Sampling for Complexity in Rendering

Jiawei Ou
Dartmouth College
Abstract

As the visual effect and movie industries are striving for realism and high fidelity images, physically based lighting, global illumination, realistic materials, and highly tessellated geometry are gradually accepted and used in movie and game industries. Modern computer graphics has reached an unprecedented level of complexity. As a result, brute-force rendering methods become prohibitively expensive. For decades, computer graphics researchers have been tackling the complexity in rendering by introducing more advanced Monte Carlo integrators, efficient sampling and reconstruction algorithms, effective filtering techniques, sophisticated data structures, and new hardware architectures.

In this thesis, we focus on deriving better sampling algorithms to improve the rendering efficiency under complex scene settings. In this context, we explore several areas in computer graphics where sampling algorithms play an important role in improving the overall performance of the renderer: (1). We introduce progressive rendering algorithms as alternatives of full-quality, final rendering under complex settings. We then conduct a user study to evaluate the effectiveness of several progressive rendering algorithms in the context of appearance design tasks. (2). We conduct an investigation into high-quality rendering algorithms for scenes with complex lighting, and propose an efficient many-light algorithm, which renders a few hundred thousand virtual
point lights for global illumination, based on matrix slice sampling and light clustering. (3). We investigate importance sampling algorithms for bidirectional scattering distribution functions (bsdfs), and present an importance sampling algorithm for hair bsdf, which can drastically reduce the number of samples required for high quality hair rendering. (4). We look into the problem of out-of-core rendering with massive datasets which cannot fit in the main memory at one time. First, we present an efficient approach to construct out-of-core bounding box hierarchy (BVH). Then, we propose a simple level-of-detail (LOD) model based on point sampling which is inexpensive to compute and compact to store. Finally, we propose a few improvements to the virtual cone tracing algorithm, and present an out-of-core path tracing implementation based on our improved virtual cone tracing algorithm.
Acknowledgments

I would like to thank all the people who have helped me during my time at Dartmouth College. First I would like to express my sincere gratitude to my advisor Professor Fabio Pellacini for his continuous support of my Ph.D study and research, for his patience, inspiration, enthusiasm, and immense knowledge. He has not only provided guidance for my research and contributed to every piece of work in this thesis, but also taught me a lot in conducting rigorous and detailed academic research. For that, I could not have imagined having a better advisor and mentor for my Ph.D study.

I would also like to thank my thesis committee members Andrew Campbell, Devin Balkcom, and Jaroslav Křivánek, whose constructive comments and suggestions greatly helped me refine my solutions and inspired new ideas for the problems I faced during my thesis writing. My sincere thanks also goes to all the research collaborators in the past four years: Jaroslav Křivánek, Ondřej Karlík, Feng Xie, Priyamvad Deshmukh, and Parashar Krishnamachari. Without their help, this thesis would not have been possible.

I also want to thank my fellow laboratory-mates: Jonathan Denning, Brandon Kerr, and Xiaobo An, for the stimulating discussions, for the sleepless nights we worked together before SIGGRAPH deadlines, and for all the fun we have had in the last four years. I also would like to thank some fellow Ph.D students: Eric Kee, Nick Foti, and Ye Xu who helped me with my studies and provided me with perspectives.
Lastly, I would like to thank my family for all their love and encouragement. For my parents who raised me with a love of science and supported me in all my pursuits. And for my loving, supportive, encouraging, and patient wife Chanjuan who faithfully supports me during my Ph.D study. Thank you!

The *Sanmiguel* scene is the courtesy of Guillermo M. Leal LLaguno. The *Museum* scene is the courtesy of Alvaro Luna Bautista. The *Slum* scene is created by Pratik Solanki, and the *Studio* scene is created by Jiří Friml. The hair geometry models used in the ray tracing examples in Chapter 5 are the courtesy of Cem Yuksel. The hair geometry models used in the production rendering examples are the properties of DreamWorks Animation.
# Contents

Abstract ................................................................. ii  
Acknowledgments ......................................................... iv  

1 Introduction ............................................................. 1  

2 Light Transport, Monte Carlo, and Path Integration ................. 5  
    2.1 Light Transport ......................................................... 5  
    2.2 Monte Carlo Integration and Importance Sampling ............... 7  
        2.2.1 Monte Carlo Integration ......................................... 7  
        2.2.2 Multi-dimensional Integration .................................. 9  
        2.2.3 Importance Sampling ........................................... 10  
        2.2.4 Multiple Importance Sampling .................................. 12  
    2.3 Path Integration ...................................................... 14  
        2.3.1 Area Formulation of Rendering Equation ..................... 14  
        2.3.2 Path Integral Formulation ...................................... 16  
        2.3.3 Local Path Sampling ............................................ 17  
    2.4 Path-based Rendering Algorithms ................................ 18  
        2.4.1 Path Tracing ..................................................... 19  
        2.4.2 Photon Mapping ................................................ 21  
        2.4.3 Virtual Point Light ............................................. 22
### 3 Evaluating Progressive Rendering Algorithms in Appearance Design Tasks

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Overview</td>
<td>26</td>
</tr>
<tr>
<td>3.2 Progressive Rendering Algorithms</td>
<td>29</td>
</tr>
<tr>
<td>3.2.1 Selection of Algorithms</td>
<td>29</td>
</tr>
<tr>
<td>3.2.2 Implementation Detail</td>
<td>36</td>
</tr>
<tr>
<td>3.2.3 Other Methods</td>
<td>37</td>
</tr>
<tr>
<td>3.3 Experiment</td>
<td>37</td>
</tr>
<tr>
<td>3.3.1 Goal</td>
<td>37</td>
</tr>
<tr>
<td>3.3.2 Test Subjects</td>
<td>38</td>
</tr>
<tr>
<td>3.3.3 Scene Datasets</td>
<td>39</td>
</tr>
<tr>
<td>3.3.4 Trial Design</td>
<td>39</td>
</tr>
<tr>
<td>3.3.5 Questionnaire</td>
<td>41</td>
</tr>
<tr>
<td>3.3.6 Trial Procedure</td>
<td>43</td>
</tr>
<tr>
<td>3.4 Analysis</td>
<td>44</td>
</tr>
<tr>
<td>3.4.1 Time to Completion</td>
<td>45</td>
</tr>
<tr>
<td>3.4.2 Scene Complexity</td>
<td>47</td>
</tr>
<tr>
<td>3.4.3 Subjective Image Quality</td>
<td>47</td>
</tr>
<tr>
<td>3.4.4 Workflow in Matching Trials</td>
<td>48</td>
</tr>
<tr>
<td>3.4.5 Workflow in Open Trials</td>
<td>49</td>
</tr>
<tr>
<td>3.4.6 Resolution Switching</td>
<td>50</td>
</tr>
<tr>
<td>3.4.7 Algorithm Ratings and Rankings</td>
<td>52</td>
</tr>
<tr>
<td>3.5 Discussion and Future Work</td>
<td>54</td>
</tr>
<tr>
<td>3.6 Conclusions</td>
<td>57</td>
</tr>
</tbody>
</table>

### 4 LightSlice: Matrix Slice Sampling for the Many-Light Problem

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>58</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Summary of the notation used in this chapter .................................. 68

4.2 Rendering statistics comparing LightSlice, multidimensional lightcuts,
and matrix row column sampling .................................................. 75

5.1 Summary of notation. ................................................................. 93

6.1 Memory layout for different node types in bricks ............................... 122
6.2 Memory layout for triangles in bricks ........................................... 123
6.3 BVH building time for each scene at each stage ............................... 135
# List of Figures

1.1 Increasing rendering hours for DreamWorks Animation’s feature movie production over the years ........................................... 2

2.1 Comparison between different choices of pdfs .......................... 12
2.2 Sampling the lights, sampling the materials, and multiple importance sampling ................................................................. 13
2.3 Path sampling pattern for different rendering algorithms ............ 20

3.1 Progressive algorithm comparison for scene Studio .................. 31
3.2 Progressive algorithm comparison for scene Kitchen ................. 32
3.3 Progressive algorithm comparison for scene Museum ................ 33
3.4 Progressive algorithm comparison for scene Lobby .................. 34
3.5 User Interface for the experiment ............................................ 41
3.6 Lighting and material matching trials ...................................... 42
3.7 Lighting and material open trials .......................................... 43
3.8 Average time to completion ................................................. 46
3.9 Subjective image rating ....................................................... 48
3.10 Average time between user interactions in seconds .................. 50
3.11 Proportion of time users spent at each resolution .................... 51
3.12 Algorithm ratings ............................................................ 52
6.7 Quality comparison of our LOD path tracing and regular non-LOD path tracing on Slum scene ................................. 136
6.8 Quality comparison of our LOD path tracing and regular non-LOD path tracing on Sanmiguel scene ................................. 137
B.1 Sampling using Box-Muller transform ................................. 154
B.2 Error Images of [HR11] and our method ................................. 155
B.3 $L^2$ error plots of area light scene in Figure B.2 ................................. 156
Chapter 1

Introduction

In the computer graphics industry, the need for realism and fidelity in movies and games has been driving people to gradually adapt more advanced lighting models, realistic material models, and detailed geometry models to capture the complexity of the real world. These advancements in numerical models substantially increase the complexity of the rendering problem and the size of the datasets. Nowadays, computer graphics has reached an unprecedented level of complexity, and the growing complexity comes with a cost of more computational resource and longer rendering time.

Although computer hardware and software has been drastically improved over the last decade, the rendering task still takes a significantly large portion of time in the production pipeline compared to other tasks, e.g., animation, physically based simulation. In today’s feature movie production, rendering a movie frame in full quality may take hours or days to complete, and the number of rendering hours keeps climbing in spite of companies’ effort in improving the rendering infrastructures (i.e., building rendering farms with advanced hardware). Tens of millions rendering hours are usually dedicated for producing a single animated film (See Figure 1.1). Therefore, it
is extremely important to derive scalable algorithms for rendering scenes with highly complex models and large datasets.

In 3D computer graphics, rendering is a time-consuming simulation process that computes the light transport in a virtual scene. It is usually done by using Monte Carlo integration technique. Monte Carlo integration technique can robustly estimate the high-dimensional integrals in the rendering equations [Kaj86], making it a powerful tool for computing the light transport in complex scenes. However, Monte Carlo integration is also known to be inefficient due to its relatively slow converging rate of $\frac{1}{\sqrt{N}}$, where $N$ is the number of samples. Using insufficient samples causes undesirable artifacts in the final image (e.g., noise, blur, and banding). A naïve Monte Carlo rendering algorithm often requires a large number of samples to be drawn and evaluated in order to reconstruct a high quality image, resulting in a long rendering time.
Introduction

In this thesis, we focus on sampling techniques in rendering, because the effectiveness of the sampling process directly affects the efficiency of a Monte Carlo integrator. In the rest of this thesis, we present a series of investigations into improving the sampling techniques for the rendering of complex datasets.

Light transport and Monte Carlo Integration. Light transport and Monte Carlo integration serve as the foundation of our work. In Chapter 2, we begin with a brief review of these basic concepts: an introduction of background material including Monte Carlo integration, light transport, rendering equation, and importance sampling. It is followed by a description of path integral formulation and local path sampling technique. Then we examine three well-established categories of rendering algorithms, i.e., path tracing, photon mapping, and virtual point light algorithms. We briefly discuss the differences of their path sampling strategies, path reusing patterns, and the artifacts they introduce to the final image.

Progressive Rendering. Given the fact that Monte Carlo based rendering algorithms take a long time to converge, progressive rendering algorithm is a good alternative when fast feedback is needed (e.g., in the case of material editing and light editing). In Chapter 3, we implement the progressive version of path tracing, photon mapping, and virtual point light algorithms discussed in Chapter 2. We conduct a user study to evaluate the effectiveness of these progressive algorithms in the context of appearance design.

Global illumination. Global illumination simulates the complex light flow of the real world based on the laws of physics. It greatly enhances the fidelity of the result image.
Introduction

However, due to the high dimensionality and the complexity of light transport problem, a brute-force Monte Carlo integration is expensive to compute. In Chapter 4, we looked into a problem of global illumination named many-light problem. We present LightSlice, a global illumination algorithm that solves many-light problem efficiently using matrix slice sampling and virtual point light clustering.

**BSDF Importance Sampling.** An object’s interactions with the light can be defined with a bidirectional scattering distribution function (bsdf). In order to faithfully model the subtle details of surfaces’ appearance in the real world, complex bsdfs are often used. The efficiency of bsdf sampling and evaluation has a direct impact to the overall efficiency of the rendering process. In Chapter 5, we tackle the problem of importance sampling for hair bsdf. We propose an efficient importance sampling algorithm for hair bsdf which is easy to implement, has no significant memory overhead, and needs no pre-computation.

**Out-of-core Rendering.** Plants, hairs, and many man-made structures have a high level of structural details. Complex geometric model are often used to capture those detail structures. In modern movie production, geometric complexity has reached an unprecedented level [PFHA10, KTO11]. The growing complexity in geometry not only increases the cost of visibility tests, but also requires more system memory and disk storage. When the dataset cannot fit in the main memory at one time, the rendering problem becomes out-of-core. Rendering algorithm that does not have an I/O-optimized sampling strategy will be I/O bounded. In Chapter 6, we look into the problem of out-of-core global illumination and present several techniques to improve the efficiency of out-of-core rendering.
Chapter 2

Light Transport, Monte Carlo, and Path Integration

2.1 Light Transport

Given the description of a virtual world, a rendering system simulates the physics of light transport to generate accurate and convincing images. One of the most important concepts of light transport, the rendering equation, was introduced by Kajiya in 1986 [Kaj86]. The rendering equation describes all light transport mechanisms as recursive integrals. The following is the hemispherical formulation of the rendering equation, which is one of the most commonly used formulations in rendering.

\[
L(x, \omega_o) = L_e(x, \omega_o) + L_r(x, \omega_o) 
\]  
(2.1)

\[
L_r(x, \omega_o) = \int_{\Omega} L(x, \omega_i) f_r(x, \omega_o \leftarrow \omega_i)(N(x) \cdot \omega_i) d\omega_i 
\]  
(2.2)
2.1 Light Transport

$L(x, \omega_o)$ is the radiance at surface point $x$ in direction $\omega_o$. $L_e(x, \omega_o)$ represents the self-emitted radiance of the surface (usually a light source) at location $x$ in direction $\omega_o$. $L_r(x, \omega_o)$ is the radiance that is reflected by the surface at location $x$ in direction $\omega_o$. $f_r(x, \omega_o \leftarrow \omega_i)$ is the bidirectional reflection distribution function (brdf) of direction $\omega_i$ and $\omega_o$ at $x$. $N(x)$ is the surface normal at $x$ and $(\cdot)$ is the dot product of two unit vectors. If we substitute Equation 2.2 into Equation 2.1, we can see that $L(x, \omega_o)$ appears on both side of the equation.

$$L(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} L(x, \omega_i) f_r(x, \omega_o \leftarrow \omega_i)(N(x) \cdot \omega_i) d\omega_i \quad (2.3)$$

In spite of the simplicity of the rendering equation, it is almost impossible to solve the light transport problem analytically. The complexity of the light transport problem lies in the complex light source models, physically-based brdf models, arbitrary scene geometry, and the intricate visibility relationship between scene objects.

Moreover, the recursive nature of the rendering equation makes light transport a high-dimensional integration problem, for which deterministic solutions will suffer from the “curse of dimensionality”. To calculate light transport numerically, we resort to a combination of Monte Carlo integration and ray tracing algorithms. In Section 2.2, we will discuss Monte Carlo integration, a numerical tool that is widely used in rendering to solve high-dimensional integrals. Then we introduce a variance reduction technique named importance sampling. In Section 2.3, we discuss the path integral formulation of light transport problem. In Section 2.4, we will discuss three famous categories of rendering algorithms that compute light transport using different path sampling and reusing strategies.
2.2 Monte Carlo Integration and Importance Sampling

The term Monte Carlo refers to a set of mathematical techniques that use statistical sampling to numerically evaluate the value of functions or approximate solutions to quantitative problems. For the history and mathematical details of Monte Carlo methods, we refer the reader to [HHW65] and [Sob94]. In this chapter, we only focus on a subset of Monte Carlo methods, namely Monte Carlo integration, a technique that allows us to solve light transport problems numerically.

2.2.1 Monte Carlo Integration

Monte Carlo method can be applied to estimate function integrals. Suppose we want to compute the following integral:

$$ I = \int_{\Omega} f(x) dx $$  \hspace{1cm} (2.4)

The basic idea of Monte Carlo integration is to approximate the function integral with a set of randomly generated samples. Specifically, $N$ random samples $X_1, X_2, \ldots, X_N$ are independently drawn from a probability density function (pdf) $p(x)$. Using these random samples, integral $I$ can be approximated as:

$$ \hat{I}_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} $$  \hspace{1cm} (2.5)
The expected value of this estimator is computed as follow:

\[
E \left[ \hat{I}_N \right] = E \left[ \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \right]
= \frac{1}{N} \sum_{i=1}^{N} E \left[ \frac{f(X_i)}{p(X_i)} \right]
= \frac{1}{N} \int_{\Omega} \frac{f(x)}{p(X_i)} p(x_i) dx
= \int_{\Omega} f(x) dx
= I
\]

The expected value of the estimator is exactly the same as integral \( I \), therefore this Monte Carlo estimator is unbiased. Unbiasedness is one of the most important properties of Monte Carlo integration. It means that Monte Carlo integration will always converge to the correct solution as long as enough samples are given. If the expected value of the estimator is different from the correct solution, the estimator is biased and the bias value can be computed as: \( B[\hat{I}_N] = E[\hat{I}_N] - I \). In Section 2.4, we will introduce two kinds of biased rendering algorithms, which are photon mapping and virtual point light algorithms. One benefit of using a biased estimator is that it usually converges faster than an unbiased one.
2.2 Monte Carlo Integration and Importance Sampling

The variance of the estimator is computed as following:

\[
V[\hat{I}_N] = V \left[ \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \right] \\
= \frac{1}{N^2} V \left[ \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \right] \\
= \frac{1}{N^2} \sum_{i=1}^{N} V \left[ \frac{f(X_i)}{p(X_i)} \right] \\
= \frac{1}{N} V \left[ \frac{f(X_i)}{p(X_i)} \right] \\
= \frac{1}{N} \int_{\Omega} \left( \frac{f(X_i)}{p(X_i)} - 1 \right)^2 p(X_i) dx
\]

As the \( N \) increases, the variance of the estimator decreases linearly with \( N \). However, the error in the estimator is proportional to the standard deviation \( \sqrt{V[\hat{I}_N]} \), which decreases linearly with \( \sqrt{N} \). That means, in order to decrease the error of the Monte Carlo estimator by a half, three times more samples need to be drawn and evaluated. This is the root cause of the relatively slow converging rate of Monte Carlo estimators. Many variance reduction techniques were proposed to reduce the variance of the estimator in order to reduce the number of samples needed for convergence. In Section 2.2.3, we will see one of those techniques, named importance sampling.

2.2.2 Multi-dimensional Integration

In Section 2.2.1, we introduced the Monte Carlo integrator for one-dimensional integrals. Although it is not the most efficient solution for computing one-dimensional integrals, Monte Carlo integration can be extended to multiple dimensions, and in some of those cases, Monte Carlo integration is the only feasible solution. This extension is
2.2 Monte Carlo Integration and Importance Sampling

straightforward:

\[ I = \int \int f(x, y) dx \, dy \quad \rightarrow \quad \hat{I}_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i, Y_i)}{p(X_i, Y_i)} \]

The expected value and variance derived in Section 2.2.1 still hold for high-dimensional integrals. More importantly, the error (or variance) of the estimator is independent of the dimensionality of the integral, which means that the computational cost will not increase exponentially as the dimensionality of the integral increases, as it would with deterministic solutions to multi-dimensional integrals. This property of Monte Carlo integrator makes it an excellent tool for high-dimensional integrations.

2.2.3 Importance Sampling

As we discussed in Section 2.2, one of the biggest disadvantages of Monte Carlo integrator is its relatively slow converging rate of \( O(1/\sqrt{N}) \), which means in order to halve the error, we need to quadruple the number of samples. Many variance reduction techniques were proposed, including importance sampling, stratified sampling, control variates, adaptive sampling, etc.. In this section, we will discuss the importance sampling technique, which is the cornerstone of many works in this thesis. For other variance reduction techniques, we refer the readers to [DBB06] and [CPF10].

Importance sampling is a variance reduction technique that uses non-uniform \( pdfs \) to generate samples. Recall the Monte Carlo integration for function \( f(x) \) in Section 2.2 (Equation 2.5). The importance sampling technique exploits the fact that the variance of a estimator \( \hat{I}_N \) reduces quickly if the samples are drawn from a \( pdf \ p(x) \) that is similar to the integrand \( f(x) \). The intuition behind this is that the estimator is
2.2 Monte Carlo Integration and Importance Sampling

more effective if we can concentrate the samples on where the integrand is relatively large, so that each sample can have relatively high contribution.

In an ideal situation, if the chosen $pdf$ is proportional to function $f(x)$, we can have an estimator with zero variance. Suppose we can choose $p'(x)$ so that $p'(x) \propto f(x)$, or $p'(x) = cf(x)$, where $c$ is a constant. Because $p'(x)$ need to be normalized to one,

$$\int_{\Omega} p'(x)dx = \int_{\Omega} cf(x)dx = 1$$

it is easy to show that

$$c = \frac{1}{\int_{\Omega} f(x)dx} = \frac{1}{I}$$

If we use $p'(x)$ in the estimator, we can estimate $I$ even with a single sample ($N = 1$).

$$\frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p'(X_i)} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{c} = \frac{1}{c} = I$$

Furthermore, since every estimate would have the same value, the variance of the estimator is zero.

However, it is not practical to have $p'(x)$, because in order to compute $c$ we need to have knowledge of integral $I$, which is the value we are trying to compute. Fortunately, as long as we can choose a $pdf$ $p(x)$ that is similar to $f(x)$, the variance can still be reduced. However, if the chosen $pdf$ is a poor match for the integrand $f(x)$, the variance will increase. Figure 2.1 shows three different choices of $p(x)$ for integrand $f(x)$.

In practice, it is also important to choose a $pdf$ that is simple and efficient to sample and evaluate. There are a few common strategies to choose a importance $pdf$ [CPF10]:

11
2.2 Monte Carlo Integration and Importance Sampling

Figure 2.1: Comparison between different choices of pdfs. (A). Uniform sampling, which is the simplest sampling strategy. The samples are drawn with uniform probability. Due to its simplicity, it is often used when there is no better sampling strategy available. (B). Importance sampling. By using a pdf \( p(x) \) that is similar to \( f(x) \), samples are concentrated on the regions where the value of integrand \( f(x) \) is relatively high, hence the variance of the estimator is reduced. (C). Sampling from a pdf that is a poor match for the integrand \( f(x) \), the variance will increase, since most samples will have low contribution while only a few of them will have very high contribution.

1. Use some parts of \( f(x) \) that can be integrated analytically.

2. Use a low-dimensional discrete approximation of \( f(x) \).

3. Use the first few terms of \( f(x) \)'s Taylor expansion.

We will see some of these strategies be applied to derive importance sampling strategies in the rest of this thesis.

2.2.4 Multiple Importance Sampling

In rendering, we often encounter integrals that are a product of two or more functions. In that case, sampling with a single pdf might not be able to capture all the importance regions. For example, when rendering scenes that contain both diffuse and glossy surfaces illuminated by small and large area lights, drawing samples from the lights' distribution is more effective for diffuse surfaces and small lights, while drawing
2.2 Monte Carlo Integration and Importance Sampling

samples from the brdfs’ distribution is a better strategy for glossy surfaces illuminated by large lights (See Figure 2.2.(A) and Figure 2.2.(B)).

![Images of sampling strategies](image)

**Figure 2.2:** Sampling the lights, sampling the materials, and multiple importance sampling. (A). Sampling only the pdf of the lights. (B). Sampling on the pdf of the brdfs. (C). Sampling both the lights’ pdf and the brdfs’ pdf. The samples are combined using multiple importance sampling (images from [VG95]).

Multiple importance sampling proposed by Veach addresses this problem by optimally combining multiple sampling strategies to minimize the variance without introducing bias [VG95]. Suppose we have $m$ different pdfs: $p_1(x), p_2(x), \ldots, p_m(x)$, which work well in different regions of the integrand but not the entire integrand. Multiple importance sampling combines all these sampling strategies using the estimator:

$$\hat{I} = \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(X_{i,j}) \frac{f(X_{i,j})}{p_i(X_{i,j})}$$

where $w_i(X_{i,j})$ is the weight for sample $X_{i,j}$ drawn from sampling strategy $p_i$. In order to have an unbiased estimator, all the weights need to be non-zero and normalized to have a sum of one. Veach and Guibas proved that near optimal weights can be
2.3 Path Integration

computed using the *balance heuristic* [VG95]:

\[ w_i(x) = \frac{n_i p_i(x)}{\sum_k n_k p_k(x)} \]

In practice, suggested by Veach, the *power heuristic* with exponent \( \beta = 2 \) might work better in combining the light samples and *brdf* samples.

\[ w_i(x) = \frac{(n_i p_i(x))^{\beta}}{\sum_k (n_k p_k(x))^{\beta}} \]

Figure 2.2.(C) shows the rendering result with multiple importance sampling, which is effective over all surfaces and light sources. In Chapter 5, we will use multiple importance sampling to combine hair scattering samples and light samples.

2.3 Path Integration

To compute the color value of a pixel, light transport paths between the light sources and the points on the virtual camera are generated and the energy carried along these paths is computed. Transforming light transport problems into path integration problems allows us to apply the Monte Carlo integration technique in Section 2.2 to compute their solution.

