using Metropolis light transport and bidirectional path tracing. In the top row of the figure, bidirectional path tracing achieves approximately the same quality as MLT. (Standard path tracing, not shown, would only produce a few bright speckles to represent the caustics.) The second row of images zooms in on one of the caustics. This has the effect of breaking down the ability of the bidirectional path tracer to sample the caustic light paths. Conversely, MLT is able to concentrate more samples on the caustic, producing a much smoother result.

7.13 A Simple and Robust Formulation of MLT

Kelemen *et al.* [2002] created an alternate formulation of MLT that is simpler in many ways than Veach's original description. Instead of relying on a measurement contribution function defined over surface areas, Kelemen *et al.* work "in the space of uniform random numbers used to build up paths." To put it another way, we can think of each light path

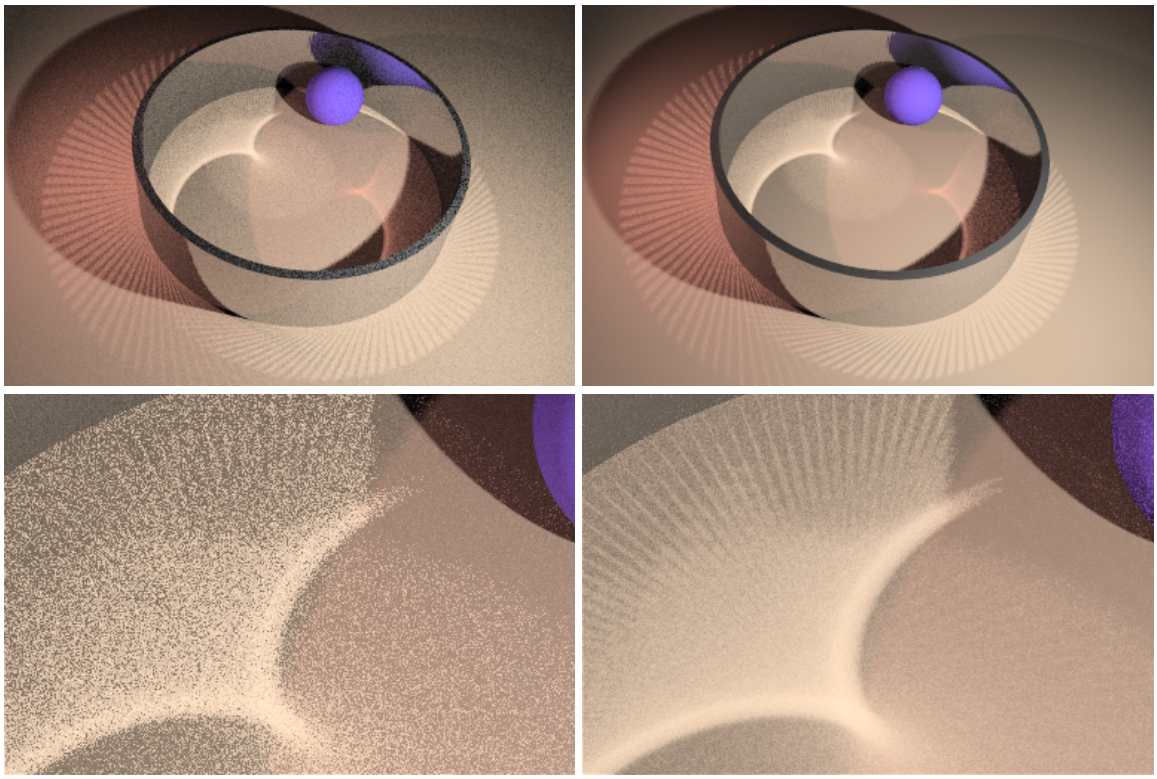


Figure 7.29: Comparison of bidirectional path tracing to Metropolis Light Transport. In this scene, the ring is faceted, creating detailed ray patterns in the caustics. The images on the left were made with bidirectional path tracing using 100 paths per pixel. On the right, the images were created with MLT using 100 mutations per pixel (approximately the same number of ray queries). As suggested by Veach and Guibas, direct lighting in the Metropolis images was computed using standard path tracing.

as being represented by a vector of numbers between zero and one. For example, the light path $\mathbf{x} = (x_0 \ldots x_n)$ might be represented by the vector of numbers $\vec{u} = <u_0 \ldots u_m>$. We use over-arrow notation and angle brackets to distinguish between the random numbers in $\vec{u}$ and the 3D points that make up $\mathbf{x}$. This space of numbers can be thought of as the *primary sample space* in which a path tracer, or bidirectional path tracer works. Some of the numbers may be used to choose a pixel location, while others may sample a BSDF or a light source. Still others may be used for Russian roulette, and so forth, but however the numbers are used, $\vec{u}$ must completely define $\mathbf{x}$.

The simplicity of the new formulation becomes apparent when one tries to create a mutation strategy and define the acceptance probability for it. First, within the area measure

of the random number space, the measurement contribution function is just the Monte Carlo estimate of the associated path: $F_n^j(\vec{u}) = \widehat{L}(\vec{u})$.[5] In addition, mutations within the space of random numbers are simpler as well. Kelemen *et al.* showed that two mutation types, "large steps" and "small steps", are sufficient to produce most of the results from Veach's original formulation. A *large step* simply discards the current vector of numbers and creates a new one from scratch, and a *small step* perturbs each number in the current vector by a small random amount (within about 0.05 units of the current value), allowing the numbers to wrap around if they go above 1 or below 0. With respect to the space of random numbers, the transition probabilities between any two points are equal for both large and small steps. Thus, $T(\vec{u} \rightarrow \vec{v}) = T(\vec{v} \rightarrow \vec{u})$ and

$$a(\vec{u} \rightarrow \vec{v}) = min\left\{1, \frac{\widehat{L}(\vec{v})}{\widehat{L}(\vec{u})}\right\}$$

for both large and small steps, a much simpler result than with standard MLT.

Algorithmically, the new form of MLT proceeds as follows: the renderer mutates the current vector $\vec{u}$ to produce $\vec{v}$, randomly choosing a large step or a small step. Next, the renderer constructs the path associated with $\vec{v}$ and evaluates $\widehat{L}(\vec{v})$. It then jumps to $\vec{v}$ with probability $a(\vec{u} \rightarrow \vec{v})$, or it retains $\vec{u}$ with probability $1 - a(\vec{u} \rightarrow \vec{v})$. Finally, the renderer deposits a single sample on the image plane at the resulting pixel location (histogram bin), and the process repeats.

One issue with the new formulation is the length of the number vectors. Because of Russian roulette or other concerns, a vector may not have enough numbers in it to define a complete light path after a mutation. This problem can be solved by lazily adding new random numbers to mutated vectors as needed during small step mutations, and resetting the size of the vector during large steps to exactly the requirements of the mutated path.

[5] This is a bit of abuse of notation, but what we mean is that the measurement contribution of the light path derived from $\vec{u}$, with respect to the area measure of the random number space, is equal to the Monte Carlo estimate of the associated path.

Figure 7.30 shows the three spheres scene rendered using the simple and robust MLT formulation. Note that while some speckles remain in the image, they are much dimmer than in standard and bidirectional path tracing. This suggests that MLT will converge to an artifact-free image more quickly than these methods.

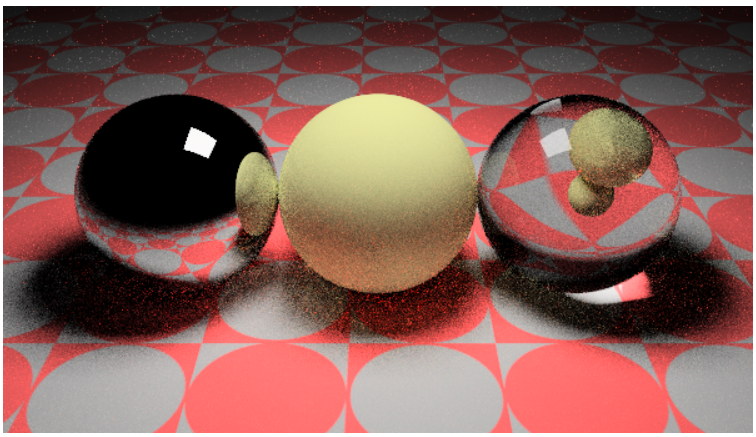


Figure 7.30: MLT using the simple and robust formualation.

Chapter 8

Energy Redistribution Path Tracing

A version of this chapter was published as:

David Cline, Justin Talbot and Parris Egbert. Energy Redistribution Path Tracing. *ACM Transactions on Graphics*, 24(3):1186-1195, 2005.

Abstract. We present Energy Redistribution (ER) sampling as an unbiased method to solve correlated integral problems. ER sampling is a hybrid algorithm that uses Metropolis sampling-like mutation strategies in a standard Monte Carlo integration setting, rather than resorting to an intermediate probability distribution step. In the context of global illumination, we present Energy Redistribution Path Tracing (ERPT). Beginning with an inital set of light samples taken from a path tracer, ERPT uses path mutations to redistribute the energy of the samples over the image plane to reduce variance. The result is a global illumination algorithm that is conceptually simpler than Metropolis Light Transport (MLT) while retaining its most powerful feature, path mutation. We compare images generated with the new technique to standard path tracing and MLT.

8.1 Introduction

Today, a number of rendering programs exist that can produce photorealistic output. Methods that create such "synthetic photographs" by measuring light transport are collectively known as *global illumination* algorithms. The distinguishing characteristic of a global illumination algorithm, as opposed to an ad-hoc lighting algorithm, is the goal of accounting for all light scattering events that lead to the creation of an image. In a very real sense, the process of global illumination is a physical simulation in which light transport paths are followed through a virtual scene and recorded on a virtual film plane.

The most versatile global illumination algorithms currently available are based on ray tracing and numerical integration. [Kajiya, 1986] was the first to publish a global illumination algorithm of this type. Drawing on heat transfer literature and Monte Carlo integration theory, Kajiya described the now classic path tracing algorithm, which samples the light reaching the image plane by tracing potential light paths backwards from the eye point.

Despite the generality of path tracing, it can be quite inefficient even in common lighting situations. The reason for this inefficiency is variance in the Monte Carlo light estimate, which shows up as noise in a rendered image. Practically speaking, the Monte Carlo sampler does not have enough global context to quickly find all of the important light transport paths. Importance sampling techniques, such as those described by Veach and Guibas [1995] and Lawrence *et al.* [2004] can provide some of this context, but usually only in a local way.

Noting the difficulty of finding all of the significant light transport paths starting at the eye point, Lafortune and Willems [1993] and Veach and Guibas [1994] independently developed bidirectional path tracing, which generates paths starting at the light sources as well as the eye point. Some parts of path space are better sampled this way, so variance is reduced.

Other techniques cache and interpolate portions of the light transport that are similar between pixels, reducing variance at the expense of biasing the solution. Irradiance caching [Ward *et al.*, 1988], density estimation [Shirley *et al.*, 1995] and photon mapping [Jensen, 1996] all take this approach.

An innovative global illumination algorithm that has received a lot of attention in recent years is Metropolis Light Transport (MLT) [Veach and Guibas, 1997]. MLT replaces the Monte Carlo integrator used in path tracing with a Metropolis sampler. The main advantage of the Metropolis algorithm over Monte Carlo integration is the ability to preserve the sampling context. This is done by using path mutation to explore path space in a localized way. Thus, when high contribution paths are found, nearby paths will likely be explored as well.

Since the original 1997 paper, researchers have sought to extend the MLT algorithm in a number of ways. Pauly *et al.* [2000] added mutation strategies to MLT that handle participating media such as smoke and fog. Kelemen *et al.* [2002] simplified the MLT algorithm, and increased the mutation acceptance rate, by defining mutation over an abstract space of random numbers rather than the geometric space of ray paths. Other work has focused on the statistical properties of MLT. Szirmay-Kalos *et al.* [1999] characterized the start-up bias problem of the algorithm, and Ashikhmin *et al.* [2001] analyzed its variance.

Combining path tracing and MLT. In this chapter we propose a new global illumination algorithm that combines Monte Carlo path tracing with Metropolis Light Transport mutation strategies. The algorithm works by redistributing the energy of initial path traced samples over the image plane, so we call it Energy Redistribution path tracing, or ERPT.

Our motivation for combining path tracing and MLT comes from the observation that Monte Carlo integration tends to be easy to stratify and control, whereas Metropolis has better convergence properties in many hard sampling situations. However, even though Metropolis sampling may exhibit better convergence properties at low sampling densities,

it will still have a worse *order* of convergence than stratified Monte Carlo sampling if the dimensionality of the integral is low enough. This effect becomes readily apparent when comparing MLT to path tracing for direct lighting.

The ERPT algorithm begins with a set of Monte Carlo samples taken from a path tracer. It then uses a filter step based on path mutation to spread the energy of the MC samples over the image plane in an unbiased way. Unlike the Metropolis algorithm, which uses a single, very long Markov chain of sample locations, our algorithm generates shorter sample chains starting at each path traced sample. We can use short chains because the initial Monte Carlo step eliminates startup bias.

Organization. The remainder of this chapter is organized as follows: section 8.2 reviews a number of ideas leading up to ER sampling, including a brief review of Monte Carlo integration, correlated integrals and energy flow. Section 8.3 describes the ER sampling algorithm in detail. Section 8.4 presents Energy Redistribution Path Tracing, followed by comparisons between ERPT, standard path tracing and MLT in section 9.3. Finally, section 8.6 concludes and suggests ways to improve the algorithm.

8.2 Sampling Issues

This section gives an overview of sampling ideas leading up to ER sampling. We give a brief overview of Monte Carlo integration, and present the concepts of correlated integrals, energy flow, and general and detailed balance. Finally, Metropolis sampling is reviewed, and we show how it relates to energy flow and detailed balance.

8.2.1 Monte Carlo Integration

Consider the problem of integrating the function f over some domain Ω:

$$\int_{\Omega} f(\bar{x})\, d\mu(\bar{x}).$$

We place a bar over the x to indicate that it may be a vector rather than just a scalar quantity. Monte Carlo integration solves this integral by creating a random variable $\mathbf{X}_f$ with expected value equal to the integral:

$$E[\mathbf{X}_f] = \int_\Omega f(\bar{x})\, d\mu(\bar{x}).$$

$\mathbf{X}_f$ is constructed starting with a sampling procedure $\mathbf{S}_p$ which generates samples from Ω according to some probability distribution, p. To complete $\mathbf{X}_f$, a sample location $\bar{x}$ is chosen using $\mathbf{S}_p$, and $\mathbf{X}_f(\bar{x})$ is evaluated

$$\mathbf{X}_f(\bar{x}) = \frac{f(\bar{x})}{p(\bar{x})}. \tag{8.1}$$

This expression forms an unbiased estimate of the integral, which may have a high variance. The usual way to reduce the variance is to average a number of samples taken from $\mathbf{X}_f$. We will refer to the quantity $\mathbf{X}_f(\bar{x})$ as the "initial energy" deposited at point $\bar{x}$.

Equation 8.1 can also be rearranged to obtain values of f in terms of $\mathbf{X}_f$ and p:

$$\mathbf{X}_f(\bar{x})\ p(\bar{x}) = f(\bar{x}). \tag{8.2}$$

We will refer to $\mathbf{X}_f(\bar{x})p(\bar{x})$ as the "expected energy" at $\bar{x}$.

8.2.2 Correlated Integrals

A good number of integration problems involve the estimation of not just one, but a large number of integrals. Path tracing is a particularly pertinent example. Each pixel in a path traced image is an integral that is evaluated using Monte Carlo integration. A standard path tracer evaluates the pixel integrals independently, but it is well known that the integrals have highly correlated integrands (see figure 8.1). The most successful correlated integral solutions tend to exploit the correlation between integrals to reduce variance. In fact, the correlation between pixel integrands is the implied basis for many of the global illumina-

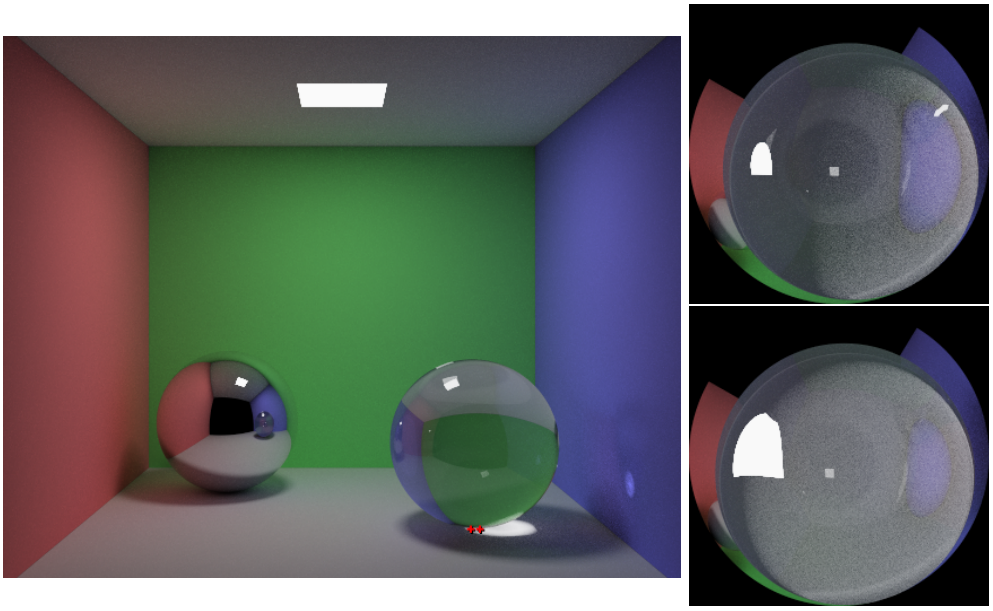


Figure 8.1: Correlated integrals in path tracing. A path tracer must integrate the light incident on surface points seen from the camera. Nearby pixels often have very similar integrands, as can be seen above. The right images show the incident light at the two pixels marked by the red dots in the left image.

tion algorithms currently in use, including irradiance caching [Ward *et al.*, 1988], photon mapping [Jensen, 1996] and Metropolis Light Transport [Veach and Guibas, 1997]. Irradiance caching and photon mapping take adavantage of inter-pixel correlation by caching incident light values, which are later used to approximate parts of the pixel integrals that are difficult to evaluate independently. MLT leverages the correlation between pixel integrals in a different way, using mutation strategies to share integrand information between pixels. Our work takes a similar approach, utilizing path mutations to spread the energy of initial Monte Carlo pixel estimates over the image plane.

