MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

ADAPTIVE METHODS FOR PHOTOREALISTIC IMAGE SYNTHESIS

M. SENCER ÇAVUŞ

MASTER THESIS
Department of Computer Science and Engineering

Thesis Supervisor
Asst. Prof. Mehmet BARAN

ISTANBUL, 2020

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

ADAPTIVE METHODS FOR PHOTOREALISTIC IMAGE SYNTHESIS

M. SENCER ÇAVUŞ
(524116037)

MASTER THESIS
Department of Computer Science and Engineering

Thesis Supervisor
Asst. Prof. Mehmet BARAN

ISTANBUL, 2020

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

M. Sencer ÇAVUŞ, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended his thesis entitled "**Adaptive Methods for Photorealistic Image Synthesis**", on Feb 03, 2020 and has been found to be satisfactory by the jury members.

Jury Members

Assist.Prof. Mehmet K. BARAN (Advisor)
Marmara University, Department of Computer Engineering .....

Prof.Dr. Çiğdem EROĞLU ERDEM (Jury Member)
Marmara University, Department of Computer Engineering .....

Assist.Prof. S. Murat YEŞİLOĞLU (Jury Member)
Istanbul Technical University, Department of Control and Automation Engineering.....

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that M. Sencer ÇAVUŞ be granted the degree of Master of Science in Department of Computer Engineering, Computer Engineering Program, on ...19.02.2020... (Resolution no: 2020/07-02.....).

Director of the Institute
Prof. Dr. BÜLENT EKİCİ

ACKNOWLEDGEMENTS

I would like to thank my parents for their invaluable support, and my grandparents, the most selfless people I have ever known, for their unconditional love and for everything they have done for me, my debts to them cannot be repaid. I would also like to express my sincere gratitude to my advisor Asst. Prof. Mehmet Baran for letting me work on a research topic I am interested in, for always encouraging me, and for being a genuine and friendly person.

TABLE OF CONTENTS

ÖZET

FOTOGERÇEKÇİ İMAJ SENTEZLEME İÇİN UYARLANABİLİR YÖNTEMLER

Fotogerçekçi imaj sentezlemede kullanılan ışık yayılımı simülasyonu algoritmalarında verimli örnekleme çözülememiş bir problemdir. Bu tezde öncelikle ışık yayılımı simülasyonunun temellerini ve verimli örneklemeye engel olan zorlukları açıklıyoruz. Bu zorluklara çare olmaya aday olan bir yaklaşım olan uyarlanabilir önem örnekleme üzerinde duruyoruz. Bunun için, daha önce yayınlanmış, uyarlanabilir önem örneklemeyi mümkün kılan ışık yolu yöneltme metodlarını inceliyoruz. Bu yöneltme metodlarının çoğunun sadece sahne yüzeylerindeki noktalara gelen radyansı yaklaşık olarak değerlendirdiği ve bu yaklaşık değere göre örnekler oluşturduğunu belirtiyoruz. Bu yaklaşım, ışığı her yöne dağıtan yüzeyler için göreceli olarak iyi çalışsa da, bu yolla oluşturulan örnekler parlak yüzeyler için işe yaramaz hale gelebiliyor, çünkü parlak materyaller bakış açısına bağlı olduğundan, gelen ışığın sadece belli bir kısmı yansıyor. Bu tezde, daha önce yayınlanmış ağaç tabanlı bir yöneltme metoduna, bakış açısına bağlı ışık yayılımını hesaba katması için yapılan basit bir ilaveyi sunuyoruz. Önceki metodda olduğu gibi, sahne uzayı uyarlamalı olarak bölgelere bölünüyor. Her bölgeye, gelen radyansı kaydeden bir dörtlü ağaç iliştiriliyor. Bizim metodumuzda bu dörtlü ağaçlar sadece ışığı her yöne dağıtan yüzeyler için kullanılıyor. Ek olarak, her bölgeye, bakış yönlerinin uzayını bölen, ve gelen radyansı kaydeden dörtlü ağaçlar içeren bir dörtlü ağaç iliştiriyoruz. Bu dörtlü ağaçlar da parlak yüzeyler için kullanılıyor. Yaptığımız ilaveyle, çok sayıda parlak yüzey bulunduran sahnelerde iyileşmeler ve bakış açısına bağlı ışık yayılımı bulundurmayan sahnelerde asgari düzeyde ek hafıza kullanımı gözlemliyoruz.

ABSTRACT

ADAPTIVE METHODS FOR PHOTOREALISTIC IMAGE SYNTHESIS

Efficient sampling in light transport simulation algorithms used for photorealistic image synthesis remains an unsolved problem. In this thesis, we first explain the basics of light transport simulation and main challenges that prevent efficient sampling. We emphasize adaptive importance sampling, which is a promising approach to overcome these challenges. Thus, we review previously published light path guiding methods that enable adaptive importance sampling. We state that most of these guiding methods only approximate incident radiance at points of scene's surfaces and generate samples according to this approximation. This approach works relatively well for diffuse surfaces. However, samples generated in this way at points on glossy surfaces may become invalid, since only a subset of incident light is reflected as glossy materials are view-dependent. In this thesis, we propose a simple extension to a previous tree-based guiding method to make it aware of view-dependent light transport. As in the previous work, spatial domain of the scene is adaptively partitioned into regions. To each of these regions, a quadtree that records incident radiance is attached. In our method these quadtrees are used only for diffuse surfaces. We attach to each spatial region an additional quadtree that partitions the space of view directions and stores quadtrees that record incident radiance. These quadtrees are then used for glossy surfaces. With our extension, we observe improvements in scenes with many glossy surfaces and minimal memory overhead in scenes without view-dependent light transport.

ABBREVIATIONS

AD	: Automatic Differentiation
BDPT	: Bidirectional Path Tracing
BSDF	: Bidirectional Scattering Distribution Function
EM	: Expectation-Maximization
GMM	: Gaussian Mixture Model
HMC	: Hamiltonian Monte Carlo
LTE	: Light Transport Equation
MCMC	: Markov Chain Monte Carlo
MIS	: Multiple Importance Sampling
MLT	: Metropolis Light Transport
NDF	: Normal Distribution Function
NEE	: Next Event Estimation
PDF	: Probability Distribution Function
PPG	: Practical Path Guiding
PSSMLT	: Primary Sample Space Metropolis Light Transport
RMSE	: Root Mean Square Error
UDPT	: Unidirectional Path Tracing

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Photorealistic image synthesis or rendering is the process of simulating light transport in space, which aims to create lifelike images using data that describe a three dimensional virtual scene containing objects or media, namely polygons, materials, textures, light sources, by obeying the laws of physics. It is widely used in visual effects for feature films, architecture and engineering.

One of the practical problems of photorealistic rendering is that it takes too much time to render even a single frame. This is because many samples are required to produce a noise-free image due to the stochastic nature of physically based rendering algorithms. Using better hardware is not a solution as the scene complexity also increases to push the boundaries of realism and the hardware is generally used to its limit. While GPU acceleration can certainly improve the render times for some workloads, the high memory demand of the scenes created, for instance in film production, do not allow them to be used in every case. The production quality final renders are done using CPUs, whereas GPUs are tend to be used in less detailed interactive preview renders where artists have an approximation of how the final image will look in the end.

Hence we try to devise more efficient and robust rendering methods. One way to achieve efficiency is to implement existing methods in a way that is more conformant to the target hardware. Such as vectorizing the existing methods to utilize SIMD instructions or making it more GPU-friendly. Another way is to develop more robust numerical methods by studying the theory behind rendering. Adaptive rendering methods try to achieve this. This is a good opportunity to apply statistical learning, which gained traction in recent years, since adaptive methods produce better results by utilizing given input data or the data generated during runtime. Adaptive rendering methods generally fall into one of the two main categories; (1) adaptive reconstruction and filtering, where the aim is to reconstruct a clean image from a noisy rendered image by adaptively assigning weights to image filters, and (2) adaptive importance sampling or path guiding methods, where the aim is to improve sample quality by constructing distribution functions adaptive to the

given scene, learning from the data generated during rendering. Our treatise will be of the second category.

Remaining part of this thesis is organized as follows: In Chapter 2 we first give a theoretical background of physically based rendering, defining basic units and quantities, introducing governing equations and the computational problem of light transport simulation. We then present basic numerical methods and algorithms used to solve this problem. These are not given in-depth but just enough to appreciate the main ideas and notions behind our work. After the basics, we touch upon some improved and more advanced rendering methods. Finally we give a brief survey of path guiding methods which constitute the main topic of this thesis, and explain our contribution. In Chapter 3 we share the results of our method as to how it performs in variety of cases. And in Chapter 4 we give our concluding remarks.

2. MATERIAL AND METHOD

2.1 Elements of Light Transport Theory

2.1.1 Radiometry

Before starting to talk about light transport simulation, we must first define some physical quantities about light. We shall briefly explain the radiometric quantities.

2.1.1.1 Radiant Flux Φ

Radiant flux is the most basic radiometric quantity. It is the radiant energy per unit time.

$$\Phi = \frac{\partial Q}{\partial t} \tag{2.1}$$

where Q denotes energy and t the time. It is measured in watts ($W = J/s$).

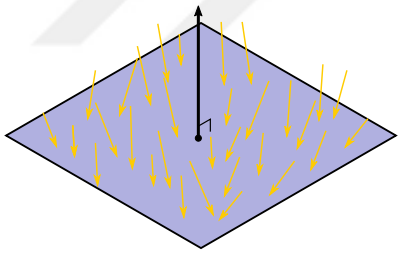


Figure 2.1: Radiant flux.

2.1.1.2 Irradiance E

Irradiance is the light flux coming from all directions per unit surface area.

$$E = \frac{\partial \Phi}{\partial A} \tag{2.2}$$

where A is the surface area. It is measured in watts per square meter ($W \times m^{-2}$).

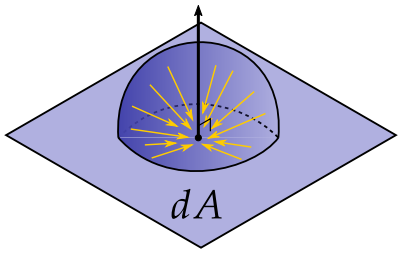


Figure 2.2: Irradiance.

2.1.1.3 Radiance L

Radiance is the most important quantity in our case. It is the light flux per unit surface area per direction.

$$L = \frac{\partial^2 \Phi}{\partial \omega \partial A^{\perp}} \tag{2.3}$$

where, ω is the solid angle and $A^{\perp}$ is projected area defined as $A^{\perp} = Acos\theta$, θ being the angle between surface normal and direction ω. Its unit of measure is watts per steradian per square meter ($W \times sr^{-1} \times m^{-2}$).

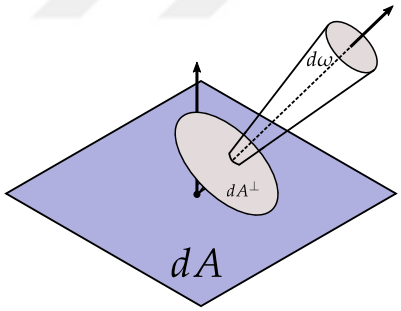


Figure 2.3: Radiance.