2.3.1 Area Formulation of Rendering Equation

In order to discuss the path formulation and path space, we need to reformulate the rendering equation presented in Section 2.1 (Equation 2.3). We first define the exitant
2.3 Path Integration

radiance from point \( x' \) to point \( x \) as

\[
L(x \leftarrow x') = L(x', \omega_o)
\]

where \( \omega = \hat{x} - \hat{x}' \) is a unit vector from \( x' \) to \( x \) defined as \( \hat{x} - \hat{x}' = \frac{x - x'}{||x - x'||} \). We can also define the \textit{brdf} at location \( x' \) as

\[
f_r(x \leftarrow x' \leftarrow x'') = f_r(x', \omega_o \leftarrow \omega_i)
\]

where \( \omega_o = \hat{x} - \hat{x}' \) and \( \omega_i = \hat{x}'' - \hat{x}' \). The new formulation contains a geometric term which consists of the projection product in the hemisphere formulation and a Jacobian that transforms the integral over direction into the one over surface area.

\[
G(x' \leftrightarrow x'') = V(x' \leftrightarrow x'') \frac{|(N(x') \cdot \omega_i)(N(x'') \cdot -\omega_i)|}{||x' - x''||^2}
\]

where \( N(x) \) is the surface normal at \( x \). \( V(x' \leftrightarrow x'') \) is the visibility function of point pair \( x' \) and \( x'' \). \( V(x' \leftrightarrow x'') = 1 \) if \( x' \) and \( x'' \) are mutually visible, and \( V(x' \leftrightarrow x'') = 0 \) otherwise. In practice, visibility functions are usually evaluated by ray tracing or shadow mapping. Substituting these terms into Equation 2.3, we rewrite the rendering equation into an integral over surface area.

\[
L(x \leftarrow x') = L_e(x \leftarrow x') + \int_{\mathcal{A}} L(x' \leftarrow x'')f_r(x \leftarrow x' \leftarrow x'')G(x' \leftrightarrow x'')dA(x'') \quad (2.6)
\]

where \( \mathcal{A} \) is the union of all scene surfaces and \( dA(x'') \) is the area measure of \( x'' \) on \( \mathcal{A} \). Equation 2.6 is called the area formulation of rendering equation, which is equivalent to Equation 2.3 [PH10].
2.3 Path Integration

2.3.2 Path Integral Formulation

The area formulation allows us to transform light transport problem into a path integral problem. To do this, we recursively expand Equation 2.6 [Vea97]:

\[
L(x_0 \leftarrow x_1) = L_e(x_0 \leftarrow x_1)
\]

\[
+ \int_{\mathcal{A}} L_e(x_1 \leftarrow x_2) f_r(x_0 \leftarrow x_1 \leftarrow x_2) G(x_1 \leftarrow x_2) dA(x_2)
\]

\[
+ \int_{\mathcal{A}} \int_{\mathcal{A}} L_e(x_2 \leftarrow x_3) f_r(x_1 \leftarrow x_2 \leftarrow x_3) G(x_2 \leftarrow x_3)
\times f_r(x_0 \leftarrow x_1 \leftarrow x_2) G(x_1 \leftarrow x_2) dA(x_2) dA(x_3)
\]

\[
+ \cdots
\]

\[
= \sum_{k=1}^{\infty} \int_{\mathcal{A}^{k-1}} L_e(x_{k-1} \leftarrow x_k)
\times \left( \prod_{i=1}^{k-1} f_r(x_{i-1} \leftarrow x_i \leftarrow x_{i+1}) G(x_i \leftarrow x_{i+1}) \right) dA(x_2) \cdots dA(x_k)
\]

Considering each \( x_i \in \mathcal{A} \) to be a vertex in a path, a path of length \( k \) can be defined as:

\[
\bar{x} = x_0 x_1 \ldots x_k
\]

where \( 1 \leq k < \infty \) and \( x_i \in \mathcal{A} \). Usually the \( x_k \) is a point on the light source while \( x_0 \) is a point on the camera. In order to apply the Monte Carlo estimator, we want to express each measurement in the form of Equation 2.4. First we define the measure of a path as a product of surface area measures of each vertex.

\[
d\mu_k(\bar{x}) = dA(x_2) \cdots dA(x_k)
\]
2.3 Path Integration

Note the missing $d\mathcal{A}(x_0)$ and $d\mathcal{A}(x_1)$ in equation 2.9, that is because vertices $x_0$ and $x_1$ are not created by path sampling. They are given as parameters of Equation 2.6. Equation 2.8 can be rewritten as:

$$L(x_0 \leftarrow x_1) = \sum_{k=1}^{\infty} \int_{\mathcal{A}^{k-1}} f_k(\bar{x}) d\mu_k(\bar{x})$$

where

$$f_k(\bar{x}) = L_e(x_{k-1} \leftarrow x_k) \times \left( \prod_{i=1}^{k-1} f_i(x_{i-1} \leftarrow x_i \leftarrow x_{i+1}) G(x_{i+1} \leftrightarrow x_i) \right)$$

We define measurement $I^k$ for paths of length $k$, and $I^*$ is the measurement of all paths.

$$I^k = \int_{\mathcal{A}^{k-1}} f_k(\bar{x}) d\mu_k(\bar{x}) \quad \text{and} \quad I^* = \sum_{k=1}^{\infty} I^k$$

The the integrals of different path length $k$ are independent to each other. We can approximate $I^k$ as

$$\hat{I}^k_N = \frac{1}{N} \sum_{i=1}^{N} f_k(\bar{x})$$

where $p_k(\bar{x})$ is the pdf of generating path $\bar{x}$.

2.3.3 Local Path Sampling

To apply the Monte Carlo estimator, we must be able to evaluate $f_k(\bar{x})$ and $p_k(\bar{x})$. $f_k(\bar{x})$ can be explicitly evaluated using the rendering equation, while $p(\bar{x})$ not only depends on the path $\bar{x}$, but also depends on how the path is generated. A path is usually generated by sampling a sequence of vertices or connecting two existing subpaths. Usually $x_k$ is on the light source while $x_0$ is on the camera.
2.4 Path-based Rendering Algorithms

Theoretically, paths can be generated by randomly sampling surface points and joining them by verifying their connectivity. In practice, this approach is usually not efficient due to the intricate visibility relationship between arbitrary scene objects. A more common approach is to generate paths incrementally using local path sampling. Local path sampling generates one vertex at a time based on local information at existing vertices (light distribution or brdf). There are two common strategies for local path sampling.

- **Generating a new vertex by sampling scene surfaces.** This strategy is usually used to generate vertices on light sources or on the camera, where some *a priori* distributions are available.

- **Generating a new vertex by sampling the brdf of existing vertex.** Another approach is to sample a scattering direction according to the brdf of an existing vertex, and then cast a ray to find the first intersection point as the new vertex.

A new path can also be created by connecting two vertices on two separate subpaths. Instead of generating a new vertex, the visibility between two vertices is tested to verify their connectivity. In this way, existing vertices are reused to form new paths. In Section 2.4, we will introduce three different categories of rendering algorithms that use different strategies to connect subpaths.

### 2.4 Path-based Rendering Algorithms

The path integral formulation does not specify how the paths are generated and reused. Different rendering algorithms can choose different strategies for path sampling and reusing. Generally speaking, most of the rendering algorithms can be classified as one of the following algorithms or their variants: *Path Tracing*, *Photon Mapping*, and
2.4 Path-based Rendering Algorithms

Virtual Point Light algorithms (Figure 2.3).

2.4.1 Path Tracing

Introduced by Kajiya [Kaj86], path tracing has recently gained popularity in industry practice for its ease of use and robustness [KFC'10]. Various effects, e.g., anti-aliasing, soft shadow, depth-of-field, motion blur, etc., can be achieved using the path tracing framework [CPC84]. A classical path tracing algorithm generates paths starting from the camera. Each path bounces multiple times in the scene and finally connects to the light sources to form a complete path. Whenever a new vertex is generated, the length of the path is extended. Therefore, shorter paths are reused to form longer paths. But paths with the same path length do not share vertices with each other. Path tracing is an unbiased algorithm, but it exhibits high-frequency noise in the image until it is converged (See Figure 2.3.(A)).

Paths can also be generated starting from the light sources. This variant of path tracing algorithm is called Light Tracing [DBB06]. As compared to classical path tracing, light tracing is very efficient in sampling some types of light transport paths, e.g., specular caustics, highly occluded light sources. However, in general, light tracing performs worse than classical path tracing because light tracing is not able to concentrate path samples on scene regions that are more important to the final images.

Bidirectional path tracing is proposed independently by both Veach [VG94] and Lafortune [LW93]. It can be viewed as a combination of classical path tracing and light tracing. Each time, bidirectional path tracing generates two subpaths. One starts from the camera and the other starts from the light sources. Then, the vertices of two subpaths are connected to form complete paths, and paths with same length are combined.
2.4 Path-based Rendering Algorithms

Figure 2.3: Path Tracing, Photon Mapping, and VPL algorithm use very different path sampling and reuse strategy, therefore they demonstrate very different artifacts patterns.
2.4 Path-based Rendering Algorithms

using multiple importance sampling. Subpaths are reused many times so the cost of sampling can be amortized. In general, bidirectional path tracing converges faster than both classical path tracing and light tracing because it combines the advantages of both algorithms and is able to efficiently sample different types of light transport.

2.4.2 Photon Mapping

Photon mapping algorithm, proposed by Jensen, is a two-pass algorithm [JC95, Jen96, JC98, Jen01]. In the first pass, a large number of light subpaths are traced starting from the light sources. For each vertex, a photon carrying flux information is deposited. Then a data structure (usually a KD-tree) is built to cache all the photons for fast spatial look up. In the second pass, the image is rendered using the information in the photon map. Typically, subpaths are traced from the camera. At each vertex of the subpaths, the density of photons is estimated to compute the outgoing radiance. Therefore, photon mapping is also a bidirectional method like bidirectional path tracing. The only difference is that photon mapping algorithms use particle density estimation instead of directly connecting vertices. In fact, it has been proved independently by Georgiev [GKDS12] and by Hachisuka [HPJ12] that photon mapping and bidirectional path tracing can be combined to work together to efficiently capture difficult light paths. They also proved that density estimation is just another way to share light subpaths. Depending on the location of density estimation events (i.e., vertex locations of camera subpaths), a light subpath may be reused multiple times or never used at all.

Photon mapping generally generates smoother results as compared to path tracing and its variants, because the density estimation can filter the high-frequency signal in light transport. However, it is also important to note that using density estimation with
2.4 Path-based Rendering Algorithms

A finite number of photons introduces bias, which manifests itself as blurriness in the final images (See Figure 2.3.(B)). To make photon mapping an unbiased algorithm, one needs an infinite number of photons. However, it is impractical to cache an infinite number of photons in memory. To solve this problem, Hachisuka et al. [HJW’08, HJ09] introduced progressive photon mapping. By intertwining the photon shooting pass and rendering pass, progressive photon mapping allows the use of an infinite number of photons with a small memory footprint. However, it is also proved to have a slower convergence rate than path tracing in general [KZ11].

2.4.3 Virtual Point Light

Virtual point light, proposed by Kelly [Kel97], is a category of algorithms that use a set of point lights to approximate global illumination. Like photon mapping, virtual point light algorithms are also two-pass algorithms. In the first pass, subpaths starting from the light sources are generated. However, instead of depositing a photon at each vertex, a virtual point light (VPL) is placed. Each VPL is associated with a source radiance, a position, a surface normal, and a radiance distribution (usually specified using a brdf). In the second pass, subpath of length one is generated starting from the camera and connected to each VPL. Thus the light subpath associated with each VPL is reused by all camera subpaths. Depending on the implementation, the VPL shooting pass and rendering pass can also intertwined. VPLs can be generated in batches at each iteration or all at once at the beginning.

It is important to note that the paths generated by VPL algorithms are highly correlated. It makes the result images look smooth without the high frequency noise manifested in path tracing images or the low frequency blurriness manifested in im-
2.4 Path-based Rendering Algorithms

images rendered with photon mapping. Moreover, the visibility tests between a VPL and all the camera paths can be computed using shadow mapping, which have extremely efficient implementations using rasterization pipelines. Due to their efficiency and the absence of visually disturbing artifacts, VPL algorithms are widely used in real-time global illuminations [DS05, DS06, LSK'07, RGK'08, CNS’11].

Unfortunately, the sampling strategy of VPL algorithms is not always ideal. Because VPLs are a discrete representation of light transport, insufficient VPLs cause banding artifacts in object shadows (See Figure 2.3.(C)). This problem can be solved by increasing the number of VPLs. The other limitation of VPL algorithms is the weak singularity problem caused by having insufficient VPLs in some regions of the scene (e.g., object corners, area around the peak of a glossy lobe). When connecting the camera subpath to one of those VPLs, the \textit{brdf} and the geometric term may have a very large value as compared to the \textit{pdf} of the path, producing a “spike” of illumination in the image. A common solution is to clamp the \textit{brdf} and the geometric terms to avoid the “spike”. However, the clamping causes energy loss, thus introduces bias. Various solutions were proposed to compensate the energy lost due to clamping. Kollig and Keller proposed an unbiased approach based on path tracing [KK04]. Hašan et al. proposed to use virtual spherical lights (VSL) instead of VPL to replace the usual point sampling by an integration over some non-zero domain [HKWB09]. Davidovič et al. extended Kollig and Keller’s solution by using the GPU to evaluate and share locally traced paths with approximated visibility [DKH’10]. A progressive clamping approach is also proposed [DGS12]. Novák et al. introduced a image-space approach for compensating the VPL bias [NED11].

VPL algorithms usually require a large number of VPLs (i.e., 100k) to achieve high
2.4 Path-based Rendering Algorithms

quality rendering. Many approaches were proposed to improve the efficiency of VPL algorithms [WFA'05, WABG06, HPB07, GS10, DGS12, PKD12]. All these algorithms can be considered as importance sampling strategies for selecting VPLs with high contribution. In Chapter 4, we propose an efficient VPL algorithm named LightSlice, which uses matrix slice sampling and VPL clustering to efficiently render a large number of VPLs for complex light transport.
Chapter 3

Evaluating Progressive Rendering Algorithms in Appearance Design Tasks

As discussed in Chapter 1, in feature movie production, it is time consuming to render images to their full quality. When artists are performing appearance editing tasks, e.g., lighting or material editing, they need to wait for a long time to see the result images of their edits before making further adjustments, resulting in a long turnaround time. An alternative is to use progressive rendering algorithms. Progressive algorithms give the artists instant feedback with render artifacts, but keep converging to the final images. Using the imperfect feedback from progressive rendering algorithms, artists are able to make early decisions and perform adjustments without waiting for a full-quality render.

In Section 2.4, we discussed three kinds of rendering algorithms and the differences in their path sampling and reusing strategies. These algorithms can be implemented as progressive rendering algorithms. However, the images created by these progressive rendering algorithms can exhibit different kinds of artifacts (i.e., noise, blurriness, banding) at the early stage of rendering. We are interested in understanding how
3.1 Overview

these artifacts effect people's judgment in appearance editing tasks. In this chapter, we present a user study that investigates the effects of these artifacts on user performance in appearance design tasks, i.e., lighting, material editing. Specifically, we asked both novice and expert subjects to perform lighting and material editing tasks with four algorithms: random path tracing, quasi-random path tracing, progressive photon mapping, and virtual point light (VPL) algorithm. We collected questionnaires and interface statistics to analyze their performance using these algorithms.

This chapter is organized as follows: we start in Section 3.1 with an overview of appearance design and progressive rendering, followed by a detailed discussion of several progressive rendering algorithms and their implementation details in Section 3.2. Section 3.3 presents a user study of progressive rendering in the context of appearance design. We discuss the goal and the setting of our experiment, including the test subjects, test environment, datasets used in the test, user interfaces, questionnaires, and the trial procedures. In Section 3.4, we analyze the data collected during the experiment, and present the result of our study along with some interesting findings of the experiment. In Section 3.5 we discuss the limitations of our study and some future works. Finally, in Section 3.6, we summarize our findings and conclude our user study.

3.1 Overview

Appearance design, i.e., the editing of lights and materials, is fundamental in the creation of computer-generated imagery, with a significant impact on the final image look. Kerr and Pellacini showed that in these tasks designers proceed mainly by trial-and-error, reporting on average 5 to 10 minutes for the manipulation of point lights and single BRDFs, among the simplest of design tasks [KP09, KP10]. These tasks would
3.1 Overview

take even longer in scenes with complex geometry and animation. More importantly, the timings reported assume that the renderer is fast enough to provide the user with immediate feedback. In practice, however, rendering realistic lighting and materials in complex environments, including global illumination effects, takes at least a few minutes.

Precomputation-based approaches have been introduced to speed-up rendering in the context of appearance design of complex environments \([\text{PVL}^*05, \text{HPB}06]\). These methods either support only one design task or do not allow artists to move geometry or camera. More importantly, they are based on approximations that cannot guarantee that exactly the same image will be generated by the final renderer, possibly misguiding the designer’s effort.

For this reason, progressive rendering is becoming a popular alternative for providing fast feedback in appearance design tasks. A progressive renderer avoids pre-computation completely. Instead, it gradually improves the image quality until it converges to the final image, while providing the user with visual feedback during the entire course of computation. During design tasks, the renderer is restarted each time a scene changes, providing instantaneous feedback. At the early stages of computation, though, the image can contain various kinds of visual artifacts, such as high- or low-frequency noise or banding, that can interfere with the design task. Which of these artifacts are least objectionable in appearance design is an open question that we strive to answer.

In this chapter, we present a user study that investigates the effects of the different artifacts produced by progressive renderers on user performance in appearance design tasks. Out of the large variety of progressive rendering algorithms, we chose the fol-
3.1 Overview

The following four: purely random unstratified path tracing (showing high-frequency uncorrelated noise), quasi-random path tracing (showing high-frequency correlated noise), progressive photon mapping (showing low-frequency noise) and virtual point lights rendering (showing banding). We chose these methods, because (1) they span different types of visual artifacts, (2) they converge to the final rendered image, (3) they are used in practice, and (4) they have no initial latency, thereby supporting frequent user interactions.

In our study, fourteen expert subjects and twelve novice subjects perform simple tasks involving lighting and material design, receiving feedback from each of the aforementioned algorithms. In the light matching task, the subjects are asked to adjust a single parameter (either position or size) of one area light to match the given target image. In the material matching task the subjects adjust one of the color, glossiness, or roughness parameters of an object. In the lighting and material open trials, the subjects are asked to choose their preferred design out of eight predefined configurations of lighting or materials respectively. Subjects work with four scenes of varying complexity and extent shown in Figure 3.1 – Figure 3.4. We collect quantitative and qualitative data by recording all user interface actions and by asking subjects to fill out questionnaires, collecting ratings, rankings, and comments on each progressive renderer. By analyzing this data, we draw the following conclusions:

- Both path tracers are strongly preferred to progressive photon mapping and VPL rendering. As suggested by the time to completion and the algorithm rating/ranking, in appearance design tasks, users can cope better with the high-frequency artifacts of the path tracers, than the low-frequency noise and banding of progressive photon mapping and VPL rendering.
3.2 Progressive Rendering Algorithms

- The random and quasi-random path tracing are not systematically preferred to one another; the same holds between progressive photon mapping and VPL rendering.
- The four different algorithms do not result in any significant difference in user workflow.
- While experts are faster and more accurate than novices, the two groups have surprisingly similar preferences for the algorithms and exhibit similar workflow.

Although the results of the study apply strictly only to our scenes and tasks, we believe that they provide a valuable insight into developing progressive algorithms that specifically target appearance design.

3.2 Progressive Rendering Algorithms

3.2.1 Selection of Algorithms

We are interested in appearance design tasks in the context of realistic imagery. A wide range of rendering algorithms have been implemented to support realistic rendering, only a limited number of which were included in our study. The following criteria guided our selection.

- **Different types of image errors.** We include the algorithms that exhibit different types of image errors, including high-frequency noise, low-frequency splotches and banding.
- **Convergence.** Recent real-time global illumination algorithms achieve high performance at the cost of approximations that may compromise image quality. To avoid affecting our results, we only consider algorithms that are known to con-
3.2 Progressive Rendering Algorithms

verge to the correct solution, including both biased and unbiased techniques.

• **Used in practice.** We include algorithms that are widely used in industry and academia.

• **Minimum latency.** To support interactivity, rendering must restart immediately after any user action. For this reason we do not use algorithms that involve significant preprocessing, such as classic photon mapping [Jen01] and Lightcuts [WFA*05].

Based on these criteria we select the following four algorithms for our study: (1). random path tracing, (2). quasi-random path tracing, (3). progressive photon mapping, and (4). virtual point light (VPL) rendering. Figure 3.1 – 3.4 show the algorithms as they converge on each scene used in our study. Note that while the different algorithms converge at similar speed according to the $L^2$ image error shown in the graphs, they exhibit very different artifacts. All images are able to be fully converged in a couple minutes at a resolution of 512×512 pixels on a 27” iMac with a 4-core 2.93 GHz Intel Core i7 processor and 16 GB RAM.

**Path tracing** Path tracing has recently gained popularity in industry practice for its ease of use and robustness [KFC*10]. Random path tracing is a Monte Carlo solution of the rendering equation that provides unbiased results. A quasi-Monte Carlo version based on strictly deterministic number sequences can provide faster convergence in some situations [Kel03]. Non-converged images generated by path tracing exhibit unstructured high-frequency noise (pure random version) or structured high-frequency patterns (quasi-random version). Bidirectional path tracing was not included in the study because it exhibits the same kind of error as a path tracer in our scenes.

Our implementation of the pure random path tracing algorithms uses a simple pseu-
3.2 Progressive Rendering Algorithms

![Converged images](image1)

![Progressive error graphs](image2)

**Figure 3.1:** Fully converged image and two partially converged images captured after 1 second and 10 seconds of scene Studio for each algorithm. Zoomed-in insets from the partially converged images show the artifacts in detail. A convergence graph of $L^2$ error plotted against time for the first 200 seconds.
3.2 Progressive Rendering Algorithms

![Converged](image)

<table>
<thead>
<tr>
<th></th>
<th>1 second</th>
<th>10 seconds</th>
<th>1 second (closeup)</th>
<th>10 seconds (closeup)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VPL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.2:** Fully converged image and two partially converged images captured after 1 second and 10 seconds of scene *Kitchen* for each algorithm. Zoomed-in insets from the partially converged images shows the artifacts in detail. A convergence graph of $L^2$ error plotted against time for the first 200 seconds.
3.2 Progressive Rendering Algorithms

![Converged](image1)

![1 second](image2)
![10 seconds](image3)

![1 second (closeup)](image4)
![10 seconds (closeup)](image5)

**Figure 3.3:** Fully converged image and two partially converged images captured after 1 second and 10 seconds of scene Museum for each algorithm. Zoomed-in insets from the partially converged images shows the artifacts in detail. A convergence graph of $L^2$ error plotted against time for the first 200 seconds.
3.2 Progressive Rendering Algorithms

![Converged Image](image1.png)

![Time vs L2 Error Graph](image2.png)

<table>
<thead>
<tr>
<th>1 second</th>
<th>10 seconds</th>
<th>1 second (closeup)</th>
<th>10 seconds (closeup)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPT</td>
<td>QPT</td>
<td>PPM</td>
<td>VPL</td>
</tr>
</tbody>
</table>

**Figure 3.4:** Fully converged image and two partially converged images captured after 1 second and 10 seconds of scene Lobby for each algorithm. Zoomed-in insets from the partially converged images shows the artifacts in detail. A convergence graph of $L^2$ error plotted against time for the first 200 seconds.
3.2 Progressive Rendering Algorithms

dorandom sequence with no stratification. One path per pixel is traced in each iteration. We use next event estimation to gather light from light sources at every path vertex. The quasi-random path tracer uses parametric quasi-Monte Carlo integration where one Sobol sequence is used for the entire image. By including both the random path tracer with no stratification and the quasi-random version which is stratified by nature, we are able to compare how stratification affects the performance of the two algorithms.

**Progressive photon mapping** Progressive photon mapping [HJW∗08] was introduced as a modification of the popular and widely used photon mapping method. It is more robust than path tracing in some difficult lighting situations but has a slower asymptotic convergence rate [KZ11]. At the initial stage, the algorithm generates smoother images than path tracing, which are affected by low-frequency noise. As the algorithm progresses, the noise becomes more fine-grained and decreases in magnitude.

Our implementation is based on stochastic progressive photon mapping where we connect photons to camera paths of one segment in length. In each iteration, we cast one camera ray per pixel, after which we trace 80,000 photons and connect them to the camera ray hit points. The photon mapping takes care of both direct and indirect illumination. The initial lookup radius was set manually; we used $\alpha = 0.7$ for the radius shrinking coefficient.

Practical implementations of classic photon mapping often compute direct illumination separately and only use the photon map for indirect illumination. In our stochastic progressive photon mapping implementation, on the other hand, we use the photon map for both direct and indirect illumination evaluation for the following reasons. First, we wanted to conform to the original progressive photon mapping pa-
3.2 Progressive Rendering Algorithms

per [HJW’08]. Second and more importantly, we wanted to keep the four algorithms as clearly separated as possible by computing both direct and indirect lighting with the same method. This has the benefit that the artifacts specific to each method are displayed clearly and do not depend on the relative contribution of direct and indirect illumination in the particular view of the scene.

**Virtual point lights** VPL rendering [Kel97] is the basis of a number of real-time global illumination solutions. The algorithm is popular because smooth images free of any noise can be obtained quickly. This comes at the price of energy losses and banding artifacts, which can be disturbing especially at the early stages of progressive computation. Krivánek et al. show that this method is not well suited to material design of highly glossy surfaces, but provides an acceptable approximation for low-gloss surfaces that we use in our study [KFB10].

Our implementation of progressive VPL rendering adds illumination from one VPL in each iteration. We use the same number of virtual lights for direct and indirect illumination. Moderate clamping of light contributions is used to suppress artifacts due to weak singularities.

### 3.2.2 Implementation Detail

All our algorithms are based on an optimized ray-tracing engine and are fully parallelized. All renderers run asynchronously from the user interface to ensure that the latter is not blocked. To simplify the algorithms' comparison, we deterministically stop all paths at length three. While this introduces bias in the solution, doing so suppresses the different impact that Russian roulette-based path termination may have on
3.3 Experiment

the convergence of the compared algorithms, and makes sure the amount of energy transfer computed by each algorithm in each render pass is identical. This ensures a fair comparison between all algorithms. Non-diffuse reflection is only computed at the first bounce from camera, while all other light bounces only consider the Lambertian component of the BRDF. This solution was chosen for compatibility with VPL rendering that does not easily support glossy reflection at the VPL [HKWB09]. Moreover, excluding glossy reflections after the first bounce and limiting the path length are common approaches to improve the renderer efficiency in production practice.

3.2.3 Other Methods

We also tested a progressive version of Lightcuts [WFA’05], where the light cut at each pixel is refined progressively. This algorithm was not included in the study because it produces visually distracting artifacts and requires a large amount of memory, making it unsuitable for practical usage. Pre-computation based approaches for interactive lighting or material design were also considered. These methods are based on moving some of the expense of rendering into a pre-computation stage. We did not include these algorithms in the study, because they do not support global illumination at all or cannot guarantee convergence to the correct image.

3.3 Experiment

3.3.1 Goal

We seek to evaluate the effectiveness of different progressive rendering algorithms for appearance design tasks, namely light and material editing. Specifically, (1) we want
3.3 Experiment

to measure how efficiently users can perform editing tasks using different progressive rendering algorithms, and (2) we want to understand how the different artifacts and convergence behavior exhibited by each algorithm affect the way users perform appearance design tasks.

3.3.2 Test Subjects

Fourteen expert subjects and twelve novice subjects participated in the experiment. All subjects had normal or corrected-to-normal vision. Subjects in the expert group perform most of their daily work using graphics design software. Most of them work in architectural or product visualization, using commercial 3D design software, such as 3ds Max and Maya, and have experience with multiple renderers, such as V-Ray and mental ray. All our experts subjects are capable of producing photorealistic images. Subjects in the novice group have limited experience with 3D design software. Most of them have never performed any appearance design before. However, the simplicity of the user interface and the design task we chose makes their performance in our experiments solely dependent on the behavior of the rendering algorithms, giving us clear measurements of algorithm qualities. We chose to include novices along with experts, because we believe they are more likely to need precise feedback from the rendering algorithms, and because our long-term goal is to make image synthesis more accessible to non-experts. Furthermore, comparing the performance and workflow of subjects of very different background gives us further insights on the generality of our results.
3.3 Experiment

3.3.3 Scene Datasets

We designed our scenes to reflect the most common cases that professional lighters encounter. All scenes are lit by one area light and tone mapped with a fixed exposure-gamma algorithm ($\gamma = 2.2$). Materials are represented as a sum of a Lambertian diffuse lobe and a Blinn-Phong specular lobe. We positioned the camera to ensure that the light source was not visible in the rendered image. The Studio (104751 triangles) and Kitchen (183997 triangles) are relatively simple scenes lit mainly by direct and indirect illumination, respectively. The Museum (745944 triangles) is a more complex scene with strong direct and indirect lighting contribution. The Lobby (628478 triangles) is our most complex scene in terms of geometric detail, shadowing, and indirect effects. We chose diffuse and moderately glossy surfaces because these are the cases most commonly encountered in professional lighters’ practice. In addition, including surfaces with more substantial gloss would handicap the VPL algorithm, which is unable to render them faithfully [KFB10].