8.2.3 Energy Flow

One way to coordinate sampling efforts between correlated integrals is to use a process of *energy flow*. (By *energy*, we simply mean the value of a real-valued function. For a color-valued function, such as an image, energy refers to the luminance.) Energy flow allows a sampling procedure to perform a directed search between similar points in the domains of

correlated integrals. To see why this can be useful, consider two correlated integrals, I_1 and I_2, with domains Ω_1 and Ω_2. Suppose that in the process of sampling, a high contribution point $\bar{x}$ is found in Ω_1 (i.e. $\mathbf{X}_f(\bar{x})$ is large). Since I_1 and I_2 are correlated, it is likely that a high contribution point $\bar{y}$ will exist in a location similar to $\bar{x}$ in Ω_2. Energy flow establishes a connection between points $\bar{x}$ and $\bar{y}$ and transfers some of the energy at $\bar{x}$ to $\bar{y}$. Figure 8.2 shows this graphically. Often, energy flow can be more efficient than standard Monte Carlo sampling because the cost of finding high contribution points is amortized over multiple integrals.

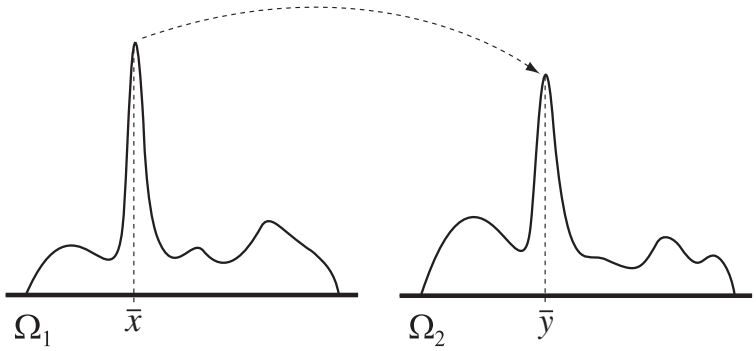


Figure 8.2: Energy flow connects points in correlated integrals, and transfers function energy between them. When done properly, energy flow provides a mechanism for directed searching within the domains of correlated integrals without biasing the integral estimates.

The expected energy flow. In practice, energy flow is created by perturbing or "mutating" a source point, $\bar{x}$, to produce a destination point, $\bar{y}$. (Imagine laying a pipe from $\bar{x}$ to $\bar{y}$ along which energy can flow.) Some of the energy at $\bar{x}$ is then tranferred to $\bar{y}$. Let $T(\bar{x} \rightarrow \bar{y})$ be the *transition probability* from $\bar{x}$ to $\bar{y}$, that is, the probability that $\bar{y}$ is chosen as the destination point given that $\bar{x}$ is the source point. In this situation, the expected flow from $\bar{x}$ to $\bar{y}$ is

$$E[\phi(\bar{x} \rightarrow \bar{y})] = E[\mathbf{X}_f(\bar{x})\ p(\bar{x})\ T(\bar{x} \rightarrow \bar{y})\ q(\bar{x} \rightarrow \bar{y})], \tag{8.3}$$

where $\phi(\bar{x} \rightarrow \bar{y})$ denotes the energy flow from $\bar{x}$ to $\bar{y}$, $E[\ \cdot\ ]$ is the expected value, $X_f(\bar{x})p(\bar{x})$ is the expected energy located at $\bar{x}$ from an initial Monte Carlo estimate (equation 8.2),

and $q(\bar{x} \to \bar{y})$ is the percentage of energy at $\bar{x}$ that flows to $\bar{y}$ once a connection has been established.

General and detailed balance. Astonishingly, energy flow can occur without biasing the integral estimates, as long as certain conditions on the flow amount are met. In particular, the integral estimates will remain unbiased as long as the expected flow of energy out of any point $\bar{x}$ equals the expected flow back in. We will refer to this property as *general balance*. More formally, we say that general balance holds if

$$E\left[\int \phi(\bar{x} \to \bar{y})\, d\mu(\bar{y})\right] = E\left[\int \phi(\bar{y} \to \bar{x})\, d\mu(\bar{y})\right] \quad \forall\, \bar{x}. \tag{8.4}$$

An even stronger constraint that guarantees unbiased-ness is called *detailed balance*. Detailed balance requires that the expected flow between any two points be equal. In other words,

$$E[\phi(\bar{x} \to \bar{y})] = E[\phi(\bar{y} \to \bar{x})] \quad \forall\, \bar{x}, \bar{y}. \tag{8.5}$$

Figure 8.3 illustrates the two kinds of balance graphically.

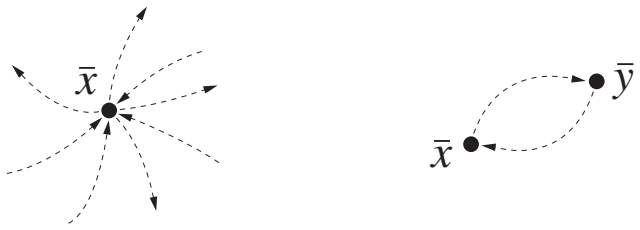


Figure 8.3: General and detailed balance of energy flow. General balance (left) requires the total expected flow out of a point to equal the expected flow back in. Detailed balance (right) requires that the expected energy flow between any two points be equal.

8.2.4 Review of Metropolis Sampling

Here we provide a brief overview of Metropolis sampling, also called the Metropolis algorithm or the Metropolis-Hastings algorithm. We refer the reader to [Pharr, 2003] for a more thorough introduction to Metropolis sampling and its application to rendering.

Suppose that it is desired to evaluate a set of correlated integrals, $I_1 \ldots I_n$. Monte Carlo integration would solve this problem by taking a separate set of samples from each of the integral domains. Sampling the integrals separately can be inefficient, however, since the high contribution points in each integral domain must be found independently. The Metropolis algorithm [Metropolis *et al.*, 1953] takes a different approach that allows sampling efforts to be coordinated among the different integrals. The main idea is to create a probability distribution (pdf) that is proportional to the correlated integrals and then draw samples from this distribution. Metropolis sampling does this by using detailed balance to migrate a single sample through the domains of the correlated integrals, $\Omega_1 \ldots \Omega_n$. As the sample moves, a histogram is kept of its location, and the number of samples deposited in the domain of each integral ends up being proportional to the value of the integral (see figure 8.4).

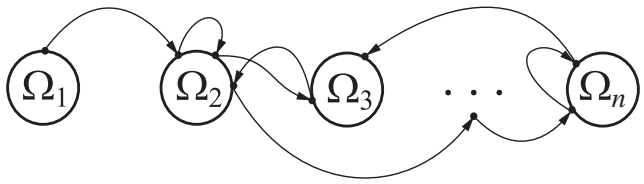


Figure 8.4: The Metropolis algorithm evaluates a set of correlated integrals by moving a sample through the domains of the integrals, tracing out a distribution proportional to the integral values. Sampling efforts are coordinated because the moving sample can jump between similar locations in the domains of different integrals.

Metropolis sampling and detailed balance. Instead of using detailed balance to define the amount of flow in the system, Metropolis sampling uses it to define the *acceptance probability*, the probability that flow will occur given a proposed mutation. When flow

does occur, a single unit of energy is transferred. Thus, the expected number of flow events between any two points, $\bar{x}$ and $\bar{y}$, must be equal for detailed balance to hold, and the ratio of the acceptance probabilities between points $\bar{x}$ and $\bar{y}$ is given by

$$\frac{a(\bar{x} \rightarrow \bar{y})}{a(\bar{y} \rightarrow \bar{x})} = \frac{f(\bar{y})T(\bar{y} \rightarrow \bar{x})}{f(\bar{x})T(\bar{x} \rightarrow \bar{y})}.$$

In practice, it is usually best to maximize the acceptance probabilities, so the actual acceptance probability used is

$$a(\bar{x} \rightarrow \bar{y}) = min\left(1, \frac{f(\bar{y})T(\bar{y} \rightarrow \bar{x})}{f(\bar{x})T(\bar{x} \rightarrow \bar{y})}\right).$$

Limitations of the Metropolis algorithm. Although Metropolis sampling has proven useful in a variety of sampling contexts, it has several limitations that make it difficult to use in a global illumination setting. For example, the Metropolis algorithm is based on the idea of drawing samples from a probability distribution, even though it doesn't explicitly calculate a pdf. This framework tends to be less flexible than working directly with function energies. Metropolis also exhibits the so called "startup bias" problem, which in practice means that it can only be used if a large number of samples will be taken. Furthermore, Metropolis sampling does not stratify well, and its convergence characteristics are hard to analyze.

Metropolis versus ER sampling. As will be seen in the next sections, Energy Redistribution sampling attempts to do away with some of the limitations of Metropolis sampling while maintaining its most powerful features. ER sampling works directly with function energies. It eliminates the startup bias problem because it begins with an unbiased Monte Carlo estimator. Consequently, ER sampling does not require a complete set of mutation strategies to work (ergodicity is ensured by the initial MC samples). Furthermore, since it

is an extension of Monte Carlo integration, ER sampling can leverage stratified sampling and other Monte Carlo variance reduction techniques.

8.3 Energy Redistribution Sampling

In the last section we saw that Metropolis sampling is closely related to the ideas of energy flow and detailed balance. In this section, we describe Energy Redistribution sampling, a new algorithm that exploits these same principles, but directly in a Monte Carlo integration setting.

Energy Redistribution sampling evaluates a set of correlated integrals in a two step process. In the first step, Monte Carlo samples are taken from the integral domains. The second step uses a process of energy flow to redistribute the energy of the MC samples over the domains of the correlated integrals in an unbiased way. Figure 8.5 summarizes this process.

The heart of ER sampling lies in choosing a flow filter to redistribute the energy of the MC samples. This section describes several *balanced energy flow filters* (rules that define energy flow in such a way that general balance holds) We start with the *detailed balance flow rule* and then modify it to produce the *equal deposition flow rule*, which forms the basis of our ER sampling algorithm.

EnergyRedistributionSampling

For each integral domain, Ω_i
  For $j = 1$ to m
    Create an MC sample, $\bar{x}$, in Ω_i according to $\mathbf{S}_p$
    Evaluate $\mathbf{X}_f(\bar{x}) = f(\bar{x})/p(\bar{x})$
    If $\mathbf{X}_f(\bar{x}) > 0$
      Redistribute the energy of $\mathbf{X}_f(\bar{x})$ using a balanced energy flow filter.

Figure 8.5: The energy redistribution sampling algorithm.

8.3.1 The Detailed Balance Flow Rule

To work, ER sampling must define a set of mutation strategies and determine the amount of energy transferred during flow events (q from equation 8.3). The main objective is to define a flow rule that reduces the variance of a set of Monte Carlo estimates while satisfying general balance. As an initial attempt, we follow the lead of Metropolis, and use *detailed* balance directly to define q. Borrowing the standard acceptance probability used by the Metropolis algorithm, we derive

$$q(\bar{x} \rightarrow \bar{y}) = min\left(1, \frac{\mathbf{X}_f(\bar{y})p(\bar{y})T(\bar{y} \rightarrow \bar{x})}{\mathbf{X}_f(\bar{x})p(\bar{x})T(\bar{x} \rightarrow \bar{y})}\right). \tag{8.6}$$

We call this flow rule the *detailed balance* flow rule for obvious reasons. In practice, we mutate the original Monte Carlo samples multiple times and transfer some energy to each of the mutated samples. For example, if the MC sample $\bar{x}$ is mutated n times, it will produce n mutated samples, $\bar{y}_1 \ldots \bar{y}_n$, and the amount of energy transferred to each of the $\bar{y}_i$ will be $X_f(\bar{x})q(\bar{x} \rightarrow \bar{y}_i)/n$.

After flow occurs, any energy that has not flowed out of $\bar{x}$ stays there, and contributes to the integral estimate at that point. Consequently, the detailed balance flow rule is unbiased, but has the serious drawback that a large portion of the energy may not flow anywhere. Thus, if a bright spot exists in one of the initial integral estimates, it will likely continue to exist after flow has occurred, and the variance will remain high.

8.3.2 The Equal Deposition Flow Rule

The main problem with the detailed balance flow rule is that some of the energy of the Monte Carlo estimates does not flow anywhere. A modification that partially solves this problem is to apply the detailed balance rule recursively on the original sample and all the mutated samples that are created. Each time the rule is applied, some of the energy at $\bar{x}$ gets "whittled off" so that after a few flow events very little of the original energy remains

The Equal Deposition Flow Rule

EqualDepositionFlow $(\bar{x},\ e,\ m,\ e_d)$
  $numChains = \lfloor \text{random}(0,1) + e/(m \times e_d) \rfloor$
  For $i = 1$ to $numChains$
    $\bar{y} = \bar{x}$
    For $j = 1$ to m
      $\bar{z} = \text{mutate}(\bar{y})$
      If $q(\bar{y} \to \bar{z}) \ \geq\ \text{random}(0,1)$
        $\bar{y} = \bar{z}$
      Deposit e_d energy at $\bar{y}$

Figure 8.6: Equal deposition flow. $\bar{x}$ is the location of a Monte Carlo sample, $e = \mathbf{X}_f(\bar{x})$ is the initial energy at $\bar{x}$, m is the sample chain length, and e_d is the deposition energy.

at that point. Unfortunately, iterating in this manner results in an exponential growth in the number of samples. In essence, the recursion creates a tree of splitting Markov chains that multiplies the number of samples at each iteration. Another problem with this approach is that the amount of energy in the mutated samples will vary wildly, leading to increased variance.

Equal deposition. A better solution is to create linear Markov chains that emanate from each MC sample point rather than splitting chains. Sample chains can be prevented from splitting by probabilistically keeping all of the energy at the current location or transferring all of it to the mutated location. These chains of samples create a set of unbiased estimates of the correlated integrals, one for each flow iteration. (Note that in order for the estimates to remain unbiased, the sample chains must all be the same length.) To reconstruct the integrals, the ER sampler desposits an equal fraction of the original energy imparted to the sample chains after each iteration. Hence, we call this rule the *equal deposition* flow rule.

To see that the equal deposition flow rule is unbiased, notice that the sample chains, taken as a whole, form n unbiased estimates of the correlated integrals. Although each sample is processed separately, the end result is nothing more than the average of the n unbiased estimates, and is therefore unbiased as well.

Equal deposition and Metropolis sampling. In the special case where all of the sample chains start with the same amount of energy, the equal deposition flow rule becomes a form of Metropolis sampling that does not exhibit startup bias. To put it another way, since the amount of energy deposited by each sample chain on each flow iteration is equal, the sampler is implicitly taking draws from a distribution proportional to f, which is exactly what Metropolis does. This process is very similar to how Veach and Guibas [1997] eliminate startup bias, except that in addition, our algorithm uses the Monte Carlo samples to provide initial coverage of the entire sample space.[1] Pseudo-code for this form of the algorithm is given in figure 8.6.

Starting the sample chains with the same energy is not the only option, however. Any number of sample chains (even zero) can be started from a given MC sample as long as the expected energy imparted to the chains equals the sample's initial energy. Another point is that there are situations in which the sample chains do not all have to have the same length. This happens when the set of available mutations splits the domains of the integrals into disjoint sets. As long as the chains in these disjoint sets are all the same length, the integral estimates will remain unbiased.

The deposition energy. An essential part of the equal deposition flow rule is the *deposition energy*, or how much energy will be deposited after flow events. To determine the deposition energy, we estimate the expected energy of MC samples within the sampling domain and divide by the desired number of mutations per integrand as follows:

$$e_d = e_{ave}/k, \tag{8.7}$$

where e_d is the deposition energy, e_{ave} is the average energy of a number of samples taken with the Monte Carlo sampler, and k is the desired number of mutations per correlated

[1] These results also imply that if Veach's resampling algorithm to eliminate startup bias is used, and multiple sample chains are created, we cannot use the energy of the original MC samples to determine the sample chain length without reintroducing bias.

integrand. Note that a poor estimate of e_{ave} will not change the accuracy of the algorithm, only its run time. Contrast this with Metropolis sampling, in which a value similar to e_{ave} acts as a global scale factor for the integral estimates.

8.4 Energy Redistribution Path Tracing

This section gives the details of our Energy Redistribution path tracing algorithm, which we will refer to as ER path tracing or ERPT. Conceptually, the algorithm is nothing more than Energy Redistribution sampling with a path tracer as the Monte Carlo sampler. Figure 8.7 gives pseudocode for ERPT. Notice that the algorithm is quite similar to a path tracer. The only difference is that the step that deposits the energy onto the image plane has been replaced by a balanced energy flow filter. The remainder of this section describes several essential details of ER path tracing. We give a brief overview of the rendering equation in the context of path tracing, and then discuss the idea of Monte Carlo path density. Next we give a set of rules to determine the relative Monte Carlo sampling density between two ray paths, which is needed to calculate the value of q. We also discuss the specific mutation strategies used by our algorithm.

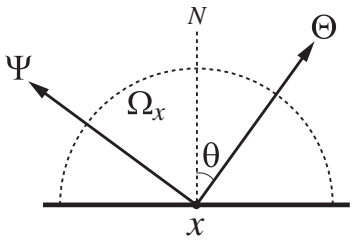
Energy Redistribution Path Tracing

```
ERPathTracing(m_c)
    Determine the deposition energy, e_d    // equation 8.7
    For each pixel in the image
        For j = 1 to n
            Create a path, x̄, in the current pixel
            X_f(x̄) = f(x̄)/p(x̄)   // evaluate the path
            If  X_f(x̄) > 0
                EqualDepositionFlow(x̄, X_f(x̄), m_c, e_d)
```

Figure 8.7: The ER path tracing algorithm. The code above specifies the equal deposition flow rule, but any balanced energy flow filter could be used. The value m_c is the user-specified sample chain length. In practice, we found values between about 100 and 1000 to work well.

8.4.1 Ray Paths and Monte Carlo Path Density

A path tracer creates an image by sampling the incoming light over the area of each pixel on the image plane. This incoming light is described by the *rendering equation* [Kajiya, 1986], one form of which is given below:



$$L(x \rightarrow \Psi) = L_e(x \rightarrow \Psi) + \int_{\Omega_x} L(x \leftarrow -\Theta) f_r(\Psi \leftrightarrow \Theta) |\cos\theta| d\omega_\Theta. \tag{8.8}$$

In brief, the rendering equation describes the light coming from a surface point x in a particular direction, Ψ, $L(x \rightarrow \Psi)$. The term $L_e(x \rightarrow \Psi)$ is the light emitted directly from from x in direction Ψ, Ω_x is the hemisphere above point x, and $f_r(\Psi \leftrightarrow \Theta)$ is the BRDF function at x. We refer the reader to [Dutré *et al.*, 2003] for a complete discussion of the various forms of the rendering equation.