Notice that one can derive other radiometric quantities from radiance simply by integrating over a respected domain. For example, integrating over hemisphere of directions, we arrive at irradiance as shown below:

$$E = \frac{\partial \Phi}{\partial A} = \int_{\mathcal{H}^2} \frac{\partial^2 \Phi}{\partial \omega \partial A^{\perp}} cos\theta d\omega.$$

2.1.2 The Light Transport Equation

Central to the photorealistic rendering is the light transport equation (LTE). There are two formulations of LTE, which we will use throughout this thesis. First of them, the solid angle formulation due to Kajiya [8], is a recursive integral equation

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\mathcal{H}^2(\mathbf{x})} f_s(\mathbf{x}, \omega_i \rightarrow \omega_o) L_i(\mathbf{x}, \omega_i) \cos\theta_i d\omega_i, \tag{2.4}$$

evaluated with differential solid angles over the hemisphere $\mathcal{H}^2$ at point $\mathbf{x}$. Here L_o denotes the total radiance leaving a point $\mathbf{x}$ on a surface of an object through direction ω_o, L_e the emitted radiance at $\mathbf{x}$ through ω_o, L_i the incident radiance at $\mathbf{x}$ coming from ω_i, f_s the bidirectional scattering distribution function (BSDF), informally it describes how much of the incoming light from ω_i is scattered through ω_o, and θ_i is the angle between ω_i and the surface normal at $\mathbf{x}$.

Second formulation, namely the path integral formulation due to Veach [23], is a Lebesgue integral based on differential areas on scene's surfaces, and integration is on the measure space $(\mathcal{P}, \mu)$ of paths which is the disjoint union of spaces of paths of all lengths:

$$\mathcal{P} = \bigsqcup_{k=2}^{\infty} \mathcal{P}^k. \tag{2.5}$$

Let j be the pixel index on the sensor, we write the LTE describing the total light arriving at j-th pixel of the sensor as

$$I^j = \sum_{k=2}^{\infty} \int_{\mathcal{P}^k} f(\bar{x}) d\mu(\bar{x}), \tag{2.6}$$

where $f : \mathcal{P}^k \rightarrow R_{\geq 0}$, the measurement contribution function denotes the light energy carried by path $\bar{x}$.

Since $\mathcal{P}$ consists of disjoint subsets, we separately evaluate the integral in subset domains $\mathcal{P}^k \subset \mathcal{P}\ \forall k \geq 2$ [1] and take the sum of the results.

f consists of other terms, namely the emitted radiance L_e, the BSDF f_s and the

[1] In practice k is bounded by a maximum value, as longer paths carry less energy and their contributions become negligible.

geometry term G:

$$f(\bar{x}) = f(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k) = L_e(\mathbf{x}_0 \to \mathbf{x}_1) \prod_{l=1}^{k} f_s(\mathbf{x}_{l-1} \to \mathbf{x}_l \to \mathbf{x}_{l+1}) G(\mathbf{x}_{l-1} \leftrightarrow \mathbf{x}_l). \tag{2.7}$$

Here, each path vertex $\mathbf{x}_l$ is an element of a set $\mathcal{M}$ called the scene manifold (i.e. $\mathbf{x}_l \in \mathcal{M}$), which is the union of all surfaces in a scene, so each path vertex is a point on a surface in the scene. This function can be visualized as in Fig. 2.4, which illustrates the light transport originating from point $\mathbf{x}_0$ on a light source, reflecting off of $\mathbf{x}_1$, and arriving at $\mathbf{x}_2$. The product of terms f_s and G

$$\mathfrak{f}(\bar{x}) = \prod_{l=1}^{k} f_s(\mathbf{x}_{l-1} \to \mathbf{x}_l \to \mathbf{x}_{l+1}) G(\mathbf{x}_{l-1} \leftrightarrow \mathbf{x}_l)$$

is called the path throughput.

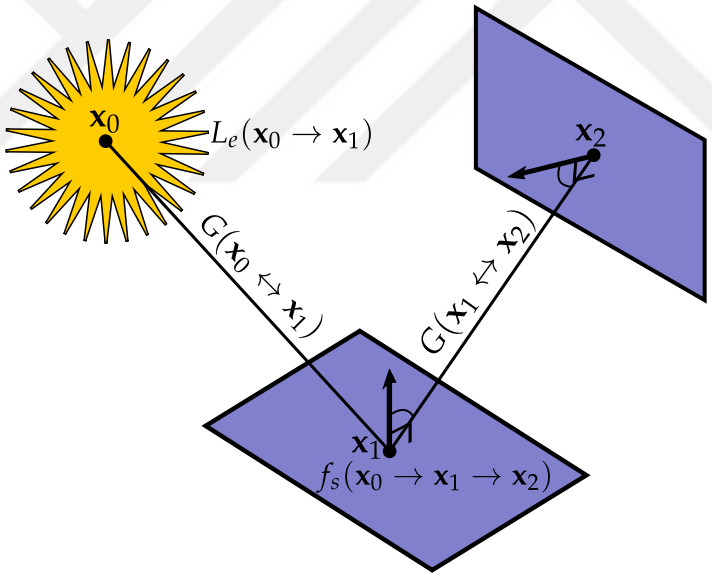


Figure 2.4: Visualization of measurement contribution function.

We will now explain the terms in LTE.

2.1.2.1 Emitted Radiance L_e

$L_e : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$ is a function that returns the radiance emitted from a point $\mathbf{x}_e$ on the emitter to another point $\mathbf{x}$ on a surface in the scene. For most basic area light sources,

this is a constant function, though more complex light sources also exist where it is not constant.

2.1.2.2 Bidirectional Scattering Distribution Function (BSDF) f_s

BSDF gives the material properties to the objects in the scene. It describes the way light rays reflect or refract at a surface. This effects the look of that surface. For instance, the surface may look rough like wood, or it may have a glossy, metallic look.

While the materials in the physical world can be multilayered and arbitrarily complex, single layered materials cover a lot of cases and can be described with varying degrees of roughness. On the one hand there are extremely rough, diffuse surfaces, usually modelled by the Lambertian BSDF. A diffuse surface can reflect an incoming light ray in any direction of the hemisphere. On the other hand there is the perfectly specular surfaces such as mirrors or glass surfaces. A specular surface just reflects or refracts the incoming ray along the surface normal in only one direction.

In between these two extremes, there are glossy surfaces. These kind of surfaces can be approximated with the help of microfacet theory. Microfacet theory assumes a microsurface underlying a seemingly flat macrosurface (see Fig. 2.5). This microsurface is assumed to have many small specular mirror-like facets that reflects the light along their normals. The distribution of these normals, along with a shadowing-masking function which we will explain later, controls the roughness of the material.

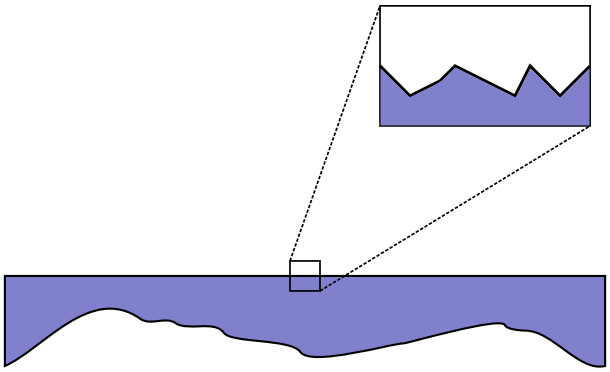


Figure 2.5: Microfacets.

One of the most frequently used BSDF models that is based on the microfacet theory

is the Cook-Torrance model [1]. The Cook-Torrance model for reflections is written as

$$f_r(\mathbf{i}, \mathbf{o}, \mathbf{n}) = \frac{F(\mathbf{i}, \mathbf{h})G(\mathbf{i}, \mathbf{o}, \mathbf{h})D(\mathbf{h})}{4|\mathbf{i} \cdot \mathbf{n}||\mathbf{o} \cdot \mathbf{n}|}. \tag{2.8}$$

Here, F is the Fresnel function describing whether ω_o should reflect or refract, given ω_i. G is the shadowing-masking function, which accounts for the directions ω_o that may be blocked by microfacets. D is the microfacet distribution function, sometimes called the normal distribution function (NDF); given ω_i, it assigns ω_o a density value. Given a parameter that effectively describes the roughness of the surface, it models the distribution of microfacet normals. What this means is that some directions carry more energy than the others, and some directions may not be able carry any energy at all, because reflecting or refracting to that direction may be impossible, hence that direction is not in the support of D, namely $\omega_o \notin \operatorname{supp} D$.

We are free to choose between different distributions or shadowing-masking functions, each resulting in slightly different looks and we may also use different approximations to Fresnel function. Now we shall give some examples to these terms.

A fast approximation to Fresnel function [20] is used frequently:

$$F(\mathbf{i}, \mathbf{h}) = F_{\text{Schlick}}(\mathbf{i}, \mathbf{h}) = F_0 + (1 - F_0) \times (1 - (\mathbf{h} \cdot \mathbf{i}))^5 F_0 = \frac{(n_1 - n_2)^2}{(n_1 + n_2)^2}, \tag{2.9}$$

where n_1 and n_2 are indices of refraction of the media.

A popular choice for distribution function is the GGX distribution [29]. Let α be the width parameter controlling the roughness of the surface, θ_m be the angle between $\mathbf{h}$ and $\mathbf{n}$, θ_v between $\mathbf{v}$ and $\mathbf{n}$ and let

$$\chi^+(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases},$$

the GGX microfacet distribution function is given as

$$D(\mathbf{h}) = D_{\text{GGX}}(\mathbf{h}) = \frac{\alpha^2 \chi^+(\mathbf{h}, \mathbf{n})}{\pi \cos^4 \theta_m (\alpha^2 + \tan^2 \theta_m)^2}. \tag{2.10}$$

It is used together with the appropriate shadowing-masking function

$$G(\mathbf{i}, \mathbf{o}, \mathbf{h}) \approx G_1(\mathbf{i}, \mathbf{h}) G_1(\mathbf{o}, \mathbf{h}) \tag{2.11}$$

where

$$G_1(\mathbf{v}, \mathbf{h}) = \chi^+ \left(\frac{\mathbf{v} \cdot \mathbf{h}}{\mathbf{v} \cdot \mathbf{n}} \right) \frac{2}{1 + \sqrt{1 + a^2 \tan^2 \theta_v}}. \tag{2.12}$$

is derived from the microfacet distribution function.

2.1.2.3 Geometry Term G

$$G(\mathbf{x}_0, \mathbf{x}_1) = G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) = V(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \left| \frac{\cos \theta_0 \cos \theta_1}{r^2} \right| \tag{2.13}$$

Here, $V : \mathcal{M} \times \mathcal{M} \rightarrow \{0, 1\}$ is the binary valued visibility function which maps to 1 if the ray from $\mathbf{x}_0$ to $\mathbf{x}_1$ does not intersect any surface in the scene, otherwise it maps to 0. Other terms are illustrated in Fig. 2.6.

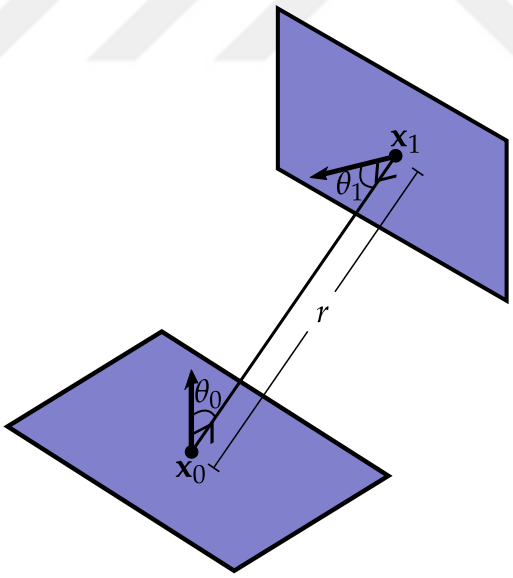


Figure 2.6: Visualization of geometry term.