Reducing complexity  In this experiment, our focus is on measuring how the users are affected by the progressive algorithms themselves, rather than by the usability of the user interface. For this reason, we asked subjects to perform drastically simplified appearance design tasks.

3.3.4 Trial Design

In matching trials, subjects were asked to match a given image by adjusting only one parameter corresponding to a property of the light or material. These simple tasks are representative of the basic editing operations that users perform with traditional
3.3 Experiment

user interfaces, where artists typically work by adjusting one parameter at a time, as shown in previous studies [KP09, KP10]. The matching trials allow us to quantitatively measure subjects’ performance, while providing them with a clear goal.

In open trials, we asked the subjects to make a subjective choice from a fixed set of predefined designs, each affecting many light or material properties. These operations are akin to picking the preferred option from a catalogue or by means of an image-based navigation interface. The open trials allow us to observe how subjects explore possible lighting/material configurations, which is a more natural task than matching. The user interfaces for the matching and open trials are shown in Figure 3.5.

Figure 3.6 and Figure 3.7 show the starting and target configurations for the matching trials, as well as one of the predefined designs and the task description for the open trials. In the light matching trials, the subjects were asked to move or resize the light, while in the material matching trials they were asked to change the diffuse and specular intensity, specular roughness, or the diffuse hue. In each task, only one parameter can be changed by means of a slider. In the open trials, the subject were asked to choose from eight designs that we created by randomly changing either the light parameters (position, orientation, size, intensity) or the material parameters (diffuse and specular colors, and roughness). We chose a wide-angle camera view for the lighting tasks, while for material tasks we focused the camera on the edited object. To alleviate learning effects, we randomized the various starting and target configurations when the subject moves from one algorithm to another. To provide the subjects with a tradeoff between the image quality and speed, they can pick one of the three rendering resolutions: $128 \times 128$, $256 \times 256$, and $512 \times 512$ pixels.
3.3 Experiment

Figure 3.5: User Interface for matching (top) and open (bottom) trials. Left: Target image or task instruction. Center: Progressive rendering. Right: Controls.

3.3.5 Questionnaire

After performing all trials with all algorithms, subjects are asked to complete a questionnaire where they rated each algorithm on a scale from 1 to 5 in the four categories corresponding to preference in lighting matching/material matching/open trials, and overall preference. Subjects also strictly ranked the algorithms in each of these categories. Immediately after finishing the trials using each algorithm, subjects were asked
3.3 Experiment

**Lighting Trials**

<table>
<thead>
<tr>
<th>trial 1</th>
<th>trial 2</th>
<th>trial 3</th>
<th>trial 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="target image" /></td>
<td><img src="image2" alt="starting cfg." /></td>
<td><img src="image3" alt="starting cfg." /></td>
<td><img src="image4" alt="starting cfg." /></td>
</tr>
</tbody>
</table>

**Material Trials**

<table>
<thead>
<tr>
<th>trial 1</th>
<th>trial 2</th>
<th>trial 3</th>
<th>trial 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5" alt="target image" /></td>
<td><img src="image6" alt="starting cfg." /></td>
<td><img src="image7" alt="starting cfg." /></td>
<td><img src="image8" alt="starting cfg." /></td>
</tr>
</tbody>
</table>

**Figure 3.6:** Lighting matching trials: Subjects are asked to change the light size (trials 1) or adjust its position (trials 2 to 4) to match the target image. Material matching trials: Subjects are asked to change the brightness of the couch (trial 1), the roughness of the counter (trial 2), the highlight intensity on the dinosaur skull (trial 3), or the hue of the lobby pillar (trial 4).
### 3.3 Experiment

<table>
<thead>
<tr>
<th>Lighting Trials</th>
<th>Material Trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>trial 5</td>
<td>trial 5</td>
</tr>
<tr>
<td>please select a case fits the mood of a museum</td>
<td>please pick the material case that is most pleasant to you</td>
</tr>
<tr>
<td>trial 6</td>
<td>trial 6</td>
</tr>
<tr>
<td>please pick the lighting case that is most pleasant to you</td>
<td>please pick the material case that you would want for an office</td>
</tr>
</tbody>
</table>

**Figure 3.7:** Lighting and material open trials: Subjects are asked to choose a pre-defined lighting or material design that fits a text description (trials 5 and 6).

...to leave free-form comments on their workflow and rate the subjective quality of the image they created. For subjects whose native language is not English, we translated the questionnaire in their language to allow them to faithfully express their opinion. See Appendix A for the questionnaire in English.

#### 3.3.6 Trial Procedure

The study consisted of four sessions, one for each algorithm (described in detail in Section 3.2). We randomized the order of algorithms given to each subject. In each session, the subject performed the following trials in order: 4 light matching trials, 2 light open trials, 4 material matching trials, and 2 material open trials. Before the study, we trained each subject individually to allow for questions and to accommodate each subject's learning needs. The instructor verified that the subject understood the
3.4 Analysis

We conducted the study in two separate labs. The subjects performed all their trials in a controlled lighting environment with negligible ambient lighting, to simulate typical working conditions of artists and maximize visibility of the screen. The study was run with screens at resolution of 1600 × 900, at approximately 1 foot from the subject. All rendered images were upsampled to 512 × 512 pixels on screen with a box filter, covering an area of roughly 4 × 4 = 16 square inches. We used an iMac with a 4-core 2.93 GHz Intel Core i7 processor and 16 GB RAM as our reference machine and synchronized the framerate of other machines to match its performance.

3.4 Analysis

We present our results in two parts. First, we analyze the output of the rendering system as the subjects proceed through each trial. Second, we compile the subjects’ feedback from the questionnaires. Unless stated otherwise, tests for statistical significance are computed with repeated measures analysis of variance (ANOVA). The ANOVA for ranking data is conducted using the Kruskal-Wallis test, which is a nonparametric alternative that does not rely on an assumption of normality. A $p$ value below 0.1 indicates a 90% confidence that the two population means differ given the measure of the sample. In all figures, error bars represent standard deviation.
3.4 Analysis

3.4.1 Time to Completion

Given enough time, all the algorithms eventually converge. The true difference between them is in the early stages of computation, so time to completion is an important measure of an algorithm’s performance. In particular, the algorithm that provides the most useful feedback for appearance design should show shorter time to completion.

Figure 3.8 shows the average time to completion for each type of trial for each subject group. In light matching trials, all subjects were able to finish sooner using one of the path tracers than using the virtual point light algorithm ($p < 0.04$). In addition, expert subjects also finished tasks sooner using the progressive photon mapping than using the virtual point light algorithms ($p < 0.06$), but such a trend is not shown in the novice group. This indicates that regardless of subjects’ experience, path tracers can provide more useful feedback, while progressive photon mapping is useful but limited to the experts. In general, subjects try to match shadows/highlight first, and only if these were not sufficiently salient, they consider other lighting features. In questionnaires, subjects left comments such as “for lighting look at the shadow, for material look at the color and brightness”, and “if there is shadow in the figure, it is much easier than without shadow”.

For material matching trials, no clear trend is present. All subjects finished in roughly the same time. We attribute this to the fact that in light matching trials subjects needed an overview of the lighting of the entire scene to make a decision. In contrast, in material editing tasks, subjects had a limited view to the scene and a limited feedback sufficed to perform the task, making the difference between algorithms less important. Comments in the questionnaires confirm this: “Tuning material is easy, the rendering of lighting is sort of slow”. 

45
3.4 Analysis

In the open trials, subjects have lower average completion time than in the matching trials with all algorithms ($p < 0.01$). However, the meaning of time to completion is less well-defined since it depends on the standard of judgment the subject has chosen. While novice subjects show no significant trend between algorithms, expert users spent much more time in both light and material open trials with VPLs than with the other three algorithms ($p < 0.05$).

In general experts subjects finished sooner than novice subjects when using the random path tracer in light matching trials ($p < 0.08$) and material matching trials ($p < 0.06$). A similar trend is also observed for quasi-random path tracing, but with less certainty.

![Figure 3.8](image-url)  

**Figure 3.8:** Average time to completion. Subjects generally finished sooner using one of the path tracers than using the VPL algorithm.
3.4 Analysis

3.4.2 Scene Complexity

We found that in lighting design, scene complexity has a large impact on the user performance. The main reason is that complicated light paths affect the different algorithms in a different manner. For example, the lobby scene in lighting trial 4 has a higher matching error for all algorithms (see Figure 3.4). In this case, progressive photon mapping performs much better but still shows artifacts that hinder users' ability to perform design tasks. However, in material design tasks, we found that the performance is independent of the geometric complexity, since subjects focus mostly on a small part of the environment.

3.4.3 Subjective Image Quality

At the end of each trial, we asked the subjects to rate their work on the scale from 1 to 5, where 1 means the worst and 5 means the best. Matching trials were rated in terms of how closely the subjects are able to match the reference image. Open trials were rated in terms of how satisfied they are with their choice. Figure 3.9 shows average rating for each kind of trial and for each user group. In the light matching trials, all subjects on average rated their work better when using the two path tracers ($p < 0.08$ for novices and $p < 0.01$ for experts\(^1\)) suggesting that they perceived themselves doing a better job with these algorithms. However, for other three types of trials, no obvious trend is shown in the novice subject group. We attribute this to the fact that, unlike light matching, material matching and open trials are too subtle for novices to properly rate their work.

\(^1\)This is the upper bound of the $p$ value for the ANOVA of (RPT vs. PPM), (RPT vs. VPL), (QPT vs. PPM) and (QPT vs. VPL).
3.4 Analysis

We also observe that expert subjects have substantially lower rating for tasks done with the virtual point light algorithm in light open trials compared to the other three algorithms ($p < 0.07$), meaning that expert subjects are unsatisfied with the feedback provided by the VPL algorithm when they need to get a general sense of the entire scene instead of matching specific lighting features. This is also confirmed in the questionnaire “Overall, it [VPL] is unpleasant to look at. I don’t see any advantage.” “It [VPL] made me wait longer. Previews kept changing.”

![Subjective Image Rating](image)

**Figure 3.9:** Subjective Image Rating. Experts are less satisfied with their results, indicating that they are more precise in appreciating appearance differences.

3.4.4 Workflow in Matching Trials

In matching trials, most subjects employed a simple search and refine approach. Subjects would first click different random positions on the slider. Once they found a rough value, they began to perform smaller adjustments to find the target. If they received
3.4 Analysis

fast feedback, they tended to click more. Some of the users would repeatedly click
the slider bar to simulate a dragging effect. This is confirmed by looking at the indi-
vidual videos for each subject. Moreover, the behavior of novice subjects and expert
subjects do not differ. Figure 3.10 shows that subjects have shorter interaction in-
terval when using the two path tracers compared to progressive photon mapping and
VPL ($p < 0.05^2$), indicating that path tracers provide useful feedback at the early stage,
which allows subjects to make decisions faster. By observing the user interactions, we
found that the workflow is similar in the two groups.

3.4.5 Workflow in Open Trials

In the open trials, subjects explored the pre-defined design options. At first, they
browsed through all the options and waited a short time to get a sense of how each
design looks like. After this first round, users would go back to a few of the designs
that interested them and made a final decision by going back and forth amongst them.
Less time was spent in each configuration than in the matching trials ($p < 0.02$) (see
Figure 3.10). This indicates that when the task is more subjective and only requires
a high-level decision, progressive algorithms are able to provide a very quick preview
that users find helpful in this context. Trends similar to matching trials are observed,
two path tracers have shorter interaction time in both light open trials ($p < 0.09$) and
material open trials ($p < 0.07$). Both groups performed their tasks using a similar
workflow.

---

2This is the upper bound of the $p$ value for the ANOVA of (RPT vs. PPM), (RPT vs. VPL), (QPT vs. PPM) and (QPT vs. VPL).
3.4 Analysis

Figure 3.10: Average time between user interactions in seconds. For matching tasks, it is the average interval between two clicks on the slider. For open tasks, it is the average interval between changing the designs. Experts take slightly less time than novices.

3.4.6 Resolution Switching

Subjects were allowed to change image resolution during each trial. Most of them started at the lowest resolution and later switched to higher resolution for finer tweaking. If they felt the result needed more than a small adjustment to match the target, they switched back to a lower resolution. One comment says: “I have been using low resolution to get an approximation and then high resolution for more precision”. Another says: “The more I approached the target, the more important the render quality was so that I could compare the image details.”

We notice that the subjects spent more time working in low and medium than in high resolution (see Figure 3.11). Most subjects only switched to high resolution at the end to validate their work; a few subjects never used the high resolution mode in
3.4 Analysis

matching tasks. This trend is even more obvious in the open trials: “[In open trials,] I switched to low resolution and was looking mainly at the colors”. We conclude that most users do not need a fully converged, high-quality image to make their choice and having a faster low-resolution mode is useful for all progressive algorithms.

![Resolution Usage Chart]

**Figure 3.11:** Proportion of time users spent at each resolution. Experts spend less time at high resolution.

In general, compared to novices, experts were more likely to stay in the low and middle resolutions, suggesting that they were comfortable making more decisions based on lower quality imagery, probably due to their familiarity. Another interesting observation is that expert subjects stayed in high resolution longer when using photon mapping compared to other three algorithms. A few users even stated in the questionnaire that they need to use higher resolution with progressive photon mapping. “I had to increase the quality.”, “I nearly did not use the low resolution.”, “I immediately switched to MID or HIGH (when using PPM).”
3.4 Analysis

3.4.7 Algorithm Ratings and Rankings

Subjects rated each algorithm in all four types of trials. Figure 3.12 summarizes average ratings. In the overall rating, novice subjects rated the two path tracers higher than the other two methods ($p < 0.05$). However, when observing the individual trials, this preference is not significant, except in the light matching trials ($p < 0.01$).

![Algorithm ratings](image)

**Figure 3.12:** Algorithm ratings. Experts have a stronger preference toward path tracing algorithms.

On the other hand, the expert subjects show a clear preference toward the two path tracers in the ratings statistics for all trial types ($p < 0.03$). This was mirrored by the comments in the questionnaires. For example, regarding progressive photon mapping, some subjects wrote: “it was hard to get an accurate match because of the artifacts even at high resolution”, and “photon mapping generates too large splotches and it is hard to find the key areas for comparing the images (edge, shadow, highlight)”, and
3.4 Analysis

finally “[I] have to wait to see small details in both light and material”. Regarding VPL method they comment: “things change very slowly and may look very different before it converges”, “it was easy to settle on a rough approximation, but I was never confident of the stability of the final choice”, “I was annoyed by the blinking from VPLs”.

There is no systematic difference between the two path tracers in terms of ratings. Subjects’ comments confirm this, for example “random path tracing and quasi-random path tracing were quite similar except quasi-random was slightly clearer”, “RPT+QPT - I did not notice any difference. I found them relatively fast and accurate”, and “Differences between QPT a RPT are not noticeable. They can provide an overall preview very swiftly.”.

Subjects were also asked to rank each algorithm in all four types of trials. While rating can have ties, rankings are a forced choice. Figure 3.13 shows the stacked frequencies of rankings. In the novice group, the two path tracers were ranked higher than the other two methods in the overall ranking and in the two lighting trials. In the material trials, progressive photon mapping received the lowest ranking among all algorithms ($p < 0.06$), but no systematic difference is shown between the two path tracers and VPL method.

On the other hand, in the expert group, subjects consistently ranked the two path tracers higher than the other two algorithms in overall ranking statistics ($p < 0.03$), but there is no systematic difference between the two path tracers in terms of ranking. The strong preference for path tracing is one of the main results of our work.
3.5 Discussion and Future Work

In this section, we summarize the major findings of our study. Before continuing, we want to remind the reader that strictly speaking our observations only apply within the boundary of the tested cases, just like all user studies. At the same time, given that the observed trends were consistent for the four different scenes included in our study, we believe that they are general enough to apply to many other scenes as well.

**Progressive Renderers.** Our most prominent result is the poor performance of progressive photon mapping and virtual point light rendering compared to the two path tracers. We found that while subjects are able to perform simple appearance design tasks well using all algorithms, their performance is better with path tracers. This trend can be observed objectively in the time to completion. Moreover, the path tracers

---

**Figure 3.13:** Algorithm rankings ($R = \text{RPT}$, $Q = \text{QPT}$, $P = \text{PPM}$, $V = \text{VPL}$). Two path tracers were ranked higher than the other two methods in the overall ranking and in the two lighting trials.
3.5 Discussion and Future Work

received higher rating and ranking, indicating that the users actually prefer working on appearance design using these methods.

In our tests, the random and quasi-random path tracers had similar performance and subjects did not consistently prefer one over the other. This result is surprising especially because our random path tracer did not use any stratification while the quasi-random path tracer is stratified by nature. Similarly, we did not observe any systematic preference of progressive photon mapping over virtual point light rendering or vice versa. This suggests that the high-frequency errors exhibited by the path tracers are easier for subjects to cope with than the low-frequency errors or banding shown by the other methods. This trend is more obvious in the expert group than in the novice one.

**Experts vs. Novices.** In general, expert subjects are more efficient than novices. This is confirmed by the statistics of average time between interactions and time to completion, and also by watching the captured videos. Expert subjects appreciate appearance differences better than novice subjects. The differences between the algorithms’ artifacts substantially influence expert subjects’ decision in the final rating and ranking. Moreover, the statistics of the expert group usually have a clearer trend with lower variance, meaning that in general, expert subjects behave more consistently. The statistics of the novice group tend to have higher variance, showing that novices are less predictable. Unlike the expert subjects, the algorithm ratings of novice subjects do not reflect a clear preference. But when it comes to ranking which subjects are forced to choose, novice subjects made decisions similar to expert subjects.

**Common Workflow.** Our subjects exhibit common workflow patterns. In matching tasks, they generally employ a search and refine approach, first finding a rough position
3.5 Discussion and Future Work

around the target and then refining it by making small changes. This workflow is independent of the progressive renderer used as well as the subject experience level. A similar trend was found for open trials, but with quicker user decisions. Furthermore, subjects are willing to sacrifice image resolution for faster feedback in the initial search, while they switch to higher resolution as they refine.

**Initial User Feedback.** The search-and-refine workflow together with the resolution switching behavior suggest that in appearance editing subjects favor algorithms that provide immediate feedback on the overall scene look while refining the details later.

**Limitations.** As in all user studies, the main limitation of our work is the scope of our investigation, in terms of the algorithms and of the lighting and material editing tasks we have explored. Moreover, we have only explored a fraction of the possible lighting models, material models, and scene settings. In material trials, we chose what we believe are the most common material design tasks, but we acknowledge that the results may not hold for very different tasks such as spatially varying material design or texture selection.

**Future Work.** These limitations suggest clear directions in expanding the scope of our work in the future. At the same time, we feel that the observed trends are general and likely to be confirmed by further studies, especially considering that similar trends are observed in different subjects’ groups. We believe that a more fruitful avenue for further exploration is the development and testing of appearance design user interfaces that work in conjunction with progressive renderers, rather than the current interfaces that fundamentally assume the renderer has perfect image quality. Kerr and Pellacini
showed that the choice of user interface has a significant impact on user performance when coupled with a real-time renderer [KP09, KP10]. The question we are interested in is how to design effective interfaces that can help users in design tasks when the feedback is given by, for example, a progressive path tracer.

### 3.6 Conclusions

This chapter presents a first step toward the evaluation of progressive rendering algorithms in the context of appearance design. By performing a series of matching and open trials and by collecting subject evaluation in questionnaires, we have measured how different progressive rendering algorithms aid both novice and expert subjects in performing specific lighting and material design tasks. In comparing path tracing with progressive photon mapping and virtual point light (VPL) rendering, we found the former to perform better in terms of objective and subjective measures. This trend was common in both subject groups, further strengthening the results. The main differences between subject groups were that experts were faster and more precise overall.

While, as in any user study, we acknowledge that our measurements are only strictly valid within the context of our experiment, we believe that the main trends found in our study generalize to other scenes and appearance design tasks. In addition, we expect that our experiment design will be used as the basis for further explorations of the effectiveness of progressive rendering algorithms. In the future, we are interested in extending our study to include more sophisticated appearance tasks and different user interfaces. More importantly though, we are interested in investigating how to design user interfaces that work in conjunction with progressive renderers.
Chapter 4

LightSlice: Matrix Slice Sampling for the Many-Light Problem

In Section 2.4.3, we gave a brief introduction about VPL algorithms which are a group of algorithms that approximate global illumination using a set of point light sources generated by sampling light subpaths. The user study in Chapter 3 shows that progressive VPL algorithms are not preferred for appearance editing tasks, because its artifacts hinder the judgement of the users. However, VPL algorithms are widely used in movie industry and game industry for final rendering. One reason for this popularity is that VPL algorithms can have extremely efficient implementations using rasterization pipelines. Another reason is that VPL algorithms can produce smooth images without disturbing high-frequency noise or low-frequency blurriness. Although VPL algorithms may introduce bias, it is usually less of a concern for movie and game industries, since accuracy is not a high-priority goal for them.

In practice, a large number of point light sources are usually needed to approximate the complex lighting effect of global illumination in high quality rendering. The final
4.1 Overview

gathering step that connects the VPLs to camera paths is known as many-light problem. Due to the large number of VPLs, computing all the VPLs’ contribution is not feasible. In this chapter, we present LightSlice, an algorithm that efficiently solves the many-light problem for large environments with complex lighting. As in prior work, we derive our algorithm based a matrix formulation of the many-light problem, where the contribution of each VPL corresponds to a column. The final image can be computed by summing of all matrix columns. Hašan et al. proved that this matrix is usually a low-rank matrix, which can be approximated using only a few columns [HPB07]. We make a further observation that if we cluster similar surface samples together, the slice of the matrix corresponding to these surface samples would have significantly lower rank than the original matrix. We exploit this observation to derive a two-step algorithm based on matrix slice sampling.

The rest of this chapter is organized as follows: we start in Section 4.1 with an overview of many-light problem, its matrix formulation, matrix structure, and some of our observations, followed by Section 4.2 which compares our method to other many-light algorithms. In Section 4.3, we present an overview of LightSlice, followed by an extensive mathematical description and implementation details. In Section 4.4, we present some results and compare LightSlice with two other state-of-the-art VPL algorithms. Section 4.5 discusses some limitations of LightSlice and future works. Finally, Section 4.6 summarizes our conclusions.

4.1 Overview
4.1 Overview

4.1.1 Realistic Rendering as Many-Lights

Fast computation of global illumination in large scenes with a complex lighting configuration is still a challenging problem in computer graphics. Many methods have been proposed to compute fast global illumination solutions, e.g., bidirectional path tracing and photon mapping (see Chapter 2). In this chapter, we focus on computing images using a variant of VPL algorithms, where direct and indirect illumination are approximated by converting the original light sources into a large number of virtual point lights (VPLs) distributed across the entire scene. In this model, computing a global illumination solution is equivalent to computing an image lit solely by a large number of point light sources, i.e., the many-light problem. Prior work in offline, high-fidelity rendering [HPB07, WFA*05] has shown that for scenes with diffuse and low gloss materials, hundreds or thousands of VPLs effectively approximate complex direct and indirect illumination effects, while having the advantage of treating both equally within the same algorithm framework. VPLs have also been used in real-time applications, where they handle a smaller number of lights at the price of the accuracy of approximation [REG*09]. VPLs have also found much use in feature film production [Chr08]. In this chapter, we focus on high-fidelity rendering in complex environments rather than interactive applications.

4.1.2 Matrix Interpretation of Many-Lights

It is useful to consider an alternative interpretation of the many-light problem as a matrix sampling problem. Let us arrange all pixels of an image as a long column vector. We can then arrange all columns corresponding to each VPL as a large unknown matrix.
4.1 Overview

The final image can be computed by summing each row in the matrix. Figure 4.1 shows an example of such matrix. While many-light algorithms handle all lights equally, it is useful to note that the lights have two common behaviors. Ignoring shadows, some lights have strong contributions to all pixels; these VPLs typically correspond to direct illumination, e.g., the sun. We term these *global lights*. When unshadowed, these appear as bright matrix columns. When shadowed, these appear as columns with bright and black sections or an entirely black column. Other lights have more local behavior affecting only a few pixels; typically, these are VPLs derived by sampling indirect lighting, and thus have lower intensity and an $r$-squared falloff. We term these *local lights*. In the matrix, these lights appear mostly as black columns with a small, low intensity section.

For hundreds of thousands of lights, a brute force solution that computes all columns of the many-light problem is prohibitively expensive. Many methods have been proposed to reduce the computation complexity of the many-light problem to be sub-linear in the number of VPLs. The two main observations that allow this is that the elements of the matrix have repeating patterns and that large areas of low contributions are present (see Figure 4.1). All scalable many-light algorithms exploit these two observations by clustering groups of similar VPLs and approximating their contribution using a single representative. In other words, all these methods subsample the matrix by approximating blocks of similar elements as constant values computed from only one element. The size and shape of these blocks changes for different algorithms. We compare our work with two main prior algorithms.
4.1 Overview

Figure 4.1: The light transport matrices of two complex scenes (subsampled from the original). Note the existence of repeating patterns and large areas of near black in the matrices.

4.1.3 Exploiting the Matrix Structure

Lightcuts [WFA’05] hierarchically clusters the lights into a light tree using geometric proximity as the cluster metric. It then renders the final image by choosing a set of representative clusters differently for each pixel. Matrix Row-Column Sampling (mrcs in short) [HPB07] clusters entire matrix columns together and renders one representative column for the entire clusters. This is motivated by the observation that the transport matrix is close to low rank. As we will discuss later in details, for large environments and complex lighting, neither lightcuts nor mrcs optimally exploits the structure found in these matrices as shown in Figure 4.1. The former works well for local lighting and mostly-visible global lights, but oversamples shadowed global lights (corresponding to bright columns with large black segments or entirely black). The latter works well for global lights, quickly determining the global visibility behavior, but is inefficient for local lighting (corresponding to low intensity columns that are mostly black) in that it samples them for all pixels. We are interested in deriving a matrix sampling algorithm that has the benefits of both approaches.
4.1 Overview

The main observation of our work is that, if we cluster similar pixels together, the slice of the matrix corresponding to these pixels has significantly lower rank than the original matrix. Intuitively this is true since for each slice, local lighting and shadowed regions can all be approximated together with a low intensity representative. This observation was already exploited in the domain of precomputed radiance transfer to either compute the transport matrix [HR10, MSRB07] or for compression [SHHS03]. However each of these approaches are significantly different from ours since they aim to approximate the whole matrix, while we only need to compute the sum of matrix columns.

4.1.4 LightSlice

In this chapter, we present LightSlice, an algorithm that efficiently solves the many-light problems by sampling matrix slices. In LightSlice, we first determine matrix slices by clustering similar image pixels based on their geometric proximity. For each of these slices we render a representative and roughly cluster all columns based on all row values. This initial clustering effectively captures the global structure of the matrix. For each slice, we then refine such global clusters into per-slice clusters based on representative rows of the given slice and its neighboring slices. This effectively captures the local structure of the matrix, including its shadowing behavior. We render each slice by choosing representative columns and only rendering the column elements corresponding to the slice rows. LightSlice combines the advantages of both lightcuts and mrcs by effectively capturing the global structure of the matrix, including its shadowing, while adapting to the local changes for each slice.