A path tracer samples the rendering equation by means of ray paths that connect the eye point to a light source through a number of scattering events (reflections or refractions). To build a path, a path tracer sends out a ray from the eye point into the scene. The path tracer then extends the ray through a number of scattering events to produce an *eye subpath*, using a probabilistic sampling function to choose the outgoing direction at intersection points. We will call this function p_d. The path tracer may connect the eye subpath to a light source in one of two ways. First, p_d may happen to choose a direction that hits a light source. We will refer to this kind of path as an *implicit path*. Second, the path tracer may connect the eye subpath directly to a point on a light source. We will refer to paths created in this way as *explicit paths*.

Since the ray paths created by a path tracer are Monte Carlo samples of the rendering equation, the path tracer evaluates them in such a way that the expected value of the

paths that contribute to a given pixel is equal to the pixel brightness. To see how this is done, consider the path in figure 8.8 below that connects the eye point to a light source:

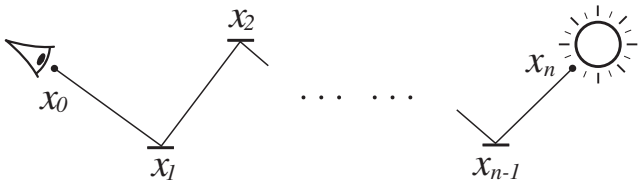


Figure 8.8: A ray path.

To form an unbiased estimate of the light reaching the eye along direction $x_1 \rightarrow x_0$, the MC sampler in a path tracer multiplies the pertinent terms of the rendering equation together (i.e. f_r, $cos\theta$ and L_e), and divides by the probability that the path was generated by the sampler. For an implicit path, the estimate is given by

$$\frac{L_e(x_n \rightarrow \Psi_n)}{p_{length}(n)} \times \prod_{i=1}^{n-1} \frac{f_r(\Psi_i \leftrightarrow \Theta_i)\ |\cos\theta_i|}{p_d(\Psi_i \rightarrow \Theta_i)}, \tag{8.9}$$

where Ψ_i and Θ_i are the incoming and outgoing directions at x_i, θ_i is the angle between Θ_i and the surface normal at x_i, $p_{length}(n)$ is the probability that the MC sampler chose to create a path of length n, and p_d is defined with respect to solid angle.

Explicit paths are evaluated similarly, except that the term $p_d(\Psi_{n-1} \rightarrow \Theta_{n-1})$ is replaced by a term that converts area sampling on the surface of a light source to sampling over the solid angle. Suppose that the ray path in figure 8.8 was made by connecting an eye subpath to light k. The path would then be evaluted

$$\frac{L_e(x_n \rightarrow \Psi_n)}{p_{length}(n)} \times \prod_{i=1}^{n-2} \frac{f_r(\Psi_i \leftrightarrow \Theta_i)\ |\cos\theta_i|}{p_d(\Psi_i \rightarrow \Theta_i)} \times \frac{f_r(\Psi_{n-1} \leftrightarrow \Theta_{n-1})\ \ |\cos\theta_{n-1}\cos\phi_n|}{p_{light}(k)\ p_{area}(x_n)\ d^2}. \tag{8.10}$$

As can be seen, the first two terms are nearly identical to the implicit case. The third term converts sampling over the surface of light source k to sampling over the solid angle from point x_{n-1}. The new terms in the expression are as follows: $\cos\phi_n$ describes the angle

between the normal at point x_n on the light source and the incoming direction Ψ_n; d is the distance between x_{n-1} and x_n; $p_{light}(k)$ is the probability that light k was chosen by the MC sampler, and $p_{area}(x_n)$ is the probability that point x_n was chosen on the light source with respect to surface area.

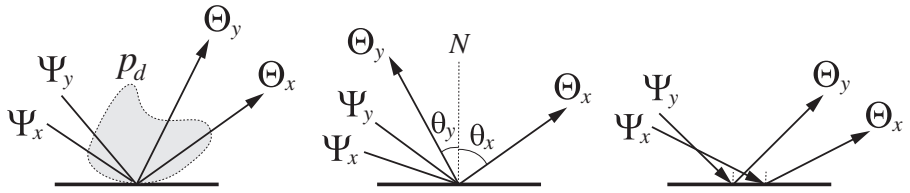
Monte Carlo path density. The product of all of the terms related to probability in a ray path (p_{length}, p_d, p_{light} and p_{area}) can be thought of as the *path density* in path space with respect to the given MC sampler. In section 8.4.2 we will use this fact to compute the relative sampling density in different parts of path space.

8.4.2 Mutation and Changes in Path Density

ER path tracing relies on two fundamental sampling steps: an initial Monte Carlo step, and an Energy Redistribution step that uses path mutation. To allow energy flow between a path $\bar{x}$ and a mutated path $\bar{y}$ during the energy redistribution step, the ER sampler must compute the ratio of the path density at $\bar{y}$ to the path density at $\bar{x}$ with respect to the path tracer's sampling routines ($p(\bar{y})/p(\bar{x})$). Cline and Egbert [2005] give a set of rules that describe these path density changes for ideal diffuse and specular surfaces sampled in a particular way. Here we give an extended set of rules that are valid for arbitrary BRDFs for the mutation types that we use.

Rule 1: Changes to pixel coordinates. Explicit changes to the pixel coordinates of a path do not change the relative path density, unless the MC sampler samples different image coordinates with different densities. Note that this rule is only applied if the pixel coordinates of the path are explicitly manipulated. Rule 5 handles incidental changes to the pixel coordinates of a path.

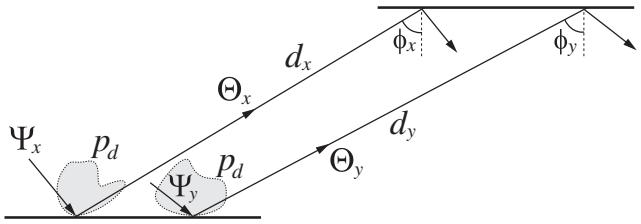
Rule 2: Changes to directions. When the MC sampler chooses an outgoing direction at a surface, it does so according to the probability distribution p_d, which was described in section 8.4.1. Perturbing an outgoing direction at a surface changes the path density in a manner proportional to the relative density of samples taken by the MC sampler in the original outgoing direction and the mutated direction. In the case of a diffuse surface sampled with a cosine-weighted distribution about the normal, the density change is proportional to the ratio of the cosines of the two angles. In the case of an ideal specular surface, perturbing the outgoing direction to lie in the specular direction changes the path density proportional to p_s, the probability that the MC sampler would choose to send a ray in the specular direction. (The value of p_s might change if, for example, the MC sampler uses a Fresnel term to decide whether to send a reflection or refraction ray.) Equation 8.11 summarizes all three cases.



$$\frac{p_d(\Psi_y \to \Theta_y)}{p_d(\Psi_x \to \Theta_x)} \qquad \frac{|\cos\theta_y|}{|\cos\theta_x|} \qquad \frac{p_s(\Psi_y \to \Theta_y)}{p_s(\Psi_x \to \Theta_x)} \tag{8.11}$$

Rule 3: Connecting points in the middle of a path. Connecting two non-specular vertices in a path is a way of sampling surface areas instead of directions, and thus we must convert between area sampling and directional sampling. If we are sampling directions according to a cosine-weighted distribution about the surface normal, the density change is proportional to the familiar geometry term $|\cos\theta\cos\phi/d^2|$. On the other hand, if some

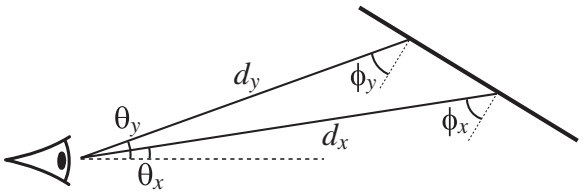
other method is used to sample directions, the density change replaces the $\cos\theta$ term with with the probability of sampling different directions, as shown in equation 8.12.



$$\frac{p_d(\Psi_y \to \Theta_y)|\cos\phi_y|}{d_y^2} \times \frac{d_x^2}{p_d(\Psi_x \to \Theta_x)|\cos\phi_x|} \tag{8.12}$$

Rule 4: Connecting points to light sources. For explicit paths, connecting to an established or perturbed point on a light source does not change the path density if the light source is sampled uniformly with respect to area. Otherwise the density change is proportional to the ratio of probabilities of choosing the mutated and original points on the light. Implicit paths, on the other hand, incur the same change in path density as connecting points in the middle of a path.

Rule 5: Connecting a point to the viewer. Assuming a pinhole camera model, if a connection is made in which one of the vertices is the eye point, the density change is proportional to the modified geometry term $|\cos\phi/(d^2\cos^3\theta)|$. The actual change in density for this case is given in equation 8.13.



$$\frac{|\cos\phi_y|}{d_y^2|\cos^3\theta_y|} \times \frac{d_x^2|\cos^3\theta_x|}{|\cos\phi_x|} \tag{8.13}$$

Applying the density change rules. The path density change rules just described must be applied beginning at the eye point regardless of the manner in which the mutated path was generated. This is a consequence of the fact that we are trying to capture path density changes **with respect to the MC sampler used by the path tracer.**[2] To compute the total density change between two paths of the same length, we apply all of the pertinent rules, and multiply their results together. Appendix 8A gives several examples of how to apply the rules.

8.4.3 Mutation Strategies

Our implementation uses two mutation types, which correspond to lens and caustic perturbations as described in [Veach and Guibas, 1997]. We are able to get away with such a small set of mutations because the path tracing step provides complete coverage of path space, and roughly distributes path energy over the image plane. Besides our descriptions here, Veach and Guibas [1997] and Cline and Egbert [2005] provide descriptions of lens and caustic perturbations as well as other mutation types.

Lens perturbations. A lens perturbation is a mutation that creates a new path $\bar{y}$ from an existing path $\bar{x}$ beginning at the eye point. To start the mutation, the pixel coordinates of $\bar{x}$ are perturbed by a random amount on the image plane.[3] A ray is cast from the eye point through this new pixel coordinate, and the new eye subpath is propagated through the same number and types of specular bounces as the original path, arriving at a non-specular vertex. If the next vertex in the original path is non-specular, $\bar{y}$ is completed by connecting the eye subpath directly to the next vertex in the original path. If the next vertex is specular, however, the outgoing direction from the diffuse vertex is perturbed, and the eye subpath is extended through another specular chain looking for two non-diffuse vertices in a row.

[2] It is not necessary to use the same MC sampler as the path tracer. Any valid sampler will work, as long as it is used to compute all parts of q.

[3] Our implementation mutates uniformly within a small square (9×9 pixels) centered on the current location.

This process repeats until either two non-diffuse vertices or the light source are found. Appendix 8A gives an example lens perturbation, and shows how to compute the relative path density between $\bar{x}$ and $\bar{y}$.

Caustic perturbations. Caustic perturbations are created in much the same way as lens perturbations, except that they start at the light source, or second diffuse vertex in the path (from the eye point). For example, consider the path $LSSDE$. The caustic mutation starts by perturbing the direction $L \rightarrow S$ by a random angle.[4] The new light subpath is propagated through two specular bounces, and arrives at a non-specular vertex, producing the light subpath $LSSD\ldots$. This subpath is then connected directly to the eye point. Note that this type of mutation can only be used on paths in which the eye point is connected to a non-specular vertex. Appendix 8A gives an example of a caustic perturbation and the path density change that it incurs.

Mutation probabilities. In practice, our strategy is to choose caustic perturbations exclusively for paths of the type $L\ldots SDE$, and lens perturbations in all other cases. Another reasonable strategy would be to choose randomly between the two mutation types when they are both valid.

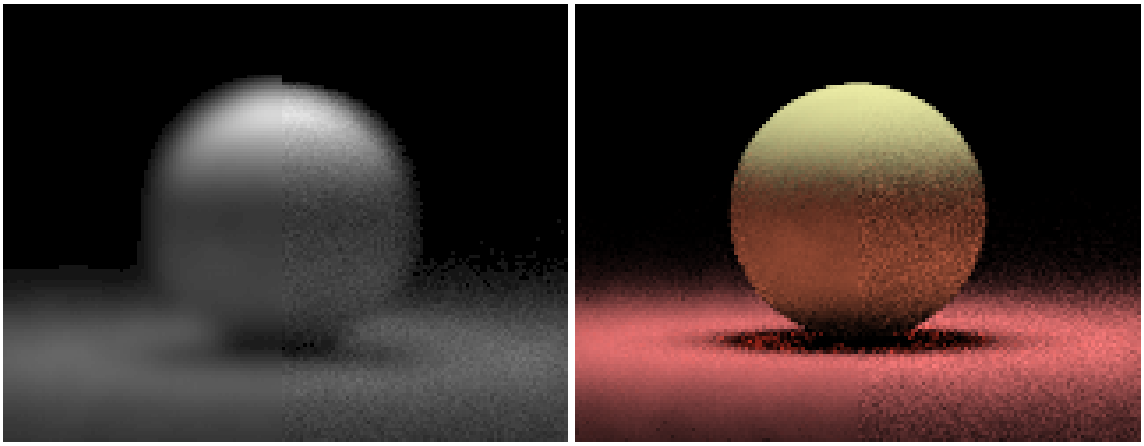
8.4.4 Noise Filtering

Since ER path tracing is a stochastic process, it can naturally introduce noise into a rendered image. Here we describe two filters that can significantly reduce the noise of images produced by ERPT. Although they are theoretically biased, we have found the filters to be effective at eliminating noise while producing few visible artifacts.

[4] When perturbing the angle, we follow the formulation of Veach and Guibas [1997], and mutate in an exponential distribution between 0.0001 and 0.1 radians. Cline and Egbert [2005] describes this procedure in detail.

Proposed mutations noise filtering. One of the main causes of noise in ERPT is an imbalance in the number of potential flow events into different pixels. We can compensate for this imbalance by keeping a tally of the number of proposed mutations to each image pixel. This "proposed mutations" image is then blurred to produce an approximate expected number of proposed mutations into each pixel. Our current implementation uses a box filter to compute this "expected proposed mutations" image. The rendered image is then de-noised by scaling the pixel values by the ratio of the expected number of proposed mutations at the pixel to the actual number. Figure 8.9 shows the "proposed mutations" and "expected proposed mutations" images for a simple scene along with the effect of applying the filter.

We have been quite pleased with the results of this filter, but there may be room for improvement. For example, a median filter might be a better choice than a box filter to smooth the "proposed mutations" image. Also, since the expected proposed mutations



(a) (b)

Figure 8.9: Proposed mutation noise filtering. (a) The right half of the image shows the number of proposed mutations into each pixel, the "proposed mutations" image. The left half of the image was produced by convolving the proposed mutations image with a 7×7 smoothing kernel. This is the "expected proposed mutations" image. (b) The left half of the image has been smoothed with our proposed mutations noise filter, and the right half has not. In spite of the large kernel applied to determine the expected number of proposed mutations, edges in the filtered image remain sharp. Very large kernel sizes can produce ringing artifacts, however.

image is essentially a convolution of the luminance image, it may be better to blur the unfiltered luminance to produce the "expected proposed mutations" image.

Consecutive sample filtering. A second noise filter that works well in practice is to refuse to accumulate more than a small number of consecutive samples on a given pixel, say 10 or 20. During energy redistribution, the ERPT sampler counts how many times in a row a sample chain deposits energy onto the same pixel. Once the maximum number has been reached, the sampler continues mutating the sample chain, but it throws away any energy that would normally be deposited until the sample chain migrates off of the offending pixel, at which time energy deposition begins again. This filter tends to clean up speckles that occur when the sampler gets stuck on a given pixel.

8.5 Results

This section demonstrates different aspects of the ERPT algorithm, and compares ERPT to standard path tracing and MLT. To make the comparison as fair as possible, the functions that mutate paths are shared between MLT and ERPT.

Comparison of flow rules. We experimented with the different flow filters described in section 8.3, plugging them in as the flow filter for our ER path tracer. One of these experiments is summarized in figure 8.10. In virtually every case, the equal deposition flow rule was superior to the others, so we use it for all of the other examples in the paper.

Sample chain length. We have found that the sample chains emanating from the Monte Carlo samples need to be fairly long to produce good results. Values between about one hundred and one thousand seem to work well. As a general rule, very short chains produce an uneven appearance, and very long chains start to lose the stratifying properies of the initial MC samples as more and more of the initial samples do not have any sample chains assigned to them. Figure 8.11 shows the effect of varying the sample chain length.

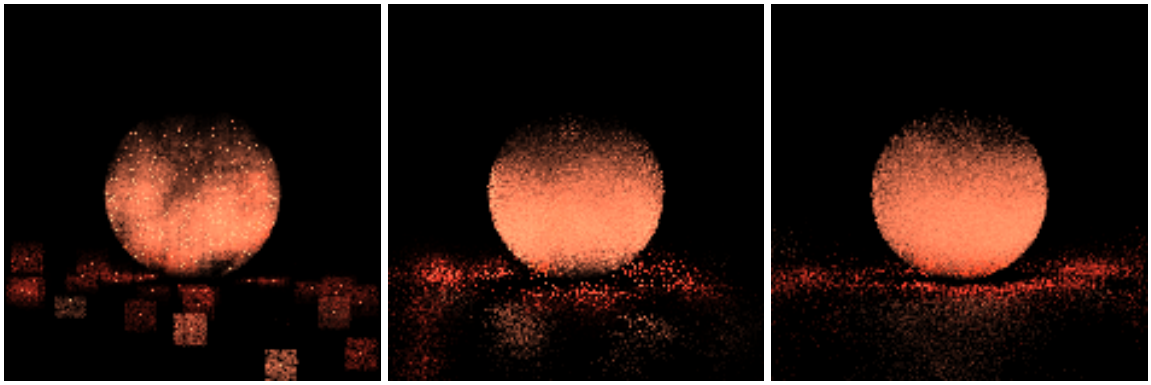


Figure 8.10: The effect of different flow rules. The images show the indirect lighting from the scene in figure 8.9, using different flow filters for the Energy Redistribution step of ERPT. (Left) The detailed balance flow rule works poorly because energy cannot migrate away from the MC sample site. (Middle) An iterated version of detailed balance works better, but the splitting sample chains produced by the rule are too short to spread evenly over the image plane. (Right) Equal deposition flow, while not perfect at this sample density, spreads the energy of the MC samples more evenly than the other two rules.