2.2 Estimating the Light Transport Equation

LTE includes an integral term which cannot be computed analytically. Therefore we aim to compute it using a numerical algorithm. While quadrature based integration methods such as the trapezoidal rule or the Gaussian quadrature may work well for the integrals with one to three dimensional domains, they suffer from the curse of dimensionality in high dimensional domains. The domain of the light transport integral $\mathcal{P}$ is potentially an infinite dimensional space, even though in practice we do not let the paths bounce infinitely. Nevertheless since $\mathcal{P}^k \in \mathcal{P}$ where k may be a sufficiently big number, we cannot utilize the aforementioned integration methods. To cope with the curse of dimensionality, we use the well-known Monte Carlo integration [14]. Below we briefly explain the Monte Carlo integration and show direct applications of it to the light transport problem.

2.2.1 Monte Carlo Integration

Let $(\mathcal{X}, \mu)$ be a measure space. For the sake of simplicity let $\mathcal{X} \subset \mathbb{R}^d$ be an interval in d-dimensional euclidean space (i.e. $\mathcal{X} = [a, b] \subset \mathbb{R}$ if $d = 1$) and μ be the volume of that interval (i.e. $\mu(\mathcal{X}) = b - a$ if $d = 1$). Let

$$I = \int_{\mathcal{X}} f(x)dx$$

be the integral we want to evaluate. Given a set of N samples $\{x_1, \ldots, x_i, \ldots, x_N\}$ drawn uniformly from the integral domain the Monte Carlo estimator of this integral is written as

$$\langle I \rangle^N = \frac{\mu(\mathcal{X})}{N} \sum_{n=1}^{N} f(x_n). \tag{2.14}$$

with the property that $E\left[\langle I\rangle^N\right] = I$ which we can easily observe:

$$\begin{aligned} E\left[\langle I\rangle^N\right] &= E\left[\frac{1}{N}\sum_{n=1}^{N} f(x_n)\right] \\ &= \frac{\mu(\mathcal{X})}{N}\sum_{n=1}^{N} E\left[f(x_n)\right] \\ &= \frac{\mu(\mathcal{X})}{N}\sum_{n=1}^{N}\int_{\mathcal{X}} f(x_n)p(x_n)dx \\ &= \int_{\mathcal{X}} f(x_n)dx \\ &= I. \end{aligned}$$

Monte Carlo estimator also has another important property which tells us that, as long as we increase the number of samples N, the estimation draws closer to the integral value we want to compute, eventually converging in the limit:

$$\lim_{N\to\infty}\langle I\rangle^N = I.$$

Since the estimator is unbiased, the convergence rate of the estimator depends on the variance which decreases with the rate of $O(1/\sqrt{N})$.

2.2.2 Light Tracing

Light tracing is a direct application of Monte Carlo integration to the light transport problem. We simply write the estimator as

$$\langle I_j\rangle^N = \sum_{k=2}^{\infty}\frac{1}{N}\sum_{n=1}^{N} f(\bar{x}_n) \approx \sum_{k=2}^{\infty}\int_{\mathcal{P}^k} f(\bar{x})d\mu(\bar{x}).$$

Since the domain of the integrand is the path space $\mathcal{P}$ we need to sample paths in the form

$$\bar{x} = \mathbf{x}_0\mathbf{x}_1\ldots\mathbf{x}_k$$

where $\mathbf{x}_0$ is on a light source. We uniformly sample a point $\mathbf{x}_0$ and a direction ω_0 on a light source and trace a ray with origin $\mathbf{x}_0$ and direction ω_0 which hits a point on the scene manifold $\mathbf{x}_1 \in \mathcal{M}$. We again sample a direction $\omega_1 \in \mathcal{H}^2$ at $\mathbf{x}_1$ and evaluate G and f_s terms. If one of them results in zero value, we stop and start sampling a new path from

the light source. Otherwise we continue sampling directions and evaluating the terms.

However, this estimator is very inefficient in most cases as probability of connecting a path vertex to the camera is extremely low, hence we use a different method in practice, which we will introduce next.

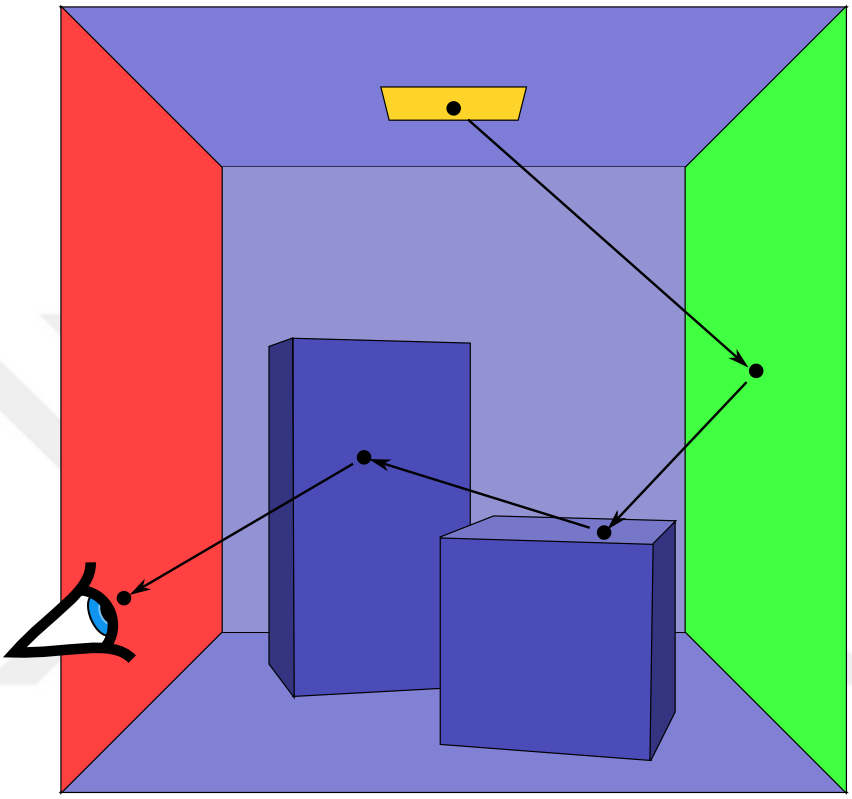


Figure 2.7: In light tracing, paths are traced from light sources to the camera.

2.2.3 Path Tracing

It was believed in ancient times that people see due to the rays coming out of their eyes. This is of course very far from the truth. But unidirectional path tracer (UDPT), or usually called simply as path tracing, works exactly like that, which is an interesting observation in itself. This is possible thanks to the terms in the throughput of LTE being symmetric functions for incoming and outgoing directions. Which means that $G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) = G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_0)$ and $f_s(\mathbf{x}_0 \rightarrow \mathbf{x}_1 \rightarrow \mathbf{x}_2) = f_s(\mathbf{x}_2 \rightarrow \mathbf{x}_1 \rightarrow \mathbf{x}_0)$. As a result, we can flip the directions between

path vertices as depicted in Fig. 2.8, constructing paths as

$$\bar{y} = \mathbf{x}_0\mathbf{x}_1 \dots \mathbf{x}_k$$

$$\bar{z} = \mathbf{x}_k\mathbf{x}_{k-1} \dots \mathbf{x}_0,$$

and get the same throughput value $\mathfrak{f}(\bar{y}) = \mathfrak{f}(\bar{z})$.

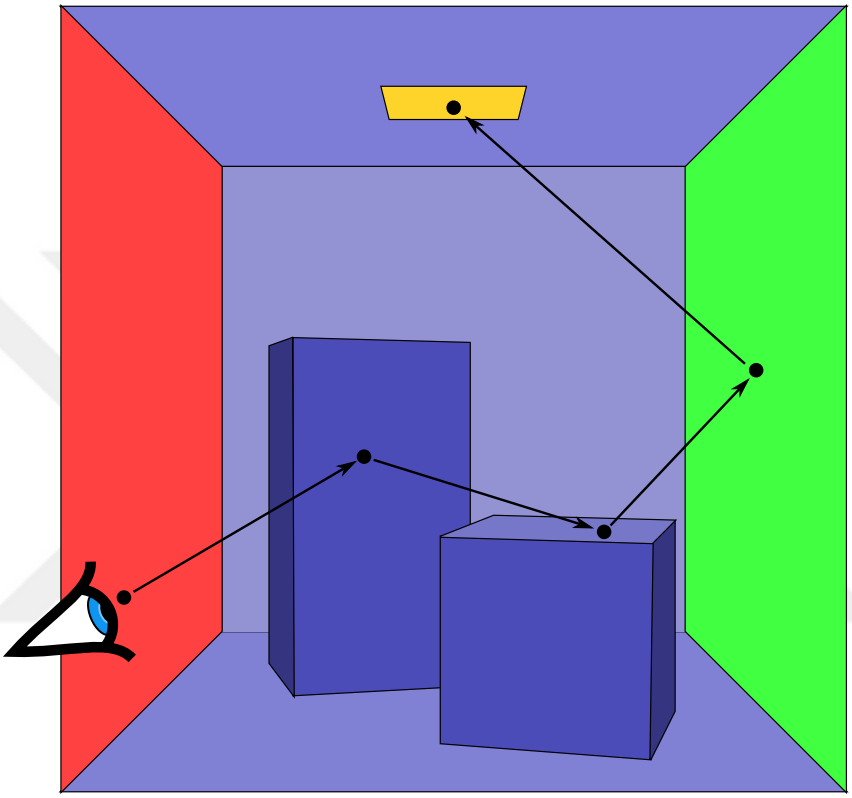
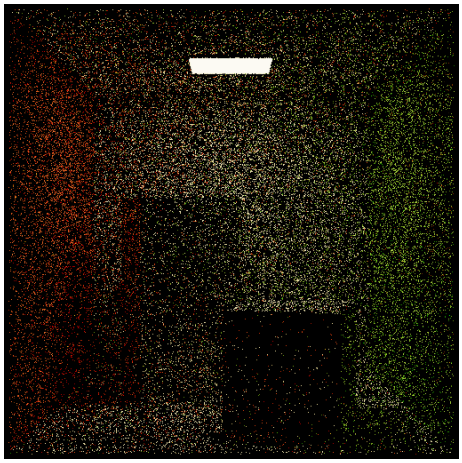
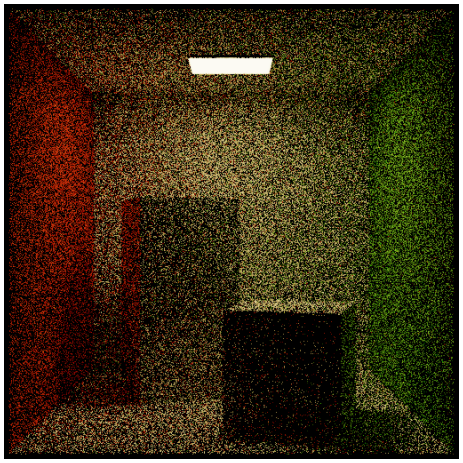


Figure 2.8: In path tracing, paths originate from the camera.