We tested our algorithm on a variety of complex scenes with indoor and outdoor
4.2 Many-Light Algorithms

illumination. \textit{LightSlice} is consistently faster than other algorithms, with between three and six times speedup. More importantly, each of these prior algorithms works well for some scenes, but becomes inefficient for others. This is due to the fact that each of them is optimal for some matrix structure but inefficient for others. \textit{LightSlice} is instead consistently efficient in all our scenes since it can adapt to the typical matrix structures found in complex lighting scenarios.

4.2 Many-Light Algorithms

We provided a brief introduction of VPL algorithms in Section 2.4.3. In this section we quickly review recent work that uses VPL as the main rendering primitive.

4.2.1 Lightcuts

\textit{Lightcuts} hierarchically clusters the lights into a light tree using geometric proximity as the cluster metric [WFA’05]. Each tree node corresponds to a light cluster where a single representative VPL is selected to approximate the cluster contribution. \textit{Lightcuts} renders the final image by choosing a set of representative clusters differently for each pixel and computing their contribution by raytracing. The choice of representative is done by hierarchically exploring the light tree, where at each node the algorithm descends into its children if the current representative is deemed to be a poor approximation of the cluster for the pixel being rendered. The estimation is made by evaluating a conservative error metric based on properties of light, pixel geometry and materials, but ignoring visibility. In a matrix interpretation, \textit{lightcuts} can be thought of as sampling each row of the matrix independently, but with a precomputed ordering in light
4.2 Many-Light Algorithms

tree construction. Typically, the columns corresponding to the highest level clusters are rendered mostly in full, while sparse elements are selected independently for each row thereafter to efficiently adapt to local lighting, for which lightcuts works very well. For global lighting, however, lightcuts is not as efficient because it ignores shadowing when estimating light contributions. This means that if large shadows are present, most pixels will sample that amount of matrix elements with zero contribution.

Multidimensional Lightcuts extends lightcuts by introducing a gather tree to reduce the cost of antialiasing [WABG06]. However, because the error metric of Multidimensional Lightcuts is designed for the use of evaluating one single integral, it cannot be easily extended to evaluate multiple integrals (e.g., bound the error of a group of gather points from different pixels) simultaneously. Thus the overhead of walking the light tree cannot be amortized across pixels.

4.2.2 Matrix Row-Column Sampling

In Matrix Row-Column Sampling (mrcs), Hašan et al. made an observation that the transport matrix is close to low rank, since the image corresponding to nearby lights is similar, so are their columns [HPB07]. To compute a final image, mrcs computes light clusters and renders one representative per cluster. The same representative is used for all pixels. To determine the optimal clustering, mrcs samples a small set of matrix rows in full to form a reduced matrix. The columns of this matrix are then clustered in such a way that similar columns that have low contribution are grouped together. mrcs works very well for global lighting since it uses subsampled rows (including shadowing) to choose the matrix columns that have the highest contribution to the final image. The cost of carefully choosing columns is amortized over all pixels in
4.2 Many-Light Algorithms

the image, making the algorithm effective. At the same time, since mrcs reconstructs
the final image by rendering entire columns, it is not efficient for local lights whose
corresponding columns have most matrix elements with zero contribution. Note that
while the original implementation of mrcs computed rows and columns on the GPU
using shadow mapping, we found that in large scenes shadow maps exhibit too many
biasing and sampling artifacts to produce high quality images. We will compare our
algorithm to an implementation of mrcs executed on a raytracer that does not suffer
from these artifacts. Hašan et al. further extended this methods to animation by using
reprojection and tensor clustering [HVAPB08].

4.2.3 Interactive Many-Lights Rendering

Kelly introduced instant radiosity, a method that uses a relatively small number of
VPLs to simulate the global illumination in a scene [Kel97]. The main advantage of
VPLs is that their contribution can be gathered efficiently by using shadow mapping
on the GPU. Today, VPL rendering is still the basic building block of many interactive
global illumination algorithms. Imperfect shadow maps used low quality shadow maps
to approximate visibility when gathering VPL contributions [RGK’08]. Ritschel et al.
proposed using small rasterization buffers warped by BRDF importance to accurately
account for the glossy reflection in the final bounce [REG’09]. VPL approaches are
also used in conjunction with precomputed radiance transport for interactive relighting.
Hašan et al. presented an interactive GPU-based system for cinematic relighting
with multiple-bounce indirect illumination from a fixed view-point [HPB06]. Cheslack-
Postava et al. used precomputed visibility cuts to perform interactive lighting and ma-
terial design [CPWAP08]. Our method is in different scope from all these approaches
4.3 *LightSlice* Algorithm

since it is designed to handle significantly larger number of VPLs without precomputation.

### 4.2.4 Improving VPL Rendering Quality

A recent extension to VPL rendering methods used virtual sphere lights [HKWB09] to reduce the loss of energy due to clamping. Křivánek et al. showed that VPL methods have trouble in accurately representing the look of highly-glossy materials [KFB10]. To handle these cases, Davidovič et al. combined row-column sampling with selective raycasting to handle glossy BRDFs [DKH*10]. We refer to our introduction for a detailed discussion of *lightcuts* and *mrcs* as they are compared to our method.

### 4.2.5 Precomputed Radiance Transfer

Precompute radiance transfer algorithms also utilize the fact that the light transfer matrix is locally low rank. The CPCA method reduces the high-dimensional transfer signal to low-dimensional by partitioning many samples into fewer clusters [SHHS03]. Mahajan et al. gave a theoretical analysis of the local low-rankness of the light transport matrix [MSRB07]. Huang and Ramamoorthi utilized the locally low-rankness to sparsely sample the spatial domain [HR10]. Although this analysis inspired our work, it is not directly applicable to our problem since PRT methods sample and reconstruct the entire matrix, while we only seek to compute the sum of matrix columns.

### 4.3 *LightSlice* Algorithm
4.3 *LightSlice* Algorithm

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Dimensionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>Number of pixels</td>
<td>scalar</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of VPLs</td>
<td>scalar</td>
</tr>
<tr>
<td>$r$</td>
<td>Number of slices</td>
<td>scalar</td>
</tr>
<tr>
<td>$A$</td>
<td>Full lighting matrix</td>
<td>$m \times n$</td>
</tr>
<tr>
<td>$R$</td>
<td>Global reduced matrix</td>
<td>$r \times n$</td>
</tr>
<tr>
<td>$A_i$</td>
<td>Column of $A$</td>
<td>$m \times 1$</td>
</tr>
<tr>
<td>$R_i$</td>
<td>Column of $R$</td>
<td>$r \times 1$</td>
</tr>
<tr>
<td>$l_i$</td>
<td>Number of pixels in slice $S^i$</td>
<td>scalar</td>
</tr>
<tr>
<td>$S^i$</td>
<td>A slice of matrix $A$</td>
<td>$l_i \times n$</td>
</tr>
<tr>
<td>$S^j_i$</td>
<td>Column of $S^i$</td>
<td>$l_i \times 1$</td>
</tr>
<tr>
<td>$k$</td>
<td>Number of k-nearest-neighbour</td>
<td>scalar</td>
</tr>
<tr>
<td>$L^i$</td>
<td>Local reduced matrix for slice $S^i$</td>
<td>$k \times n$</td>
</tr>
<tr>
<td>$C^R_i$</td>
<td>Column (light) clusters of $A$ estimated from $R$</td>
<td>set of VPLs</td>
</tr>
<tr>
<td>$C^L_i$</td>
<td>Column (light) clusters of $S^i$ derived from $L^i$</td>
<td>set of VPLs</td>
</tr>
<tr>
<td>$c$</td>
<td>Number of $A$'s clusters</td>
<td>scalar</td>
</tr>
<tr>
<td>$c^i$</td>
<td>Number of $S^i$'s clusters</td>
<td>scalar</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of the notation used in this chapter

4.3.1 Algorithm Overview

We formulate the many-light problem as a matrix sampling problem, where matrix $A$ is the transport matrix of size $m \times n$, where $m$ is the number of surface samples and $n$ is the number of virtual point lights collected in the scene. Each element $A(i, j)$ of matrix $A$ is the contribution of virtual light $j$ to surface sample $i$, and the final rendering of surface sample $i$ is the sum of the contributions from all VPLs: $I(i) = \sum_j A(i, j)$. In our implementation, we generate VPLs for direct illumination by randomly sampling area lights and environment maps using stratified sampling, and VPLs for indirect illumination by particle tracing as in [Kel97]. For our test scenes, we use 150-300K VPLs taking roughly 3 to 5 seconds to generate.

With this high number of VPLs, the cost of computing matrix $A$ with a brute force
4.3 *LightSlice* Algorithm

algorithm is prohibitively high. *LightSlice* is an algorithm that provides efficient and accurate approximation of the many-light problem by exploring and exploiting the structure of the light transport matrix. Figure 4.2 shows the steps of our algorithm:

- **Matrix Slicing.** We first partition the surface samples based on their geometric proximity. This is equivalent to slice the rows of the transport matrix in such a way that each slice contains the contribution of close-by surface samples, thus the lighting effects are likely similar within the slice. For each slice, we seek to optimally choose the important lights for each slice.

- **Slice Sampling.** Next, we pick a representative point per slice and compute the corresponding row of the matrix $A$. These sampled rows form a reduced transport matrix $R$, which is a submatrix of $A$. As shown in [HPB07], $R$ contains enough information to capture the global structure of $A$ while being significantly smaller in size. Intuitively, one can interpret $R$ as a stratified sampling of $A$.

- **Initial Light Clustering.** Since the structure of global lighting effects is captured well by $R$, we perform a rough initial clustering of the columns of $R$, i.e., the scene’s lights, just as in *mrcs*. This provides a good initial estimate of light clusters, which can be further refined on a per-slice basis to better capture local effects. Overall, this initial clustering significantly reduces the cost of determining per-slice light clusters.

- **Per-slice Cluster Refinement.** Starting from the initial clustering, our algorithm further refines the light clusters for each slice independently, by splitting global clusters into local ones in order to efficiently capture local lighting effects. In matrix form, this can be interpreted as splitting global clusters of high rank, while leaving the low rank ones unsplit. Since the rank of local lighting effects varies
4.3 **LightSlice Algorithm**

strongly per slice, this generates different light clusters for slice, each of which captures well the rank of the sub-matrix.

- **Per-slice Reconstruction.** For each slice, we reconstruct the final image by rendering the slice elements of one representative column for each light cluster.

![Diagram](image)

**Figure 4.2:** Algorithm overview: starting from an unknown light transport matrix, first we determine matrix slices by clustering surface samples based on the geometric proximity. For each of these slices, a representative sample point is chosen and the corresponding row of $A$ is computed. These sampled rows form a reduced matrix $R$ on which an initial light clustering is performed to capture the global structure of $A$. For each slice, we then refine the initial light clusters based on the neighboring slices to effectively capture local lighting effects. Finally, we render each slice by choosing representative columns (lights).

### 4.3.2 Matrix Slicing

We cluster surface samples using a top-down KD-partition scheme proposed in [WABG06]. First, we map the surface samples onto a 6D space, where the 6D coordinates encompass both position and normal at each point. We then iteratively split the 6D bounding box along the longest axis until either the bounding box or the number of samples is small enough. We will also stop refining adaptively by checking if less than 10000 gather points are contained in a leaf. Each cluster $S^i$ corresponds to a slice of matrix $A$:
4.3 *LightSlice* Algorithm

\[
A = \begin{bmatrix}
S^1 \\
S^2 \\
\vdots \\
S^r
\end{bmatrix}
\]

### 4.3.3 Slice Sampling

We sample each slice by randomly picking one representative point and compute the full corresponding row using raytracing. We assemble the sampled rows for all slices in a reduced matrix \( R \) that represents a stratified sampling of the full matrix \( A \).

### 4.3.4 Initial Light Clustering

As pointed out by previous work, since \( A \) is close to low rank, the structure of \( A \) that is shared by all slices, typically originating from global lights, is captured well by \( R \). For example, bright lights visible within the entire scene (e.g., bright, long columns in Figure 4.1) correspond to columns of \( R \) with dense, large numbers. Therefore, we perform a rough clustering on the columns of \( R \) as initial light clusters for all slices. This greatly reduces computation time needed to compute light clusters for each slice. In our implementation, we adapted the cluster metric proposed in [HPB07] to compute clustering. More specifically, we cluster columns of \( R \) by minimizing the summed cost of its \( c \) clusters \( C^R_k \):

\[
\sum_{k=1}^{c} \text{cost}(C^R_k) = \sum_{k=1}^{c} \sum_{p,q \in C^R_k} d_R(p,q) \tag{4.1}
\]

\[
d_R(p,q) = ||R_p|| \cdot ||R_q|| \cdot ||\tilde{R}_p - \tilde{R}_q||^2 \tag{4.2}
\]
where \( d_R(p,q) \) is the distance between two column \( p \) and \( q \) in the reduced matrix \( R \), \( ||x|| \) denotes the norm of vector \( x \) and \( \tilde{x} \) is the normalized vector \( \tilde{x} = x/||x|| \). Intuitively, the objective of this cost function is to partition the columns into clusters such that the similar, low-intensity columns are grouped together. We implement the clustering using the multi-set sampling algorithm described in [HPB07]. In our implementation, the number of initial clusters is set to roughly 30% of the total cluster budget.

### 4.3.5 Per-slice Cluster Refinement

The initial light clustering roughly captures the global structure of the matrix, but cannot adapt efficiently to the local matrix structure. We further refine this clustering for each slice to compute per-slice light clusters that adapt well to local lighting. For each slice, we assemble a local matrix \( L^i \), a submatrix of \( R \), by combining the representative row for the slice \( S^i \) with the representative rows for spatially close slices that are likely to exhibit similar local structure. We find these spatial neighbors by performing a nearest neighbor search in the 6D-space kd-tree used for matrix slicing. We do this to ensure that spatially close slices have similar local clustering, thus avoiding image-space discontinuity artifacts commonly found when rendering image blocks separately. We then cluster the columns of the local matrix \( L^i \) to determine the light clusters for the slice \( S^i \).

We determine light clusters for each slice by iteratively splitting the highest cost cluster until a maximum number of clusters, corresponding to our reconstruction budget, is reached. We initialize the procedure by assigning each slice column to the same clusters found in the global clustering. In other words, for each slice \( S^i \), we initialize
the clusters as $C^L_k = C^R_k$ for the rows in $L^i$. At each iteration, we evaluate the clustering cost function in equation 4.2 with the elements of $L$, i.e., $\sum_{k=1}^{C_i} \sum_{p,q \in C^L_k} d_L(p,q)$. We then split the cluster with the largest cost. We can do so efficiently by maintaining a priority queue. To split the cluster, we first randomly pick two columns with probability proportional to their norm $||L_p||$ and compute the corresponding line in $r$-dimensional space. We then project all other columns onto this line and find the best position to cut the line into two. After this procedure, we have clustered the columns of $L^i$ in $c_i$ as clusters $C^L_i$.

### 4.3.6 Per-slice Reconstruction

Similarly to [HPB07], we render each slice $S^i$ by summing the contribution of each of its clusters $C^L_k$. We estimate the contribution of each cluster $C^L_k$ by rendering the elements of one representative column $j_k$ belonging to the slice $C^L_k$ using raytracing. We choose the representative column randomly with probability proportional to its global norm $||R_j||$ and estimate its total weight as $(\sum_{p \in C^L_k} ||R_p||) / ||R_j||$. We consider $||R_j||$ as a measure of the contribution a virtual point light to the scene and it is proportional to $||L^i_j||$. We use $||R||$, instead of $||L^i_j||$, to compute the weighting, because it contains more row samples and can provide a more numerically-stable estimation.

### 4.3.7 Anti-aliasing and Multiple Representatives

In the case of anti-aliasing, we render $s$ gather points for each pixel. To obtain better anti-aliasing of lighting effects (e.g., such as soft shadows), we modified our per-slice reconstruction. For each cluster, instead of selecting one representative, we randomly pick a set of $s$ representatives as described above. Each gather point in the pixel is
paired with a different representative in the set and this set of representative is shared by all pixels in the slice. By having each gather point in a pixel connected to a different representative, we enable a better shadow ray distribution, while still maintaining per-pixel noise to a minimum as in \textit{mrcs}.

### 4.4 Results and Discussion

#### 4.4.1 Overview

Table 4.2 shows statistics for the four scenes we tested, with geometric complexity ranging from 0.6 to 1.6 million triangles. We include timings for \textit{LightSlice}, \textit{lightcuts}, and \textit{mrcs}, where we show equal-time and equal-quality timings for the latter two methods. For each scene we tested, Figure 4.3 shows the equal-time renderings obtained with the three methods as well as a reference solution computed by rendering all VPLs. We generate all \textit{LightSlice} results using the same parameters for the gather kd-tree construction and the number of columns. Note that since we use an adaptive scheme to select the slices during the kd-tree build, the number of slices is different for each scene. Figure 4.4 shows a visualization of the gather point clusters for each scene, where each patch in the image corresponds to a slice of the transport matrix. We report timing results for parallel implementations of the various algorithms running on a machine with four Intel Xeon 7560 processors, each with 8 cores, running at 2.27GHz. \textit{LightSlice} converges within minutes. We report the average per-pixel relative error and the $L^2$ image error for all renderings.

For the \textit{lightcuts} algorithm, we chose to compare with the multidimensional \textit{lightcuts} variant [WABG06], because this is more efficient in the presence of antialiasing.
4.4 Results and Discussion

Table 4.2: Rendering statistics comparing LightSlice, multidimensional lightcuts, and matrix row column sampling. Note how our algorithm consistently gives us a significant speed up in equal quality comparisons or a smaller error in equal time comparisons.

<table>
<thead>
<tr>
<th>Scenes</th>
<th>Sanmiguel</th>
<th>Museum</th>
<th>Condo</th>
<th>Lobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>triangles</td>
<td>1.6M</td>
<td>1.5M</td>
<td>1.4M</td>
<td>1.4M</td>
</tr>
<tr>
<td>resolution</td>
<td>1024 × 1024</td>
<td>1024 × 1024</td>
<td>1200 × 900</td>
<td>1200 × 900</td>
</tr>
<tr>
<td>samples per pixel</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>VPL number</td>
<td>550k</td>
<td>153k</td>
<td>305k</td>
<td>317k</td>
</tr>
<tr>
<td>LightSlice</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg. rays per pixel</td>
<td>4372</td>
<td>3841</td>
<td>4093</td>
<td>4031</td>
</tr>
<tr>
<td>slice number</td>
<td>1471</td>
<td>1658</td>
<td>1698</td>
<td>1425</td>
</tr>
<tr>
<td>column number</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>sample slice time(s)</td>
<td>32.5</td>
<td>7.5</td>
<td>14.9</td>
<td>17.5</td>
</tr>
<tr>
<td>init clustering time(s)</td>
<td>55.9</td>
<td>24.9</td>
<td>75.9</td>
<td>65.6</td>
</tr>
<tr>
<td>refine &amp; render time(s)</td>
<td>651.5</td>
<td>376.69</td>
<td>222.28</td>
<td>747.76</td>
</tr>
<tr>
<td>total time(s)</td>
<td>775.1</td>
<td>425.9</td>
<td>351.9</td>
<td>866.5</td>
</tr>
<tr>
<td>avg. relative error</td>
<td>6.75%</td>
<td>2.77%</td>
<td>2.12%</td>
<td>1.14%</td>
</tr>
<tr>
<td>$L^2$ error</td>
<td>37.88</td>
<td>26.57</td>
<td>5.97</td>
<td>2.44</td>
</tr>
<tr>
<td>Matrix Row</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Column Sampling</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equal Quality</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg. rays per pixel</td>
<td>27315</td>
<td>36087</td>
<td>27169</td>
<td>22676</td>
</tr>
<tr>
<td>row number</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>column number</td>
<td>3000</td>
<td>4000</td>
<td>3000</td>
<td>2500</td>
</tr>
<tr>
<td>time (s)</td>
<td>4237(×5.46)</td>
<td>2076(×4.87)</td>
<td>1621(×4.60)</td>
<td>2542(×2.93)</td>
</tr>
<tr>
<td>avg. relative error</td>
<td>6.82%</td>
<td>2.84%</td>
<td>1.98%</td>
<td>1.13%</td>
</tr>
<tr>
<td>$L^2$ error</td>
<td>32.06</td>
<td>26.20</td>
<td>4.41</td>
<td>1.73</td>
</tr>
<tr>
<td>Matrix Row</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Column Sampling</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equal Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg. rays per pixel</td>
<td>4815</td>
<td>9087</td>
<td>9169</td>
<td>7376</td>
</tr>
<tr>
<td>column number</td>
<td>500</td>
<td>1000</td>
<td>1000</td>
<td>800</td>
</tr>
<tr>
<td>row number</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>avg. relative error</td>
<td>22.5%</td>
<td>8.83%</td>
<td>5.60%</td>
<td>2.51%</td>
</tr>
<tr>
<td>$L^2$ error</td>
<td>93.83</td>
<td>47.83</td>
<td>13.12</td>
<td>5.2</td>
</tr>
<tr>
<td>Multidimensional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lightcuts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equal Quality</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg. rays per pixel</td>
<td>15913</td>
<td>21548</td>
<td>15851</td>
<td>14993</td>
</tr>
<tr>
<td>max cut size</td>
<td>16000</td>
<td>30000</td>
<td>26000</td>
<td>22000</td>
</tr>
<tr>
<td>time (s)</td>
<td>3618(×4.66)</td>
<td>2816(×6.61)</td>
<td>1972(×5.60)</td>
<td>2738(×3.16)</td>
</tr>
<tr>
<td>avg. relative error</td>
<td>6.65%</td>
<td>2.96%</td>
<td>2.23%</td>
<td>1.41%</td>
</tr>
<tr>
<td>$L^2$ error</td>
<td>29.06</td>
<td>40.26</td>
<td>7.55</td>
<td>2.74</td>
</tr>
<tr>
<td>Multidimensional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lightcuts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equal Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg. rays per pixel</td>
<td>3999</td>
<td>5975</td>
<td>3952</td>
<td>4999</td>
</tr>
<tr>
<td>max cut size</td>
<td>4000</td>
<td>6000</td>
<td>4000</td>
<td>5000</td>
</tr>
<tr>
<td>avg. relative error</td>
<td>16.3%</td>
<td>7.4%</td>
<td>5.7%</td>
<td>3.7%</td>
</tr>
<tr>
<td>$L^2$ error</td>
<td>62.58</td>
<td>55.40</td>
<td>20.68</td>
<td>5.83</td>
</tr>
</tbody>
</table>
Figure 4.3: Equal-time comparison of VPL rendering methods. From the top: Sanmiguel, Museum, Condo and Lobby. Note that while LightSlice reproduces the complex lighting effects in the reference image, both other algorithms show various artifacts.

We augment the conservative error bound of *lightcuts*, which we set to 0.01, with a maximum cut size to induce stopping in equal-time comparisons and in some scenes locations where we found that the conservative bound would induce too many subdivisions. For our *mrcs* implementation, we chose to use raycasting rather than shadow mapping since our environments are large and contain fine geometric details. In these
4.4 Results and Discussion

Figure 4.4: A visualization of the gather point clusters corresponding to the matrix slices.

situations, shadow maps show severe artifacts due to sampling as well as biasing, as noted in the original paper.

4.4.2 Scenes

The Sanmiguel model is an outdoor scene with complex geometry lit by an environment map where most of the illumination come from the sun. This is our most challenging scene. For most parts of the image, global lighting is dominant, but with complex shadows. In the archway region, local lighting is dominant. mrcs quickly finds the
4.4 Results and Discussion

dominant global lights but has trouble converging on the archway region dominated by local effects. *Lightcuts* works well on those regions, but invest significant resources in rendering environment map lights, some of which are mostly occluded, since it lacks a visibility bound. Our algorithm combines the advantages of both prior methods since it can quickly exclude occluded global VPLs while converging quickly on local VPLs. These same trends are visible on all scenes tested.

The *Museum* model is an indoor scene with relatively simple geometry lit by an outdoor sun shining through small windows. The archway and ceiling area are lit only by indirect illumination. This scene is challenging for all VPL methods since most of the image is lit by strong local lighting. Our algorithm performs better than others, which exhibit more banding in equal time comparisons.

The *Condo* model is a relatively simple environment mostly lit by direct illumination coming from the ceiling. We chose this scene because it highlights the effect of multiple bounces of indirect lighting, which is particularly visible in the region below the stairs. Even in this lighting scenario, our algorithm performs very well, while the other two are less efficient in converging in the area mentioned.

Finally, the *Lobby* model is an indoor scene with an open ceiling covered by a thin metal frame. The illumination is dominated by global lighting from an environment map with regions of local lighting in the bottom floor. We chose this scene because it exhibits sharp and soft shadow from thin as well as large geometry, covering the range of direct shadowing characteristics. We found that even in this case, our algorithm works very well compared to the other two.
4.4 Results and Discussion

4.4.3 Performance in Equal Quality Comparison

To compare the three algorithms in terms of performance, we perform an equal-quality comparison by finding the best settings for \textit{mrcs} and \textit{lightcuts} with an exhaustive search. We chose to match the average relative error that better captures perceptual issues and is related to the \textit{lightcuts} stopping criteria. With these settings, we have a speed up of between roughly three to six times when compared to the other methods. \textit{LightSlice} allocates most of its resources to cluster refinement and final gathering, which are executed together in our parallel implementation. This, combined with the speed up we obtain, demonstrated that the exploration phase of our algorithm is well worth executing since it allows us to better direct resources according to the matrix structure.

In the scenes we tested, we found that \textit{mrcs} works best when we dedicate most of its resources to column sampling, as summarized in Table 4.2. This results in a significantly larger number of gather rays per pixel than our method for final image reconstruction. This slow convergence is due to the locality of lighting that \textit{mrcs} cannot capture. Since local lights only affect a small portion of the scene, the global clustering error is small while the per-slice error is large for some slices. \textit{mrcs} will not allocate resources (e.g., column samples) to these areas quick enough since it only minimizes a global clustering metric. This problem is inherent in global clustering and cannot be solved by increasing the number of row samples. Compared to \textit{mrcs}, our algorithm requires a larger number of rows to effectively discover the matrix structure. The benefit of this more expensive exploration phase though is a significant reduction in the number of columns needed for careful final reconstruction. This can be explained by observing that, since the rank of each matrix slice is low, per-slice clustering allows
4.4 Results and Discussion

our algorithm to dedicate most of its reconstruction resources to only the lights that are locally important for each slice.

In the scenes we tested, we found that lightcuts' bound is too conservative. While the original implementation of the algorithm continues to subdivide the tree until a conservative error bound is reached, we had to set a maximum per-pixel budget in the form of a maximum cut size, without which the algorithm would perform worse. This is due to the predefined light tree order, forcing some parts of the scene to explore lights that have little weight locally, and to the fact that the error metric does not account for visibility, which performs poorly in scene with complex shadowing. In these cases, the scalability of lightcuts is lost. We also found that while lightcuts might appear to have no explicit exploration cost, the cost of light tree refinement is not negligible, just like our cluster refinement. This leads to a slightly lower budget dedicated to ray shooting. Compared to lightcuts, our algorithm requires larger exploration time to discover the matrix structure, but the benefit of this more expensive exploration phase is a significant reduction in the number of gather rays needed for careful final reconstruction. We believe this is mostly due to the fact that LightSlice accounts for visibility during exploration and can thus quickly discard heavily shadowed lights that would be sampled aggressively by lightcuts.