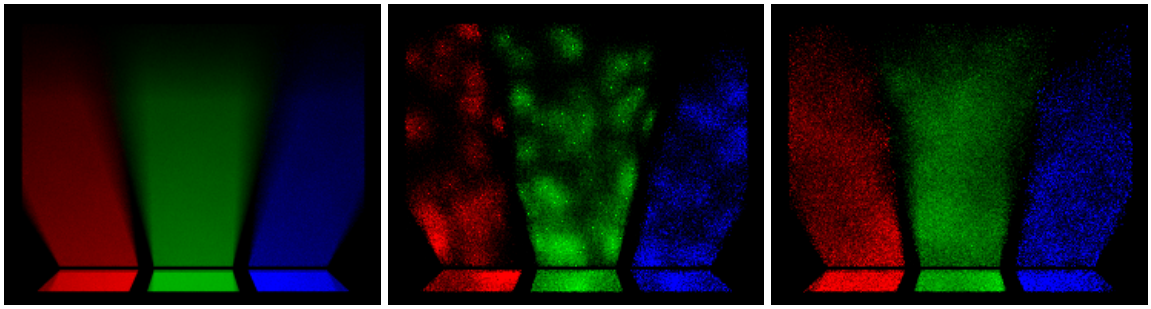


Figure 8.11: Sample chain length. In the scene above, light reflects off of three colored mirrors onto a diffuse wall. The desired image (left) is reproduced with ERPT using sample chains of length 10 (middle), and 100 (right).

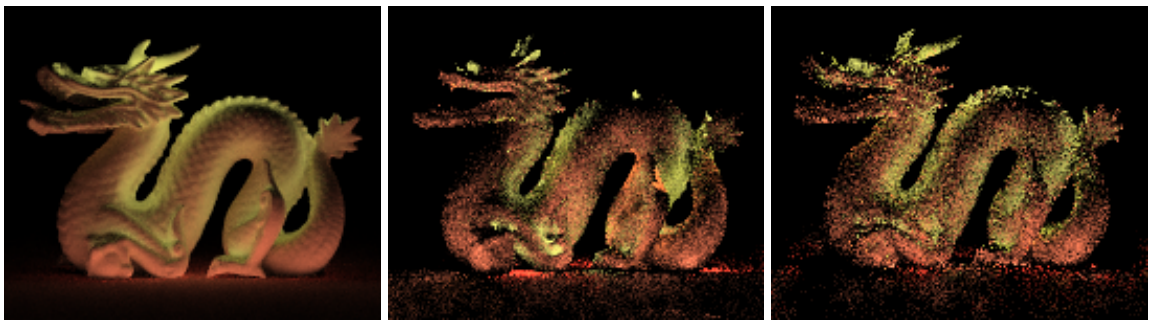
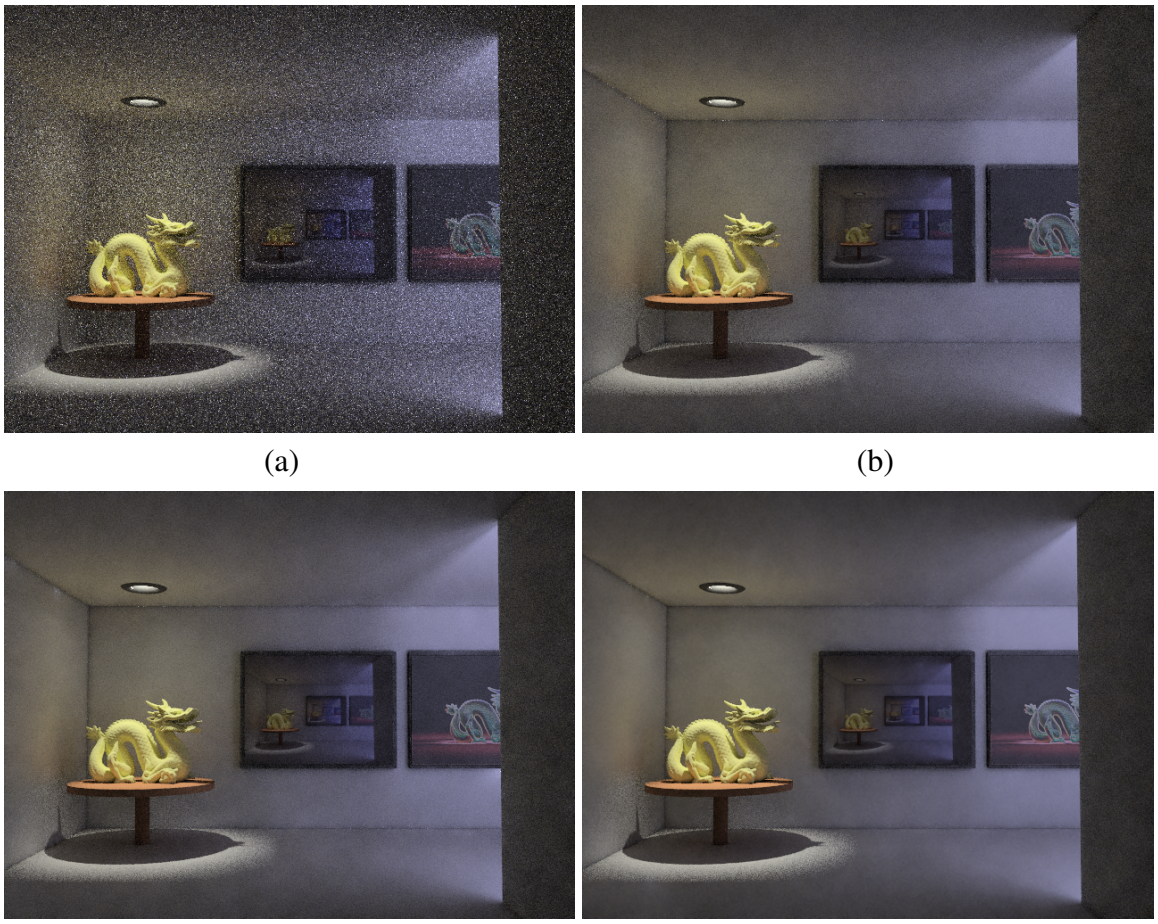


Figure 8.12: Comparison of stratification over the image plane between MLT (middle) using 2 samples per pixel and ERPT (right) using one MC sample and one mutation per pixel.

Approximate stratification of path space. In most lighting situations, MLT and ERPT produce similar results. However, ERPT tends to stratify a little better over the image plane. For example, consider the images in figure 8.12 showing indirect lighting of a yellow dragon placed on a red ground plane. The left image gives the desired result, computed by path tracing using a large number of samples. The middle image was produced by MLT using two mutations per pixel, and the right image was produced with ER path tracing using one Monte Carlo sample and one mutation per pixel. Note that the ERPT image more faithfully reproduces the contours of the dragon than MLT, and the lighting in the MLT image has more bright artifacts. This is because ERPT creates an initial distribution of energy that covers path space evenly. By contrast, MLT must rely on large mutations to fully account for all contributing paths.

Indirect lighting example. Figure 8.13 shows renderings of a gallery scene with strong indirect lighting, taking approximately equal time, for path tracing, MLT and ERPT. The only direct illumination visible in the scene comes from the overhead spot light shining on the dragon. Image (a) was computed with standard path tracing. Since path tracing cannot coordinate sampling efforts between pixels, it wastes a lot of time sampling unimportant regions of the path space, resulting in a very noisy image. Image (b) was produced with Metropolis Light Transport. MLT is able to coordinate sampling efforts between pixels, producing a better image; however, some noise is still visible. The bottom row shows ERPT without (c) and with (d) the noise filters described in section 8.4.4. ERPT without the noise filters achieves a slightly better result than MLT in this scene. Adding the flow filters further reduces the noise, producing a much smoother result.

Difficult caustic lighting. Figure 8.14 compares ERPT to MLT and path tracing for a difficult lighting situation in which a large portion of the lighting comes from implicit caustic paths seen through a glass surface. Even the "direct" lighting on the torus is made up of these paths. (b) Path tracing produces a very noisy image. (c) MLT does much better,



(a) (b)

(c) (d)

Figure 8.13: Comparison of a scene with strong indirect lighting. All render times are approximately 20 minutes. (a) Standard path tracing with 84 paths per pixel has a hard time finding the indirect lighting paths, resulting in a very noisy image. (b) MLT with 200 mutations per pixel produces a much better result, but some noise is still visible. (c) ERPT with 36 MC samples and 200 mutations per pixel achieves a slight improvement over MLT. (d) Adding the noise filters described in section 8.4.4 to ERPT removes most of the remaining noise.

but still results in a splotchy appearance. A more full set of mutation strategies might remedy this problem; however, ERPT using the exact same set of mutations (d) has a much smoother appearance. This is because the deposition energy of any given sample chain is limited. Image (e) shows the effect of applying the noise filters from section 8.4.4. The top image (a) shows a higher quality ERPT rendering of the same scene.

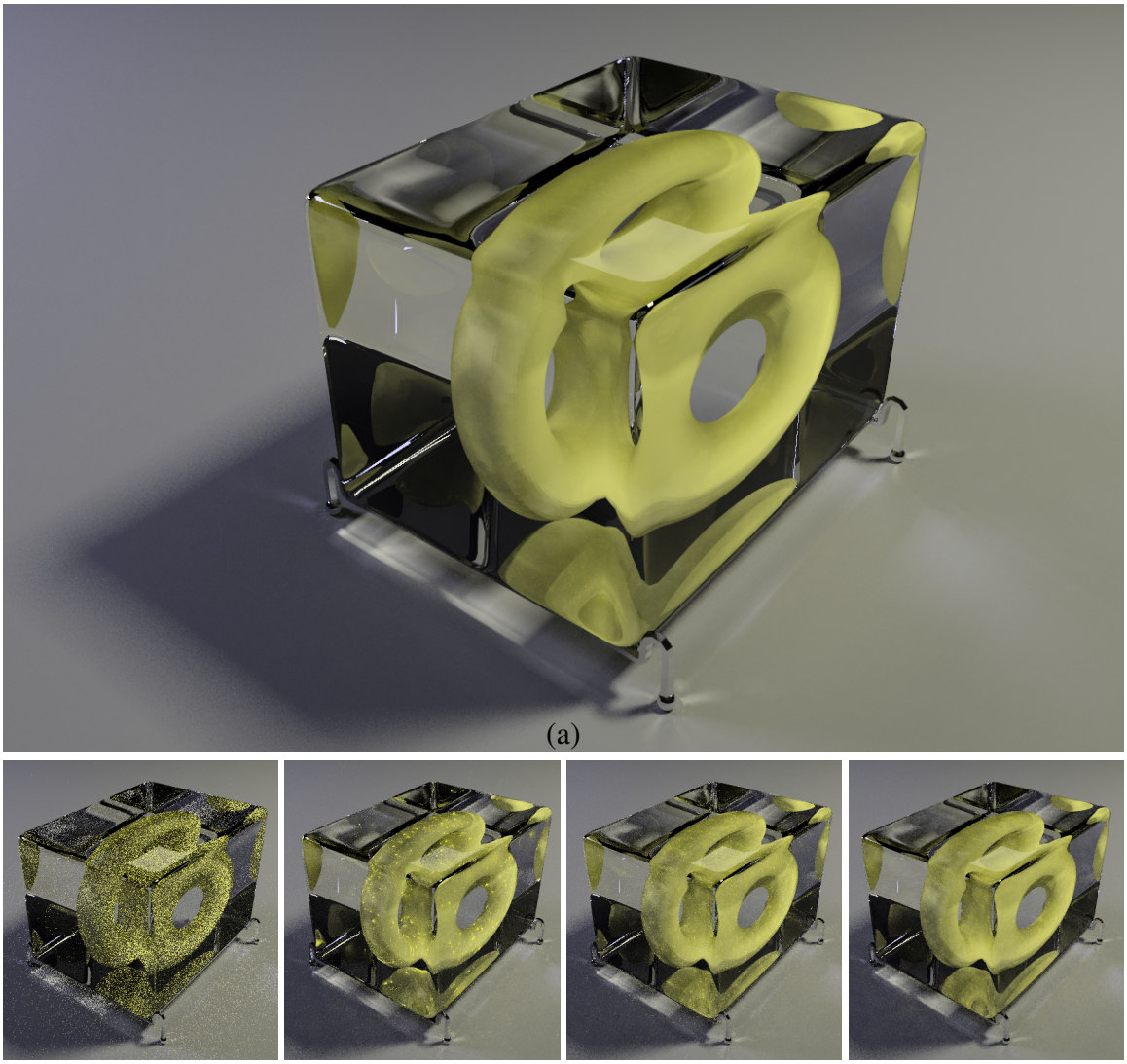
Objective Image Quality. Figure 8.15 shows graphs the convergence of ERPT against path tracing and MLT for the ball, gallery and paperweight scenes. In two of the three examples, ERPT converges slightly faster than MLT, likely because of its improved stratification. Once again, however, subjective image quality is better as well, particularly in the paperweight scene, which is very blotchy in the MLT rendering.

8.6 Conclusion and Ideas for Future Study

This paper presented Energy Redistribution path tracing as an algorithm to solve the general global illumination problem. The algorithm has at its core a novel sampling technique called Energy Redistribution sampling, which can efficiently solve correlated integral problems. We compared images generated by ER path tracing to images created with standard path tracing and Metropolis Light Transport, and demonstrated several situations in which the new algorithm outperforms standard path tracing and MLT.

In addition to the algorithmic contributions of the paper, we feel that some of the most important contributions of this work are pedagogical. The concepts of correlated integrals and energy flow offer profound insights into the inner workings of Metopolis as well as ER sampling, and a new perspective from which to view numerical intergration problems in general.

We have only scratched the surface in comparing ER to Metropolis sampling. Because it works directly with function energy, we have found the ER sampling framework to be more malleable than Metropolis sampling. For example, ER sampling can be easily



(a)

(b) (c) (d) (e)

Figure 8.14: Difficult caustic lighting. In this scene, a large portion of the light transport comes from implicit "caustic" paths. (b) Path tracing with 100 paths per pixel produces a very noisy image. (c) MLT, using 100 mutations per pixel, gets stuck on some of the caustic paths, producing a splotchy appearance. (d) ERPT using 36 MC samples and 50 mutations per pixel. Although some bright spots are visible, they are much less pronounced than in the MLT case. This is so despite the fact that both algorithms use the same mutation strategies. (e) Adding the noise filters to ERPT removes most of the small speckles in the image. (a) A high quality ERPT rendering of the scene using 192 MC samples and 800 mutations per pixel, again using the noise filters. The bottom row images were rendered at a 640×480 resolution in about fifteen minutes, and the top image was rendered at a resolution of 1200×800 in about seven and a half hours on a 3.2 Ghz Pentium 4.

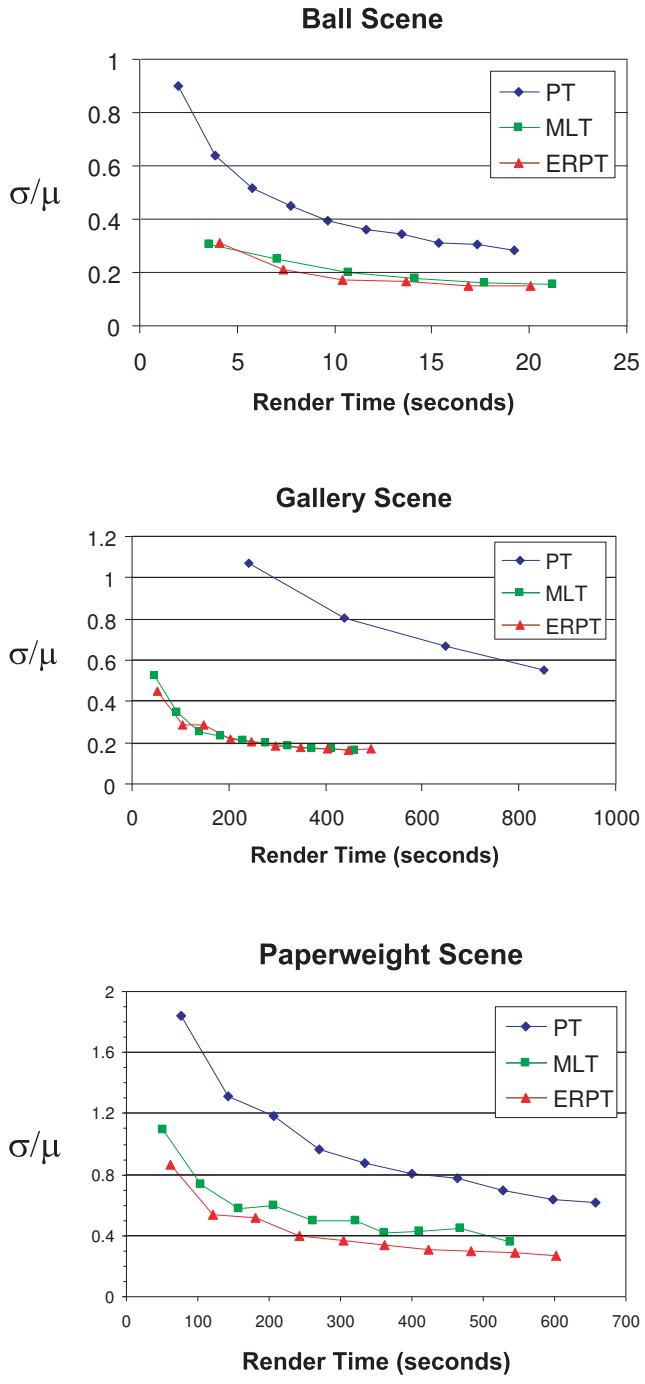


Figure 8.15: Convergence properties of ERPT compared to path tracing and MLT.

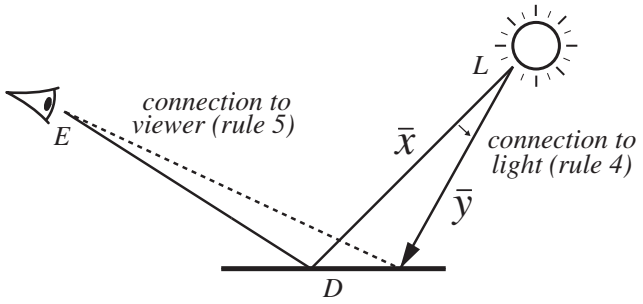
adapted to handle negative-valued functions as well as all positive ones. Are there domains besides global illumination that would benefit from ER sampling rather than Metropolis sampling? In what situations does ER sampling work better?

There are a number of questions that remain in regards to ER sampling flow rules. For example, are there ways to define better flow rules based on the less restrictive general balance condition rather than detailed balance? What is the optimal tradeoff between Monte Carlo samples and ER samples? Can flow rules and mutation strategies be defined which stratify better?