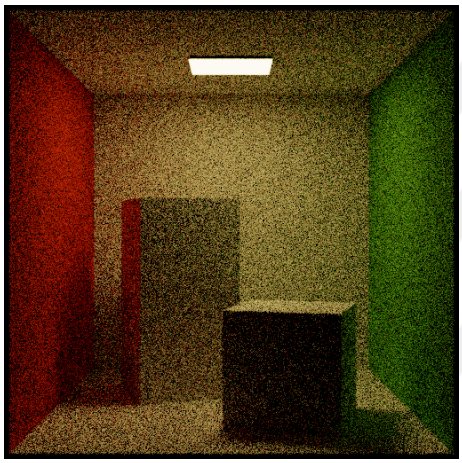
This new estimator we have by trivial modification of light tracing produces a cleaner image, as the probability of a path hitting a light source with finite surface area is much higher than hitting the camera. A pseudocode of path tracing is given in Algorithm 1. As can be seen in Fig. 2.9, by increasing the number of paths traced per pixel, we decrease the root mean square error (RMSE) of the rendered images and reduce the noise.



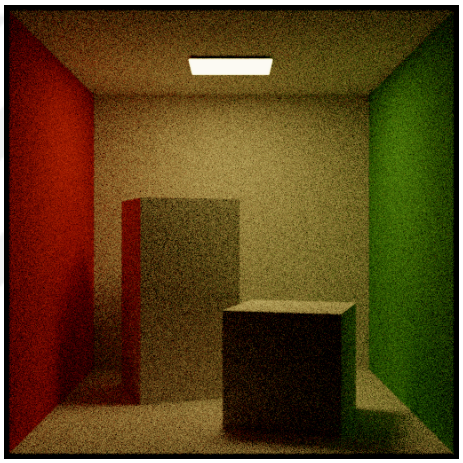
(a) $N = 8$, RMSE 0.476



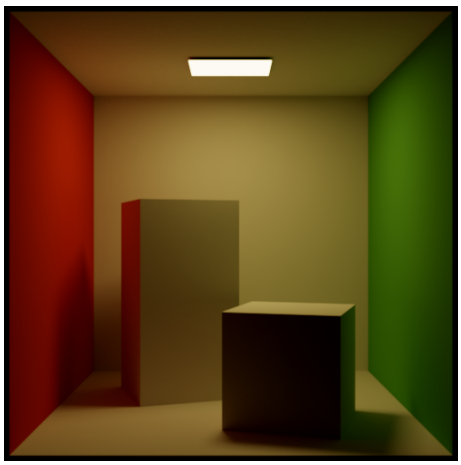
(b) $N = 32$, RMSE 0.383



(c) $N = 128$, RMSE 0.220



(d) $N = 512$, RMSE 0.108



(e) Reference

Figure 2.9: A scene rendered using UDPT. Increasing the number of samples per pixel N reduces the error and produces cleaner images.

Algorithm 1 Path Tracing

for all $j \in \{1, \ldots, J\}$ **do**
  for all $n \in \{1, \ldots, N\}$ **do**
    $\mathbf{p} \leftarrow \textsc{SamplePointInsidePixel}(j)$
    $\mathbf{x}' \leftarrow \textsc{TraceRay}((\mathbf{x}, \omega_i))$
    $k \leftarrow 0$
    while $(L_e(\mathbf{x}' \rightarrow \mathbf{x}) = 0) \wedge (k < K)$ **do**
      $\mathbf{x} \leftarrow \mathbf{x}'$
      $\omega_o \leftarrow -\omega_i$
      $\omega_i \sim S_{\mathcal{H}^2}$
      $\mathfrak{f} \leftarrow \mathfrak{f} \times f_s(\mathbf{x}, \omega_i \rightarrow \omega_o) \times |\mathbf{n}(\mathbf{x}) \cdot \omega_i|$
      $x' \leftarrow \textsc{TraceRay}((x, \omega_i))$
      $k \leftarrow k + 1$
    end while
    $L_j \leftarrow L_j + L_e(\mathbf{x}' \rightarrow \mathbf{x}) \times \mathfrak{f}$
  end for
  $\langle I_j \rangle^N \leftarrow \frac{L_j}{N}$
end for

2.2.3.1 Next Event Estimation

In the basic version of UDPT, we only sample random directions and hope that following those directions brings paths to light sources. While hitting a light source has a much higher probability than hitting the camera, we can still do much better by utilizing the light source information in the scene.

In next event estimation (NEE) we utilize this information in path sampling process as follows: At each path vertex $\mathbf{x}_n$ we randomly sample a point $\mathbf{p}$ on a light source. We then trace a ray from $\mathbf{x}_n$ to $\mathbf{p}$. This constructs a path of length $n+1$. Instead of blindly sampling a direction and hoping that it hits a light source, we exploit the fact that we know where all the light sources are and at each step try to explicitly connect the paths to them. This is shown in Fig. 2.10.

As a result, this considerably shrinks the search space of sampling non-zero valued paths, even though it is not guaranteed, as the ray $\mathbf{x}_n \rightarrow \mathbf{p}$ may be occluded or $\mathbf{x}_{n-1} \rightarrow \mathbf{x}_n \rightarrow \mathbf{p}$ may not be in the support of f_s causing the contribution to be zero.

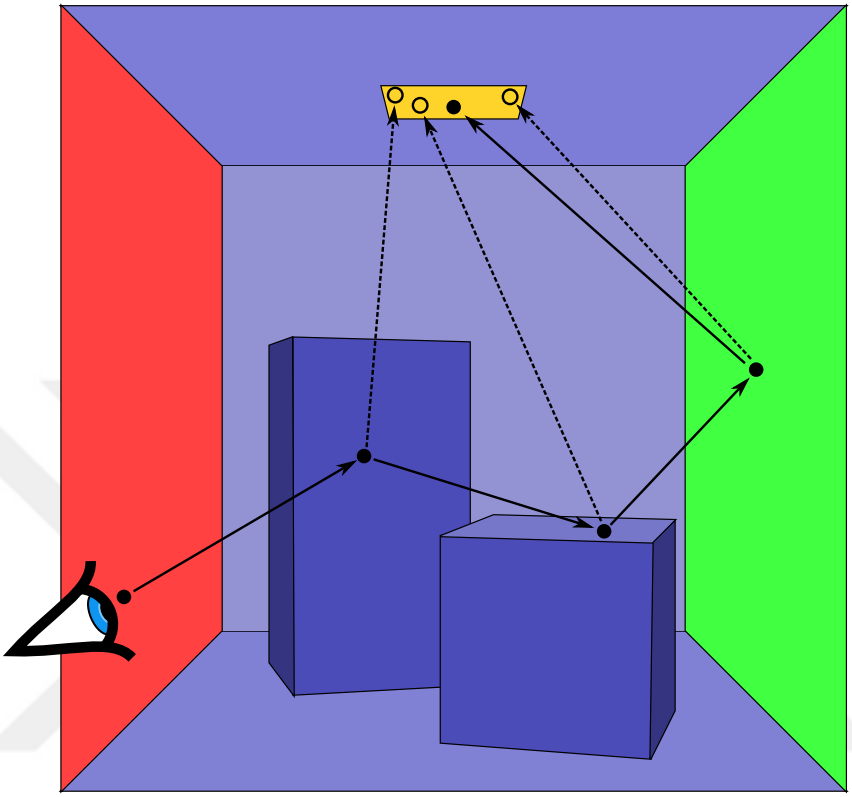
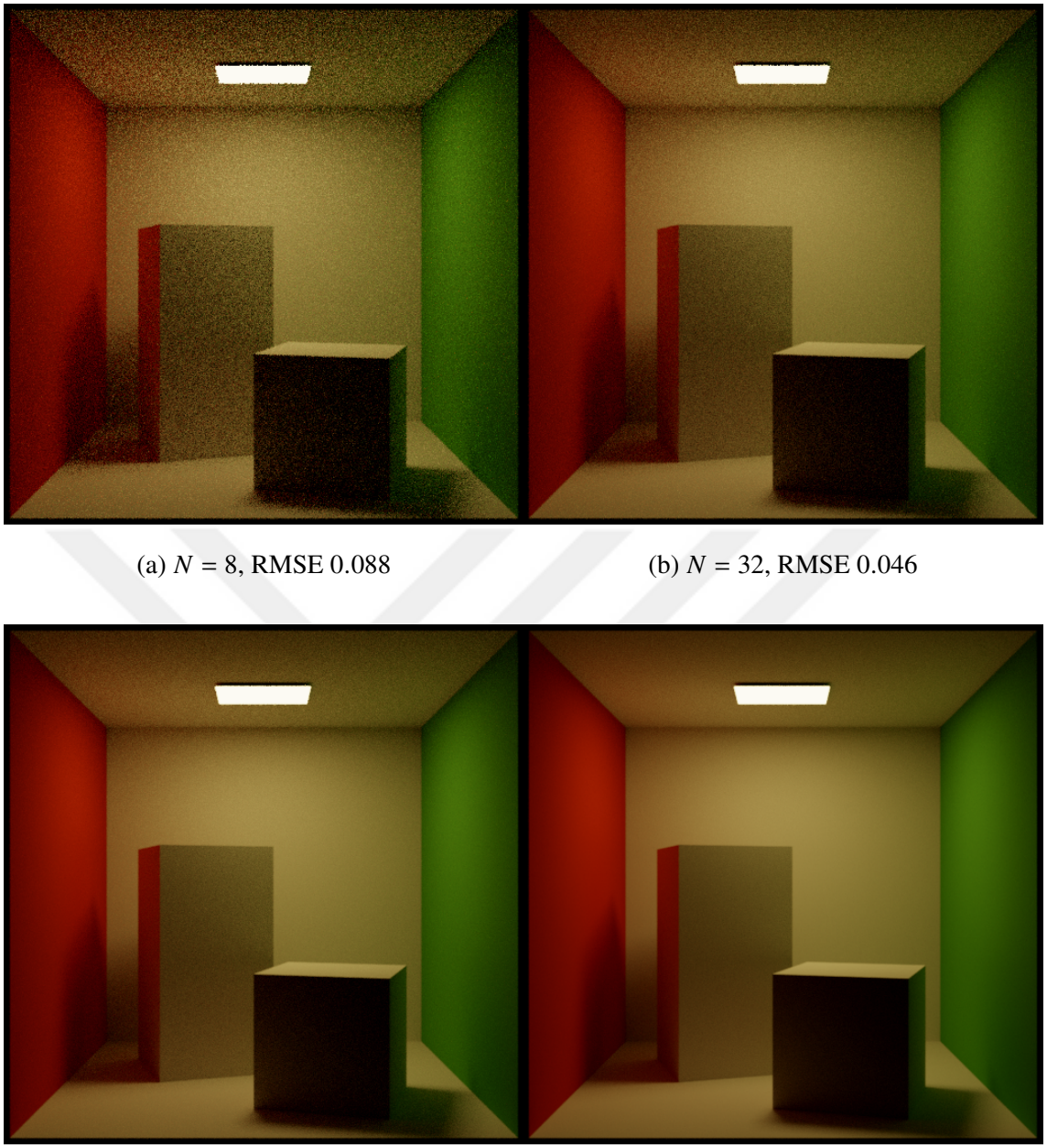


Figure 2.10: UDPT with next event estimation. Dashed lines show explicit connections to the light source.