4.4.4 Performance in Equal Time Comparison

We performed an equal-time comparison on all three algorithms by finding settings for mrcs and lightcuts that use roughly the same rendering time as LightSlice. Figure 4.5 plots the average relative error versus time for each algorithm obtained by varying the number of columns or maximum cut size. This comparison shows that LightSlice can
reduce error quicker than mrcs and lightcuts. Figure 4.6 shows the per-pixel relative error for the images in Figure 4.3. LightSlice achieves significantly lower error than the other two VPL methods given the same amount of time.

4.5 Limitations and Future Work

Parameters Selection. LightSlice implicitly avoids spatial discontinuities by using a high number of slices and including neighboring slices in the cluster refinement. Compared to mrcs which inherently has no discontinuities, LightSlice is more conservative when refining clusters and splits more aggressively. When lowering the number of slices, the coarse matrix slicing causes two types of artifacts, shown in Figure 4.7 (left). For some regions (e.g., red box), the algorithm does not capture the local matrix structure and thus does not allocate sufficient resources to locally-important VPLs. In re-
4.5 Limitations and Future Work

Figure 4.6: Per-pixel relative error for the equal-time comparisons of Figure 4.3. From the top: Sanmiguel, Museum, Condo and Lobby. The images show that given the same amount of time, LightSlice has lower error than other methods.
4.5 Limitations and Future Work

Figure 4.7: Left: Museum scene render with 378 slices and 400 columns. Banding artifacts are caused by illumination dissimilarities between gather point clusters as a result of the coarse matrix slicing. Right: The artifacts can be alleviated by increasing the number of columns 1200.

regions with strong dissimilarity between neighboring slices (e.g., yellow box), the discontinuities between gather point clusters become visible. Increasing the number of columns lowers the error thus alleviates these artifacts considerably, but also significantly increase the rendering time, as shown in Figure 4.7 (right).

Glossy Surfaces. All VPL methods have difficulty in handling highly glossy materials, since the number of VPLs needed grows too significantly for these algorithms to remain efficient. Moreover, glossy transports increase the local matrix rank, causing additional inefficiencies. In the case of LightSlice, a higher number of columns is needed to avoid banding artifacts coming from the dissimilarity between neighboring slices. A promising approach to support glossy surfaces if to combine selective raycasting with LightSlice as shown in [DKH’10].
4.6 Conclusion

**Animation.** In general, we expect a larger number of VPLs to be necessary for rendering flicker-free animations, lengthening rendering times. Hašan et al. proposed a tensor rendering approach that relies on reprojection to amortize final gathering cost [HVAPB08]. Applying this approach directly to our scenes would not work though because it would cause severe artifacts on fine geometry, preventing us from amortizing gathering cost. We leave it to future work to investigate an alternative approach to speed animations.

**Matrix Sparsity.** On the theoretical side, if the transport matrix is very sparse, LightSlice cannot efficiently converge on the correct solution, because it will have trouble finding the sparse elements. mrcs will have a similar behavior but with slower render times. Lightcuts will eventually converge on these cases, but at the price of rendering most of the matrix in an attempt to find the sparse elements. Besides the case of highly glossy surfaces, we have not been able to find such a case in rendering various scenes, including the ones that have difficult shadowing, such shadows from tree leaves, small windows, or fine geometry. Hašan et al. drew similar conclusions [HPB07]. In other words, while we cannot theoretically prove that all transport matrices have low rank structure, or at least have locally low rank structure, we have yet to find a case for which this is not the case. This makes this limitation worthwhile to highlight from a theoretical standpoint, but unlikely to become important in practice.

4.6 Conclusion

In this chapter, we present LightSlice, an algorithm that solves the many-light problem by clustering and rendering per-slice columns in the transport matrix. Our algorithm
4.6 Conclusion

can take advantage of the global and local behavior of VPL lighting, thus exploiting the transport matrix structure effectively. Compared to prior methods, our algorithm shows consistent speedups for a variety of lighting scenarios.
Chapter 5

ISHair: Importance Sampling for Hair Scattering

In Section 2.3.2, we introduced the path formulation of light transport and local path sampling, a strategy for new vertices generation. One option of local path sampling is to sample a new scattering direction using the local material property (i.e., $brdf$) of an existing vertex, and use ray tracing to find a new vertex. The effectiveness of sampling scattering directions has a direct impact to the overall variance of the estimator.

In this chapter, we look into the importance sampling algorithm for a super set of $brdf$, named bidirectional scattering distribution function ($bsdf$). In reality, light is not only reflected or absorbed by object surfaces, it can also transmits through object surfaces. A bidirectional transmission distribution function ($btdf$) models the transmission property of a surface point. $Bsdf$ is a union of $brdf$ and $btdf$. It allows us to represent the effects of reflection and transmission all under a unified framework. $Bsdf$ is defined
ISHair: Importance Sampling for Hair Scattering

as a function of incoming direction $\omega_i$ and outgoing direction $\omega_o$

$$f_s(x, \omega_o \leftarrow \omega_i)$$

where $x$ is the location where the bsdf is defined. Bsdf has a very similar definition as brdf. The only difference is that the incoming and outgoing directions (i.e., $\omega_i$ and $\omega_o$) of bsdf are defined over the sphere, instead of the hemisphere, because light may travel through the surface, opposite the surface normal.

In reality, some objects (e.g., wax, milk, and human skin) can have very complex, multiple-bounce inter-reflection and transmission property, create a non-local subsurface scattering effect, leading to a more complicated bidirectional subsurface scattering distribution function (bssdf) [JMLH01, dI11]. However, in this chapter, we only focus on the importance sampling algorithm for bsdf.

Importance sampling can effectively reduce the variance of an estimator, thus it is worth finding good sampling distributions for bsdfs. There has been extensive research on importance sampling different surface bsdfs. Analytic methods exist only for simple bsdfs such as Phong [Pho75], Lafortune [LFTG97], Ward [Lar92], and Cook-Torrence [KSK01]. For more complex bsdfs (e.g., hair bsdf), straightforward analytic solutions are usually not available. In these cases, various degrees of approximations discussed in Section 2.2.3 are used. In this chapter, we apply these approximations to derive an analytical solution for importance sampling hair bsdf.

A more general solution is to derive importance sampling functions using factorized representations or basis projections of bsdfs [LRR04, CJAMJ05, JCJ09]. Factorized representations or basis projections allow importance sampling for arbitrary bsdfs and for measured materials which are not defined analytically. However, these methods
5.1 Overview

Hair is a ubiquitous element of human and animal characters. High-quality hair rendering is essential to provide believable appearance in digitally-created content. We are interested in rendering hair lit by area and environment lights without precomputation to support dynamic scenes. These physically based light sources have become prevalent in both visual effects and animated feature films.

Marschner et al. introduced a physically-based scattering model that captures all the nuances of hair’s appearance [MJC’03]. However, this model is computationally expensive, requiring the solution of a cubic equation derived by internal path analy-
5.1 Overview

sis. Moreover, it is cumbersome for artists to directly control the appearance of hair by changing the model's parameters. To address these problems, Sadeghi et al. proposed an artist-friendly shading model for hair that approximates Marschner's model using only elementary functions, making it easier for artists to control than the purely physical based model [SPJT10]. In this thesis, we concentrate on this latter model.

Both these hair shading models have narrow peaks in their specular lobes, especially for shiny hair. This causes severe noise in Monte Carlo based rendering methods, especially when combined with large area lights and environment maps. Importance sampling is a widely used variance reduction technique for Monte Carlo numerical integration. In the context of rendering, importance sampling offers a means to reduce the variance by concentrating samples in regions with significant contribution to the illumination integral.

In this chapter, we present an efficient importance sampling method for the hair scattering $bsdf$ of [SPJT10]. Our method is capable of significantly improving the quality of the rendered image with negligible overhead. We reduce noise by drawing samples from a distribution that approximates well the scattering function in [SPJT10]. We do so efficiently since drawing samples requires only the evaluation of a few analytic functions, with no precomputation or significant memory footprint. We found our method easy to implement both in a prototype path tracer and in a micropolygon based production renderer. In both cases, results are further improved by using importance sampling of the $brdf$ in conjunction with importance sampling of lighting, a technique commonly known as multiple importance sampling (See Section 2.2.4).

The main contribution of our work is to provide a sampling algorithm for hair scattering that is effective (at reducing noise), robust, simple to implement and efficient
5.2 Hair Rendering Algorithms

to evaluate.

5.2 Hair Rendering Algorithms

Photorealistic Hair Rendering. There is a large body of work regarding hair modeling and shading. Here we review only the publications most closely related to our work, referring the reader to [WBK’07] for a detailed review. Kajiya and Kay proposed the first prominent model for hair rendering where they modeled the hair bsdf by computing light scattering from thin cylinders [KK89]. Marschner et al. improved upon this model by incorporating internal path analysis of hair strands [MJC’03]. Marschner’s work was the first complete physically-based hair shading model, capable of capturing the complex scattering behavior of hair. Zinke and Weber proposed a more general framework for filaments scattering [ZW07]. Inspired by Marschner’s model, Sadeghi et al. derived a practical hair shading model that is more efficient and easier for artist to control [SPJT10]. d’Eon et al. proposed an energy conserving hair reflectance model that includes several modifications to Marschner’s model to ensure energy conservation during scattering [dFH’11]. These models focus on providing accurate bsdfs for hair, but none provides an efficient method to importance sample the scattering functions. This is the focus of our work. While our method speeds up multiple scattering using Monte Carlo methods, it can also be integrated with more efficient multiple scattering solutions such as [MM06, MWM08, ZYWK08].

Importance Sampling Hair. Moon and Marschner proposed to sample the scattering directions by tracing rays through a rough elliptical cylinder, instead of importance sampling the hair bsdf [MM06]. Moon et al. sped up this process using a precom-
5.2 Hair Rendering Algorithms

Figure 5.1: Error images of [HR11] and our method. The edge cases of Box-Muller transform is not correctly handled in [HR11], resulting in incorrect energy estimation at grazing angles. [HR11] will not converge to correct solution as sample count increases (The error images are computed using per-pixel $L^2$ difference).

puted lookup table [MWM08]. These methods were either computationally expensive or required precomputation. Neulander et al. derived a practical importance sampling algorithm based on a cone-shell hair bsdf model, which was a variant of the Kajiya-Kay model [Neu10]. Their method, however, does not support hair models that have multiple specular lobes with different widths and offsets, so it does not apply to Marschner’s hair model or its variants. Hery and Ramamoorthi proposed an importance sampling method for the reflection lobe of hair bsdf [HR11]. Their method relied on the Box-Muller transform to sample the Gaussian distribution. This approach has several difficult edge cases and the solution presented in [HR11] has significant error and bias that can result in rendering artifacts (see Figure 5.1 and Appendix B.2).
5.3 Hair Shading Function

**Hair Rendering under Environment Lighting.** Hair rendering under environment lighting benefits from an efficient bsdf importance sampling algorithm. This case is so common that algorithms have been developed specifically for it [RZL*10, XMR*11]. While these methods work well in their problem domain, they are limited to environment lighting and require a considerable amount of precomputation. Moreover, they are all derived by approximations of the illumination integral, which makes it hard to integrate shadowing from complex dynamic occluders into these methods. These constraints limit their applicability in production rendering.

5.3 Hair Shading Function

We start the presentation of our importance sampling method with a summary of the hair shading model it supports, followed by the derivation of sampling functions for all the lobes of the hair bsdf and a complete description of our algorithm and its implementation. We follow the notation summarized in Table 5.1.

Sadeghi et al. propose an artist-friendly hair shading model [SPJT10], where the scattering function \( S(\theta_i, \phi_i, \theta_r, \phi_r) \) of hair fibers is decomposed into four individual components: reflection (R), refractive transmission (TT), secondary reflection without glint (TRT-g) and Glint (g). Each component is represented as a separate lobe and further factored as the product of a longitudinal term \( M \) and an azimuthal term \( N \).
5.3 Hair Shading Function

Table 5.1: Summary of notation.

<table>
<thead>
<tr>
<th>symbol</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S(\theta_i, \phi_i, \theta_r, \phi_r)$</td>
<td>hair bsdf</td>
</tr>
<tr>
<td>$M_R, M_{TT}, M_{TRT}$</td>
<td>longitudinal scattering functions</td>
</tr>
<tr>
<td>$N_R, N_{TT}, N_{TRT-g}, N_g$</td>
<td>azimuthal scattering functions</td>
</tr>
<tr>
<td>$\omega_i$</td>
<td>incoming direction</td>
</tr>
<tr>
<td>$\omega_r$</td>
<td>reflected direction</td>
</tr>
<tr>
<td>$u$</td>
<td>hair direction, pointing from the root to the tip</td>
</tr>
<tr>
<td>$v, w$</td>
<td>axes of the normal plane, orthogonal to $u$</td>
</tr>
<tr>
<td>$\theta_i, \theta_r$</td>
<td>inclination of $\omega_i$ and $\omega_r$ w.r.t the normal plane where $0^\circ$ is perpendicular to $u$, $90^\circ$ is $u$, and $-90^\circ$ is $-u$</td>
</tr>
<tr>
<td>$\phi_i, \phi_r$</td>
<td>azimuthal angles of $\omega_i$ and $\omega_r$ in the normal plane where $v$ is $0^\circ$ and $w$ is $90^\circ$</td>
</tr>
<tr>
<td>$\phi$</td>
<td>relative azimuthal angle, $\phi = \phi_r - \phi_i$</td>
</tr>
<tr>
<td>$\theta_d$</td>
<td>longitudinal difference angle $\theta_d = (\theta_r - \theta_i)/2$</td>
</tr>
<tr>
<td>$\theta_h$</td>
<td>longitudinal half angle $\theta_h = (\theta_r + \theta_i)/2$</td>
</tr>
</tbody>
</table>

The full scattering model is:

$$S(\theta_i, \phi_i, \theta_r, \phi_r) = I_R M_R(\theta_h) N_R(\phi) / \cos^2 \theta_d$$

$$+ I_{TT} M_{TT}(\theta_h) N_{TT}(\phi) / \cos^2 \theta_d$$

$$+ I_{TRT} M_{TRT}(\theta_h) N_{TRT-g}(\phi) / \cos^2 \theta_d$$

$$+ I_{TRT} M_{TRT}(\theta_h) I_g N_g(\phi) / \cos^2 \theta_d$$

$I_R$, $I_{TT}$ and $I_{TRT}$ are the colored intensities of the corresponding lobe while $I_g$ is the additional intensity of the Glint lobe.

$M_R$, $M_{TT}$ and $M_{TRT}$ model the longitudinal variation of each lobe. All three are
5.3 Hair Shading Function

Figure 5.2: Shapes of each hair lobes: Longitudinal lobes (Left). Azimuthal lobes (Right).

Gaussian functions of the longitudinal half angle $\theta_h$ as

\[
M_R = g(\beta_R^2, \alpha_R, \theta_h) \quad M_{TT} = g(\beta_{TT}^2, \alpha_{TT}, \theta_h) \quad M_{TRT} = g(\beta_{TRT}^2, \alpha_{TRT}, \theta_h)
\]

where $\beta_R$, $\beta_{TT}$, $\beta_{TRT}$ and $\alpha_R$, $\alpha_{TT}$, $\alpha_{TRT}$ are the widths and means of corresponding Gaussian functions. $\alpha$ controls the highlight shift of each lobe, while $\beta$ changes the roughness of the hair. In our notation,

\[
g(\beta^2, \alpha, \theta_h) = \exp \left[ -\frac{(\theta_h - \alpha)^2}{2\beta^2} \right]
\]

$N_R$, $N_{TT}$, $N_{TRT-g}$ and $N_g$ model the azimuthal variation of each lobe. All azimuthal terms are functions of the relative azimuthal angle $\phi = \phi_r - \phi_i$ and are defined respec-
5.4 Importance Sampling for Gaussian

Importance sampling is a technique used to reduce variance in Monte Carlo integration. To efficiently reduce variance, we seek to draw samples from a distribution whose probability distribution function (pdf) is proportional to the function we are integrating. In the context of hair rendering, we want to sample $\omega_i$ such that $p(\omega_i) \propto S(\theta_i, \phi_i, \theta_r, \phi_r)$.

We first describe how to efficiently sample each individual lobe, then we show how to combine all the lobes by randomly selecting a lobe based on an estimate of its energy. The longitudinal terms and azimuthal terms can be sampled independently since they depend on different variables. Specifically, we sample the spherical angles $\theta_i$ and $\phi_i$ separately, and then convert them into the direction $\omega_i$. The pdf of the sample is a product of the longitudinal pdf and the azimuthal pdf as $p(\omega_i) = p(\theta_i)p(\phi_i)$. We use the inverse cumulative distribution function (cdf) technique described in [PH10] to derive our analytic sampling functions.

where $\gamma_{TT}$ is a user controllable azimuthal width for $N_{TT}$. $N_g$ has two Gaussian functions with widths $\gamma_g$ that are symmetric about the axis $\phi = 0$, and $\phi_g$ is the half angle between its peaks.
5.4 Importance Sampling for Gaussian

**Sampling Gaussians.** Equation 5.1 uses Gaussian functions to model the variation in longitudinal and azimuthal scattering. Box-Muller transform is a general approach to draw samples from a Gaussian distribution with an infinite domain [BM58]. However, for this specific problem, we have to draw samples from a Gaussian distribution with a finite domain, e.g., \([-\pi/2 + \theta_r, \pi/2 + \theta_r]\). Using Box-Muller transform can result in samples outside of the finite domain (edge cases) that are difficult to handle (See Appendix B.2).

Inverse cdf can be used to constrain the samples to fall within the finite domain, but the lack of a closed form anti-derivative for Gaussian makes this approach infeasible. Although there are numerical approximations for the \(pdf\) and \(cdf\) of the Gaussian, they require the evaluation of error functions or Taylor series [PTVF07]. These methods are either computationally expensive or unstable at the tail of the Gaussian. To overcome these limitations, we would like to draw samples from a \(pdf\) that has a similar shape to the Gaussian function and a closed-form antiderivative. Observing that the Gaussian is a bell-shaped function with varying width and center, we can approximate it using another bell-shaped function.

**Cauchy distribution.** The Cauchy distribution is a probability distribution mainly used in physics, and it was recently used by computer graphics researchers as a sampling distribution [KF11]. It is defined as:

\[
f(\gamma, x - x_0) = \frac{1}{\pi} \left[ \frac{\gamma}{(x - x_0)^2 + \gamma^2} \right]
\]
5.4 Importance Sampling for Gaussian

Similar to the Gaussian, the Cauchy distribution is a bell-shaped function with offset $x_0$ and width $\gamma$. In contrast to the Gaussian, it has an analytic antiderivative

$$P(x) = \frac{1}{\pi} \tan^{-1} \left( \frac{x - x_0}{\gamma} \right)$$

This simple form of the antiderivative makes it possible to derive a sampling algorithm using the inverse $cdf$ technique. The offset and width of a Gaussian distribution can be directly used as the offset and width of the Cauchy distribution correspondingly. Figure 5.3 shows the plot of a set of Gaussian and Cauchy functions with same widths and offsets. The fact that Cauchy distributions have wider tails than Gaussians guarantees that using the Cauchy distribution to approximate the Gaussian in importance sampling will not increase variance. Using this approximation, we derive our sampling method for each lobe.

![Figure 5.3: Cauchy and Gaussian distributions with same widths and offsets. Both of distributions are normalized in the domain $[-\pi/2, \pi/2]$.](image)
5.5 Importance Sampling for Hair BSDF

5.5.1 Sampling Longitudinal Terms

Since the three longitudinal terms have the same form, we describe the approach using generic symbols $M$, $\beta$ and $\alpha$. Note that we ignore the $1/\cos^2 \theta_d$ terms for simplicity, since $M$ alone accounts for most of the variation in the longitudinal terms\textsuperscript{1}. Substituting the Gaussian functions in the $M$ terms with Cauchy distributions allows us to derive the sampling functions for incoming inclination $\theta_i$. Given a random variable $\xi$ uniformly drawn from range $[0, 1)$, we can sample $\theta_i$:

$$\theta_i = 2\beta \tan(\xi(A - B) + B) + 2\alpha - \theta_r$$

where $A = \tan^{-1}\left(\frac{\pi/4 + \theta_r/2 - \alpha}{\beta}\right)$ and $B = \tan^{-1}\left(-\frac{\pi/4 + \theta_r/2 - \alpha}{\beta}\right)$. The longitudinal pdf can be computed as

$$p(\theta_i) = \frac{1}{2\cos \theta_i(A - B)(\theta_h - \alpha)^2 + \beta^2} \frac{\beta \xi}{\beta^2}$$

5.5.2 Sampling Azimuthal Terms

All azimuthal terms are functions of relative azimuthal angle $\phi = \phi_r - \phi_i$. In our approach, we first sample $\phi$, then compute $\phi_i = \phi_r - \phi$. The pdf of $\phi$ is the same as the pdf of $\phi_i$ since $p(\phi_i) = p(\phi) \left| \frac{d\phi_i}{d\phi} \right|^{-1} = p(\phi)$.

\textsuperscript{1}$1/\cos^2 \theta_d$ term has a singularity when both $\theta_i$ and $\theta_r$ approach $\pi/2$ or $-\pi/2$. However, the projection term $\cos \theta_i$ in the rendering equation cancels its effect because $\cos \theta_i/\cos^2 \theta_d$ is a smooth function. Therefore the $M$ term remains the dominant source of variance.
5.5 Importance Sampling for Hair BSDF

**Sampling** $N_R$. $N_R$ is evaluated as $\cos(\phi/2)$. Deriving a sampling function for this term is trivial. Given a uniform random variable $\xi$ in $[0, 1)$, we sample $\phi$ as

$$\phi = 2 \sin^{-1}(2\xi - 1)$$

then we can compute $\phi_i = \phi_r - \phi$ and the azimuthal pdf $p(\phi_i) = p(\phi) = \frac{1}{4} \cos \frac{\phi}{2}$.

**Sampling** $N_{TT}$. $N_{TT}$ is defined with a Gaussian that is positive in the range $[0, 2\pi]$. We take an approach similar to the longitudinal terms. Given a uniform random variable $\xi$ in $[0, 1)$, we draw a sample of $\phi$ as

$$\phi = \gamma_{TT} \tan \left[ C_{TT} \left( \frac{\xi - 1}{2} \right) \right] + \pi$$

where $C_{TT} = 2 \tan^{-1} \left( \frac{\pi}{\gamma_{TT}} \right)$. We then compute $\phi_i = \phi_r - \phi$ and the azimuthal pdf $p(\phi_i) = p(\phi) = \frac{1}{C_{TT}} \left[ \frac{\gamma_{TT}}{(\phi - \pi)^2 + \gamma_{TT}^2} \right]$.

**Sampling** $N_{TRT-g}$. $N_{TRT-g}$ is approximated as $\cos(\phi/2)$. Since it is the same as the $N_R$ term, we follow the same approach as sampling $N_R$.

**Sampling** $N_g$. *Glint* models the lighting effect caused by the caustic light path inside hair strands. The azimuthal term of *Glint* is defined as two Gaussian functions symmetric about the $\phi = 0$ axis. Given a uniform random variable $\xi$ from $[0, 1)$, we choose one of two glint lobes by setting the sign of $\phi$ and remap $\xi$ back to the range $[0, 1)$ accordingly. Specifically, for $\xi < 1/2$, we set $\phi$ positive and map $\xi \leftarrow 2\xi$. For $\xi \geq 1/2$, we set $\phi$ negative and map $\xi \leftarrow 2(1 - \xi)$. After that, we sample $\phi$ using the remapped
5.5 Importance Sampling for Hair BSDF

\[ \xi \] as

\[ \phi = \gamma_g \tan(\xi(C_g - D_g) + D_g) + \phi_g \]

where \( C_g = \tan^{-1}\left(\frac{\pi/2 - \phi_g}{\gamma_g}\right) \), \( D_g = \tan^{-1}\left(-\frac{\phi_g}{\gamma_g}\right) \). Once we have \( \phi \), we compute \( \phi_i = \phi_r \pm \phi \), and compute its pdf as 

\[ p(\phi_i) = \frac{1}{2} \frac{1}{p(\phi)} = \frac{1}{2(C_g-D_g)} \left[ \frac{\gamma_g}{(\phi|\phi_g|^2+\gamma_g^2)} \right], \]

taking into account our remapping of the random variable.

5.5.3 Energy-based Lobe Selection

We have discussed how to sample each individual lobe. To sample the complete brdf, we distribute samples to each lobe. A simple solution is to uniformly select a lobe. To better match the energy distribution of the brdf, however, we use an energy-based lobe selection scheme. For each sample, we select a lobe with a probability proportional to an estimate of the energy of each lobe. We estimate these energies as the product of the integrals of the longitudinal and azimuthal terms. This results in the following estimates:

\[ E_R = 4\sqrt{2}\pi\beta_R I_R \]
\[ E_{\text{TRT-g}} = 4\sqrt{2}\pi\beta_{\text{TRT}} I_{\text{TRT}} \]
\[ E_{TT} = 2\pi\beta_{TT} \gamma_{TT} I_{TT} \]
\[ E_g = 4\pi\beta_{TRT} \gamma_g I_{TRT} I_g \]

We use the Gaussian integral in the domain \([-\infty, \infty]\) instead of \([-\pi/2, \pi/2]\) to compute the estimated energy. Although this is not accurate in general, it is easy to compute and works well as an estimation. The approximation error is less than 1% for \( \beta < 30^\circ \) and \( |\alpha| < 20^\circ \) and 0.003% for \( \beta < 20^\circ \) and \( |\alpha| < 10^\circ \).

We provide step-by-step derivations and the Python code for the sampling method in Appendix B.
5.6 Results

5.5.4 Implementation Details

Amortizing Constants Computation. It is important to note that $A_R, A_{TT}, A_{TRT}, B_R, B_{TT}, B_{TRT}, C_{TT}, C_g$ and $D_g$ in the sampling functions are constant for all the samples of the same gathering point and reflective direction $\omega_r$. We compute these constants once and amortize the cost over all the samples.

Longitudinal Grazing-angle PDF. Notice that the longitudinal pdf has a singularity when $\theta_i$ approaches $-\pi/2$ or $\pi/2$. The sample evaluation becomes numerically unstable at grazing angles. To avoid this problem, our implementation discards the sample if the angle between $\omega_i$ and $u$ or $-u$ is smaller than a predefined epsilon ($10^{-5}$ in our case). Although in theory this may bias the result, in practice, it only rejects a small percentage of samples ($< 0.001\%$) and all discarded samples have negligible contribution with weights ($< 0.0001$), resulting in negligible bias.

5.6 Results

5.6.1 Sample Distribution.

Figure 5.4 shows the sample distributions using the described importance sampling scheme. We use the Halton quasi-random sequence to generate the samples since it is repeatable and stratified [PH10]. Compared to uniform sampling (Figure 5.4.a), our importance sampling method (Figure 5.4.b) concentrates samples in regions of high importance. Figures 5.4.c – 5.4.f show the sample distribution of each individual lobe.
5.6 Results

Figure 5.4: Comparison of samples distributed using (a) uniform and (b) importance sampling. We use the pseudo-random Halton sequence to generate well-distributed random numbers. We show the sample distributions of each individual lobe using our importance sampling scheme. Notice the computed sample distribution match the energy distribution of the brdf.