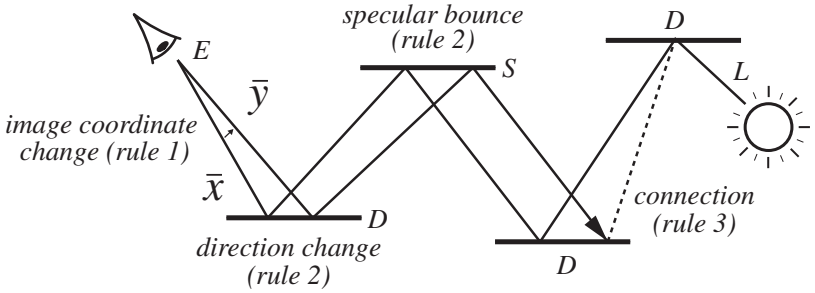
Noise filtering also seems to be a good avenue for further exploration. We were quite pleased with the results of the two simple filters presented in the paper, and are currently looking at the use of other simulation statistics besides "proposed mutations" to reduce the noise. Finally, we note that the noise filters that were defined in this paper are biased. Ideally, we would like to create effective noise filters that are unbiased.

Appendix 8A: Computing Path Density Change

Since computing the change in path density is essential to ERPT, we give several examples of how to do it here. As a first example, let us assume that we are mutating a path of the form *LDE* using a caustic mutation, as shown below:



The direction starting at the light source is perturbed, and a ray is cast from the light source in this direction, creating the light subpath *LD*…. This subpath is connected to the eye point, once again creating a path of the form *LDE*. Note that even though the mutation was generated starting at the light source, the path density change rules must be applied starting at the eye point. From the eye point, we apply the following rules: A connection was made from the eye point, so we apply rule 5. Now note that even though the mutation was generated by perturbing a direction from the light source, from the point of view of the path tracer, we must make a direct connection to the light source from the *D* vertex. Thus, we apply rule 4. As another example, consider a lens subpath mutation on a path of the form *LDDSDE*:



In this case, the pixel coordinates of the ray from the eye are changed, and we cast a ray in the new direction. The ray hits a diffuse surface, which is followed by a specular surface.

In this case, the outgoing direction from this D vertex is perturbed, and extended through a specular bounce to produce the eye subpath of the form $\ldots DSDE$. This eye subpath is then connected directly to the next vertex in the path, to once again produce a path of the form $LDDSDE$. Now we apply the density change rules. First, we changed the pixel coordinates, so we apply rule 1. Next, we perturbed the outgoing direction, so we apply rule 2. Then, we extended the path through a specular bounce, so we apply rule 2 again. Finally, we connected two diffuse vertices in the middle of the path, so we apply rule 3.

Chapter 9

Sample Swarming

A version of this chapter was submitted to the 2007 Eurographics Symposium on Rendering.

Additional authors for the submission include Parris Egbert and Daniel Adams.

Abstract. Monte Carlo rendering algorithms generally rely on some form of importance sampling to evaluate the measurement equation. However, most of these importance sampling methods only take local information into account, so that the actual importance function used may not closely resemble the light distribution in the scene. In this paper, we describe a sampling technique that augments existing local importance functions with tabular "swarming maps" that direct sampling towards undersampled regions of path space. The swarming maps are constructed lazily, relying on information gathered during the course of sampling. When a bright sample is found, the swarming maps update to significantly increase the probability of sampling near the same location in path space. Subsequent samples then naturally "swarm" to that part of path space, exploring the bright region. We show that swarming maps can be effective in a number of common rendering situations.

9.1 Introduction

9.1.1 Dimensionality of the Rendering Problem

The core goal of almost all global illumination algorithms is to measure the light transport in a virtual scene that leads to a photorealistic image. Each pixel in such an image must record the output of an idealized light sensor (e.g. a CCD or tiny section of film) during the camera exposure. Mathematically, the output from one of these pixel sensors can be described by the *measurement equation*, an integral of the incoming radiance striking the sensor through the camera lens. In its full form, the measurement equation is a five-dimensional integral of incoming radiance, with two dimensions for the area of the pixel sensor, two for the lens, and one for time.

Beyond these five dimensions, the incoming radiance that strikes the lens must be evaluated using the *rendering equation*, another high dimensional integral. To account for participating media and surface reflection, the rendering equation must integrate the change in radiance along a ray's trajectory, accounting for the radiance contributions of every possible scatter point along the way. This leads to a three dimensional integral for each level of scattering. Thus, to account even for single scattering, the measurement equation must be eight dimensional, with three more dimensions being added for each additional scattering event.

9.1.2 Monte Carlo, World Space and Preimage Space

Owing to the complexity of the measurement equation, analytical solutions are generally not possible, and most renderers resort to some form of Monte Carlo sampling. In world space, a Monte Carlo sample of the measurement equation consists of a ray path that connects a point on a pixel sensor to a point on a light source, through the camera lens and a number of light scattering events. However, a ray path sample generally starts out as a uniformly distributed point in $[0,1)^n$. Each coordinate of the point represents some deci-

sion that must be made while constructing the ray path. For example, one coordinate may represent the instant in time for which the ray path is constructed, two more may represent a point on the pixel sensor, two more a point on the lens, and so forth. In this paper, we will refer to a ray path sample as being in *world space* or *path space*, while the term *preimage space* will be used to refer to points in $[0,1)^n$, since this is the preimage (domain) of the function that maps points to ray paths.

9.1.3 Evaluating a Ray Path Sample

Once constructed, a Monte Carlo ray path sample can be evaluated as the product of a number of terms, each one corresponding to a decision that the renderer made while building the path. Of course, the details of the product will vary from renderer to renderer, but a typical evaluation might look something like:

$$\widehat{L}(\mathbf{x}) = \frac{time}{p_{time}} \times \frac{pixel}{p_{pixel}} \times \frac{lens}{p_{lens}} \times \frac{dist.}{p_{dist.}} \times \frac{dir.}{p_{dir.}} \times \cdots \tag{9.1}$$

where $\widehat{L}(\mathbf{x})$ is the value assigned to ray path $\mathbf{x}$, the numerators of the terms correspond to the geometric and light scattering properties of the path, and the denominators of the terms represent the probability with which each part of the path was generated. $\widehat{L}(\mathbf{x})$ forms an unbiased estimate of the pixel's value, which may have a high variance. In fact, the variance of $\widehat{L}$ may be so high that the renderer must average thousands of samples per pixel to produce an image with acceptable error.

9.1.4 Importance Sampling in Global Illumination

Importance sampling can be a powerful tool for reducing the variance of a Monte Carlo estimate. In global illumination, importance sampling methods work by changing the way in which preimage space points are converted to world space samples to make $\widehat{L}$ from equation 9.1 as constant as possible. Flattening $\widehat{L}$ can be quite difficult, however, since the

renderer constructs ray paths as Markov chains of ray segments. In practice, this means that each term in equation 9.1 not only depends on the local decision, but on previous decisions as well. Consider the process of choosing a point on the lens to sample. This decision affects which surface in the scene will be struck by the ray path, which in turn determines the BRDF function that must be evaluated in the next term of the product, and so on.

Because of these issues, most importance sampling methods for global illumination only consider one or two terms of $\widehat{L}$ at a time. Nevertheless, importance sampling a few terms of the measurement equation can be quite successful in some situations, and a large number of importance sampling methods have been developed around this idea.

BRDF and phase function sampling. One of the terms of the measurement equation that has received a lot of attention is the BRDF function. The result of this work is that a large number of analytical and sampled BRDF representations can be importance sampled [Blinn, 1977; Ward, 1992; Lafortune *et al.*, 1997; Ashikhmin and Shirley, 2000; Lawrence *et al.*, 2004]. Along with BRDFs, some phase function representations, which describe the scattering of light within a participating medium, can also be directly importance sampled [Blasi *et al.*, 1993].

Light source sampling. Light sources in the scene are another common target for importance sampling. This work can roughly be divided into two categories, algorithms that define an importance function over a large number of light sources [Ward, 1991; Shirley *et al.*, 1996; Fernandez *et al.*, 2002], and algorithms that define an importance function over the area of a large light source such as an environment map [Agarwal *et al.*, 2003; Kollig and Keller, 2003; Ostomoukhov *et al.*, 2004; Debevec, 2005].

Combining BRDF and light source samples. Since the measurement equation is evaluated as a product of terms, a number of methods have been developed to importance sample two terms in the product simultaneously, usually the BRDF and light sources.

Multiple importance sampling [Veach and Guibas, 1995], for instance, allows a renderer to sample the BRDF with some samples and the light sources with others, combining the probabilities in a way that preserves the good properties of both methods. Other work has gone on to sample or approximately sample the product of the lighting and BRDF, either by discarding some samples taken from a simpler distribution [Burke *et al.*, 2005; Talbot *et al.*, 2005], or by constructing an explicit representation of the product suitable for importance sampling [Clarberg *et al.*, 2005; Cline *et al.*, 2006].

Sampling directions in space. Beyond product sampling, a number of algorithms attempt to provide a more global context to the importance sampler by using probability maps to indicate good sampling directions in space. The probability maps in these methods [Jensen, 1995; Hey and Purgathofer, 2002; Steinhurst and Lastra, 2006] are generally constructed using a photon map. Our sample swarming algorithm also uses probability maps to provide more of a global context to the sampler, but we construct the maps lazily during the process of sampling rather than using a photon map. Furthermore, our probability maps can take into account terms of the measurement equation, such as the area of the lens, that are ignored by methods that simply specify sampling directions in space.

9.2 Sample Swarming

This section describes our sample swarming algorithm in detail. The basic idea behind sample swarming is to supplement existing importance sampling methods with tabular probability maps, called *swarming maps*. The renderer uses a custom set of swarming maps for each image pixel to augment any existing importance sampling functions. The purpose of the swarming maps is to direct sampling efforts towards undersampled regions of the measurement equation that may not be properly accounted for by other sampling methods. For example, one of the swarming maps in our system helps determine what direction to

sample from a non-specular surface, which can be useful in sampling caustics and other indirect lighting phenomena.

The swarming maps for a given pixel are assembled lazily from the Monte Carlo samples that have already been taken in the pixel's neighborhood. When a bright sample is found, the swarming maps update to increase the probability of sampling near the same location in path space. Subsequent samples then naturally "swarm" to that part of path space, exploring the bright region and reducing variance. This swarming behavior can be quite effective at finding features in path space that are not captured by local importance functions such as BRDF or light source sampling.

9.2.1 World Space and Preimage Space Maps

The swarming maps that we define in this paper can be grouped into two main categories, those that act in world space, and those that act in preimage space.

World space swarming maps work in parallel with existing importance sampling methods. That is, some preimage space points get transformed to world space using standard importance sampling methods, while others get transformed based on the world space swarming maps, as shown in figure 9.1. Hence, a world space swarming map is really just another importance sampling method to be combined with existing methods using multiple importance sampling. Assuming the balance heuristic, the resulting probability distribution is a weighted average of the swarming map probability p_{map} and the density induced by other importance sampling methods p_{is}:

$$p_{tot} = \alpha p_{map} + (1-\alpha)p_{is}, \tag{9.2}$$

where α is the fraction of the points that are transformed using the swarming maps.

Preimage space swarming maps, on the other hand, serve as a preprocess to standard importance sampling. The sampler feeds uniformly distributed points in $[0,1)^n$ into the preimage space map, which transforms them into non-uniformly distributed points, but

still in preimage space. The non-uniform points are then transformed to world space with standard importance sampling methods. Figure 9.2 shows this process graphically. The end probability resulting from this double transformation is the product of the probability specified in the swarming map and the natural probability created by importance sampling:

$$p_{tot} = p_{map}\, p_{is}. \tag{9.3}$$

9.2.2 Maps Used in our System

In theory, one could define a single, high dimensional swarming map to help make all of the decisions related to constructing ray paths. However, this would cause memory overhead and "curse of dimensionality" issues (a high dimensional space is more difficult to explore than a low dimensional one). To avoid these issues, we define a series of one and two dimensional maps to guide the renderer in making specific decisions. The maps currently defined in our renderer include *time*, *lens area*, *scatter distance* within a participating medium, *bifurcation* (whether to reflect or transmit), *light selection*, *point on light*, and *scatter direction*. Table 9.1 gives a brief description of the swarming map types defined by our renderer. Figures 9.3 through 9.9 show the kinds of results that can be achieved by including each of the map types, compared against a converged image and standard path tracing. In our system, the maps can be individually turned on or off by the user.

Note that while the scenes in figures 9.3 through 9.9 may seem contrived, each one actually corresponds to a fairly common rendering task. For example, cinematographers often utilize a very shallow depth of field like that shown in figure 9.4 to de-emphasize background elements or create a dreamy appearance. Furthermore, ordinary household objects such as glasses and pitchers exhibit layering of transparent surfaces, much like the scene in figure 9.7. Other common effects in the examples include light scattering in fog, and scenes with large numbers of light sources. Sample swarming forms a flexible tool that can aid existing importance sampling methods in all of these situations.

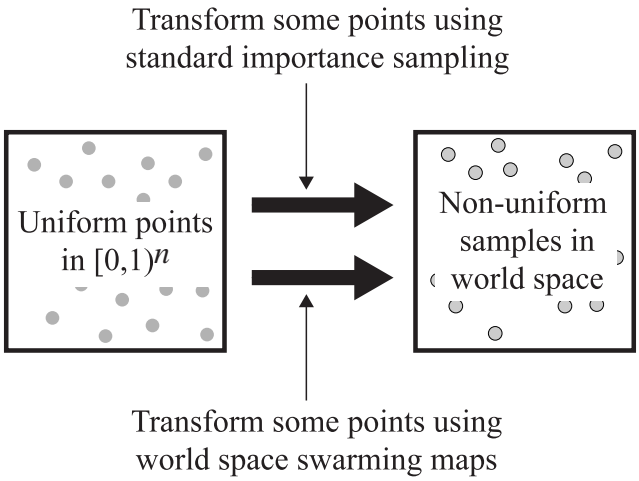


Figure 9.1: World space swarming maps act as an additional importance sampling method to transform points from preimage space into samples in world space. The resulting probability distribution is a weighted average of the distribution induced by standard importance sampling and the one induced by the swarming maps.

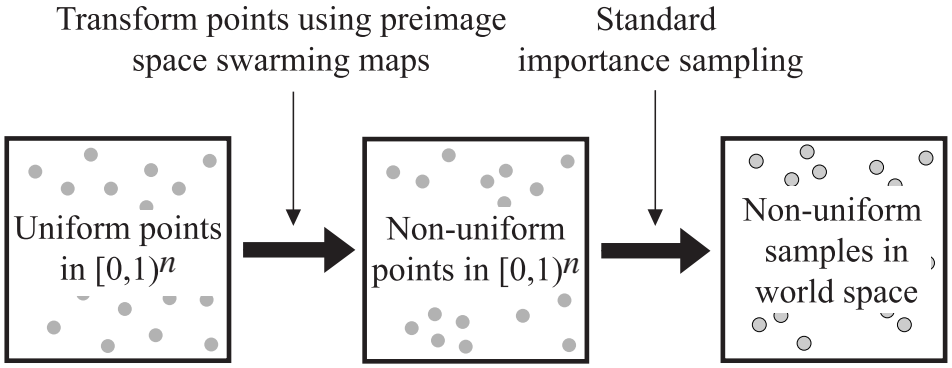


Figure 9.2: Preimage space swarming maps act as a preprocess to standard importance sampling, changing the distribution of points in preimage space that are fed into the importance sampling routines. The resulting distribution of samples in world space is proportional to the product of the distribution specified by the preimage space maps and the one induced by standard importance sampling.

Map Name	Helps to sample	Dim.	Space
Time	Motion blur	1D	Preimage
Lens	Depth of field	2D	Preimage
Scatter dist.	Participating media	1D	Preimage
Bifurcation	Reflect vs. transmit	Discrete	World
Light select.	Many light sources	Discrete	World
Point on light	Large light sources	2D	Preimage
Scatter dir.	Caustics	2D	World

Table 9.1: Types of swarming maps. The table lists the different types of swarming maps defined in our system, along with short descriptions of the kinds of rendering situations each type helps to sample, the map dimensionality, and the space in which the map resides (preimage or world space).

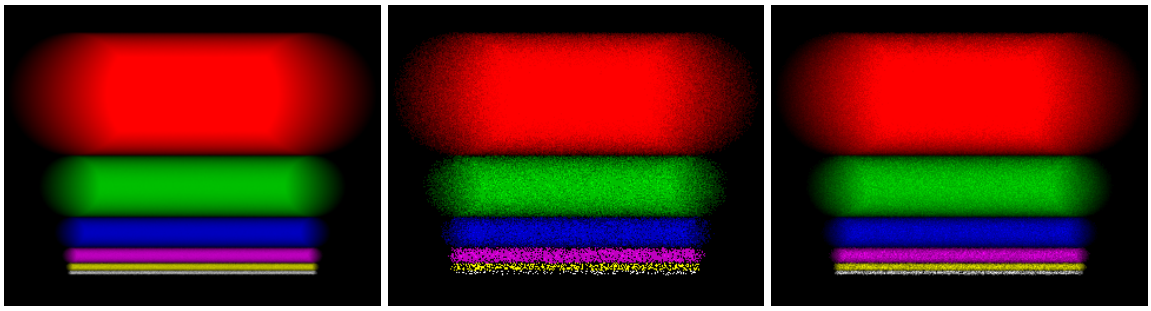


Figure 9.3: The time map helps the renderer choose what instant in time to sample. The scene shows six motion blurred spheres. Path tracing (middle) cannot reliable predict when the spheres will be within view for a given pixel. Adding a time map (right) provides this information. Both examples were rendered using 16 samples per pixel.

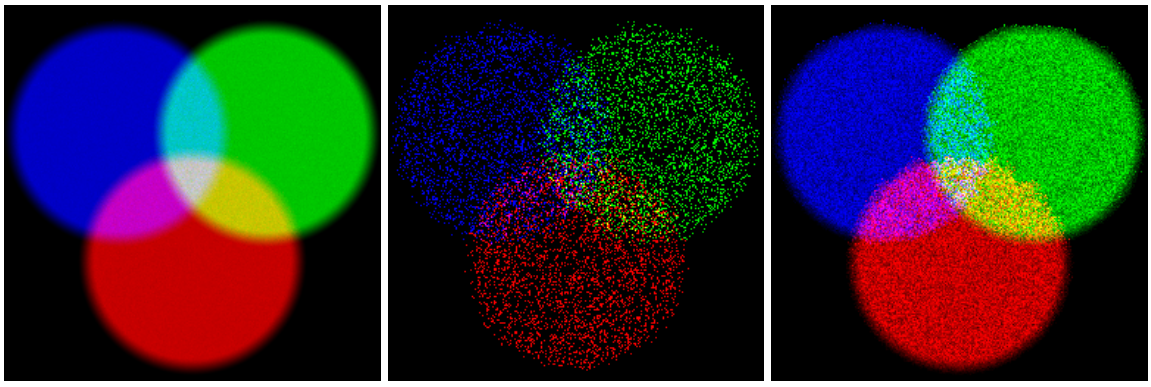


Figure 9.4: The lens map aids in sampling the lens area. Three colored lights are rendered out of focus, producing a color wheel pattern. Since path tracing always samples the lens uniformly, the three light sources are barely discernible. Adding a lens map significantly improves the render quality with the same sample density (32 samples per pixel).