(a) $N = 8$, RMSE 0.088

(b) $N = 32$, RMSE 0.046

(c) $N = 128$, RMSE 0.023

(d) $N = 512$, RMSE 0.011

Figure 2.11: A scene rendered using UDPT with NEE.

2.3 Improved Sampling for Monte Carlo Integration and Variance Reduction

While Monte Carlo integration is a generally applicable and a simple to implement numerical integration technique that gives unbiased estimations, it suffers from slow convergence due to the slow decrease rate of variance with respect to the number of samples. In the context of rendering, the estimates we get are noisy unless we use very high number of samples. In this section we review some of the techniques that can improve the sampling quality such that the variance reduces faster and the estimates become cleaner with relatively less number of samples.

2.3.1 Importance Sampling

The standard Monte Carlo estimator given in Eq. 2.14 works with samples taken from the uniform distribution. But we need not sample strictly from the uniform distribution, we can sample from an arbitrary PDF, the estimator then becomes

$$\langle I \rangle^N = \frac{1}{N} \sum_{n=1}^{N} \frac{f(x_n)}{p(x_n)}. \tag{2.15}$$

and its expected value again equals to the original integral:

$$\begin{aligned} E\left[\langle I \rangle^N\right] &= E\left[\frac{1}{N} \sum_{n=1}^{N} \frac{f(x_n)}{p(x_n)}\right] \\ &= \frac{1}{N} \sum_{n=1}^{N} E\left[\frac{f(x_n)}{p(x_n)}\right] \\ &= \frac{1}{N} \sum_{n=1}^{N} \int_{\mathcal{X}} \frac{f(x_n)}{p(x_n)} p(x_n) dx \\ &= \int_{\mathcal{X}} f(x_n) dx \\ &= I. \end{aligned}$$

This is essentially the same as the standard estimator, when p is chosen to be a uniform distribution, it becomes a constant and can be taken outside of the integral as it was done in Eq. 2.14, where $p(x) = 1/\mu(\mathcal{X})$.

Notice that if we sample from a hypothetical PDF perfectly proportional to the integrand

$$p(x) = \frac{f(x)}{\int_{\mathcal{X}} f(x')dx'} = \frac{f(x)}{I},$$

and substitute it into Eq. 2.15

$$\begin{aligned}
\langle I \rangle^N &= \frac{1}{N} \sum_{n=1}^{N} \frac{f(x_n)}{\frac{f(x_n)}{I}} \\
\langle I \rangle^N &= \frac{1}{N} \sum_{n=1}^{N} I \\
\langle I \rangle^N &= I,
\end{aligned}$$

the estimator becomes equal to the value of the integral and has zero variance. This implies that we could have estimated the integral without error with a single sample, i.e. $N = 1$, if we had access to an ideal PDF. Although this is generally impossible in practice, we still strive for PDFs with shapes as close to the shape of the integrand as possible, for such PDFs increase the convergence rate of the estimator, with its variance decreasing faster than $O(1/\sqrt{N})$ in comparison to uniform-only sampling.

2.3.1.1 Path Tracing with Importance Sampling

It is impossible to derive a PDF that is perfectly proportional to the LTE integrand. Therefore we try to sample from PDFs that are as proportional as possible to the integrand. To this end, PDFs proportional to BSDFs are usually used. While uniform sampling from the hemisphere is proportional to the Lambertian BSDF, it is not for other view-dependent microfacet-based BSDFs. Nevertheless, it is generally possible to derive PDFs that have shapes similar to microfacet-based BSDFs. For instance, a PDF used when importance sampling from the BSDF (reflection only) with GGX microfacet distribution from §2.1.2.2 is given by

$$p(\mathbf{m}) = \frac{D(\mathbf{m})|\mathbf{m} \cdot \mathbf{n}|}{4|\mathbf{o} \cdot \mathbf{h}|}. \tag{2.16}$$

BSDF importance sampling lowers the variance, and reduces noise as a result, in comparison to uniform sampling.

2.3.2 Multiple Importance Sampling

Importance sampling allows us to sample from a single PDF that is approximately proportional to the integrand. This PDF may have a shape similar to that of integrand in a subset of the integral domain, but other alternative PDFs, each being proportional to the integrand in different subsets of the domain, may also exist. The question then arises: How can we combine multiple estimators that use different PDFs into a superior estimator in an unbiased manner? Multiple importance sampling (MIS) [26] is a solution to this problem.

We shall intuitively explain the technique by giving a simple example: Let $p_1 \propto f_1$ and $p_2 \propto f_2$ be probability distributions with arbitrary domain $\mathcal{X}$ and proportional to f_1 and f_2 respectively and let $x \in \mathcal{X}$ be sampled from $p_1, x \sim p_1$. But we could have sampled x from p_2, and that might have resulted in lower variance, when $p_2(x) > p_1(x)$. What we want to achieve is to weight $\frac{f_1(x)f_2(x)}{p_1(x)}$ with w_1 and $\frac{f_1(x)f_2(x)}{p_2(x)}$ with w_2 that results in effect as if x was sampled from an ideal hypothetical distribution $p^* \propto f_1(x)f_2(x)$:

$$w_1 \frac{f_1(x)f_2(x)}{p_1(x)} + w_2 \frac{f_1(x)f_2(x)}{p_2(x)} \approx \frac{f_1(x)f_2(x)}{p^*(x)}. \tag{2.17}$$

What MIS does is to determine these weights w_1 and w_2 through heuristic functions, such that $w_1 + w_2 = 1$ and $w_2 > w_1$ whenever $p_2 > p_1$ and vice versa.

There are several heuristics devised to assign weights. Two of them are most frequently used; (1) the balance heuristic

$$w_i(x) = \frac{N_i p_i(x)}{\sum_j N_j p_j(x)}, \tag{2.18}$$

and (2) the power heuristic

$$w_i(x) = \frac{(N_i p_i(x))^\beta}{\sum_j (N_j p_j(x))^\beta}. \tag{2.19}$$

These heuristics guarantee the unbiasedness of combined estimators. Proofs of unbiasedness are given in [23].

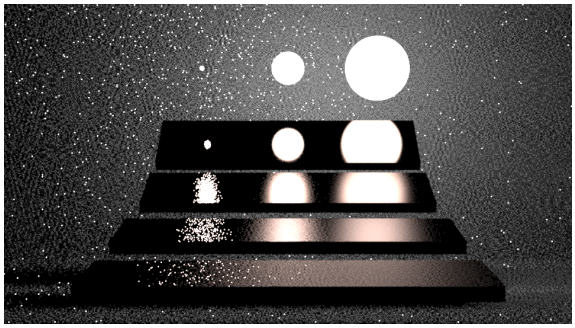
2.3.2.1 Next Event Estimation with MIS

Previously we have stated that NEE is one of the most basic methods to improve path tracing and it is usually implemented as an application of MIS. Assume that we are tracing a path in a scene with a single light source and the current path vertex is $\mathbf{x}$. Let $p_1 = p_{L_e}$ be the PDF used to sample the light source and let $p_2 = p_{f_s}$ be the PDF used to sample the BSDF at $\mathbf{x}$. In the version of NEE with MIS, we sample a direction ω as in the original version of NEE from p_1, but sampling ω from p_2 might have been more robust. Thus we have two estimators that are combined

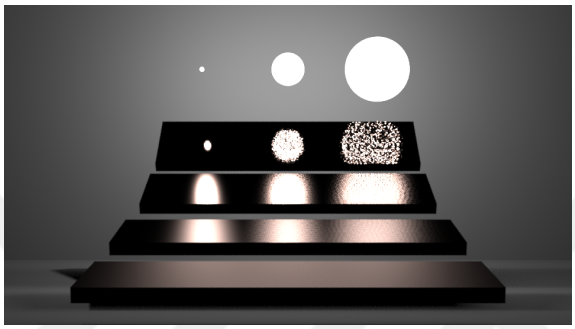
$$\langle I \rangle = w_1 \frac{L_e \cdot \mathfrak{f}}{p_1} + w_2 \frac{L_e \cdot \mathfrak{f}}{p_2}$$

by weighting the path throughput value $\mathfrak{f}$ with PDF values p_1, p_2 and their respective MIS weights, where w_i is the arbitrary MIS heuristic.

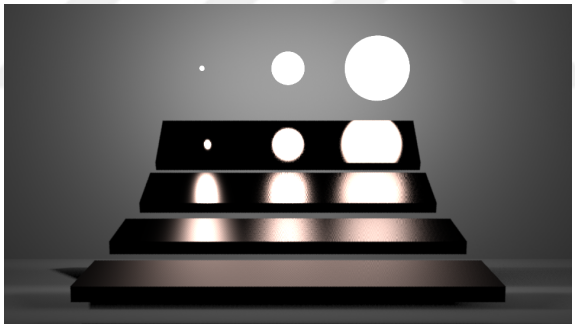
This is beneficial especially in cases such as; (1) the light source has a relatively large surface area, but the BSDF at $\mathbf{x}$ is highly glossy, where sampling according to BSDF may be more beneficial, and (2) the light source has a relatively small surface area, but the BSDF at $\mathbf{x}$ is very smooth, where sampling the light source may be more beneficial. MIS heuristics then assign a bigger weight, in case (1) to p_2, and in case (2) to p_1. This is demonstrated in Fig. 2.12 below.



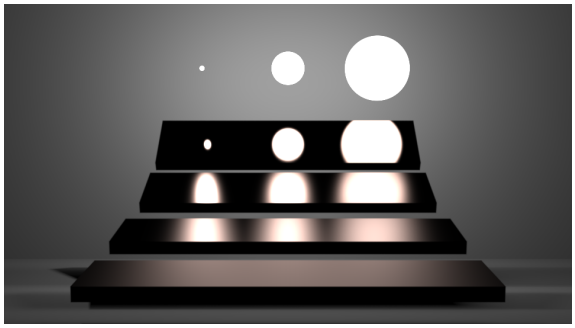
(a) BSDF-only



(b) Emitter-only



(c) MIS



(d) Reference

Figure 2.12: NEE with multiple importance sampling.

2.4 Improved Estimators

Apart from importance sampling, another way to reduce variance is to employ more sophisticated estimators tailored for the problem at hand. Naturally, implementing these estimators is more involved and iterations take more time than simple estimators such as UDPT. But in certain scenes with complex visibility conditions or with a lot of near-specular surfaces, these improved estimators may produce cleaner images in shorter amounts of time. While some of these are unbiased like UDPT, there are also biased estimators. Here we briefly introduce some of the unbiased ones.

2.4.1 Bidirectional Path Tracing

Previously in §2.2 we have seen that there are two basic ways to estimate the integral of light transport. One of them, UDPT, constructs paths starting from the camera origin, and the alternative, light tracing, starts the construction from a light source in the scene. It would be advantageous to somehow effectively combine these two estimators especially in scenes with difficult visibility conditions. Bidirectional path tracing (BDPT) [11, 24] is an estimator that does exactly this.

Intuitively, in BDPT, we trace two independent paths for each pixel in every iteration, one from the camera as in UDPT and another from an arbitrary light source as in light tracing, and combine these two paths in multiple ways.

We denote each full path $\bar{x}$ by dividing it into two sub-paths as below:

$$\bar{x} = \bar{y}\bar{z}.$$

Here, $\bar{y}$ and $\bar{z}$ denote vertices of sampled sub-paths starting from a point on a light source, and from the camera respectively. Henceforth we will call $\bar{y}$ the light sub-path and $\bar{z}$ the eye sub-path.