5.6.2 Rendering Result

Overview. We implemented our importance sampling scheme for hair bsdf in a ray-tracing renderer written in C++. Moreover, to test our approach in a movie production environment, we also implemented our algorithm in a production renderer. Figure 5.5, Figure 5.7, and Figure 5.6 present the comparisons of our method with uniform sam-
5.6 Results

For both sampling schemes, we stratify the random numbers and render the images using multiple importance sampling (MIS) for direct illumination\(^2\). These are the best conditions for uniform sampling. Since we found our method to have negligible cost, we just report sample count rather than timing.

**Figure 5.5:** Comparison of our importance sampling approach and simple uniform sampling in a ray tracing renderer. (a). Direct illumination with an area light, our method is able efficiently sample the longitudinal lobes. (b). Direct illumination with environment lighting, our method generates smooth result with significantly fewer samples than uniform sampling which fails to converge on the glint and transmission highlights.

**Area Lighting.** Physically correct area lights have become widely adopted in production rendering. Figure 5.5.(a) is a simple scene with a large area light above the hair geometry rendered with our ray tracer. Uniform sampling exhibits significant noise at low sampling rates (32 samples), while the importance sampled result is relatively smooth. With 128 samples, the importance sampling image has no visible noise, while

\(^2\)When using MIS, a sample count of 16 corresponds to 16 bsdf samples and 16 light samples.
5.6 Results

![Figure 5.6: Comparison of our importance sampling approach and simple uniform sampling in a production renderer: (a). Direct illumination with area lighting; (b). direct illumination with environment lighting. While uniform sampling has trouble converging to a smooth image, our method generates noise-free images with only a few samples.](image)

the uniform sampled image still has some distracting noise. Figure 5.6.(a) is a production model lit with a large area light rendered with a production renderer. Due to the production renderer’s antialiasing techniques, only a few (32) importance samples are required to generate a smooth image; while uniform sampling required over 1k samples to converge.

**Environment Lighting.** Environment maps are used to add realistic lighting to a scene. Figure 5.5.(b) is a simple scene with an environment map of *Pisa Courtyard*. The illumination from this environment map is smooth with high color variation. In this case, while uniform sampling is not able to clean up the noise in the transmission and *Glint* highlights even at 256 samples, our importance sampling method is able to provide a smooth result with just a few samples (64 samples). Figure 5.6.(b) is
5.6 Results

Figure 5.7: (a). Global illumination with area lighting and (b). global illumination with environment lighting show that our method is able to efficiently sample the scattering direction for multiple bounces and drastically reduce the sample number needed for convergence. (c). Global illumination in a Cornell box shows our method efficiently gathers radiance from surrounding geometry.

a production model lit with the environment map of Ennis-Brown House, where our importance sampling method delivers better image quality than images rendering with $4 \times$ number of uniform samples.

Global Illumination. Global illumination enhances the overall realism of a scene. Of the many available algorithms, we use a path tracer since it is simple to implement and unbiased. In this case, scattering rays for indirect illumination are generated with $bsdf$
5.7 Discussion and Limitations

sampling only, either uniform or with importance. Although resolving multiple scattering using brute force path tracing is inherently inefficient, it guarantees a physically correct result.

Figure 5.7.(a) is a hair model lit by three area lights, and Figure 5.7.(b) is a hair model lit by the Grace Cathedral environment map. In both scenes, the hair model is the only geometry. All the indirect illumination is the result of multiple scattering inside the hair geometry. Figure 5.7.(c) is a simple scene with a hair model inside the Cornell Box, where indirect illumination comes from both the outside geometry and inside the hair geometry. For both uniform and importance sampling, we have to use a lot more samples in these tests than the previous direct lighting tests. While uniform sampling takes a long time to converge, our method converges to the correct result much faster with significantly fewer samples. Each reference image used more than 200M uniform samples per pixel and took over 60 – 80 hours to render.

5.7 Discussion and Limitations

Sampling Efficiency. Although our importance sampling method always yields better sample efficiency than uniform sampling, the magnitude of improvement highly depend on the setting of the scene and the hair bsdf. In the case of rough hair and high frequency environment map, the improvement is less obvious because in that case the light sampling become dominant in the MIS weighting.

Multiple Scattering. Our importance sampling algorithm is derived for the single scattering function. We show multiple scattering results rendered by path tracing, whose performance is drastically improved by using our importance sampling algorithm. Ap-
5.7 Discussion and Limitations

Approximation algorithms for multiple scattering are beyond the scope of this work. Although our sampling algorithm is not specifically designed for these algorithms, we believe that they can benefit from our work. For example, our importance sampling can be used to drive the photon shooting of [MM06] and light tracing of [MWM08].

**Extension to Marschner’s Model.** We believe our approach can be extended to support Marschner’s model [MJC’03]. It can be directly applied to sample the longitudinal (Gaussian) terms of the Marschner model\(^3\). However, applying it to the azimuthal terms is not trivial. Xu et al. proposed several approximations for fitting the azimuthal terms to Gaussian functions [XMR’11]. With these approximations, it would be possible to derive a sampling algorithm using our approach, with some precomputation and a small amount of overhead for each sample. We leave this extension to future work.

**Integration with Other Sampling Techniques.** Since our sampling algorithm does not require additional data structures, it can be easily integrated with other sampling techniques. We have only shown our method applied in a path tracer with multiple importance sampling, but it can also be used with other Monte Carlo techniques, e.g., photon mapping, bidirectional path tracing, or even more sophisticated unstructured illumination sampling techniques [WÅ09]. It may also be used to sample from point based global illumination [Chr08].

\(^3\)Marschner’s model uses normalized Gaussians instead of the unnormalized ones used in [SPJT10]’s model, but this will not affect the pdf derivation because the pdf is normalized by definition.
5.8 Conclusions

In this chapter, we presented an importance sampling algorithm for the hair \(bsdf\), that is simple to implement and efficient to evaluate. By approximating the Gaussian functions in the hair \(bsdf\) with Cauchy distribution, we were able to derive an analytic sampling algorithm with significantly reduced variance. We show results of applying our importance sampling method to render scenes with area lighting, environment lighting, indirect lighting and multiple scattering, in both a production renderer and a research raytracer.
Chapter 6

Out-of-core rendering

In this chapter, we look into some system issues of rendering scenes with high geometric complexity. We present a LOD-based path tracing framework that effectively solves out-of-core light transport problems.

Our approach employs LOD techniques to regulate the amount of details used during rendering, minimizing both the required system memory and rendering time. We take advantage of a hierarchical LOD, which is integrated into an out-of-core BVH, to speed up ray tracing operations. In our system, we use a simple LOD model, which is inexpensive to compute and compact to store, enabling us to greatly reduce the preprocessing time for building the BVH and the hierarchical LOD. To avoid the artifacts caused by using this less-accurate LOD model, we present a novel pdf-adaptive virtual cone sizing approach, and a Test-Verify shadow rays scheme.

This chapter is organized as follows: we first address some problems of out-of-core rendering, followed by a brief overview of our out-of-core LOD-based path tracing system in Section 6.1. After that, we give a brief review of out-of-core rendering researches in Section 6.2. Then we present an I/O-friendly algorithm for out-of-core
6.1 Overview

BVH construction in Section 6.3, and a simplified LOD model in Section 6.4. In Section 6.5, we discuss the virtual cone tracing algorithm and propose a few improvements that help mitigate some artifacts caused by using LOD. In Section 6.6, we present an \textit{out-of-core} path tracing implementation based on our improved virtual cone tracing algorithm. At the end, we present the some results in Section 6.7 and conclude our findings in Section 6.8.

6.1 Overview

As the visual effect and movie industries are striving for realism and high fidelity images, more and more polygons are used to capture the structural detail of real world objects. Memory usage grows as the complexity of scene geometry grows, and when the size of the scene dataset exceeds the size of total main memory in the system, the rendering problem becomes \textit{out-of-core}.

Although computer hard disks have been improved drastically in both speed and capacity over the last decade, disk I/O is still more than an order of magnitude slower than system memory access. Algorithms that frequently access data that does not reside in the main memory will be penalized by the I/O latency. Moreover, as the scene size grows, the size of the acceleration structure also grows linearly, forcing the rendering algorithm to perform computation in an \textit{out-of-core} manner. Given the existing complexity of global illumination, \textit{out-of-core} scenes may take hours even days to render. Therefore it is crucial to investigate \textit{out-of-core} rendering algorithms that can handle massive datasets efficiently.

As conventional algorithms are usually bounded by I/O when rendering \textit{out-of-core} datasets, our goal is to develop a global illumination algorithm that can efficiently ren-
6.1 Overview

nder scenes with a few hundred million to a few billion triangles. We choose to use path tracing to compute global illumination because of its simplicity and unbiased nature. Moreover, path tracing can be easily implemented to take advantage of the latest parallel system architectures. Our path tracing implementation uses a hierarchy LOD that is integrated into the BVH acceleration structure to speed up the ray intersection operations. The design of our path tracer can be split into three subproblems:

**Out-of-core BVH Construction.** BVH is a primitive subdivision scheme where primitives are partitioned into a hierarchy of disjointed sets. In order to obtain a high quality BVH build, the surface area heuristic (SAH) is often used to decide the splitting plane at each level [Hav00]. A typical SAH-based BVH construction algorithm usually needs to touch all the primitives multiple times, resulting in a less I/O-optimal access pattern. Therefore, designing a scalable, I/O-efficient algorithm for out-of-core BVH construction is the first step in computing out-of-core global illumination.

**Lightweight LOD Model.** LOD is a widely used in computer graphics to trade off between complexity and speed. In our out-of-core path tracing implementation, we use a hierarchical LOD to improve the coherency of memory access, reducing data accesses to the disk. Since we are targeting single-pass final rendering, time consuming pre-processing is not desirable. Furthermore, the out-of-core nature of the problem also imposes a memory constraint on the LOD storage. Therefore, the LOD model we are using needs to be inexpensive to compute, and has minimum memory overhead.

**Virtual Cone Tracing.** Given a hierarchical LOD, the virtual cone tracing technique is used to perform LOD selection in ray intersection operations [LBBS08, CNLE09,
6.2 Out-of-core Rendering

PFHA10, CNS’11]. One problem of the virtual cone tracing technique is that, if high-
quality LODs are not available, using a low-quality LOD model could introduce too
much error (i.e., bias) to the final images.

In following sections, we conduct a series of investigations into these three sub-
problems and present our out-of-core path tracing implementation.

6.2 Out-of-core Rendering

Out-of-core rendering falls into the category of external memory algorithms [Vit08].
It has been studied extensively in the field of scientific visualization and photorealistic
rendering. Here we only address the literatures that are mostly related to our works.
We refer interested readers to [SjCC’02] and [KDG’08] for a full review.

Rasterization vs. Ray tracing. Many out-of-core rendering researches are based on
rasterization rendering [ACW’99, VM02, BSGM02, YSM03, GLY’03, YSGM05]. Ras-
terization pipelines does not require sophisticated acceleration structures, and have a
small memory footprint. Moreover, rasterization pipelines tend to have a very coherent
and localized data access pattern, since the triangles are usually accessed sequentially
and only one triangle is processed at a time. However, non-local lighting effects (e.g.,
reflection, refraction, global illumination, etc.) cannot be achieved using rasterization
without applying various tricks. Moreover, the performance of rasterization is linearly
in the number of triangles. As the number of triangles grows, rasterization will become
prohibitively expensive.

In recent years, there has been increasing interest in ray tracing out-of-core datasets
because ray tracing has an asymptotic performance that is logarithmic in the number
6.2 Out-of-core Rendering

of triangles for a given image resolution. There has been a large amount of literature on ray tracing massive scenes \[PKGH97, WSB01, WDS04, YLM06, LBBS08, GMG08, CNLE09, BBS^*09, PFHA10, CNS^*11, GBPG11, Á12\]. Furthermore, ray tracing is able to simulate various lighting effects without special treatments, making it an ideal solution for computing out-of-core global illumination.

**Visibility Culling.** Visibility culling reduces the size of in-memory working set by rejecting objects that are invisible to the camera. The culling process is often based on viewing frustum, faces orientation, sub-pixel coverage, and occlusions \[ACW^*99, VM02, ZMHI97, ESSS01, WFP^*01, BSGM02, YSGM05, CHPR07\]. Unfortunately these techniques share the similar problem as rasterization pipelines in handling non-local effects, for which accesses to invisible objects are required.

**Static Level-of-detail.** Some rendering systems use static LOD, which is constructed off-line using various surface simplification algorithms \[GH97, ESV98, GP00, Lin00, WHDS04\]. Static LOD reduces memory accesses and working set size by exploiting multi-resolution data representations. Various models of different resolutions are constructed to represent the same object during the preprocessing phrase. The processing time and memory overhead varies depending on the number of levels in use (usually 3 – 10 levels). Because of the discrete nature of static LOD, these methods usually suffer from the discontinuous artifacts caused by the transitions between levels.

**Hierarchical Level-of-detail.** Hierarchical LOD allows relatively smooth transitions between levels. It can be used with point splatting technique to render large point clouds. *QSplat* used a point-based LOD to visualize out-of-core meshes with simple
6.2 Out-of-core Rendering

lighting [NBB04, RL00, RL01]. Gobbetti and Marton presented FarVoxel, which renders out-of-core models by employing a LOD model based on cubical view-dependent voxels [GM05]. Nevertheless, these systems support only simple material (i.e., Lambertian) and lighting (i.e., point/directional lights). Point-based LOD is also used in movie production to compute global illumination [CLF03, KTO11]. These methods use point clouds to represent directly illuminated geometry in the scene. These points are clustered into a octree. Directional-varied illumination is approximated using spherical harmonics stored in octree nodes. Unfortunately, these methods are not able to produce physically-correct images, due to the limited rasterization resolution and spherical harmonics truncations.

Ray tracing applications can also benefit from hierarchical LOD, since LOD proxies can be naturally embedded into hierarchical acceleration structure nodes. Yoon et al. used principle component analysis to approximate the subtrees of kd-tree nodes as planar surfaces [YLM06]. Wald et al. used ray tracing to compute the directionally-varied opacity of each subtree in a kd-tree. By averaging surface properties of the subtree, a LOD proxy is created for each kd-tree node [WDS04]. Lacewell et al. speeded up the occlusion computation for aggregated structures (i.e., plants, hairs) by using directionally-varied LODs computed by rasterizing the geometry under BVH subtrees [LBBS08]. Áfra proposed a hybrid approach that combines rasterization and ray tracing to build high-quality LODs, which allow them to perform interactive ray tracing on massive models [Á12]. These methods target fast/interactive rendering. Their LOD approximation often involves ray tracing or rasterization, thus it is time consuming to compute, resulting in a long preprocessing time (e.g., [Á12] reports 5 – 11 hours preprocessing time).
6.2 Out-of-core Rendering

Pantaleoni et al. presented PantaRay system, which efficiently compute directional occlusion and spherical integrals at arbitrary points in the scene [PFHA10]. They used a crude LOD approximation is inexpensive to compute and works well for the incoherent geometry in their settings. However, as we show later, this LOD approximation can cause problems when rendering planar surfaces. Given the fact that the quality of the LOD mode is a trade off between preprocessing time and accuracy, our method use a simplified LOD model to minimize the preprocessing time, while improving the rendering accuracy by employing a novel pdf-adaptive virtual cone sizing approach and a Test-Verify shadow rays scheme.

Resource Management. Resource management techniques reorganize the rendering workflow for coherent data accesses. Pharr et al. proposed a grid-based ray reordering scheme, which reorders the rays to trace only against primitives in memory [PKGH97]. Navratil et al. presented a ray scheduling approach that improves cache utilization and reduces DRAM-to-cache bandwidth usage [NFLM07]. Budge et al. improved upon the solution of Pharr et al. by introducing an data management scheme on heterogeneous architectures for out-of-core path tracing [BBS*09]. Moon et al. reordered rays using hit-point heuristic computed by intersecting the rays with simplified representations of the original models [MBK*10]. Yoon et al. presented several methods to improve the memory layout of the geometry or the acceleration structures for better data access coherence [YLPM05, YL06, YM06]. Gribble and Ramani organized rays into streams and used filtering operations to make wider-than-four SIMD viable for ray tracing [GR08]. We consider these works orthogonal to ours, since they mainly focused on workflow optimization. We believe that some of the ideas can be applied to our work with some modification to further speed up our implementation.
## 6.3 Out-of-core BVH Construction

Acceleration structure construction is usually the most time consuming task in the pre-processing phrase. Moreover, the quality of the acceleration structure has a direct impact to the overall efficiency of the renderer. Therefore, how to construct a high quality *out-of-core* acceleration structure efficiently is the first problem we are facing.

A considerable amount of literature has been published on fast acceleration structure construction algorithms [GPBG11, GPM11, CKL’10, BWW’12]. However, most of these studies target in-core acceleration structure construction. Here we only review works related to acceleration structure construction for *out-of-core* datasets.

Pharr et al. proposed using regular grids for rendering massive scenes [PKGH97]. Although grids are easy to construct and manage, their performance degrades when the scene geometry is not evenly distributed. Many works suggested the use of *two-level* acceleration structures. Budge et al. used a *two-level* kd-tree for *out-of-core* datasets, but the detail of how it is built is not addressed [BBS’09]. Hanika et al. proposed a *two-level* BVH for ray tracing massive data [HKL10]. However, they assumed that scene details can be generated on demand by tessellation and displacement mapping, thus the lower parts of the BVH can be constructed on the fly. These assumptions are not always valid for scenes exported from modeling softwares. In this work, we assume that scene datasets are exported from modeling tools, pre-tessellated, and saved on the disk.

Pantaleoni et al. presented *PantaRay* system, which built SAH-based BVH on massive microgrid datasets generated by RanderMan [PFHA10]. They proposed a stream-based approach that subdivides the data into buckets, chunks, and bricks. They mini-
6.3 Out-of-core BVH Construction

mized the number of times the stream is rewound by always working on subsets of data that fit in the main memory. Kontkanen et al. presented an algorithm to constructed an octree of point clouds. They used external sort to reorder points by its morton code, which allowed them to perform a bottom-up octree construction in a linear manner [KTO11]. Áfra proposed constructing a Kd-Tree using a top-down splitting scheme for triangle soups (i.e., a simple list of triangles without shared vertex data) [Á12]. Our BVH construction follows a similar procedure as [PFHA10], with a few differences due to different system constraints and assumptions.

6.3.1 System Constraints and Assumptions

Different from [PFHA10], our input datasets are organized as scene graphs with many surfaces. Each surface is a triangle mesh associated with a transform that maps local space to world space. Triangle meshes are defined in indexed forms. Each of them has a vertex list and a index list. A vertex could be shared by multiple triangles. Some surfaces have normal and/or texture coordinates in vertex data, while some others need to compute those properties on the fly during intersection. Surfaces are stored on disk as files. They can be loaded into memory for processing and unloaded when they are not needed. All these assumptions are very common for most exported datasets and scenes with animations.

Furthermore, we assume that a single surface can fit the main memory, that is typically the case when it is generated by modeling softwares. However, if a surface is larger than available memory, it can be split into smaller surfaces using the meta-cell technique beforehand [CSS98]. The surfaces may not be evenly distributed in the scene, and the number of triangles are not uniform among all the surfaces. In out tests,
6.3 Out-of-core BVH Construction

A few large surfaces can have more than 100 million triangles, while there are many small surfaces with 2–3 triangles. Although a single surface can fit in memory, building a BVH locally for each surface may not result in a high quality BVH (See Figure 6.1).

![Image](A) ![Image](B) ![Image](C)

**Figure 6.1:** Quality comparison between local build and global build. (A). Consider the scenario of a room. It is very common that the wall of the room is modeled as one surface and the objects in the room are modeled as their individual surfaces. (B). Building local BVH for each surface results in low quality build, since a lot of bounding boxes are overlapped. (C). Building a global BVH for the entire scene has a better subdivision result and a higher quality build.

6.3.2 Bucketing

To divide the input scene into subregions that fit in main memory, we need to have knowledge about the triangle distribution across the scene (Figure 6.2.(A)). We obtain this information by binning all triangles into a regular grid.

We first stream in all the surfaces to compute the scene's bounding box and the bounding box of each surface, all in world space\(^1\). After the scene's bounding box is computed, a regular 3D grid is initialized to cover the entire scene. In all our experiments, we use a grid size of \(1024 \times 1024 \times 1024\). We then stream in all the surfaces again to compute how many triangles fall in each grid cell (Figure 6.2.(B)). Specific-

\(^1\)If the bounding information is known beforehand (i.e., exported by the modeling software), this streaming pass can be skipped.
6.3 *Out-of-core* BVH Construction

cally, we load one surface into the memory at a time. For each triangle in a surface, we transform it to world space and compute its barycenter to determine which grid cell it falls into, then we increase the triangle count of the cell and recompute its bounding box. After processing all the triangles, the surface is unloaded from memory. In our implementation, we use a hash table to store the cells, which allows a memory-compact representation of the grid. After all the surfaces are processed, we extract all non-empty grid cells into a list of buckets (See Algorithm C.2 in Appendix C).

Note that our approach does not populate the buckets onto the disk as [PFHA10] did, because writing data to the disk is significantly slower than reading data in modern operating system, we want to wait until the actually subregion partition is refined.

6.3.3Chunking

**KD-Tree on Buckets.** After the bucketing pass, we have a coarse representation of the triangle distribution of the scene. A chunking pass is then performed to partition the scene into subregions that we can later build BVH on. We build a KD-tree over all the buckets by recursively splitting the bucket list along the longest axis of their bounding box. If the number of triangles inside the bucket list is less than a user-defined threshold (2 million triangles in our tests), we create a leaf node which represents a chunk. Otherwise, we split the bucket list using the SAH heuristic and create a branch node that contains the splitting plane (Figure 6.2.(C)). In this step, the KD-tree is built in memory using only the bucket list generated from previous step, thus no surfaces streaming is required.
6.3 Out-of-core BVH Construction

Figure 6.2: (A). Triangles unevenly distributed in the scene, grouped by surfaces (B). Bucketing pass estimates density of triangles throughout the scene. (C). Build KD-Tree for buckets. (D). Partition the scene into chunks (sub-regions) that have a manageable number of triangle. (E). The surfaces in blue is completely enclosed by a chunk, while the surfaces in red need to be split into multiple surfaces. (F). Scene surface after splitting. Each chunk is associated with one or more surfaces.

Surface Distribution and Splitting. The KD-tree built in pervious step partitioned the scene into several chunks (Figure 6.2.(D)). In this step, we propagate triangle data into corresponding chunks (leaf nodes). For each surface, we traverse the KD-tree using its bounding box computed in Section 6.3.2. For a surface that is totally enclosed by a chunk, it will reach a leaf node without intersecting any splitting planes. In that case, the surface is transformed into world space and deposited into the corresponding chunk. When a surface is deposited into a chunk, it is streamed into a disk file reserved
6.3 *Out-of-core BVH Construction*

for that chunk.

For a surface that is spread across multiple chunks, its bounding box will intersect with a splitting plane when descending the KD-tree. When that happens, we load the surface into memory and invoke a surface splitting subroutine to split the surface into multiple new surfaces using the KD-tree (Figure 6.2.(E)). The new surfaces are then deposited into their corresponding chunks (See Algorithm C.3 in Appendix C).

6.3.4 *BVH Construction*

**Building In-core BVH.** After the chunking pass, each chunk contains a set of surfaces that will fit in the main memory. We build an BVH subtree for each chunk in an in-core fashion. We first load all the surfaces in a chunk into the memory, and arrange all the triangles into a list. We then perform a top-down BVH build using the SAH heuristic. After the in-core BVH is built, LOD proxies are computed and stored into the BVH. We will discuss this in Section 6.4.

**Bricking.** The in-core BVHs are built using less compact data structures which has data redundancy. To improve cache efficiency and reduce space overhead, each in-core BVH is split it into smaller treelets, which stored as continuous segments called bricks with compact data structures. To reduce the storage overhead of LOD proxies, instead of storing a LOD proxy in each node, we only store them in the transitions between bricks. We introduced a new kind of nodes named *brick nodes*. A brick node contains the index of a LOD proxy. It serves as a transition indicator between bricks. The LOD proxy associated with a brick node can be used to determine whether a transition is necessary. Each brick can contain up to 256 nodes, triangles, vertices, or LOD proxies.
6.3 Out-of-core BVH Construction

This allows a very compact 8-bit representation.

Table 6.1 shows the layout of BVH nodes in a brick. A node can either be a branch node that contains the indices of its children within the brick, or a leaf node that refers to a list of triangles within the brick. It can also be a brick node that contains an index to a LOD proxy.

Table 6.1: Memory layout for different node types in bricks

<table>
<thead>
<tr>
<th>bytes</th>
<th>branch node</th>
<th>leaf node</th>
<th>brick node</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-23</td>
<td>bounding box</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>24</td>
<td>node type = {leaf, branch, brick}</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>25</td>
<td>splitting axis</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>26</td>
<td>left child index</td>
<td>triangle offset</td>
<td>LOD proxy index</td>
</tr>
<tr>
<td>27</td>
<td>right child index</td>
<td>triangle count</td>
<td>—</td>
</tr>
</tbody>
</table>

As noted in Section 6.3.1, triangles in surfaces are given in an indexed form. To avoid out-of-core dereferencing, the triangles are stored inside the brick to keep them close to the leaf nodes referencing them. Each vertex contains a 3D position, a 3D normal, and a 2D texture coordinate. For surfaces which do not have normal or texture coordinate information, corresponding fields are undefined. Triangles in bricks use a compact 5-byte representation (See Table 6.2). It consists of three 8-bit vertex index, a 2-bit triangle type, and a 14-bit material index in a triangle. Triangle type is a bit-field that indicates whether the normal field and/or texture coordinate field in its vertices are valid. It is used to determine whether those properties need to be computed on the fly during intersection. A 14-bit unsigned material index allows up to 16384 unique materials in a scene.

A in-core BVH is split into bricks using a post-order traversal. At each node, we keep track of a few properties of the subtree below it, including the number of unique vertices, the number of triangles, and the number of nodes. For leaf nodes, those statis-
6.4 Level-of-detail

Table 6.2: Memory layout for triangles in bricks

<table>
<thead>
<tr>
<th>bits</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-23</td>
<td>three 8-bits vertex index</td>
</tr>
<tr>
<td>24-25</td>
<td>triangle type</td>
</tr>
<tr>
<td>26-40</td>
<td>material index</td>
</tr>
</tbody>
</table>

tics can be compute directly. For branch nodes, the statistics are computed by merging the statistics of itself and both its children. If any of the properties exceeds 256, both of its children are serialized into bricks. After a subtree is serialized into a brick, it is replaced by a placeholder node and deleted from the memory. The placeholder node contains the brick’s ID to the brick and a LOD proxy, and it will be later serialized as a brick node (See Algorithm C.1 in Appendix C). In our implementation, we use the file offset of a brick as its brick ID. Note that the in-core BVHs can be built and serialized in parallel. In our implementation, up to 8 threads are used to process the chunks concurrently.

**Top-level BVH.** After all the chunks are processed, a two-level SAH-based BVH is built over all the chunks and serialized into bricks. This concludes our BVH building process. The entire BVH will be stored as a continuous file on disk.