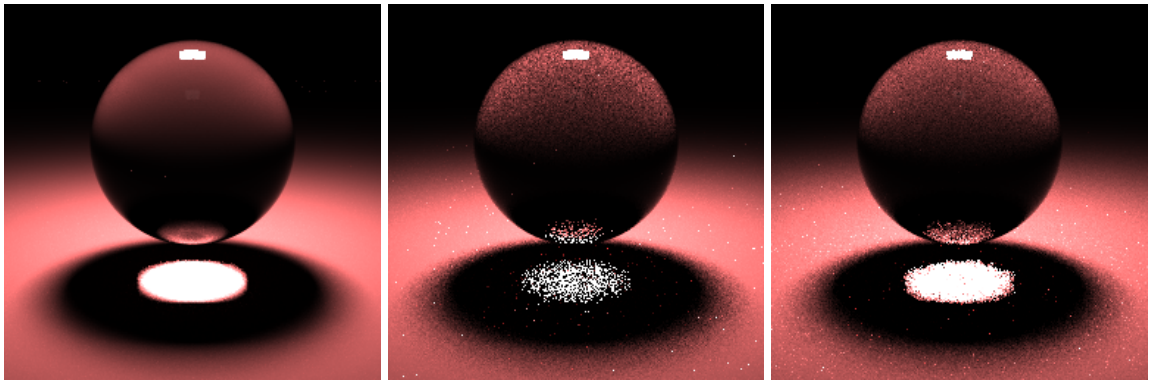


Figure 9.5: A glass ball focuses light onto a plane, creating a bright caustic. Path tracing must sample randomly to find those directions that will refract through the glass ball to hit the light source. The scatter direction maps share this information between pixels, producing a more accurate result (32 samples per pixel).

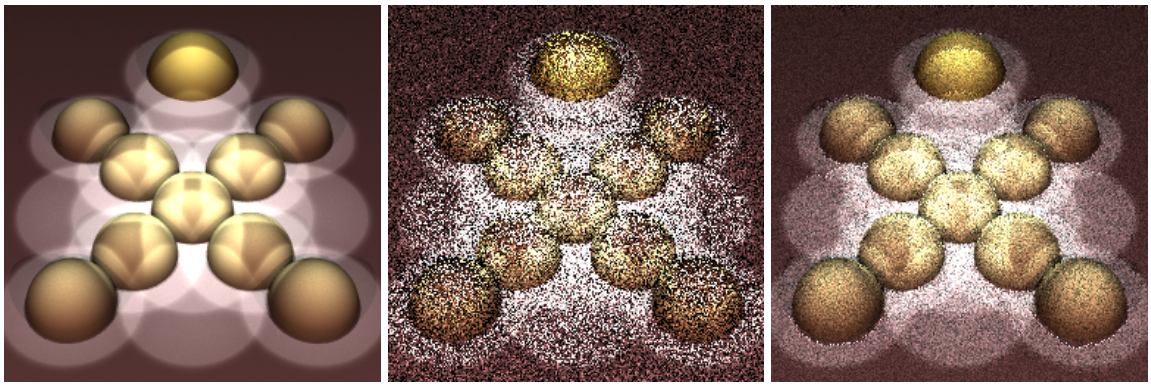


Figure 9.6: A scene is lit by 22 spot lights. Path tracing with 6 samples per pixel produces a poor image because it simply chooses lights at random to sample. Sample swarming using a light selection map does much better because information about which lights illuminate a given region of the image propagates through the image during rendering.

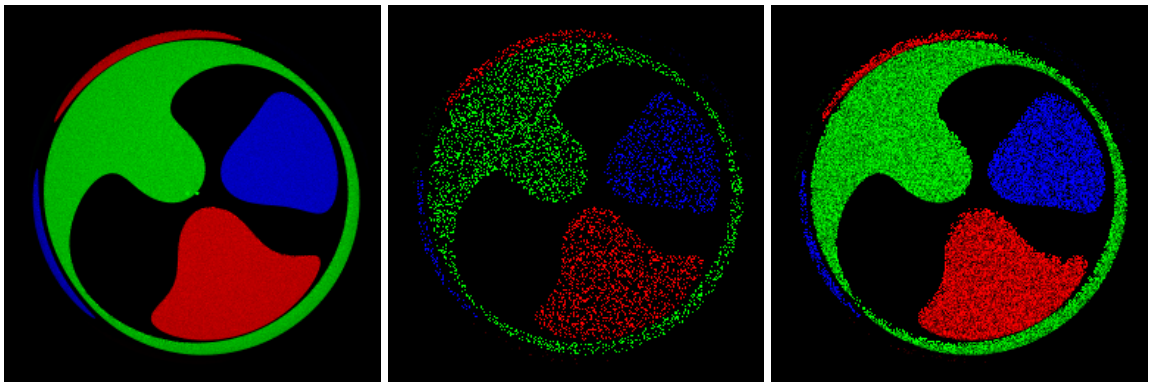


Figure 9.7: A bifurcation map helps decide whether the renderer should reflect or transmit when creating a ray path. In the above scene, three spherical light sources are viewed through three glass spheres that act as lenses. Our path tracer naturally chooses both reflection and transmission 50% of the time, so only $1/64$ of the samples make it through the glass spheres to reach the light sources. Adding a bifurcation map increases the odds of reaching the light sources by nearly an order of magnitude (16 samples per pixel).

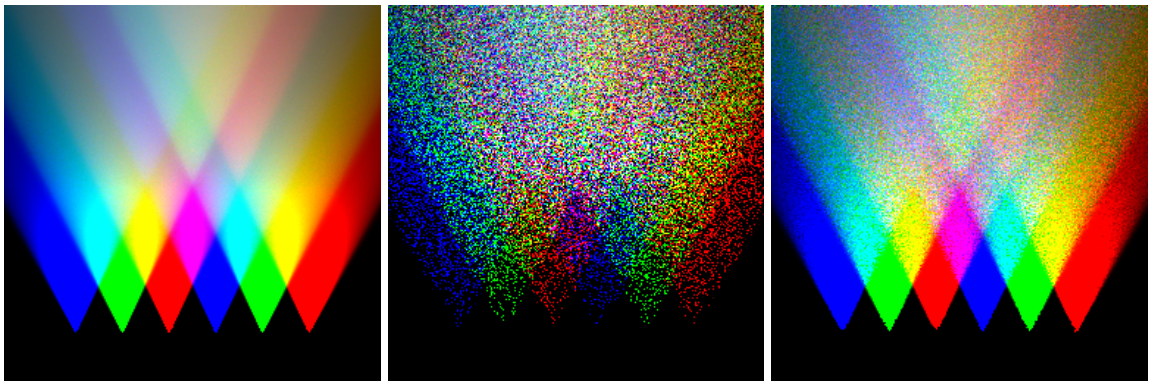


Figure 9.8: The scatter distance map helps determine how far a ray path should penetrate a participating medium before generating a scattering event. In the above scene, six colored spot lights shine up into a uniform fog. To receive any light, a sample must produce a scattering event within the cone of one of the spot lights. Path tracing has no global information about where the spot lights may be shining, so it relies solely on the scattering properties of the fog to determine where to place scattering events. The scatter distance map provides the renderer with this context, leading to an improved result (32 samples per pixel).

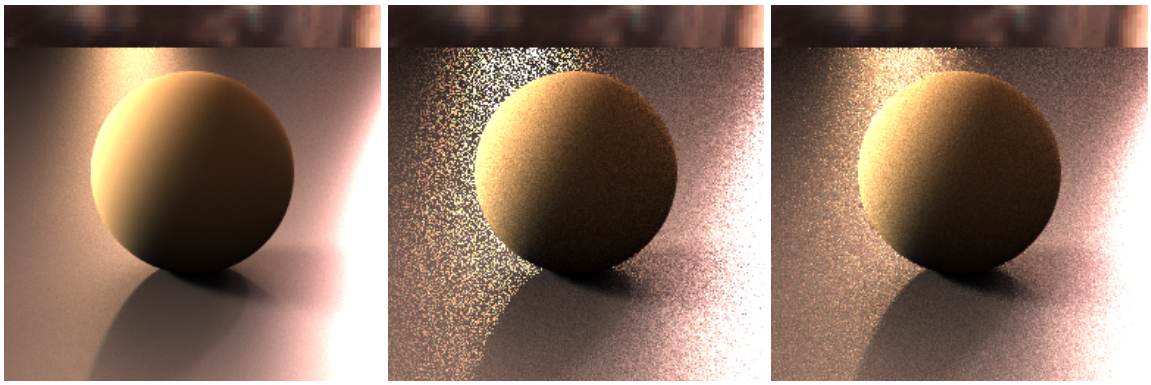


Figure 9.9: A diffuse ball sitting on a glossy plane is illuminated by an environment map. Even while sampling both the environment map and surface BRDFs, path tracing is not able to accurately reconstruct the illumination on the glossy plane. Adding a "point on light" map guides the sampler towards regions of the light source that are responsible for the glossy highlights (32 samples per pixel).

9.2.3 Map Storage and Importance Propagation

Map storage. Memory overhead for the swarming maps is kept to a minimum by only storing maps for two rows of pixels in the image at a time−the current row being rendered and the row just completed. To render a pixel, the renderer first assembles swarming maps for the pixel by averaging the maps from the neighboring pixels that have already been rendered. Our implementation averages seven maps, as shown in figure 9.10. Typically, the maps are kept fairly small in size. For 2D maps, a size of 32×32 is typical, and 1D maps generally range between 64 and 256 entries. When rendering an image, we scan back and forth to allow importance to propagate both left and right, as well as down, in the image.

The temporary maps. Ideally, we would like to add each Monte Carlo sample directly to the current pixel's swarming maps, but several difficulties get in the way. First, the maps for the current pixel are inverted to facilitate quick lookup, and it would be expensive to add samples directly to the inverted maps. Second, the pixel maps are normalized, and it is unclear how to scale individual samples so that they combine properly with normalized maps. For these reasons, we keep a set of *temporary swarming maps* to hold importance information obtained while rendering the current pixel. After the pixel has been rendered, the temporary maps can be scaled and combined with the maps for the pixel, and this information will propagate to subsequent pixels.

Before processing a pixel, the renderer clears the temporary maps. Then, each Monte Carlo sample produced while rendering the pixel is added to the temporary swarming maps. As an example of how this is done, suppose that the renderer has created the ray path $\mathbf{x}$ with value $\widehat{L}(\mathbf{x})$ from point u in preimage space. Now suppose that the coordinates $(0.2, 0.1)$ from u were used to choose a point on the lens while constructing $\mathbf{x}$. Since the lens map is a preimage space map, the new sample should be added to the temporary lens map at location $(0.2, 0.1)$. To do this, we find the map coordinates associated with

$(0.2, 0.1)$ and add $\widehat{L}(\mathbf{x})$ to this entry in the temporary map. Additionally, we add $\widehat{L}(\mathbf{x})$ to the 8 neighboring map entries, to help promote exploration.

For a world space map, we still add $\widehat{L}(\mathbf{x})$ to a 3×3 neighborhood in the map, but the location in the map is determined by the geometric properties of $\mathbf{x}$ rather than coordinates in u. The scatter direction map, for instance, encodes the first non-specular bounce direction of the path in latitude-longitude format.

Two of the swarming maps in our system make discrete rather than continous decisions, and adding values to these maps works somewhat differently than in the continuous case. One of these discrete maps is the light selection map. Each entry in the light selection map corresponds to a specific light source in the scene, so we only add $\widehat{L}(\mathbf{x})$ to a single table entry in this map. The second discrete map in our system is the bifurcation map. The bifurcation map helps decide at each surface intersection whether to reflect or transmit. To update this map, we keep a list of all the reflect/transmit decisions that were made while creating the path, and add $\widehat{L}(\mathbf{x})$ to corresponding entries in the map. For example, suppose that path $\mathbf{x}$ has the form *ERTTL* (eye, reflect, transmit, transmit, light). To add $\mathbf{x}$ to the bifurcation map, $\widehat{L}(\mathbf{x})$ would be added to three entries in the map, $reflect_0$, $transmit_1$ and $transmit_2$.

Combining in the temporary maps with the current pixel. Once a pixel has been rendered, we can combine the temporary maps with the maps for the pixel. The purpose of this operation is to improve the swarming maps for future pixels, and it is performed as follows:

- First, we normalize the temporary maps (divide them by their sum).
- Next, we add a constant value to each entry in the preimage space maps. Since all of the samples are fed through the preimage space maps, adding a constant value to the maps helps the renderer balance between exploration and exploitation. In the current implementation, we add the value $1/n$ to each map entry, where n is the number

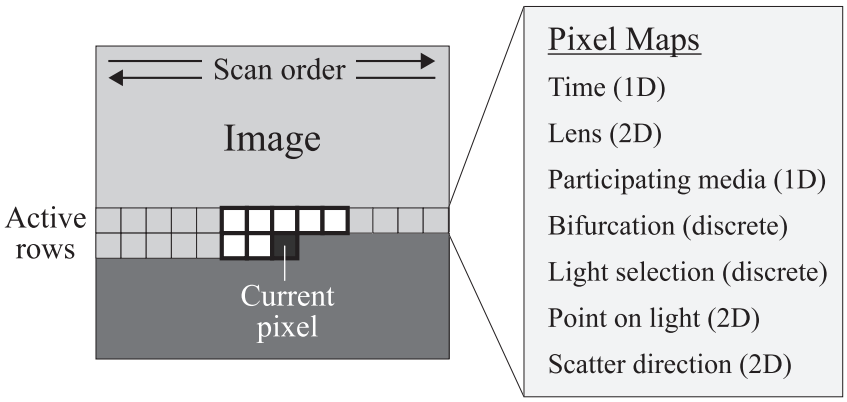


Figure 9.10: Importance propagation through map averaging. The renderer creates swarming maps for the current pixel by averaging the maps from seven neighboring pixels within the two active pixel rows. (The maps for pixels to the right of the current pixel in the figure cannot be included since they have not yet been defined.) Importance information gleaned while rendering the pixel is stored in a set of temporary maps which are combined pixel's maps after it has been rendered.

of entries in the map. World space maps (scatter direction), do not need this step, since exploration is provided by standard importance sampling. Nevertheless, we still add a tiny value to each world space map entry to avoid underflow during later map averaging. We then renormalize the temporary maps.

- The temporary maps are then inverted so that they can be added to the already inverted pixel maps. Inversion of a 1D map simply means computing a CDF of the map. We invert 2D maps similary, creating a CDF for each row of the map, as well as an extra CDF of the Y axis marginal. A lookup in a 1D map then requires a single binary search, and a lookup in a 2D map takes 2 binary searches.

- Finally, the inverted temporary maps are added to the pixel maps, and the pixel maps are renormalized. (Note that this can be done directly to the inverted maps.)

After the temporary maps have been added in, the maps for the pixel contain a mix of information gleaned while rendering the pixel and information propagated from the pixel's neighbors. It is this blending of importance data between old and new samples that causes

the swarming behavior in our algorithm. Path space regions that remain bright over several pixels get reinforced, while regions that become dim quickly lose importance.

9.3 Results

We have implemented sample swarming as an extension to the PBRT renderer. In this section we discuss the results of our method and compare it to standard path tracing.

Numerical comparison of the example renderings. Table 9.2 gives the numerical error of the images in figures 9.3 through 9.9. All of the renderings made with sample swarming have significantly lower error than path tracing with the same number of samples per pixel. To put these numbers into perspective, the lens map example from figure 9.4 required more than 18 times as many path traced samples to equal the error of the sample swarming rendering.

	Path tracing	Sample swarming
Time map (fig. 9.3)	1.045	0.207
Lens map (fig. 9.4)	3.185	0.539
Scatter direction (fig. 9.5)	2.025	0.822
Light selection (fig. 9.6)	0.904	0.312
Bifurcation map (fig. 9.7)	3.924	0.920
Scatter distance (fig. 9.8)	4.944	1.481
Point on light (fig. 9.9)	0.429	0.166

Table 9.2: Numerical results for figures 9.3 through 9.9, comparing RMSE (σ/μ) of path tracing and sample swarming.

More complicated examples. Figures 9.11 through 9.14 show a number of more complicated scenes than the examples from section 9.2.

Figure 9.11 demonstrates the ability of sample swarming to find shadow boundaries by using the scatter direction map. In the scene, a bunny is enclosed in a box that has a

small hole in the top to let light in. The swarming maps share information between pixels about the location of the hole, improving the rendering.

Figure 9.12 shows how effective the scatter direction map can be in the presence of specular surfaces. The swarming maps are able to latch on to important directions within path space and share this information between pixels.

In figure 9.13, a glass blender and matching glasses sit on a shiny counter top. While the objects in the scene are ordinary, they pose great difficulties to the renderer. The glasses are arranged one behind the other, which in effect places eight refractive surfaces in series in the scene. Also, the countertop is quite shiny, producing bright secondary reflections throughout the scene. Turning on the scatter direction and bifurcation swarming maps improves the rendering significantly, especially in the glass regions. However, even the "converged" image, rendered with sample swarming using 1280 samples per pixel, contains discernible noise. All of the shiny surfaces in this scene make it particularly difficult to sample effectively.

Finally, the scene in figure 9.14 demonstrates the use of the point on light and lens maps. This scene is difficult for a number of reasons. First, the BRDF on the ground plane is spatially varying, making it difficult to use precomputed methods like wavelet importance sampling. Next, since the scene is out of focus, methods that only send a few primary rays per pixel will not work. Finally, the rendering includes indirect lighting effects. Our path tracing render of this scene sampled both the BRDF function and environment map light source using multiple importance sampling. Nevertheless, adding the swarming maps appreciably improved the final result over this advanced sampling strategy.

9.4 Discussion

Areas for improvement. Although sample swarming can be effective in many situations, there are a number of limitations to the algorithm as it stands. First, the maps can only help explore important regions of path space once they have been found by random sampling.