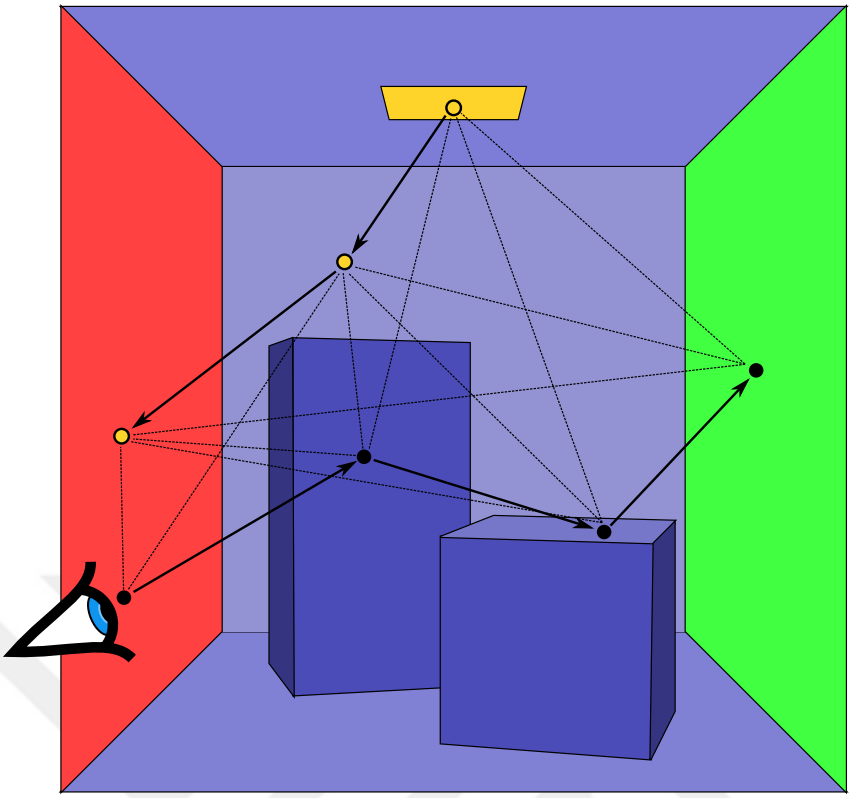


Figure 2.13: In BDPT vertices of eye path and light path are combinatorially connected to construct many new paths.

Assuming the length of a path is measured in number of vertices, let K be the length of the path $\bar{y}$ sampled by light tracing and L be the length of path $\bar{z}$ sampled by UDPT, and let $0 \leq k < K$ and $0 \leq l < L$. We denote a full path of length $k + l$ we can construct by combining sub-paths —and by sub-paths we mean the paths consisting of the first k and l vertices of $\bar{y}$ and $\bar{z}$ respectively— as below:

$$\bar{x}_l^k = \mathbf{y}_0\mathbf{y}_1 \ldots \mathbf{y}_{k-1}\mathbf{z}_{l-1} \ldots \mathbf{z}_0.$$

Consider a simple case with $k + l = 3$, for which we may have 4 different paths by

connecting the sub-path endpoints;

$$\bar{x}_0^3 = \mathbf{y}_0\mathbf{y}_1\mathbf{y}_2$$
$$\bar{x}_1^2 = \mathbf{y}_0\mathbf{y}_1\mathbf{z}_0$$
$$\bar{x}_2^1 = \mathbf{y}_0\mathbf{z}_1\mathbf{z}_0$$
$$\bar{x}_3^0 = \mathbf{z}_2\mathbf{z}_1\mathbf{z}_0.$$

Out of these paths, $\bar{x}_0^3$, $\bar{x}_3^0$, $\bar{x}_2^1$, or in general, the paths $\bar{x}_0^k$, $\bar{x}_l^0$, and $\bar{x}_l^1$ correspond to light tracing, path tracing and path tracing with NEE respectively.

This is very beneficial as we can construct 2 new paths of length 3 by only tracing 2 rays instead of 4, because we reuse the sub-paths of previously sampled paths $\bar{y}$ and $\bar{z}$. And in general by reusing sub-paths, we can construct $k + l - 1$ new paths of length $k + l$ by only tracing $k + l - 1$ rays.

In the end, the contributions from newly constructed paths are weighted and combined:

$$F = \sum_{k\geq 0}\sum_{l\geq 0} w_l^k \frac{f(\bar{x}_l^k)}{p_l^k(\bar{x}_l^k)}. \tag{2.20}$$

where the weights w_l^k are computed through MIS to increase the robustness of the estimator.

While UDPT works well in outdoor scenes with environment lights, BDPT is better suited for indoor scenes with relatively difficult illumination (see Fig. 2.14).

(a) UDPT, RMSE 0.105

(b) BDPT, RMSE 0.058

Figure 2.14: A scene rendered with UDPT and BDPT using $N = 32$ samples per pixel.

2.4.2 Metropolis Light Transport

So far, all of the estimators we have presented estimate the LTE by sampling explicitly from predefined PDFs. In this section we present a different sampling method that do not need sampling functions derived from PDFs to generate samples, and estimators based on it.

2.4.2.1 Metropolis Sampling

Metropolis sampling [15] is a general and simple Markov-chain Monte Carlo (MCMC) method used when sampling from a target distribution π with an unknown inverse cumulative distribution function (CDF) P^{-1} that may be impossible to derive analytically.

Let $\pi : \mathcal{X} \to \mathbb{R}_{\geq 0}$ be a potentially unnormalized PDF and let $x, x' \in \mathcal{X}$ with x being the current sample. In Metropolis sampling, a new potential sample x' is generated using a proposal distribution in the neighborhood of the current sample x, usually from a symmetric Gaussian distribution centered at x. In rendering literature this is also called a mutation. This new sample x' is then assigned an acceptance probability with

$$a(x'|x) = \frac{\pi(x')}{\pi(x)} \tag{2.21}$$

and either accepted as the next sample or rejected:

$$x = \begin{cases} x' & a(x'|x) > \xi \\ x & \text{otherwise} \end{cases} \tag{2.22}$$

where $\xi \in [0, 1]$ is a random number.

A more general version of this method, namely the Metropolis-Hastings sampling [4], allows non-symmetric proposal distributions. For instance, we may use anisotropic Gaussian distributions with non-symmetric covariance matrices. This way, we can generate proposals that are conformal to the geometry of target distribution.

The acceptance ratio for Metropolis-Hastings is

$$a(x'|x) = \frac{\pi(x')T(x'|x)}{\pi(x)T(x|x')}. \tag{2.23}$$

It is easy to see that Metropolis sampling is a special case of Metropolis-Hastings sampling, where proposal distributions are symmetric, implying $T(x|x') = T(x'|x)$, and thus cancel each other out.

Shown below, is an example of using Metropolis sampling to generate samples from a distribution.

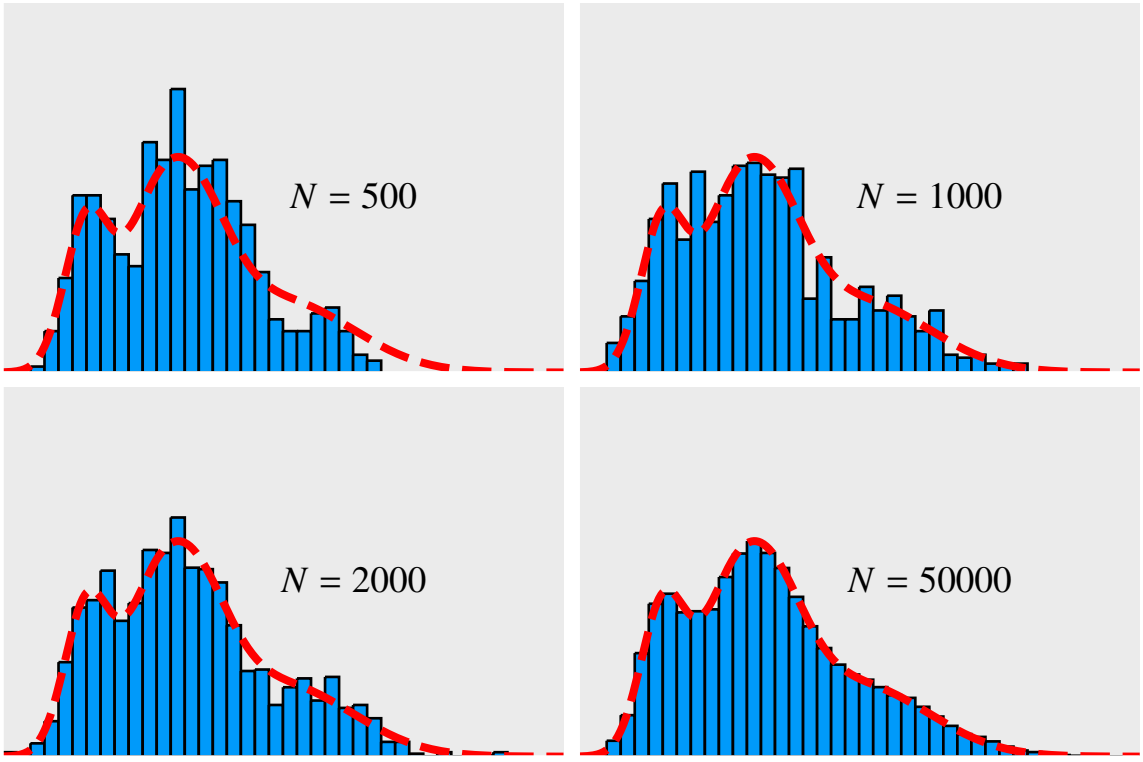


Figure 2.15: Metropolis sampling used to generate samples from an arbitrary target distribution π (shown with dashed line). As the number of samples N increases, distribution of samples converges to the target distribution.

2.4.2.2 Primary Sample Space Metropolis Light Transport

Use of Metropolis sampling in the context of rendering is first proposed in the seminal work by Veach and Guibas [25]. Now, in light transport simulation, the target distribution π becomes the measurement contribution function f. New samples are generated through mutations in the path space. A path $\bar{x}$ is mutated using one of the specifically designed mutation strategies, into a new path $\bar{x}'$. This path is either accepted or rejected depending on the acceptance probability

$$a(\bar{x}'|\bar{x}) = \frac{f(\bar{x}')T(\bar{x}'|\bar{x})}{f(\bar{x})T(\bar{x}|\bar{x}')}.$$

In practice, a path of length k, $\bar{x} \in \mathcal{P}^k$ is sampled with a vector of $O(k)$ random numbers $\bar{u} \in [0,1]^{O(k)} \subseteq \mathcal{U}^{O(k)}$. Here, $O(k)$ denotes the number of random numbers needed to sample a path of length k, and is usually taken as $2k$ or $3k$ depending on the

implementation. This random vector is fed into a joint importance sampling function $S : \mathcal{U}^{O(k)} \rightarrow \mathcal{P}^k$ such that $S(\bar{u}) = P^{-1}(\bar{u}) = \bar{x}$. S may be composed of any sampling function, such as the BSDF sampling functions [2] or geometric shape sampling functions.