6.4 Level-of-detail

As stated in Section 6.3.4, LOD proxies are computed after a in-core BVH is constructed. A LOD proxy contains a coarse representation of the subtree below a node. In our implementation, it also contains a brick ID of another brick and acts as a transition point between bricks. A LOD approximation consists of two parts: *surface properties* and *subtree opacity.*
6.4 Level-of-detail

**Surface Properties.** A LOD proxy approximates the surface properties of a subtree, including the surface normal, the texture coordinate, and the bsdf. Most of the prior works compute the approximation by simply averaging these properties under the subtree [PFHA10, CNS’11, Á12]. Our preliminary tests show that this approach does not necessarily provide a better approximation in practice. Moreover, it limits the supported material type to be Lambertian or Phong, because a unified model is required for averaging.

Instead of averaging the surface properties, our LOD model base on point sampling. We simply randomly pick a surface point in the subtree and use its surface properties as the approximation. The intuition behind is that if the subtree is very far away, each surface point has the same probability to be intersected by a ray. Although this assumption is not necessarily true for aggregated geometry, our approach shows little problems in practice and it is easy to implement. More importantly, it poses no constraints on scene materials.

For the in-core BVHs, the triangles are already loaded into the memory. At each node, we randomly select a triangle in the subtree with a probability proportional to its area. We then pick a random point on the triangle as a approximation of the subtree. For the top-level BVH, the triangles are not in the memory. In that case, for each node, we randomly select the approximation from one of its children with a probability propositional to the area sum of all triangles in the subtree.

**Subtree Opacity.** To approximate opacity of a subtree, we use the same heuristic as [PFHA10].

\[
\text{opacity}(\text{subtree}) = \min \left( 1, \frac{\sum_i \text{area}(\text{triangle}_i)}{\text{surface_area(\text{bbox}(\text{subtree}))}} \right)
\]  

(6.1)
Equation 6.1 is a very rough approximation for the occlusion in the subtree. Although Pantaleoni et al. argued that it is well suited for the incoherent geometry found in the vegetation, the lack of directional variance makes this approximation inaccurate for planar surfaces. However, in spite of its inaccuracy, this opacity approximation is extremely efficient to compute and compact to store compared to other methods [WDS04, LBBS08, Á12]. In Section 6.5.4, we present a Test-Verify shadow ray scheme that helps mitigating some artifacts caused by using this less-accurate opacity approximation.

### 6.5 Virtual Cone Tracing

The virtual cone tracing technique associates a virtual cone to each ray, and uses it as a metric for LOD selection. The size of a cone is defined by a solid angle $\sigma$. The cross sectional area of a cone at distant $r$ can be approximated as $\sigma r^2$. Therefore the cross sectional radius of the cone is

$$R_r = \sqrt{\frac{r^2 \sigma}{\pi}}$$

#### 6.5.1 LOD Selection

Our LOD selection scheme is similar to [LBBS08, PFHA10, CNS’11, Á12]. When a ray is intersecting with the BVH, we determine if the LOD of a subtree should be selected by testing whether the virtual cone encloses the subtree. Specifically, when a brick node is reached, we compare the cross sectional radius of the cone $R_r$ with the radius of the subtree bounding box $R_b$. The distance $r$ is the distance between the ray origin and the
6.5 Virtual Cone Tracing

center of the bounding box. \( R_b \) is defined as half of the bounding box's diagonal \( d \).

\[
R_b = \frac{d}{2} = \frac{\text{diagonal}(\text{bbox}(\text{subtree}))}{2}
\]

If \( R_r \geq R_b \), the ray stops descending and the LOD of the subtree is used, otherwise a brick is loaded into memory and the ray is descended into a new subtree (See Figure 6.3).

![Figure 6.3: Virtual cone tracing. (A). A cone is associated with each ray, its cross sectional radius of the cone \( R_r \) is calculated using distance \( r \) and solid angle \( \sigma \). (B). The radius of the subtree \( R_b \) is approximated as half of its bounding box diagonal. (C). LOD selection is performed using metric \( R_r \geq R_b \).](image)

In our implementation, instead of testing \( R_r \geq R_b \), we avoid the square root operations by testing

\[
d^2 \frac{\pi}{4} < r^2 \sigma
\]

Note that we do not test intersections with the cone, but use it as the footprint of
6.5 Virtual Cone Tracing

the ray for LOD selection (See Algorithm 6.1 for pseudocode). A virtual cone can be considered as an isotropic form of the ray differential [CLF’03, Ige99].

**Algorithm 6.1: LOD Ray Intersect**

```python
def intersectLOD(ray, node, sigma):
    if ray.intersect(node.bbox):
        if isLeaf(node):
            intersectTriangles(ray, node)
        elif isBrick(node):
            d2 = squareDiagonal(node.bbox)
            r2 = squareDistance(node.bbox.center, ray.origin)
            if d2 * pi / 4 < sigma * r2:
                resolveLOD(node.lod)  # use LOD proxy
            else:  # load subtree and recursive descend
                brick = loadBrick(node.brickID)
                intersectLOD(ray, brick.root, sigma)
        else:
            intersectLOD(ray, node.left, sigma)
            intersectLOD(ray, node.right, sigma)
```

6.5.2 Pixel of Error Metric

For primary rays, we calculate the solid angle $\sigma$ of the virtual cones by reformulating the *pixels-of-error* metric (PoE) [YLM06, Á12]. Assuming each pixel sample is projected as a circle on the screen, its projected radius at unit distance is

$$R' = \frac{4 \tan \frac{\phi}{2}}{w}$$

where $\phi$ is the field–of–view of the camera, and $w$ the number of pixel samples along $\phi$. $w$ can be computed as

$$w = w_{\text{image}} \sqrt{\frac{n_{\text{pixel}}}{\pi}}$$
6.5 Virtual Cone Tracing

where $w_{\text{image}}$ is the width of image and $n_{\text{pixel}}$ is the samples per pixel. Since solid angle is defined as the surface area of a unit sphere enclosed by the cone, we can approximate it as

$$\sigma = R^2 \pi = \left( \frac{4 \tan \frac{\phi}{2}}{w} \right)^2 \pi$$

6.5.3 PDF-Adaptive Cone Size

For indirect rays or shadow rays, the virtual cone is defined over the hemisphere. Most of prior works assumed that each virtual cone subtends a uniform solid angle of $2\pi/n$ ($4\pi/n$ for spherical cases), where $n$ is the number of samples. While this assumption is sufficient for diffuse surfaces, it fails to adapt to glossy surfaces. If importance sampling is used, the sample density for glossy bsdf is not uniform. As shown in Figure 6.4, glossy bsdf has much higher sample density in the “peak” region. Moreover, the “peak” region has a higher contribution to the integral and demand a more accurate estimation, hence a smaller cone size.

![bsdf samples](image)

**Figure 6.4:** (A). Uniform sample distribution of diffuse bsdf. (B). Non-uniform sample distribution of glossy bsdf

In [OP10], we made an observation that the sample’s pdf is proportional to the
6.5 Virtual Cone Tracing

inversion of its corresponding solid angle measure. Here we derive an adaptive cone size metric based on the sample's pdf.

\[ \sigma = \left( \frac{1}{p(x)} \right) / \tau = \frac{\tau}{p(x)} \]

where \( p(x) \) is the pdf of sample \( x \), and \( x \) can be a shadow ray sample or a indirect ray sample. \( \tau \) is a user-defined parameter. Using the a small \( \tau \) results in a large overall cone size. Many rays can have early exits at the upper part of the BVH. However, it can also introduce large error to the image. Using a large \( \tau \), on the other hand, leads to a small overall cone size. The estimator will be more accurate but less efficient. Figure 6.5 shows a comparison between using our pdf-adaptive cone sizing approach and using a uniform cone size.

6.5.4 Test-Verify Shadow Ray Scheme

Unlike primary rays and indirect rays, a shadow ray determines the visibility between two points. The accuracy of shadow ray query greatly affects the accuracy of direct illumination. For non-LOD BVHs, a shadow ray query returns a boolean value indicating whether there is a occluder blocking the ray. For LOD BVHs, a shadow ray query returns a float point value between 0.0 to 1.0, indicating the occlusion value along the virtual cone. This float point occlusion value is usually computed by blending the opacity of all the LOD proxies along the virtual cone [LBBS08, PFHA10]. It can be interpreted as the probability that a random ray enclosed by the virtual cone is occluded.

In our implementation, we simply accumulate the opacity of the subtree along the ray (See Algorithm 6.2). The intersection routine may have early exit when the occlu-
6.5 Virtual Cone Tracing

Figure 6.5: Compare pdf-adaptive cone size and uniform cone size. (A). Using a uniform cone size causes visual artifacts for glossy surfaces. (B). Using pdf-adaptive cone size provides a more accurate estimation, since narrower cones are used for glossy reflection rays. (C). Reference Images.

Algorithm 6.2: LOD Shadow Ray Intersect

```python
def intersectAnyLOD(ray, node, sigma):
    occlusion = 0
    if ray.intersect(node.bbox):
```

130
6.5 Virtual Cone Tracing

```python
if isLeaf(node) and intersectAnyTriangles(ray, node):
    occlusion = 1
elif isBrick(node):
    d2 = squareDiagonal(node.bbox)
    r2 = squareDistance(node.bbox.center, ray.origin)
    if d2 * pi / 4 < sigma * r2:
        occlusion = node.opacity
    else:
        # load subtree and recursive descend
        brick = loadBrick(node.brickID)
        occlusion = intersectAnyLOD(ray, brick.root, sigma)
else:
    occlusion += intersectAnyLOD(ray, node.left, sigma)
    occlusion += intersectAnyLOD(ray, node.right, sigma)
return occlusion
```

As mentioned in Section 6.4, our opacity approximation of a subtree is very rough, and it is not reliable for planar surfaces. In our case, it tends to underestimate the occlusion and causes light leaking. To avoid this problem, we propose a Test-Verify shadow ray scheme. The intuition is that since the shadow ray test using LOD is not reliable, we should only use it as a heuristic to decide whether a non-LOD shadow ray should be shoot in an out-of-core manner. The basic idea is to first shoot a LOD shadow ray to test the occlusion along the ray.

- If the occlusion value is 0.0, we are sure that the ray does not intersect with any subtrees or triangles.
- If the occlusion value is equal to or larger than 1.0, we have high confidence that the ray is blocked. That usually happens when there is a lot of occluders along the ray.
- If the occlusion value is between 0.0 and 1.0, we are uncertain whether the ray is blocked or not. Maybe only a small number of LOD proxies are intersected with ray and/or the triangles inside them are sparsely distributed. In this case, we shoot a non-LOD shadow ray to verify the exact occlusion. The portion of
6.6 Out-of-core Path Tracing Implementation

shadow rays that need to be verified various depends on the scene structure. In the Slum scene, roughly 5% of the shadow rays needs the additional verification step.

Algorithm 6.3: Test-Verify Shadow Ray Intersect

```python
def intersectAnyTV(ray, node, sigma):
    occlusion = intersectAnyLOD(ray, node, sigma);
    if occlusion >= 1.0: return True;
    elif occlusion <= 0.0f: return False;
else: return intersectAny(ray, node)
```

Algorithm 6.3 is the pseudocode of our Test-Verify shadow ray scheme. By using this scheme, we significantly reduced the error introduced by virtual cone tracing (See Figure 6.6).

6.6 Out-of-core Path Tracing Implementation

We implemented an out-of-core path tracer based on our improved virtual cone tracing technique. The path tracer is written using C++ and parallelized on CPU using Intel’s TBB library [Rei07]. The out-of-core BVH built in Section 6.3 is mapped\(^2\) into the memory space of the process for direct access. In order to increase the coherency between threads, we partition all the image pixels into square blocks. All the pixels in a block are rendered in parallel. In all our tests, we use the block size of 16 \(\times\) 16 pixels. To improve the data access locality between blocks, we reorder the image blocks into a Z-curve. Specifically, we sort the image blocks by their Morton code [Mor66].

We follow the framework of the classical path tracing algorithm introduced in Section 2.4. Rays are generated starting from the camera, and the virtual cone size of

\(^2\)we use the mmap command on unix system
6.6 Out-of-core Path Tracing Implementation

Figure 6.6: Comparison of cone tracing images with Test-Verify and without Test-Verify. Without using the Test-Verify scheme, the over-simplified opacity approximation causes inaccurate occlusion estimation. Using our Test-Verify scheme, our approach effectively suppresses the light leaking artifacts.
6.7 Results

each ray is computed using the pixel error metric in Section 6.5.2. Multiple importance
sampling is used for direct lighting estimation, the virtual cone size of a shadow ray
is computed using the pdf-adaptive metric in Section 6.5.3. Shadow ray queries are
carried out using the Test-Verify scheme in Section 6.5.4. We compute at maximum six
bounces of indirect lighting, the virtual cone size of a indirect ray is also determined
by our pdf-adaptive metric.

6.7 Results

6.7.1 Experimental Environment

All the experiments are run on a iMac with a 4-core 2.93 GHz Intel Core i7 processor,
16 GB RAM, 256GB SSD drive. We tested our out-of-core path tracer on test scenes
with different features: a highly tessellated Sanmiguel scene, a Slum scene, and a
Forrest scene. The Sanmiguel scene and the Slum scene are combinations of plane
surfaces (e.g., wall and flat ground) and aggregated surfaces (e.g., tree and grass). The
geometry distribution of Sanmiguel is extremely uneven. The Forest scene is mostly
consisted of aggregation surfaces. Both the Slum scene and the Forest scene have over
a billion triangles. All tests scenes contain only triangle meshes without instancing,
and they also contain a various range of materials.

6.7.2 BVH Construction

Table 6.3 shows the processing time for each stage of the BVH building process. The
result shows that our method has a shorter turnaround time compared to [PFHA10]
and [KTO11], with similar amount of geometry.
6.7 Results

<table>
<thead>
<tr>
<th>Triangle Count</th>
<th>Sanmiguel</th>
<th>Slum</th>
<th>Forest</th>
</tr>
</thead>
<tbody>
<tr>
<td>World Bound</td>
<td>124.0s (7%)</td>
<td>223.0s (8%)</td>
<td>300.5s (8%)</td>
</tr>
<tr>
<td>Bucketing</td>
<td>210.6s (13%)</td>
<td>378.0s (13%)</td>
<td>440.0s (12%)</td>
</tr>
<tr>
<td>Chunking</td>
<td>570.8s (34%)</td>
<td>921.3s (31%)</td>
<td>1326.3s (35%)</td>
</tr>
<tr>
<td>BVH &amp; Bricking</td>
<td>751.2s (45%)</td>
<td>1447.8s (48%)</td>
<td>1688.1s (45%)</td>
</tr>
<tr>
<td>Total Time</td>
<td>1664.6s</td>
<td>2976.8s</td>
<td>3775.4s</td>
</tr>
</tbody>
</table>

6.7.3 Comparison to Non-LOD Path Tracing

We compare our out-of-core LOD path tracing implementation with a regular path tracing implementation without LOD.

Figure 6.7 shows the comparison on the Slum scene. The Slum scene is illuminated by an environment light. Since the scene is a outdoor scene, it is dominated by direct illumination. The scene geometry contains a lot of planer surfaces distributed and oriented in a random fashion. The scene has an open sky, therefore most of the indirect rays and shadow rays are not occluded. The memory accesses for indirect rays and shadow rays are local and coherent. As a result, although the scene has over a billion triangles, both algorithms are able to converge quickly.

Figure 6.8 shows the comparison on the Sanmiguel scene. Although the highly tessellated Sanmiguel scene has less triangles compared to the other two scene, it is proved to be a very hard case for out-of-core path tracing. Although it is an outdoor scene, the main part of the scene is surrounded by tall walls. The scene is illuminated by a far-away area light as the sun, and a highly occluded environment map. Therefore
6.7 Results

<table>
<thead>
<tr>
<th>LOD</th>
<th>1024 × 512</th>
<th>32 samples</th>
<th>177.528s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-LOD</td>
<td>1024 × 512</td>
<td>4 samples</td>
<td>168.776s</td>
</tr>
<tr>
<td>Reference</td>
<td>1024 × 512</td>
<td>1024 samples</td>
<td>3723.76s</td>
</tr>
</tbody>
</table>

Figure 6.7: Quality comparison of our LOD path tracing and regular non-LOD path tracing on Slum scene.

This scene is dominated by indirect illumination and the direct illumination with high variance. Moreover, the memory access pattern for indirect rays is highly incoherent, e.g., an indirect ray originates from a wall could shoot across the entire scene and hit the wall on the opposite side. In this case, both methods need significantly more samples and rendering time for the images to converge.

The result shows that, in both scenes, our LOD path tracer implementation runs significantly faster than a regular path tracer, thus it is able to process nearly 3 times more samples compared to the regular path tracer implementation, and drastically reduce the noise in the images.
6.8 Conclusions

In this chapter, we investigated the rendering of datasets with high geometry complexity, i.e., out-of-core scenes. We first looked into the construction of BVH acceleration structure for out-of-core datasets, and presented an I/O optimized algorithm that is able to build BVH for scenes with a few hundred million triangles to a few billion triangles. We then discussed the LOD technique for out-of-core rendering, the virtual cone tracing technique and its limitations. We proposed several improvement to virtual
cone tracing technique, including a novel \textit{pdf}-adaptive virtual cone sizing approach, and a \textit{Test-Verify} shadow rays scheme. In the end, we presented an \textit{out-of-core} path tracing implementation based on our improved virtual cone tracing technique. The result showed that our \textit{out-of-core} path tracing implementation significantly improved the rendering performance for \textit{out-of-core} scenes compared to a regular path tracing implementation.
Chapter 7

Conclusion

This thesis seeks to address the problem of increasing complexity in rendering and tackle this problem using various sampling techniques. We investigated several areas of rendering that benefit from advanced sampling techniques.

We began by laying out the theoretical basis of our research in Chapter 2 by first introducing light transport problem in the context of rendering, followed with the basic concept of Monte Carlo integration. We also pointed out the limitation of Monte Carlo integration, which is its slow convergence rate of $O(\sqrt{N})$. Given this limitation, we then moved onto the discussion of importance sampling, which is one of the most important variance reduction techniques for Monte Carlo integration, followed by a brief introduction of multiple importance sampling. After that, we examined the path integral formulation of light transport, which allows us to apply Monte Carlo integration to solve light transport problems. At the end of Chapter 2, we introduced local path sampling techniques and three categories of rendering algorithms (i.e., Path Tracing, Photon Mapping, and Virtual Point Light Algorithms), which use different path sampling and reusing strategies.
Conclusion

Given the fact that scenes with complex settings take a long time to render, as alternatives, progressive rendering algorithms are often used to provide quick but imperfect feedback to users. In Chapter 3, we presented a user study to evaluate how the artifact patterns generated by different progressive rendering algorithms affect users' performance in the context of appearance design. In the study, we asked 26 subjects with different skill levels to perform a series of appearance editing tasks using four different progressive rendering algorithms (i.e., Random Path Tracing, Quasi-Random Path Tracing, Photon Mapping, and Virtual Point Light). User feedback of the algorithms are collected through questionnaires and system logging. The experimental result suggests that path tracing is strongly preferred to progressive photon mapping and VPL rendering by most subjects. Moreover, there is no indication that quasi-random path tracing is systematically preferred to random path tracing or vice-versa; the same holds between progressive photon mapping and VPL rendering. This user study not only strengthened our understanding of progressive rendering, but also gave us insights for designing appearance editing tools, content creation tools, and other interactive applications.

The second area of the thesis concerns the rendering of complex light transport. Despite the fact that the VPL algorithm is less favored in performing appearance design tasks, it is widely used in movie and game industries. In Chapter 4, we focused on high quality VPL rendering with complex light transport. We tackled the gathering problem of VPL algorithm (i.e., the many-light problem). By formulating the many-light problem into a matrix form, we observed that if we cluster similar surface samples together, the slice of the matrix corresponding to these surface samples has significantly lower rank than the original matrix. We exploited this observation and proposed Light-Slice, a two-step algorithm based on matrix slice sampling. By taking advantage of the
Conclusion

global and local behavior of VPL lighting and exploiting the transport matrix structure effectively, LightSlice achieves a significant speed up compared to prior state-of-the-art many-light algorithms for a variety of scenes.

For the third part of the thesis, we looked into the importance sampling technique for complex surface material. In Chapter 5, we started with a brief discussion of bidirectional scattering distribution function (bsdf) followed by an introduction to hair rendering in movie production. In this investigation, we aimed to derive an efficient importance sampling algorithm for complex hair bsdf. We first introduced the artist-friendly hair bsdf, which is a complex multi-lobe function. We separated the artist-friendly hair bsdf into individual lobes and further divided each lobe into longitudinal terms and azimuthal terms. We proposed using Cauchy distribution to approximate Gaussian distribution and derived an analytic importance sampling algorithm for each lobe. Finally, we combined samples from each lobe using an energy-based lobe selection algorithm. We tested our importance sampling algorithm in both an experimental ray tracing renderer and a movie production renderer. The results showed that our importance sampling algorithm is able to greatly reduce the number of samples needed for convergence and significantly speed up hair rendering processes under various lightings conditions.

For the last part of the thesis, we set out to tackle the problem of rendering extremely large and complex geometry. As the rendering problem becomes out-of-core when the scene dataset cannot fit the main memory at one time, the rendering process will be substantially slowed down by I/O latency. In Chapter 6, We investigated the problem of out-of-core rendering. We first presented an I/O optimized out-of-core BVH construction algorithm for massive models. Then we presented our hierarchical LOD
model, which is integrated into the BVH. We used a random point sampling scheme and a rough opacity approximation to compute LOD proxies. To utilize the hierarchical LOD, we introduced the virtual cone tracing algorithm. After discussing some limitations of the virtual cone tracing algorithm, we proposed two improvements: a pdf-adaptive virtual cone sizing approach, and a Test-Verify shadow rays scheme. In the end, we presented an out-of-core LOD path tracing algorithm based on our improved virtual cone tracing algorithm. We tested our LOD path tracing implementation on various scenes compared to a regular non-LOD path tracing implementation. The result showed that our implementation performed significantly faster than a regular path tracing implementation, thus it is able to evaluate many more samples in the same amount of time and reduce the variance much quicker.

In the context of sampling for complexity, this thesis covers a wide range of topics in rendering. The works we presented in this thesis made several noteworthy contributions to progressive rendering, light sampling, material sampling, and out-of-core rendering. We believe this research enhanced our understanding of rendering datasets under complex settings, and will give insights into future rendering research.
Appendix A

Questionnaire for the Progressive Rendering User Study

Knowledge of lighting and material design

Please rate your previous experience with lighting design in 3D computer graphics.
(1) No experience
(2) Have tinkered with it
(3) Have worked on a project
(4) Have worked on more than one project
(5) Work as a professional

Please rate your previous experience with material design in 3D computer graphics.
(1) No experience
(2) Have tinkered with it
(3) Have worked on a project
(4) Have worked on more than one project
(5) Work as a professional

Random Path Tracing Ratings
Lighting Design Tasks

Please rate how well you matched the goal for matching trials.
(5 is the best)
Trial 1 (studio, scale):
1 2 3 4 5
Trial 2 (kitchen, move):
1 2 3 4 5
Trial 3 (museum, move):
1 2 3 4 5
Trial 4 (lobby, move):
1 2 3 4 5

Please rate how satisfied you are with your result for open trials.
(5 is the best)
Trial 5 (museum, all):
1 2 3 4 5
Trial 6 (lobby, all):
1 2 3 4 5
Questionnaire for the Progressive Rendering User Study

**Random Path Tracing Ratings**

**Material Design Tasks**

Please rate how well you matched the goal for matching trials.
(5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (studio, brightness):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>2 (kitchen, roughness):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>3 (museum, glossiness):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>4 (lobby, color):</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

Please rate how satisfied you are with your result for open trials.
(5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (studio, all):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>6 (kitchen, all):</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

**Random Path Tracing Comments**

1. How would you compare lighting and material adjustments in terms of your workflow?

2. How well you were able to identify the features you were supposed to adjust?

3. How did the artifacts (errors) in the image affect your workflow?

4. Did the artifacts (errors) in the image affect your final choice in the open trial?

---

**Quasi-Random Path Tracing Ratings**

**Lighting Design Tasks**

Please rate how well you matched the goal for matching trials.
(5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (studio, scale):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>2 (kitchen, move):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>3 (museum, move):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>4 (lobby, move):</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

Please rate how satisfied you are with your result for open trials.
(5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (museum, all):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>6 (lobby, all):</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

**Quasi-Random Path Tracing Ratings**

**Material Design Tasks**

Please rate how well you matched the goal for matching trials.
(5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (studio, brightness):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>2 (kitchen, roughness):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>3 (museum, glossiness):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>4 (lobby, color):</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

Please rate how satisfied you are with your result for open trials.
(5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (studio, all):</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>6 (kitchen, all):</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>
Quasi-Random Path Tracing Comments

1. How would you compare lighting and material adjustments in terms of your workflow?

2. How well you were able to identify the features you were supposed to adjust?

3. How did the artifacts (errors) in the image affect your workflow?

4. Did the artifacts (errors) in the image affect your final choice in the open trial?

Photon Mapping Ratings

Lighting Design Tasks

Please rate how well you matched the goal for matching trials. (5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (studio, scale)</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>2 (kitchen, move)</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>3 (museum, move)</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>4 (lobby, move)</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

Please rate how satisfied you are with your result for open trials. (5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (museum, all)</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>6 (lobby, all)</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

Material Design Tasks

Please rate how well you matched the goal for matching trials. (5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (studio, brightness)</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>2 (kitchen, roughness)</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>3 (museum, glossiness)</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>4 (lobby, color)</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

Please rate how satisfied you are with your result for open trials. (5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (studio, all)</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>6 (kitchen, all)</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

Photon Mapping Comments

1. How would you compare lighting and material adjustments in terms of your workflow?

2. How well you were able to identify the features you were supposed to adjust?

3. How did the artifacts (errors) in the image affect your workflow?

4. Did the artifacts (errors) in the image affect your final choice in the open trial?
Questionnaire for the Progressive Rendering User Study

**Virtual Point Lights Ratings**

**Lighting Design Tasks**

Please rate how well you matched the goal for matching trials.
(5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Studio, Scale</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial 1</td>
<td>1 2 3 4 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trial 2</td>
<td>1 2 3 4 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trial 3</td>
<td>1 2 3 4 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trial 4</td>
<td>1 2 3 4 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please rate how satisfied you are with your result for open trials.
(5 is the best)

<table>
<thead>
<tr>
<th>Trial</th>
<th>Museum, All</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial 5</td>
<td>1 2 3 4 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trial 6</td>
<td>1 2 3 4 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Virtual Point Lights Comments**

1. How would you compare lighting and material adjustments in terms of your workflow?

2. How well were you able to identify the features you were supposed to adjust?

3. How did the artifacts (errors) in the image affect your workflow?

4. Did the artifacts (errors) in the image affect your final choice in the open trial?

**Final Ratings and Rankings**

**Lighting Design Tasks**

1) Your preference for working on lighting matching trials:
How does each interface rate on a scale of 1 to 5?
(1 implies do not prefer. 5 implies prefer.)

<table>
<thead>
<tr>
<th>Interface</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Path Tracing</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>Quasi-Random Path Tracing</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>Photon Mapping</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>Virtual Point Lights</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

If forced to choose, how would you rank the interfaces?