When the sampler does find an important region, information about the region cannot propagate back to pixels that have already been rendered. Another issue is that the maps do not always cooperate to reduce error. Often, one of the maps provides nearly all of the variance reduction, while the others do not help appreciably. This effect seems to be a result of the fact that the maps are 1 and 2D projections of a higher dimensional space. Finally, we note that path space coherence cannot always be found and exploited by swarming maps. For example, we experimented with defining swarming maps for the second bounce of the path, but these provided only a marginal benefit.

Despite these shortcomings, we believe sample swarming to be a practical method to reduce error in global illumination images. Several of the maps address parts of the measurement equation that are usually glossed over by other sampling methods, such as the point on the lens and instant of time to sample.

Relation to particle swarm optimization. Sample swarming is similar in a number of ways to the particle swarm optimization methods described in the machine learning literature. Both particle swarm optimization and sample swarming attempt to define a sampling distribution based on samples previously taken, and both algorithms must balance between exploration of the problem space and exploitation of high valued areas already found. On the other hand, sample swarming does not keep a population of particles, and the end goal of sample swarming is to integrate over the problem space rather than to perform a search within it.

More types of swarming maps. Besides the seven maps that we implemented, other types of swarming maps are certainly possible. In fact, almost any decision made by the renderer could conceivably be augmented by a swarming map. As an example, a spectral renderer could use a map to help determine which wavelength to sample.

Sample swarming and other global illumination algorithms. This paper demonstrated sample swarming in the context of Monte Carlo path tracing. However, swarming maps may be even more effective when used in conjunction with algorithms that reduce the dimensionality of the measurement equation by precomputing indirect lighting. Photon mapping with a final gather step [Jensen, 1996] and multidimensional light cuts [Walter *et al.*, 2006] come to mind as two algorithms that could benefit from the addition of swarming maps.

9.5 Conclusion

In this paper we presented sample swarming as a technique to supplement existing importance sampling functions in global illumination. The new method works by defining tabular swarming maps that hold importance information gleaned from the samples taken while rendering an image. We showed that sample swarming can be quite effective at reducing variance in a number of common rendering situations.

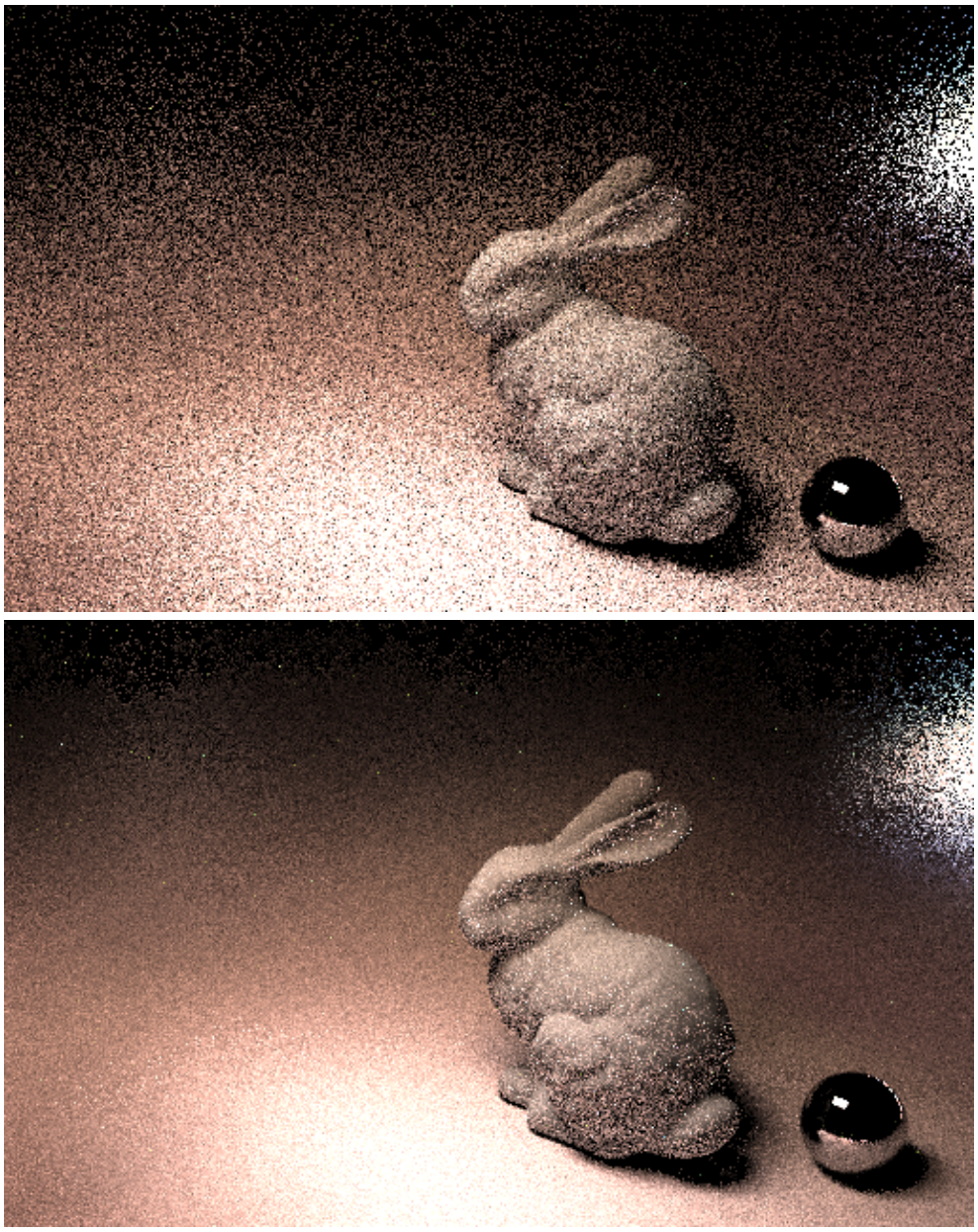


Figure 9.11: "Hare Noire" scene with indirect illumination. Light shines through a small hole in a box, illuminating a bunny model. Path tracing (top) must send out many random samples to find the hole, but sample swarming with a scatter direction map (bottom) shares information about unshadowed directions between pixels, reducing noise appreciably. Both examples use 64 samples per pixel.

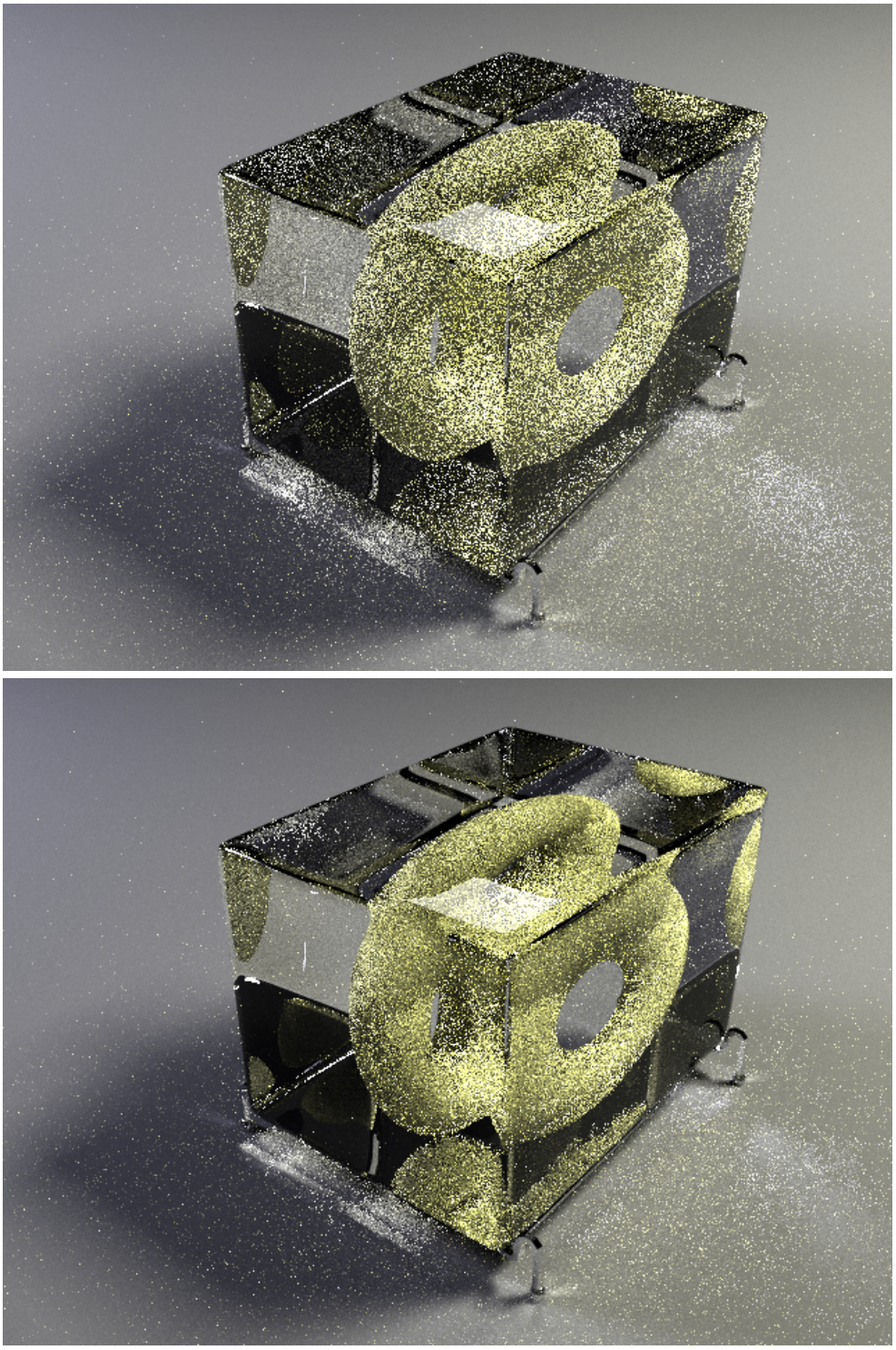


Figure 9.12: The paperweight scene above was rendered with path tracing (top) and sample swarming using the scatter direction and bifurcation maps (bottom). Note the many improvements in the sample swarming image compared to standard path tracing. For example, the torus is much smoother in the sample swarming image, as is the caustic underneath the paperweight. Both examples use 256 samples per pixel.

Figure 9.13: A blender and matching glasses. This scene is quite difficult because of the glass and shiny tile. Path tracing (bottom left) has a lot of trouble with the glass surfaces and caustic regions of the image. Sample swarming using the scatter direction and bifurcation maps (bottom right) does a much better job at sampling these effects. Note how much smoother the glass appears in the sample swarming render. Both examples are rendered with 128 samples per pixel. Cranking up the number of samples per pixel to 1280 (top image) produces a much better result, but even with sample swarming a number of bright speckles are still present.

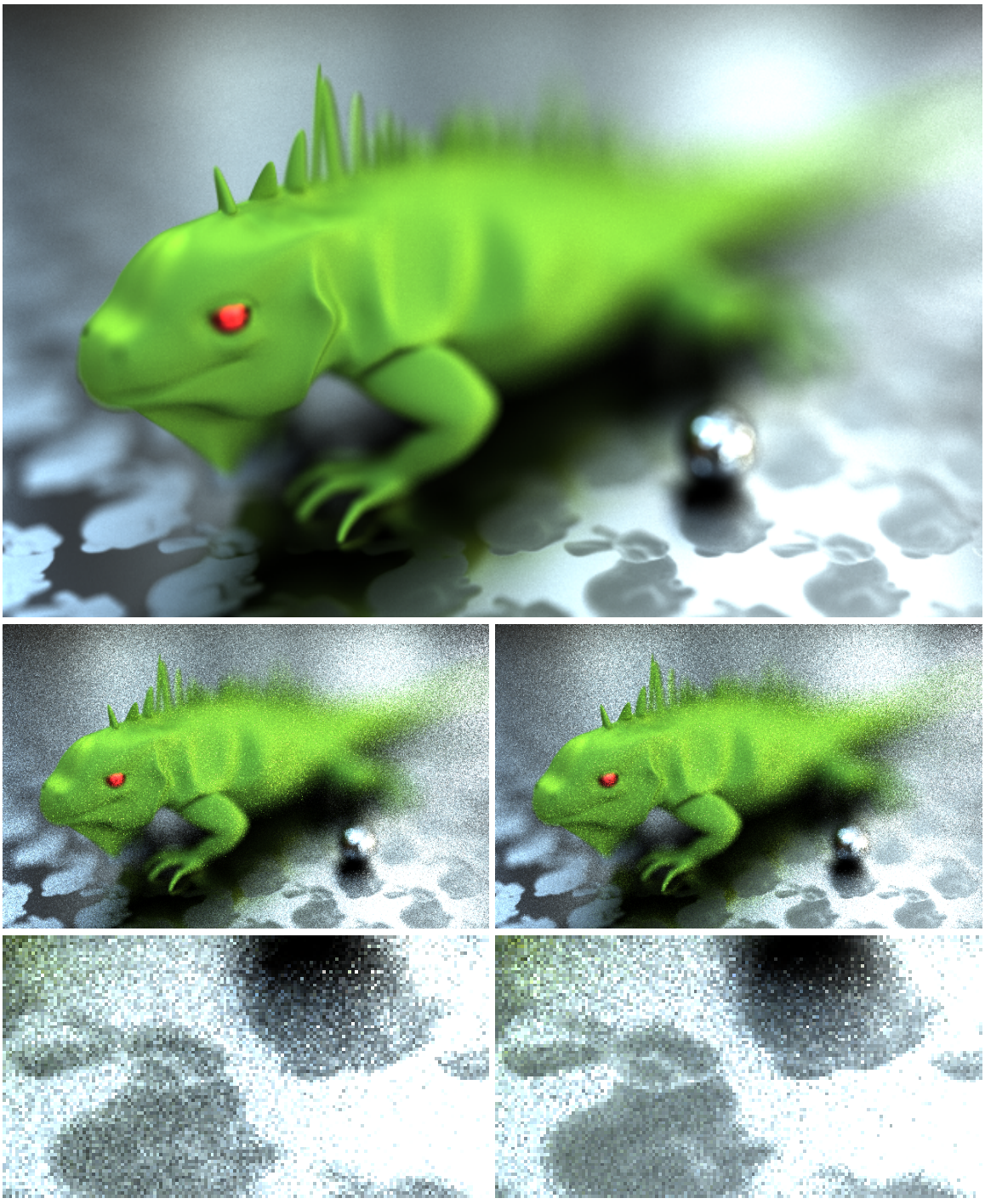


Figure 9.14: An out of focus iguana, rendered with path tracing (left images) and sample swarming using the lens and point on light maps (right images). Both renderings use 64 samples per pixel. The close-ups on the bottom row demonstrate the improvement in quality afforded by adding the swarming maps. The top image shows a converged result rendered with 8000 samples per pixel.

Chapter 10

Conclusion

This dissertation has presented a number of algorithms related to three topics in ray based global illumination: low level ray casting, direct lighting, and general global illumination. Here we conclude by summarizing the contributions made in each of these areas.

10.1 Ray Casting

Part I of the dissertation discussed ray casting methods, with particular emphasis on acceleration data structures. In chapter 2 we discussed the three types of acceleration data structures commonly used by modern ray tracers, voxel grids, BSP trees and bounding volume hierarchies, emphasizing the memory requirements of each.

The research in part I continued along these lines. In chapter 3 we presented a new acceleration data structure, the *lightweight bounding volume hierarchy* (LBVH), specifically designed to have a small memory footprint compared to the geometry that it bounds. An LBVH reduces its memory footprint over standard bounding volumes through three space saving techniques. First an LBVH eliminates pointers by storing the hierarchy in a heap-like structure. Leaf pointers are also eliminated by means of implicit indexing. Memory usage can be further reduced by encoding the bounding box extents with two-byte integers rather than floating point numbers. Finally, by increasing the branching factor of the hierarchy from two to four, a third of the nodes in the hierarchy can be eliminated while

maintaining the same number of leaves. The resulting data structure requires only 16 bytes of memory per leaf node, and does not significantly increase render times.

10.2 Sampling Methods in Direct Lighting

In Part II, we discussed sampling methods related to direct lighting. Chapter 4 presented the direct lighting equation and reviewed a number of sampling strategies to solve it, including BRDF and light source sampling, multiple importance sampling (MIS) and resampled importance sampling (RIS). We also established the goal of distributing samples according to the product of several terms of the direct lighting equation.

Following this introduction, the next chapters covered two algorithms to solve the direct lighting equation. Chapter 5 presented *two stage importance sampling* as a method to distribute samples according to the product of the BRDF and light source. Two stage importance sampling works by partitioning the light source into a hierarchy of regions based on the BRDF function. Once made, the partition can be used to transform a uniform distribution of points to approximate the product distribution. The approximation improves as more samples are added, and visibility becomes the main source of variance after a few dozen samples per pixel.

Chapter 6 addressed the visibility issue by defining visibility maps at sparsely sampled locations on the image plane. These visibility maps were then used to cull some of the samples in occluded regions. By eliminating some of the non-visible samples from a double product distribution, the surviving samples would be approximately distributed according to the *triple product* of the BRDF, lighting and visibility. This chapter also introduced the concepts of *world space* and *preimage space* samples. As the name implies, world space samples are ray samples defined in the geometric space of ray paths. Preimage space samples, on the other hand, reside in the abstract space of random numbers that will be transformed to world space. Our work demonstrated that it is often more efficient to

work in preimage space, since an expensive transformation is required to convert points from preimage to world space.

10.3 Sampling Methods in Global Illumination

Part III of the dissertation focused on sampling methods related to general global illumination. Chapter 7 described the core equations of global illumination, the rendering and measurement equations. It then went on to review a number of the most important ray based global illumination algorithms currently in use, including path tracing, irradiance and radiance caching, light tracing, bidirectional path tracing, photon mapping, instant radiosity, and Metropolis light transport (MLT).

The original research in part III described new sampling algorithms that work by sharing information obtained during the course of sampling between pixels to reduce variance. Chapter 8 presented *energy redistribution path tracing* (ERPT), which attempts to combine path tracing with Metropolis Light Transport to improve the stratification characteristics of MLT. ERPT works by spreading the energy of an initial set of path traced samples over the image plane by means of Metropolis-style mutations. The energy spreading has the effect of sharing information about bright samples between pixels, reducing the overall variance.