In primary sample space Metropolis light transport (PSSMLT) [9], we build upon this simple fact to sample paths using MCMC state vector $\bar{u} = (u_1, \ldots, u_{O(k)})$. Unlike Veach's algorithm which makes changes directly in the path space, PSSMLT works by mutating this vector which is in a space traditionally called the primary sample space $\mathcal{U}$, basically a hypercube of random numbers $[0, 1]^{O(k)}$, and lets the importance sampling functions S automatically generate the paths (see Fig. 2.16).

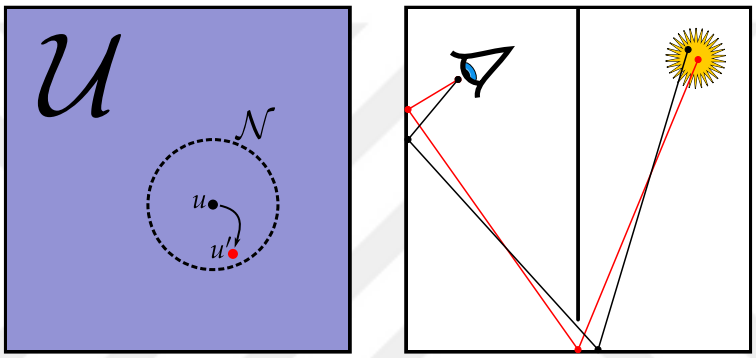


Figure 2.16: PSSMLT generates proposals in a unit hypercube, which corresponds to full paths in path space.

Since we are working on a different space ($\mathcal{U}$ rather than $\mathcal{P}$), we have to redefine our measurement contribution function and light transport equation. We redefine our new measurement contribution function in primary sample space as

$$\hat{f}(\bar{u}) = f(S(\bar{u})) = f(\bar{x}).$$

Next, we redefine the integral:

$$\int_{\mathcal{U}} \hat{f}(\bar{u}) \left| \frac{d\mu(\bar{x})}{d\bar{u}} \right| d\bar{u} = \int_{\mathcal{P}} f(\bar{x}) d\mu(\bar{x}).$$

Here, $\left| \frac{d\mu(\bar{x})}{d\bar{u}} \right|$ is the Jacobian determinant arising from the change of variables between

[2] Note that BSDF sampling functions generate directional vectors which are then used to trace a ray in that direction, eventually hitting a point on the scene surface, making that hit point the sampled path vertex.

$\mathcal{U}$ and $\mathcal{P}$. Its value is simply $\frac{1}{\hat{p}(\bar{u})}$ as we will show now.

For the sake of simplicity let us assume that $dim(\mathcal{U}) = 1$. From the previous redefinition we know that

$$\left|\frac{d\mu(\bar{x})}{d\bar{u}}\right| = \left|\frac{dP^{-1}}{d\bar{u}}\right|.$$

The CDF P is defined as

$$P(\bar{u}) = \int_0^{\bar{u}} \hat{p}(\bar{u}) d\bar{u}.$$

Differentiating both sides with respect to $\bar{u}$,

$$\frac{dP(\bar{u})}{d\bar{u}} = \hat{p}(\bar{u}).$$

From elementary calculus,

$$\frac{dP(\bar{u})}{d\bar{u}} = \frac{1}{\frac{dP^{-1}(\bar{u})}{d\bar{u}}},$$

thus, $\left|\frac{dP^{-1}(\bar{u})}{d\bar{u}}\right| = \left|\frac{d\mu(\bar{x})}{d\bar{u}}\right| = \frac{1}{\hat{p}(\bar{u})}$.

PSSMLT defines two different mutation strategies; (1) the small-step strategy, which generates a new sample vector $\bar{u}'$ by sampling from a symmetric probability distribution, usually a Gaussian distribution $\mathcal{N}$ with small covariance, centered at the current sample vector $\bar{u}$, and (2) the large-step strategy, which completely discards the current sample vector $\bar{u}$ and generates the new sample vector $\bar{u}'$ by uniformly sampling the hypercube.

These two strategies are designed to address the exploration-exploitation dilemma. (1) exploits the information on previous sample vector $\bar{u}$, which may correspond to a path with high contribution value, by generating the new sample vector $\bar{u}'$ in the neighborhood of $\bar{u}$, hoping that $\bar{u}'$ also corresponds to a path with contribution value closer to, or even higher than the previous one. This way, we can sample paths with high energy more frequently.

The benefits of (2) is twofold. By occasionally generating a completely random sample vector $\bar{u}'$ regardless of $\bar{u}$, new high energy regions in the domain can be explored. In addition, as a consequence, the ergodicity condition of MCMC is also obeyed. If one uses only the strategy (1), the method may get stuck in isolated regions. These regions,

sometimes informally called islands, are surrounded by zero-energy regions. And hence we need bigger jumps as used in (2) to escape them (see Fig. 2.17).

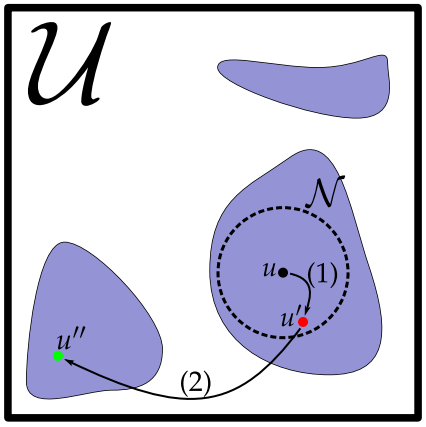


Figure 2.17: Two mutation strategies are used in PSSMLT. (1) the small-step mutation and (2) the large-step mutation.

2.4.2.3 Anisotropic Proposals through Hamiltonian Monte Carlo

As shown previously, rendering algorithms based on MCMC methods often use simple isotropic Gaussian distributions for local sample propositions with the same covariance matrix throughout the whole domain of the target function. This simplifies the calculation of acceptance ratio as the transition probabilities in Eq. 2.23 cancel each other out. However, symmetric distributions are suboptimal in cases where the target distribution has isolated nonzero regions with uneven shapes. In such cases, utilizing anisotropic proposal distributions becomes more appropriate. In recently published paper [12], the authors use Hamiltonian Monte Carlo (HMC), taking the geometry of target distribution into account to more adaptively sample from it.

HMC views the current sample as a particle which resides on the landscape of the negative target function, applies a random momentum to this particle and finds its position after t time units by solving a Hamiltonian mechanics problem, and this new position becomes the proposed sample. This way, better proposals can be generated, since HMC uses more information compared to previous classical MLT methods by generating samples according to the underlying manifold of light transport.

Solving this problem requires relatively slow numerical integration methods. As a

remedy to inefficiency, the authors approximate the target function with second-order Taylor expansion and derive closed form solutions to the underlying mechanics problem.

The resulting algorithm outperforms other MCMC based rendering methods in complex scenes with highly glossy materials. Because it needs the automatic differentiation (AD) of rendering system, it is hard to implement the algorithm in an existing rendering library or framework without AD support. AD is also slower to compute which may cause the algorithm to perform worse in simpler scenes.

2.5 An Adaptive Approach to Sampling: Guided Path Tracing

Heretofore we have talked about Monte Carlo estimators that sample from fixed, a priori distributions or from a combination of them through MIS. However, none of these a priori distributions are ideal in that they are not perfectly proportional to the integrand. The ideal distribution depends on the parameters of the scene being rendered, and thus it is different for each scene. As we cannot derive this ideal distribution analytically, how can we approximate it better than a priori distributions?

In this section we answer this question by giving a brief overview of previously published scene-adaptive sampling techniques known as path guiding methods. After reviewing previous work, in §2.5.5 we present our contribution, which is an extension to a previous path guiding method. The leitmotif behind these methods is that, influenced by the principles of statistical learning from data, they all gather and encode information from previously sampled paths about the scene (or the integral domain) and use that information to guide new paths so as to enable improved adaptive importance sampling. [3]

Recall the solid angle formulation of LTE:

$$L_o = \int_{\mathcal{H}^2} L_i f_s \cos\theta_i d\omega.$$

Standard UDPT estimates the integral by importance sampling the PDF derived from BSDF f_s. However we cannot generate samples according to L_i, as deriving a PDF proportional to it

$$p \propto \frac{L_i}{\int_{\mathcal{H}^2} L_i d\omega},$$

requires evaluation of an integral which is impossible to evaluate analytically. Except for very simple, specifically modelled scenes, we are simply unable to construct an analytic form of L_i, as we need to trace paths to evaluate it. Hence, unlike the PDF derived from f_s, we do not have access to an a priori PDF derived from L_i. The aim of most of the path

[3] We can also view MLT as an adaptive sampling method, or even as a path guiding method which only leverages local information, namely the the local neighborhood of the previously sampled path. That is exploiting the fact that previously sampled path carries a lot of energy and samples the next path $\bar{x}'$ that is in the neighborhood of the current path $\bar{x}$ in path space.

guiding methods is then to learn PDFs that are as proportional to L_i as possible.

2.5.1 Histogram Based Path Guiding

One of the very first path guiding methods is proposed by Jensen [7] and it was influential for the path guiding methods that were published after it. It is a relatively simple method that tries to approximate L_i at various points in the scene, and derives discrete PDFs from those approximations which are then used to sample directions, from where most of the direct or indirect illumination comes, by only utilizing a k-d tree and histograms.

First, light paths traced from light sources and each vertex position of these paths together with the incoming direction and carried radiance are recorded in a k-d tree. Then, while tracing paths from camera, at each path vertex we query its k nearest neighbors using the k-d tree and build a histogram of incoming directions, accumulating the carried radiance values. We then sample a new direction according to a CDF derived from this histogram. In effect, this results in better importance sampling, as at each point we have access to the approximate information of the directions where most of the illumination is coming from.

While the idea of using information gathered from previous samples to enable better sampling in the next iterations seems better than just using fixed, handcrafted a priori PDFs, it is not without flaws. Weaknesses of this method result mainly from the use of the parameter-free histogram-based model that approximates the irradiance at a cache point in the scene. Since the model is parameter-free, all samples must be stored in memory at all times, which are used in k nearest neighbors queries at path vertices to build histograms. This becomes a significant problem especially in more complex scenes where relatively larger amounts of samples may be needed to gain information to properly guide paths. Limited amount of memory is not enough to record all these samples, causing the robustness of the method to decrease drastically.

Another point of concern arises from the fact that no view-dependent information is recorded. That is because only the incident radiance L_i is encoded in histograms and not the product of incident radiance and the BSDF value $L_e \times f_s$. As a result, this method is better suited for scenes abundant with view-independent, diffuse surfaces.

2.5.2 Gaussian Mixture Model Based Path Guiding

To cope with the limitations of the histogram based guiding method, in [27] the authors build upon it by utilizing a parameterized model. To be precise, they train Gaussian mixture models (GMM) to encode, at finite set of points $\{\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_N\} \subset \mathcal{M}$, the information gathered from previous samples to build PDFs approximately proportional to L_i:

$$p_{\mathbf{x}}(.) = \sum_i \pi_i \mathcal{N}(.|\mu_i, \Sigma_i) \approx \frac{L_i}{\int_{\mathcal{H}^2} L_i d\omega}.$$

This way, information from previous samples can be captured by a relatively small number of parameters and need not be stored in the memory at all times. However, standard Expectation-Maximization (EM) method [16] (see Algorithm 2) used to compute GMM weights do not work in on-line scenarios, meaning that previously sampled data must be stored in the memory together with newly sampled data to recompute the parameters. This of course eliminates the promised memory usage improvement. Another important point to address is learning from weighted data, due to the training data consisting of direction vectors with associated radiance values. Unfortunately, standard EM cannot account for weighted data.