1st preference: _____________________
2nd preference: _____________________
3rd preference: _____________________
4th preference: _____________________

2) Your preference for working on lighting open trials:
How does each interface rate on a scale of 1 to 5?
(1 implies do not prefer. 5 implies prefer.)

<table>
<thead>
<tr>
<th>Interface</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Path Tracing</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>Quasi-Random Path Tracing</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>Photon Mapping</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>Virtual Point Lights</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

If forced to choose, how would you rank the interfaces?

1st preference: _____________________
2nd preference: _____________________
3rd preference: _____________________
4th preference: _____________________
3) Your preference for working on material matching trials:
How does each interface rate on a scale of 1 to 5?
(1 implies do not prefer. 5 implies prefer.)
Random Path Tracing: 1 2 3 4 5
Quasi-Random Path Tracing: 1 2 3 4 5
Photon Mapping: 1 2 3 4 5
Virtual Point Lights: 1 2 3 4 5
If forced to choose, how would you rank the interfaces?
1st preference: _____________________
2nd preference: _____________________
3rd preference: _____________________
4th preference: _____________________

4) Your preference for working on material open trials:
How does each interface rate on a scale of 1 to 5?
(1 implies do not prefer. 5 implies prefer.)
Random Path Tracing: 1 2 3 4 5
Quasi-Random Path Tracing: 1 2 3 4 5
Photon Mapping: 1 2 3 4 5
Virtual Point Lights: 1 2 3 4 5
If forced to choose, how would you rank the interfaces?
1st preference: _____________________
2nd preference: _____________________
3rd preference: _____________________
4th preference: _____________________

5) Your overall preference:
How does each interface rate on a scale of 1 to 5?
(1 implies do not prefer. 5 implies prefer.)
Random Path Tracing: 1 2 3 4 5
Quasi-Random Path Tracing: 1 2 3 4 5
Photon Mapping: 1 2 3 4 5
Virtual Point Lights: 1 2 3 4 5
If forced to choose, how would you rank the interfaces?
1st preference: _____________________
2nd preference: _____________________
3rd preference: _____________________
4th preference: _____________________

6) Describe why you choose the previous overall ranked:
(For example, suggest briefly what was good and bad about each method).
Appendix B

Derivation and Pseudocode for ISHair

B.1 Derivations

B.1.1 Derivation of Longitudinal Sampling

Given a uniform random variable $\xi$ in $[0, 1)$, we want to draw a sample of $\theta_i$ from the

PDF

$$p(\theta_i) \propto \left[ \frac{\beta}{(\frac{\theta_i + \theta_r}{2} - \alpha)^2 + \beta^2} \right] \frac{1}{\cos \theta_i}$$

The $1/\cos \theta_i$ term is the correcting factor when transforming integrals over solid angle into integrals over spherical coordinates. The normalization gives

$$\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} c \left[ \frac{\beta}{(\frac{\theta_i + \theta_r}{2} - \alpha)^2 + \beta^2} \right] \frac{1}{\cos \theta_i} \cos \theta_i \, d\theta_i = 2c \tan^{-1} \left( \frac{\theta_i - \alpha}{\beta} \right) \bigg|_{-\pi/2 + \theta_r}^{\pi/2 + \theta_r} = 1$$
B.1 Derivations

Therefore \( c = \frac{1}{2(A-B)} \), where \( A = \tan^{-1} \left( \frac{\pi/2 + \theta_r}{\beta} - a \right) \) and \( B = \tan^{-1} \left( \frac{-\pi/2 + \theta_r}{\beta} - a \right) \). The pdf of \( \theta_i \) is:

\[
p(\theta_i) = \frac{1}{2 \cos \theta_i (A-B)} \frac{\beta}{\left(\frac{\theta_i + \theta_r}{2} - \alpha\right)^2 + \beta^2}
\]

The cdf can be computed by integrating the pdf

\[
P(\theta_i) = \int_{-\pi/2}^{\theta_i} c \left[ \frac{\beta}{\left(\frac{\theta_i' + \theta_r}{2} - \alpha\right)^2 + \beta^2} \right] \frac{1}{\cos \theta_i'} \cos \theta_i' d\theta_i'
\]

\[
= \tan^{-1} \left( \frac{\theta_i + \theta_r}{\beta} - a \right) - B
\]

\[
= \frac{\tan^{-1} \left( \frac{\theta_i + \theta_r}{\beta} - a \right)}{A-B} - B
\]

By inverting the cdf, we sample \( \theta_i \), given a uniform random variable \( \xi \) from \([0, 1]\), as \( \theta_i = 2\beta \tan(\xi(A-B) + B) + 2\alpha - \theta_r \).

**B.1.2 Derivation of \( N_R \) and \( N_{TRT-g} \) Sampling**

Given a uniform random variable \( \xi \) from \([0, 1]\), we want to draw a sample of \( \phi \) from the pdf

\[
p(\phi) \propto \cos \frac{\phi}{2}
\]

The normalization gives that

\[
\int_{-\pi}^{\pi} c \cos \frac{\phi}{2} d\phi = c \int_{-\pi/2}^{\pi/2} 2 \cos x dx = 2 \cos x \bigg|_{-\frac{\pi}{2}}^{\frac{\pi}{2}} = 2c = 1
\]
B.1 Derivations

Therefore, \( c = 1/4. \) The pdf of \( \phi \) is

\[
p(\phi) = \frac{1}{4} \cos \frac{\phi}{2}
\]

The cdf can be computed by integrating the pdf

\[
\int_{-\pi}^{\phi} \frac{1}{4} \cos \frac{\phi}{2} \, d\phi = \frac{1}{2} \sin \frac{\phi}{2} \bigg|_{-\pi}^{\frac{\phi}{2}} = \frac{1}{2} \left( \sin \frac{\phi}{2} + 1 \right)
\]

By inverting the cdf, we sample \( \phi \), given a uniform random variable \( \xi_2 \) from \([0, 1)\), as

\[
\phi = 2 \sin^{-1}(2\xi_2 - 1)
\]

Then we can compute \( \phi_i = \phi_r - \phi \). We have to transform the pdf \( p(\phi) \) to \( p(\phi_i) \) and it can be proved that \( p(\phi_i) = \left| \frac{d\phi_i}{d\phi} \right|^{-1} p(\phi) = p(\phi) \)

B.1.3 Derivation of \( N_{TT} \) Sampling

Given a uniform random variable \( \xi \) from \([0, 1)\), we want to draw a sample of \( \phi \) from the pdf

\[
p(\phi) \propto \frac{\gamma_{TT}}{\left( \phi - \pi \right)^2 + \gamma_{TT}^2}
\]

The normalization gives that

\[
\int_{0}^{2\pi} c \left[ \frac{\gamma_{TT}}{\left( \phi - \pi \right)^2 + \gamma_{TT}^2} \right] d\phi = c \left[ \tan^{-1} \left( \frac{\phi - \pi}{\gamma_{TT}} \right) \right]_{0}^{2\pi} = 1
\]
B.1 Derivations

Therefore $c = \frac{1}{\gamma_{TT}}$ where $\gamma_{TT} = 2\tan^{-1}(\pi/\gamma_{TT})$. Then we can compute the $pdf$ of $\phi$

$$p(\phi) = \frac{1}{\gamma_{TT}} \left[ \frac{\gamma_{TT}}{(\phi - \pi)^2 + \gamma_{TT}^2} \right]$$

The $cdf$ can be computed by integrating the $pdf$

$$\int_0^\phi c \left[ \frac{\gamma_{TT}}{(\phi' - \pi)^2 + \gamma_{TT}^2} \right] d\phi' = \frac{1}{\gamma_{TT}} \left[ \tan^{-1}\left( \frac{\phi' - \pi}{\gamma_{TT}} \right) \right]_0^\phi = \frac{\tan^{-1}\left( \frac{\phi - \pi}{\gamma_{TT}} \right)}{\gamma_{TT}} + \frac{1}{2}$$

By inverting the $cdf$, we sample $\phi$, given a uniform random variable $\xi$ from $[0, 1)$, as

$$\phi = \gamma_{TT} \tan \left[ \frac{\xi - \frac{1}{2}}{\gamma_{TT}} \right] + \pi$$

Then we can compute $\phi_i = \phi_r - \phi$ and $p(\phi_i) = p(\phi)$

B.1.4 Derivation of $N_g$ Sampling

Given a uniform random variable $\xi$ from $[0, 1)$, we want to draw samples of $\phi$ from the $pdf$

$$p(\phi) \propto \frac{\gamma_g}{(|\phi| - \phi_g)^2 + \gamma_g^2}$$

We first use $\xi$ to randomly pick a half of the lobe and remap the random variable $\xi_2$ back to $[0, 1)$. Specifically, for $\xi < 1/2$, we set $\phi$ positive and map $\xi \leftarrow 2\xi$. For $\xi \geq 1/2$, we set $\phi$ negative and map $\xi \leftarrow 2(1 - \xi)$. Then we sample $\phi$ in the domain
B.1 Derivations

[0, π/2). The normalization gives

\[ \int_0^{\pi/2} c \left[ \frac{\gamma_g}{(|\phi| - \phi_g)^2 + \gamma_g^2} \right] d\phi = c \left[ \tan^{-1} \left( \frac{\phi - \phi_g}{\gamma_g} \right) \right]_0^{\pi/2} = 1 \]

Therefore \( c = \frac{1}{C_g - D_g} \) where \( C_g = \tan^{-1} \left( \frac{\pi/2 - \phi_g}{\gamma_g} \right) \) and \( D_g = \tan^{-1} \left( -\frac{\phi_g}{\gamma_g} \right) \). We can compute the pdf of \( \phi \)

\[ p(\phi) = \frac{1}{C_g - D_g} \left[ \frac{\gamma_g}{(\phi - \phi_g)^2 + \gamma_g^2} \right] \]

The cdf can be computed by integrating the pdf

\[ \int_0^\phi c \left[ \frac{\gamma_g}{(\phi' - \phi_g)^2 + \gamma_g^2} \right] = \frac{1}{C_g - D_g} \left[ \tan^{-1} \left( \frac{\phi' - \phi_g}{\gamma_g} \right) \right]_0^\phi \]
\[ = \frac{\tan^{-1} \left( \frac{\phi - \phi_g}{\gamma_g} \right) - D_g}{C_g - D_g} \]

We sample \( \phi \), given a uniform random variable \( \xi \) from [0, 1), as

\[ \phi = \gamma_g \tan(\xi(C_g - D_g) + D_g) + \phi_g \]

Then we can compute \( \phi_i = \phi_r \pm \phi \). The sign of \( \phi \) is determined by the value of the original random variable \( \xi \) before remapping. We also transform the pdf to account the remapping of the random variable.

\[ p(\phi) = \frac{1}{2} p(|\phi|) = \frac{1}{2(C_g - D_g)} \left[ \frac{\gamma_g}{(|\phi_r - \phi_i - \phi_g|^2 + \gamma_g^2} \right] \]
B.2 Issues of the Use of Box-Muller Transform

Although Box-Muller transform is easy to implement and works well for sampling Gaussian function in general, it has a major shortcoming for our specific problem. By definition, the incoming longitudinal angle $\theta_i$ has a valid range of $[-\pi/2, \pi/2]$. Ignoring offset $\alpha$ for brevity, $\theta_h$ has a valid range of $[\frac{-\pi/2+\theta_r}{2}, \frac{\pi/2+\theta_r}{2}]$ (red shaded area in Figure B.1.a-b). Since Box-Muller transform generates samples from $(-\infty, \infty)$, many samples generated fall out of the valid range. Figure B.1.c shows a possible $\theta_h$ sample that is out of the valid range. These invalid samples can be handled by rejection sampling but incurs the cost of wasted samples. Attempts to keep the samples are complicated and prone to error. Moreover, Box-Muller transform does not provide a way to compute the Gaussian integral over finite intervals (required for normalizing the pdf).

While dealing with edge cases is inevitable and complicated using Box-Muller transform, our approach does not have this short coming because it is based on an inverse CDF technique that can sample $\theta_i$ directly. As a result, we can ensure both $\theta_i$ and $\theta_h$ always fall into valid range.

Hery and Ramamoorthi introduced a method to importance sample the R lobe of hair using the Box-Muller transform[HR11]. We implemented their approach based on the pseudocode in their paper and compared it to our method using scenes in this thesis. Note that [HR11] did not provide solutions for sampling lobes other than the R lobes. To make an fair comparison, we only rendered the R lobe in all the examples.

Figure B.2 shows the rendering result of [HR11] and our method. The error images shows that Hery and Ramamoorthi’s method generate extra energy at grazing angles. Figure B.3 is the error plots of area light scene in Figure B.2. As the sample count
B.2 Issues of the Use of Box-Muller Transform

Figure B.1: (a). For case $\theta_r = -\frac{\pi}{4}$, $\theta_h$ has a valid range $[-\frac{3\pi}{8}, \frac{\pi}{8}]$ in order to make sure $\theta_i \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. (b). The valid range of $\theta_h$ only cover a portion of the entire Gaussian distribution. (c) Box-Muller transform draw samples from interval $(-\infty, \infty)$, some sample will end up outside the valid range. In this case, $\theta_h = \frac{3\pi}{4}$. (d). As a result, $\theta_i = 2\theta_h - \theta_r = \frac{3\pi}{4}$, which is not within valid range. In [HR11], this is handled by setting $\theta_i = \frac{\pi}{2} - \theta_i = \frac{\pi}{4}$. However, this also changed $\theta_h$ from $\frac{\pi}{4}$ to $0$, causing inconsistency between pdf, bsdf value and the sampling direction $\omega_i$.

increases, our method constantly produces lower error compared to [HR11].
B.2 Issues of the Use of Box-Muller Transform

Figure B.2: Error Images of [HR11] and our method. The edge cases of Box-Muller transform are not correctly handled in [HR11], resulting in incorrect energy estimation at grazing angles. The images will not converge to correct solution as sample count increases (The error images are computed using per-pixel $L^2$ difference).
B.2 Issues of the Use of Box-Muller Transform

(a). $L^2$ error vs. sample count

(b). Maximum pixel error vs. sample count

Figure B.3: $L^2$ error plots of area light scene in Figure B.2 rendered using [HR11] and our method. Error is measured as the differences in comparison to the reference image. As the sample count increases our method consistently achieves lower error than [HR11].
B.3 Pseudocode

B.3 Pseudocode

Here is the Python pseudo code for our importance sampling algorithm. To keep the code simple, we used the simple uniform lobe selection instead of the energy-based lobe selection. Moreover, it is done without amortizing the cost of constants computation.

```python
pi = 3.1415926

# sample the primary lobe
def sample_R_lobe(uv, I):
    (theta_r, phi_r) = compute_angle(I)
    a_R = arctan(((pi/2 + theta_r)/2 - alpha_R) / beta_R)
    b_R = arctan((-pi/2 + theta_r)/2 - alpha_R) / beta_R
    t = beta_R * tan(uv[0] * (a_R - b_R) + b_R)
    theta_h = t + alpha_R
    theta_i = (2 * theta_h - theta_r)
    phi = 2 * arcsin(1 - 2 * uv[2])
    phi_i = phi_r - phi
    phi_pdf = cos(phi/2) / 4
    return compute_direction(theta_i, phi_i)

# sample the transmission lobe
```

B.3 Pseudocode

```python
def sample_TT_lobe(uv, I):
(\theta_r, \phi_r) = compute_angle(I)

a_{TT} = \arctan(((\pi/2 + \theta_r)/2 - \alpha_{TT}) / \beta_{TT})
b_{TT} = \arctan(((\pi/2 + \theta_r)/2 - \alpha_{TT}) / \beta_{TT})
c_{TT} = 2 * \arctan(\pi/2 / \gamma_{TT});

t = \beta_{TT} * \tan(uv[0] * (a_{TT} - b_{TT}) + b_{TT})
theta_h = t + \alpha_{TT}
theta_i = (2 * \theta_h - \theta_r)

double p = \gamma_{TT} * \tan((v - 0.5) * c_{TT})
double phi = p + \pi
double phi_i = \phi_r - phi

return compute_direction(theta_i, phi_i)

# sample the secondary highlight lobe
def sample_TRT_G_lobe(uv, I):
(\theta_r, \phi_r) = compute_angle(I)

a_{TRT} = \arctan(((\pi/2 + \theta_r)/2 - \alpha_{TRT}) / \beta_{TRT})
b_{TRT} = \arctan(((\pi/2 + \theta_r)/2 - \alpha_{TRT}) / \beta_{TRT})

t = \beta_{TRT} * \tan(uv[0] * (a_{TRT} - b_{TRT}) + b_{TRT})
theta_h = t + \alpha_{TRT}
theta_i = (2 * \theta_h - \theta_r)

phi = 2 * \arcsin(1 - 2 * uv[2])
phi_i = \phi_r - phi
phi_pdf = \cos(\phi/2) / 4

return compute_direction(theta_i, phi_i)

# sample the glint lobe
def sample_G_lobe(uv, I):
(\theta_r, \phi_r) = compute_angle(I)

a_{TRT} = \arctan(((\pi/2 + \theta_r)/2 - \alpha_{TRT}) / \beta_{TRT})
b_{TRT} = \arctan(((\pi/2 + \theta_r)/2 - \alpha_{TRT}) / \beta_{TRT})
```

158
B.3 Pseudocode

c_G = atan((pi/2 - phi_g) / gamma_G)
d_G = atan(-phi_g / gamma_G)

t = beta_TRT * tan(uv[0] * (a_TRT - b_TRT) + b_TRT)
theta_h = t + alpha_TRT
theta_i = (2 * theta_h - theta_r)

if(uv[1] < 0.5):
    uv[1] = 2 * uv[1]
    sign = 1
else:
    uv[1] = 2 * (1 - uv[1])
    sign = -1

phi = sign * (p + phi_g)
phi_i = phi_r - phi

return compute_direction(theta_i, phi_i)

# compute the pdf of primary highlight
def compute_R_pdf(L, I):
    (theta_r, phi_r) = compute_angle(I)
    (theta_i, phi_i) = compute_angle(L)

    if(pi/2 - theta_i < epsilon):
        return 0

    a_R = arctan((((pi/2 + theta_r)/2 - alpha_R) / beta_R)
b_R = arctan((-pi/2 + theta_r)/2 - alpha_R) / beta_R)

    theta_h = (theta_i + theta_r) / 2
    t = theta_h - alpha_R
    theta_pdf = beta_R / (t*t + beta_R*beta_R) /
        (2*(a_R - b_R) * cos(theta_i))

    phi = phi_r - phi_i
    phi_pdf = cos(phi/2) / 4

    return theta_pdf * phi_pdf
B.3 Pseudocode

# compute the pdf of transmission

def compute_TT_pdf(L, I):
    (theta_r, phi_r) = compute_angle(I)
    (theta_i, phi_i) = compute_angle(L)

    if(pi/2 - theta_i < epsilon):
        return 0

    a_TT = arctan(((pi/2 + theta_r)/2 - alpha_TT) / beta_TT)
    b_TT = arctan(((pi/2 - theta_r)/2 - alpha_TT) / beta_TT)
    c_TT = 2 * arctan(pi/2 / gamma_TT);

    theta_h = (theta_i + theta_r) / 2
    t = theta_h - alpha_R
    theta_pdf = beta_R / (t*t + beta_R*beta_R) / (2*(a_R - b_R)*cos(theta_i))

    phi = abs(phi_r - phi_i)
    if phi < pi/2:
        phi_pdf = 0
    else:
        p = pi - phi
        phi_pdf = (gamma_TT / (p*p + gamma_TT*gamma_TT)) / c_TT

    return theta_pdf * phi_pdf

# compute the pdf of secondary highlight without glint

def compute_TRT_G_pdf(L, I):
    (theta_r, phi_r) = compute_angle(I)
    (theta_i, phi_i) = compute_angle(L)

    if(pi/2 - theta_i < epsilon):
        return 0

    a_TRT = arctan(((pi/2 + theta_r)/2 - alpha_TRT) / beta_TRT)
    b_TRT = arctan(((pi/2 - theta_r)/2 - alpha_TRT) / beta_TRT)
    c_TT = 2 * arctan(pi/2 / gamma_TT);

    theta_h = (theta_i + theta_r) / 2
    t = theta_h - alpha_R
B.3 Pseudocode

\[
\theta_{\text{pdf}} = \frac{\beta_R}{(t^2 + \beta_R^2)} / \left(2(a_R - b_R) \cos(\theta_i)\right)
\]

\[
\phi = \phi_r - \phi_i
\]

\[
\phi_{\text{pdf}} = \cos(\phi/2) / 4
\]

\[
\text{return } \theta_{\text{pdf}} \times \phi_{\text{pdf}}
\]

# compute the pdf of glint term

def compute_G_pdf(L, I):
    (theta_r, phi_r) = compute_angle(I)
    (theta_i, phi_i) = compute_angle(L)
    if(pi/2 - theta_i < epsilon):
        return 0
    a_TRT = arctan(((pi/2 + theta_r)/2 - alpha_TRT) / beta_TRT)
    b_TRT = arctan((-pi/2 + theta_r)/2 - alpha_TRT) / beta_TRT
    c_G = arctan((pi/2 - phi_g) / gamma_G)
    d_G = arctan(-phi_g / gamma_G)
    theta_h = (theta_i + theta_r) / 2
    t = theta_h - alpha_R
    theta_pdf = \frac{\beta_R}{(t^2 + \beta_R^2)} / \left(2(a_R - b_R) \cos(\theta_i)\right)
    phi = abs(phi_r - phi_i)
    p = phi - phi_g
    phi_pdf = \frac{\gamma_G}{(p^2 + \gamma_G^2)} / (2 * (c_G - d_G))
    \text{return } \theta_{\text{pdf}} \times \phi_{\text{pdf}}

def compute_pdf(L, I):
    pdf_R = compute_R_pdf(L, I)
    pdf_TT = compute_TT_pdf(L, I)
    pdf_TRT_G = compute_TRT_G_pdf(L, I)
    pdf_G = compute_G_pdf(L, I)
    \text{return } (pdf_R + pdf_TT + pdf_TRT_G + pdf_G) / 4

def sample_brdf(uv, I):

B.3 Pseudocode

```python
if uv[0] < 0.5 and uv[1] < 0.5:
    # Sample R lobe
    uv[0] = 2 * uv[0]
    uv[1] = 2 * vv[1]
    L = sample_R_lobe(uv, I)

elif u >= 0.5 and v < 0.5:
    # Sample TT lobe
    uv[0] = 2 * (1 - uv[0])
    uv[1] = 2 * uv[1]
    L = sample_TT_lobe(uv, I)

elif u < 0.5 and v >= 0.5:
    # Sample TRT-G lobe
    uv[0] = 2 * uv[0]
    uv[1] = 2 * (1 - uv[1])
    L = sample_TRT_G_lobe(uv, I)

else:
    # Sample glint lobe
    uv[0] = 2 * (1 - uv[0])
    uv[1] = 2 * (1 - uv[1])
    L = sample_G_lobe(uv, I)

pdf = compute_pdf(L, I)
return (L, pdf)
```
Appendix C

Pseudocode for *Out-of-core* BVH Construction

C.1  Pseudocode for Bricking Algorithm

```
Algorithm C.1: Bricking

class Stat:  #subtree statistic
  verts = Set()  # to track unique vertices
  tris = 0
  nodes = 0

def writeBrick(node):
  brickID = serializeToDisk(node)
  # make a new brick node as place holder
  placeHolder = makePlaceHolder(brickID, node.lod)
  deleteTree(node)  # delete subtree
  return placeHolder

def postOrderWalk(node):
  if isLeaf(node):
    for v in vertice_of(node):
      verts.insert(v)
```
C.2 Pseudocode for Bucketing Algorithm

```python
# one leaf node with vertices and triangles
return Stat(verts, node.count, 1)

elif isPlaceHolder(node):
    # one node, no triangle and no vertices
    return Stat(None, 0, 1)

elif isBranch(node):
    l = postOrderWalk(node.left)
    r = postOrderWalk(node.right)
    vert = l.vert + r.vert  # merge vertices
    tris = l.tris + r.tris
    nodes = l.nodes + r.nodes
    if (len(verts)>256 or faces>256 or nodes>256):
        node.left = writeBrick(node.left);
        node.right = writeBrick(node.right);
        # three nodes, no triangles and no vertices
        return Stat(None, 0, 3)
    else:
        # return the union of two subtree
        return Stat(vert, tris, nodes)

def bricking(root, tris, verts):
    postOrderWalk(root);
    return writeBrick(root);
```

Algorithm C.2: Bucketing

```python
def bucketSurface(grid, surf):
    h = {}  # hash map for local buckets
    for t in surf.mesh:  # triangles in the mesh
        tt = transform(t, surf.xform)
        bbox = getBoundingBox(tt)
        gi = getGridIndex(tt, grid)  # gi is a 3D index
        h[gi].count += 1
        h[gi].bound = union(h[gi].bound, bbox)
    return h

def bucketing(Scene):
```
C.3 Pseudocode for Chunking Algorithm

```python
wbox = computeWorldBound(Scene)
grid = initGrid(wbox)
H = {}  #global hash table for buckets
for surf in Scene:
    h = bucketSurface(grid, surf)
    for k in h.keys():
        H[k].count += h[k].count
        H[k].bound = union(H[k].bound, h[k].bound)
return list(H)
```

Algorithm C.3: Chunking Algorithm

```python
def locate(node, p):
    if isLeaf(node): return node.index
    else:
        if onLeft(p, node.split):
            return locate(node.left, p)
        else:
            return locate(node.right, p)

def splitSurface(chunks, node, surf):
    h = {}  #hash table for triangles in each chunk
    for t in surf.mesh:
        tt = transform(t, surf.xform)
P = getCenter(tt)
        index = locate(node, P)  #find chunk
        h[index].append(tt)
    for k in h.keys():
        s = makeSurface(h[k])  #make flattened surface
        chunks[k].append(s)

def distribute(chunks, node, surf, bbox):
    if isLeaf(node):  #insert surface to chunk
        s = transform(surf.mesh, surf.xform)
        chunks[node.index].append(s)
    else:
        if onLeft(bbox, node.split):
```
C.3 Pseudocode for Chunking Algorithm

```python
    distribute(chunks, node.left, surf, bbox)
    elif onRight(bbox, node.split):
        distribute(chunks, node.right, surf, bbox)
    else:
        splitSurface(chunks, node, surf)
        delete(surf)

    def chunking(buckets):
        root, chunks = buildKDTree(buckets)
        for surf in Scene:
            bbox = getBoundingBox(surf)
            distribute(chunks, root, surf, bbox)
```
Bibliography


massive model rendering system using geometric and image-based acceleration. In Proceedings of the 1999 symposium on Interactive 3D graphics 

[BBS’09] BUDGE B., BERNAVIN T., STUART J. A., SENGUPTA S., JOY K. I., OWENS J. D.: 


[BSGM02] BAXTER III W. V., SUD A., GOVINDARAJU N. K., MANOCHA D.: Gigawalk: 
interactive walkthrough of complex environments. In Proceedings of the 13th Eurographics workshop on Rendering (Aire-la-Ville, Switzerland, 


2009 symposium on Interactive 3D graphics and games (New York, NY, USA, 2009), I3D ’09, ACM, pp. 15–22.


BIBLIOGRAPHY


[Hav00] **Havran V.** *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.


BIBLIOGRAPHY

(Toronto, Ont., Canada, Canada, 2010), GI '10, Canadian Information Processing Society, pp. 145–152.


173


BIBLIOGRAPHY


BIBLIOGRAPHY