The technique described in chapter 9 shares sampling information between pixels in a more explicit way that ERPT. In this technique, called *sample swarming*, each pixel in the image keeps a set of tabular "swarming maps" that store importance information gained during the course of sampling. Whenever a bright sample is found, the swarming maps update to increase the probability of sampling that part of path space. Subsequent samples then "swarm" to the bright region, increasing the sampling density there. This swarming behavior often decreases variance because the measurement equation in a neighborhood of pixels is usually highly correlated. Building on the ideas of preimage space and world space presented in chapter 6, we defined both preimage and world space swarming maps.

World space swarming maps provide an additional importance function that can help make up for the shortcomings of other sampling methods by reinforcing bright portions of path space. Preimage space maps, on the other hand, can be even more effective since they act as a preprocess to other importance sampling methods. The resulting distribution of samples is proportional to the product of the map distribution and distribution induced by the importance sampling method. Because of this behavior, preimage space maps are a natural fit for sampling function products such as those found in the rendering equation.

Bibliography

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

[Arvo, 1995] James Arvo. Stratified sampling of spherical triangles. *Computer Graphics*, 29(Annual Conference Series):437–438, 1995.

[Ashikhmin and Shirley, 2000] Michael Ashikhmin and Peter Shirley. An anisotropic phong BRDF model. *J. Graph. Tools*, 5(2):25–32, 2000.

[Ashikhmin *et al.*, 2001] Michael Ashikhmin, Simon Premože, Peter Shirley, and Brian Smits. A Variance Analysis of the Metropolis Light Transport Algorithm. *Computers and Graphics*, 25(2):287–294, 2001.

[Ben-Artzi *et al.*, 2006] Aner Ben-Artzi, Ravi Ramamoorthi, and Maneesh Agrawala. Efficient shadows from sampled environment maps. *journal of graphics tools*, 11(1):13–36, 2006.

[Blasi *et al.*, 1993] Philippe Blasi, Bertrand Le Saëc, and Christophe Schlick. A rendering algorithm for discrete volume density objects. *Computer Graphics Forum (Eurographics '93)*, 12(3):201–210, 1993.

[Blinn, 1977] James F. Blinn. Models of light reflection for computer synthesized pictures. In *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 192–198, New York, NY, USA, 1977. ACM Press.

[Burke *et al.*, 2005] David Burke, Abhijeet Ghosh, and Wolfgang Heidrich. Bidirectional importance sampling for direct illumination. In Kavita Bala and Philip Dutré, editors, *Rendering Techniques 2005 Eurographics Symposium on Rendering*, pages 139–146, Aire-la-Ville, Switzerland, 2005. Eurographics Association.

[Burke, 2004] David Burke. Bidirectional importance sampling for illumination from environment maps. *Master's Thesis, University of British Columbia*, 2004.

[Chiu *et al.*, 1994] Kenneth Chiu, Peter Shirley, and Changyaw Wang. Multi-jittered sampling. *Paul Heckbert, editor, Graphics Gems IV*, pages 370–374, 1994.

[Christensen *et al.*, 2003] Per H. Christensen, David M. Laur, Julian Fong, Wayne L. Wooten, and Dana Batali. Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes. In *Proceedings of Eurographics 2003*, number 3, 2003.

[Clarberg *et al.*, 2005] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. Wavelet importance sampling: efficiently evaluating products of complex functions. *ACM Trans. Graph.*, 24(3):1166–1175, 2005.

[Cline and Egbert, 2005] David Cline and Parris Egbert. A Practical Introduction to Metropolis Light Transport. *Technical Report: Department of Computer Science, Brigham Young University*, pages 1–19, 2005.

[Cline *et al.*, 2005] David Cline, Justin Talbot, and Parris Egbert. Energy redistribution path tracing. *ACM Transactions on Graphics*, 24(3):1186–1195, 2005.

[Cline *et al.*, 2006] David Cline, Parris K. Egbert, Justin F. Talbot, and David L. Cardon. Two stage importance sampling for direct lighting. In T. Akienne-Möller and W. Heidrich, editors, *Rendering Techniques 2006 Eurographics Symposium on Rendering*, pages 103–113, Aire-la-Ville, Switzerland, 2006. Eurographics Association.

[Cohen *et al.*, 2003] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Trans. Graph.*, 22(3):287–294, 2003.

[Cranley and Patterson, 1976] R. Cranley and T. N. L. Patterson. Randomization of number theoretic methods for multiple integration. *SIAM Journal of Numerical Analysis*, (13):904–914, 1976.

[Crow, 1984] Franklin C. Crow. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 207–212, New York, NY, USA, 1984. ACM Press.

[Debevec, 1998] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 189–198, New York, NY, USA, 1998. ACM Press.

[Debevec, 2005] Paul Debevec. A median cut algorithm for light probe sampling. *SIGGRAPH 2005 Poster*, 2005.

[Dunbar and Humphreys, 2006] Daniel Dunbar and Greg Humphreys. A spatial data structure for fast poisson-disk sample generation. *ACM Trans. Graph.*, 2006.

[Dutré and Willems, 1994] Phillip Dutré and Yves D. Willems. Importance-driven monte carlo light tracing. In *EUROGRAPHICS Workshop on Rendering*, 1994.

[Dutré *et al.*, 2003] Phillip Dutré, Philippe Bekaert, and Kavita Bala. *Advanced Global Illumination*. A. K. Peters, 2003.

[Fernandez *et al.*, 2002] Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Local Illumination Environments for Direct Lighting Acceleration. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 7–14. Eurographics Association, 2002.

[Floyd and Steinberg, 1976] R. Floyd and L. Steinberg. An adaptive algorithm for spatial grayscale. In *Proceedings of the Society for Information Display*, volume 17(2), pages 75–77, 1976.

[Fujimoto *et al.*, 1986] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. ARTS: Accelerated Ray-Tracing System. *IEEE Computer Graphics & Applications*, pages 16–26, April 1986.

[Fussell and Subramanian, 1988] Donald S. Fussell and K. R. Subramanian. Fast Ray Tracing Using K-d Trees. Technical Report CS-TR-88-07, University of Texas, Department of Computer Science, 1988.

[Ghosh *et al.*, 2006] Abhijeet Ghosh, Arnaud Doucet, and Wolfgang Heidrich. Sequential sampling for dynamic environment map illumination. In T. Akienne-Möller and W. Heidrich, editors, *Rendering Techniques 2006 Eurographics Symposium on Rendering*, pages 115–126, Nicosia, Cyprus, 2006. Eurographics Association.

[Glassner, 1984] Andrew Glassner. Space Subdivision for Fast Ray Tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[Goldsmith and Salmon, 1987] Jeffrey Goldsmith and John Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics & Applications*, 7(5):14–20, 1987.

[Haines, 2004] Eric Haines. BSP plane cost function revisited. *Ray Tracing News*, May 2004.

[Heckbert, 1990] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In *Computer Graphics (SIGGRAPH 90 Proceedings)*, volume 24, pages 145–154, August 1990.

[Hey and Purgathofer, 2002] Heinrich Hey and Werner Purgathofer. Importance sampling with hemispherical particle footprints. In *SCCG '02: Proceedings of the 18th spring conference on Computer graphics*, pages 107–114, Budmerice, Slovakia, 2002. ACM Press.

[Immel *et al.*, 1986] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A Radiosity Method for Non-diffuse Environments. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 133–142. ACM Press, 1986.

[Jensen, 1995] Henrik Wann Jensen. Importance driven path tracing using the photon map. In *Rendering Techniques '95 (Proceedings of the 6th Eurographics Workshop on Rendering)*, pages 326–335, Vienna, 1995. Springer-Verlag.

[Jensen, 1996] Henrik Wann Jensen. Global Illumination Using Photon Maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, pages 21–30, New York, NY, 1996. Springer-Verlag/Wien.

[Jensen, 2001] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., 2001.

[Jones, 2006] Thouis R. Jones. Efficient generation of poisson-disk sampling patterns. *journal of graphics tools*, 11(2):27–36, 2006.

[Kajiya, 1986] James T. Kajiya. The Rendering Equation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, pages 20, 4, 143–150. ACM Press, 1986.

[Kaplan, 1985] Michael Kaplan. Space tracing: A constant time ray tracer. *ACM SIGGRAPH '85 Course Notes 11*, July 1985.

[Kay and Kajiya, 1986] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 269–278, New York, NY, USA, 1986. ACM Press.

[Kelemen *et al.*, 2002] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm. *Computer Graphics Forum*, 21(3):1–10, 2002.

[Keller, 1997] Alexander Keller. Instant Radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1997.

[Kirk and Arvo, 1991] David Kirk and James Arvo. Unbiased sampling techniques for image synthesis. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 153–156, NY, USA, 1991. ACM Press.

[Klimaszewski and Sederberg, 1997] Krzysztof S. Klimaszewski and Thomas W. Sederberg. Faster Ray Tracing Using Adaptive Grids. *IEEE Computer Graphics & Applications*, 17(1):42–51, 1997.

[Kollig and Keller, 2002] T. Kollig and A. Keller. Efficient multidimensional sampling. *Computer Graphics Forum*, 21(3):557–563, September 2002.

[Kollig and Keller, 2003] Thomas Kollig and Alexander Keller. Efficient illumination by high dynamic range images. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 45–50, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Kopf *et al.*, 2006] Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive wang tiles for real-time blue noise. *ACM Transactions on Graphics*, 25(3):509–518, 2006.

[Křivánek *et al.*, 2006] Jaroslav Křivánek, Kadi Bouatouch, Sumanta N. Pattanaik, and Jiří Žára. Making radiance and irradiance caching practical: Adaptive caching and neighbor clamping. In Tomas Akenine-Mller and Wolfgang Heidrich, editors, *Rendering Techniques 2006, Eurographics Symposium on Rendering*, pages 127–138, Nicosia, Cyprus, June 2006. Eurographics Association.

[Lafortune and Willems, 1993] Eric P. Lafortune and Yves D. Willems. Bi-directional Path Tracing. In H. P. Santo, editor, *Proceedings of the Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, 1993.

[Lafortune *et al.*, 1997] Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Non-linear approximation of reflectance functions. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 117–126, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Lawrence *et al.*, 2004] Jason Lawrence, Szymon Rusinkiewicz, and Ravi Ramamoorthi. Efficient BRDF Importance Sampling Using a Factored Representation. *ACM Transactions on Graphics*, 23(3):494–503, 2004.

[Lloyd, 1983] S. Lloyd. "an optimization approach to relaxation labeling algorithms". *Image and Vision Computing*, 1(2), 1983.

[Logie and Patterson, 1995] J. R. Logie and J. W. Patterson. Inverse displacement mapping in the general case. *Computer Graphics Forum*, 14(5):261–273, December 1995.

[MacDonald and Booth, 1990] David J. MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. *Vis. Comput.*, 6(3):153–166, 1990.

[Mahovsky, 2005] Jeffrey Mahovsky. Ray tracing with reduced-precision bounding volume hierarchies. *PhD thesis, University of Calgary*, 2005.

[Martin *et al.*, 2000] William Martin, Elaine Cohen, Russell Fish, and Peter Shirley. Practical ray tracing of trimmed nurbs surfaces. *Journal of Graphics Tools*, 5(1):27–52, 2000.

[McCool and Harwood, 1997] Michael D. McCool and Peter K. Harwood. Probability trees. In Wayne A. Davis, Marilyn Mantei, and R. Victor Klassen, editors, *Graphics Interface '97*, pages 37–46. Canadian Human-Computer Communications Society, 1997.

[Metropolis *et al.*, 1953] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines. *Chemical Physics*, 21:1087–1091, 1953.

[Mitchell, 1987] Don P. Mitchell. Generating antialiased images at low sampling densities. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 65–72, New York, NY, USA, 1987. ACM Press.

[Mitchell, 1991] Don P. Mitchell. Spectrally optimal sampling for distribution ray tracing. *SIGGRAPH Comput. Graph.*, 25(4):157–164, 1991.

[Mitchell, 1996] Don P. Mitchell. Consequences of stratified sampling in graphics. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 277–280, New York, NY, USA, 1996. ACM Press.

[Oren and Nayar, 1994] Michael Oren and Shree K. Nayar. Generalization of lambert's reflectance model. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 239–246, New York, NY, USA, 1994. ACM Press.

[Ostomoukhov *et al.*, 2004] Victor Ostomoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast Hierarchical Importance Sampling with Blue Noise Properties. *ACM Transactions on Graphics*, 23(3):486–493, 2004.

[Owen, 1995] Art B. Owen. Randomly permuted (t, m, s)-nets and (t, s)- sequences. *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pages 299–317, 1995.

[Pauly *et al.*, 2000] Mark Pauly, Thomas Kollig, and Alexander Keller. Metropolis Light Transport for Participating Media. In B. Peroche and H. Rushmeier, editors, *Rendering Techniques 2000 (Proceedings of the Eleventh Eurographics Workshop on Rendering)*, pages 11–22, New York, NY, 2000. Springer Wien.

[Pharr and Hanrahan, 1996] Matt Pharr and Pat Hanrahan. Geometry caching for ray-tracing displacement maps. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 31–40, New York City, NY, 1996. Springer Wien.

[Pharr and Humphreys, 2004] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2004.

[Pharr *et al.*, 1997] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. *Computer Graphics*, 31(Annual Conference Series):101–108, 1997.

[Pharr, 2003] Matt Pharr. Chapter 9: Metropolis Sampling. In *Monte Carlo Ray Tracing, SIGGRAPH 2003 Course 44, course notes*, 2003.

[Phong, 1975] Bui Thong Phong. Illumination for computer-generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[Rubin and Whitted, 1980] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *SIGGRAPH '80: Proceedings of the*

7th annual conference on Computer graphics and interactive techniques, pages 110–116, New York, NY, USA, 1980. ACM Press.

[Shirley and Morley, 2003] Peter Shirley and R. Keith Morley. *Realistic Ray Tracing (Second Edition)*, chapter 9, pages 141–142, ISBN: 1–56881–198–5. A K Peters, 2003.

[Shirley *et al.*, 1995] Peter Shirley, Bretton Wade, Philip M. Hubbard, David Zareski, Bruce Walter, and Donald P. Greenberg. Global Illumination via Density Estimation. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques 1995 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 219–230, New York, NY, 1995. Springer-Verlag.

[Shirley *et al.*, 1996] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte Carlo Techniques for Direct Lighting Calculations. In *ACM Transactions on Graphics*, volume 15, pages 1–36. ACM Press, 1996.

[Shirley, 1991] Peter Shirley. Discrepancy as a quality measure for sampling distributions. *Eurographics '91*, pages 183–194, 1991.

[Smits *et al.*, 2000] Brian Smits, Peter Shirley, and Michael Stark. Direct ray tracing of displacement mapped triangles. In *Eurographics Workshop on Rendering*, pages 307–318, Brno, Chzech Republic, June 2000.

[Smits, 1998] Brian Smits. Efficiency issues for ray tracing. *Journal of Graphics Tools*, 3(2):1–14, 1998.

[Snyder and Barr, 1987] John M. Snyder and Alan H. Barr. Ray tracing complex models containing surface tessellations. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 119–128. ACM Press, 1987.

[Steinhurst and Lastra, 2006] Joshua Steinhurst and Anselmo Lastra. Global importance sampling of glossy surfaces using the photon map. In Ingo Wald and Steven G. Parker, editors, *IEEE Symposium on Interactive Ray Tracing 2006*, pages 133–138, 2006.

[Sung and Shirley, 1992] Kelvin Sung and Peter Shirley. *Graphics Gems III*, chapter VI. 1. Ray Tracing with the BSP Tree, pages 271–274. Academic Press, 1992.

[Szirmay-Kalos *et al.*, 1999] László Szirmay-Kalos, Péter Dornbach, and Werner Purgathofer. On the Start-up Bias Problem of Metropolis Sampling. *Technical Report: Department of Control Engineering and Information Technology, Technical University of Budapest*, pages 1–8, 1999.

[Tabellion and Lamorlette, 2004] Eric Tabellion and Arnauld Lamorlette. An Approximate Global Illumination System for Computer Generated Films. *ACM Transactions on Graphics*, 23(3):467–474, 2004.

[Talbot *et al.*, 2005] Justin F. Talbot, David Cline, and Parris K. Egbert. Importance resampling for global illumination. In Kavita Bala and Philip Dutré, editors, *Rendering Techniques 2005 Eurographics Symposium on Rendering*, pages 139–146, Aire-la-Ville, Switzerland, 2005. Eurographics Association.

[Terdiman, 2001] Pierre Terdiman. Memory-optimized bounding-volume hierarchies. *http://www.codecorner.com/Opcode.pdf*, 2001.

[Veach and Guibas, 1994] Eric Veach and Leonidas J. Guibas. Bidirectional Estimators for Light Transport. In *Rendering Techniques 1994 (Proceedings of the Fifth Eurographis Workshop on rendering)*, pages 147–162, 1994.

[Veach and Guibas, 1995] Eric Veach and Leonidas J. Guibas. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proceedings of ACM SIGGRAPH 1995*, pages 419–428. ACM Press, 1995.

[Veach and Guibas, 1997] Eric Veach and Leonidas J. Guibas. Metropolis Light Transport. In *Proceedings of ACM SIGGRAPH 1997*, pages 65–76. ACM Press, 1997.

[Veach, 1997] Eric Veach. Robust monte carlo methods for light transport simulation. *Ph.D. dissertation, Stanford University*, December 1997.

[Wald *et al.*, 2001] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In Alan Chalmers and Theresa-Marie Rhyne, editors, *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001*, volume 20, pages 153–164. Blackwell Publishers, Oxford, 2001.

[Wald *et al.*, 2002] Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. Interactive Global Illumination using Fast Ray Tracing. In *Proceedings of the 13th Eurographics Workshop on Rendering*. Held in Pisa, Italy, 2002.

[Walter *et al.*, 2005] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: a scalable approach to illumination. *ACM Trans. Graph.*, 24(3):1098–1107, 2005.

[Walter *et al.*, 2006] Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. Multidimensional lightcuts. *ACM Trans. Graph.*, 25(3):1081–1088, 2006.

[Ward and Heckbert, 1992] Gregory J. Ward and Paul Heckbert. Irradiance Gradients. In *Third Eurographics Workshop on Rendering*, pages 85–98, Bristol, UK, 1992.

[Ward *et al.*, 1988] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A Ray Tracing Solution for Diffuse Interreflection. In *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, pages 22, 4, 85–92. ACM, 1988.

[Ward, 1991] Gregory Ward. Adaptive Shadow Testing for Ray Tracing. In *2nd Eurographics Workshop on Rendering*, volume 3, 1991.

[Ward, 1992] Gregory J. Ward. Measuring and modeling anisotropic reflection. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 265–272, New York, NY, USA, 1992. ACM Press.