To address the first issue they consider the step-wise EM algorithm [13] (see Algorithm 3). Step-wise EM can be easily modified to train GMM on-line, but again, it cannot be used to learn from weighted data. To learn from weighted data and therefore to address the second issue they further modify the online version of step-wise EM algorithm. Hence they use this modified step-wise EM algorithm to address both of these issues.

Algorithm 2 Standard Expectation-Maximization algorithm

1: $\gamma_{qj} = \frac{\pi_j \mathcal{N}(s_q|\theta_j^{old})}{\sum_{h=1}^{K} \pi_h \mathcal{N}(s_q|\theta_h^{old})}$
2: **repeat**
3: $\quad u_{N-1}^{j} \leftarrow \frac{1}{N} \sum_{q=0}^{N-1} \gamma_{qj} u(s_q)$
4: $\quad \theta^{new} \leftarrow \bar{\theta}(u_{N-1}^{j} \dots u_{N-1}^{K})$
5: **until** convergence

Algorithm 3 Step-wise Expectation-Maximization algorithm

$i \leftarrow 0$
repeat
  for all $q \in 0, \ldots, N-1$ **do**
    $u_i^j \leftarrow (1-\eta_i)U_{i-1}^j + \eta_i \gamma_{qj} u(s_q)$
    if $(i+1) \mod m = 0$ **then**
      $\theta^{new} \leftarrow \bar{\theta}(u_{N-1}^j \ldots u_{N-1}^K)$
    end if
    $i \leftarrow i+1$
  end for
until convergence

The method works as follows: Similar to the photon map based method, in a learning phase we trace paths from light sources and also from the camera, and record vertices, incoming directions and radiance values in k-d trees. Then, in rendering phase while tracing paths, we look for nearby caches at points x^c in the neighborhood of the current path vertex $x^c \in \mathcal{B}(x_i)$. If there exist none, we build a GMM cache at that vertex x_i by gathering nearby data stored in k-d trees, and using the aforementioned modified step-wise EM algorithm. If there exist caches in the neighborhood of the current vertex, we interpolate parameters of GMMs in these caches, and sample a new direction using this GMM with interpolated parameters.

Caches that use data from light paths are called photon caches and from camera paths are called importon caches. Intiuitively, photon caches guide paths closer to light sources and importon caches guide paths visually important regions of the scene, i.e. the regions in direct view of the camera. These caches are respectively stored in separate k-d trees.

Recording both photons and importons allows us to combine this method with bi-directional light transport algorithms such as BDPT which we introduced in §2.4.1. Extending the method to guide paths in a bi-directional light transport setting is fairly straightforward: Light paths are guided by sampling appropriate importon caches, and eye paths are guided, as in standard guided path tracing presented above, by sampling photon caches.

As in the histogram based method, the main focus of this method is to approximate the incident radiance term L_i, thus it cannot handle the view-dependent domain properly and it offers no improvement over the histogram based method in this matter. GMMs are better suited to model smooth functions. As the scene geometry may change abruptly, trained GMMs may not be sharp enough to guide paths correctly, whereas piecewise functions may be better suited.

2.5.3 Path Guiding through Reinforcement Learning

The method proposed by [2] is in a similar vein to GMM based method, with the main exception that it uses Q-learning [30], a tabular reinforcement learning method, to guide paths. Reinforcement learning is a class of machine learning techniques, initially influenced by ideas from psychology of animal learning [22]. The basic idea is this: By interacting with a given environment, an agent learns to take the actions that, based upon the feedbacks taken from the environment, benefits her the most in the end. Put formally, let $\mathcal{S}$ be the set of states that an agent can be in and let $\mathcal{A}$ be the set of actions that an agent can take in an environment with a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. At time t the agent at a state $s_t \in \mathcal{S}$ takes an action $a_t \in \mathcal{A}$, arrives at a new state $s_t \in \mathcal{S}$ and gets a reward value $R(s_t, a_t)$. The objective of the agent then is to learn to take the actions a_t at states s_t, such that the cumulative reward is maximized.

Q-learning, the reinforcement learning technique employed in this paper is formulated by the following recursive equation:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a')\right). \tag{2.24}$$

As our main purpose is to guide paths by sampling a direction ω from a path vertex x, an environment of light transport simulation can be trivially described in the context of reinforcement learning:

$$\mathcal{S} = \mathcal{M},$$
$$\mathcal{A} = S^2,$$
$$R = L_i$$

Where the set of states corresponds to the set of surface points, set of actions corresponds to the set of directions defined by hemisphere. Indeed, this maps quite naturally to the essence of reinforcement learning. We can now describe the light transport in terms of reinforcement learning: At each state $x_i \in \mathcal{M}$, take an action $\omega \in S^2$ and arrive at a new state x_{i+1} and get the reward L_i.

However, there is a pressing issue with this direct application of reinforcement learning, as in light transport we do not seek to construct a path that carries the highest energy, but want to sample the paths proportional to a term in the integrand, or ideally proportional to the whole integrand. To remedy this, the authors stress that Eq. 2.24 can be modified to update the Q-function. Instead of the maximum Q value of actions, it can be updated with the value computed by averaging Q values with a kernel π

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(R(s,a) + \gamma \sum_{a' \in \mathcal{A}} \pi(s',a')Q(s',a')\right) \tag{2.25}$$

which can be rewritten as

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(R(s,a) + \gamma \int_{\mathcal{A}} \pi(s',a')Q(s',a')da'\right) \tag{2.26}$$

if the domain is continuous, as in the case of light transport.

Now, observe that the second term in the left hand side of Eq. 2.26 looks interchangeable with the right hand side of the rendering equation. After substituting, the new Q function compatible with light transport thus becomes

$$Q(\mathbf{x},\omega) = (1-\alpha)Q(x,\omega) + \alpha\left(L_e + \gamma \int_{\mathcal{H}^2} f_s Q(\mathbf{x}',-\omega)d\omega'\right) \tag{2.27}$$

where in essence, it approximates the incident radiance L_i. A PDF derived from this function then can be used to generate samples proportional to L_i, guiding the paths to light sources.

In practice, the authors discretize this function by approximating the state and action spaces with finite sets. They discretize the state space with a fixed grid. Furthermore for each cell inside that grid they discretize the action space likewise with finite number of strata. Consequently, the Q-function becomes the Q-table so that in $Q(\mathbf{x},\omega)$, $\mathbf{x}$ corresponds to a cell of the grid and ω to a stratum.

As the data structure used to discretize the Q-function is not adaptive, it may not approximate the Q-function well, especially in more complex scenes where more refined data structures are needed, which guarantees a high memory consumption. This easily becomes the main drawback of this method.

2.5.4 Tree-based Path Guiding

The use of tree data structures is a recurring theme in path guiding methods. Tree based data structures are generally used to partition the spatial dimensions of training data. Usually the statistical models such as histograms or GMMs are used to learn the directional distributions as mentioned in previous sections. In this section we cover two methods, out of which the second one is influenced by the first, that use trees both for partitioning of the spatial dimensions and to learn directional distributions.

2.5.4.1 5D-tree

One of the very first algorithms to build a tree data structure that encodes data for adaptive importance sampling is presented in [10]. They approximate the incident radiance L_i by recording the incident radiance at each path vertex in a 5D tree which partitions the spatio-directional domain of the radiance field. This is analogous for instance to the octree which is used to partition the 3D space, where each node is partitioned to 2^3 child nodes with equal volumes. The 5D tree then has a branching factor of 2^5. The incident radiance L_i from direction ω at path vertex $\mathbf{x}$ is accumulated in the nodes that contain $(\mathbf{x}, \omega)$. Nodes in this tree are partitioned when the number of radiance values accumulated in that node is above a threshold. When tracing paths this tree is then used to derive at each path vertex $\mathbf{x}$ a piecewise constant PDF approximately proportional to L_i. And from that PDF, a new direction ω is sampled to continue tracing the path. As a result, similar to the previously presented guiding methods, paths are guided to regions of the scene with more illumination.

Because of the high branching factor this method tends to use too much memory and traversing the tree to construct piecewise-constant PDFs is relatively inefficient. The next method aims to solve these problems with a better designed tree-based data structure.

2.5.4.2 Practical Path Guiding

The data structure called SD-tree (spatio-directional tree) proposed by Müller et al. [18] approximates L_i more efficiently. Instead of partitioning the whole 5D spatio-directional domain with a single tree, the spatial and directional domains are partitioned with different trees with low branching factors. Specifically, the 3D spatial domain is partitioned with a binary tree, where at each level, the nodes are split according to a partition axis, which cycles through the principal axes x, y or z. Each leaf node of this binary tree holds a quadtree, which partitions the 2D directional domain. The resulting tree effectively covers the 5D domain of the radiance field, while lowering the branching factor substantially.

Two trees are used while rendering. $\mathcal{T}^k_{\text{sampling}}$ is used for sampling, $\mathcal{T}^k_{\text{building}}$ is used to record new data. Let $\mathcal{T}(x)$ denote the quadtree connected to the spatial binary tree leaf node containing the point x, and $\mathcal{T}(\mathbf{x}, \omega)$ the leaf node of that quadtree containing the direction ω. At each path vertex $\mathbf{x}_l$, a direction ω is sampled from the piecewise-constant PDF derived from $\mathcal{T}^k_{\text{sampling}}(\mathbf{x}_l)$. After a full path with nonzero contribution is traced, for each vertex x_i of the path, incident radiance from ω is accumulated into the quadtree leaf node $\mathcal{T}^k_{\text{building}}(\mathbf{x}_l, \omega)$ and the nodes accessed during the traversal to reach that leaf node. A pseudocode of this method is given in Algorithm 4.

The method works in iterations with exponentially increasing samples per pixels. Specifically, in iteration k, 2^k paths are traced per pixel. Prior to each iteration k, the new sampling tree $\mathcal{T}^k_{\text{sampling}}$ is constructed from $\mathcal{T}^{k-1}_{\text{building}}$. First, quadtrees are refined in parallel. If the accumulated energy L in a leaf node of the quadtree is above a predetermined threshold, then that node is split into four child nodes with equal energy adding up to L. This process continues until the energies stored in new leaf nodes are below the threshold. Then, the binary tree is refined. Refinement condition is based on the number of samples recorded in quadtrees. If the number of samples recorded in a quadtree is more than a predetermined threshold, then the binary tree leaf node containing that quadtree is split according to the partition axis of that level, and the quadtree is copied to the new leaf node.

Notable drawbacks of this method are (1) as in other path guiding methods, it only approximates L_i, and (2) the algorithm works incrementally: In each iteration k, rendering

starts from scratch and the samples from previous iterations are wasted. Also, during the iteration, $\mathcal{T}^k_{\text{sampling}}$ is kept constant and not updated until the end of the iteration.

Algorithm 4 Path Tracing with Practical Path Guiding

for all $k \in \{1, \ldots, K\}$ **do**
  $(\mathcal{T}^k_{\text{sampling}}, \mathcal{T}^k_{\text{building}}) \leftarrow$ RebuildSDTree$(\mathcal{T}^{k-1}_{\text{building}})$
  for all $j \in \{1, \ldots, J\}$ **do**
    for all $n \in \{1, \ldots, 2^k\}$ **do**
      $\mathbf{p} \leftarrow$ SamplePointInsidePixel(j)
      $\mathbf{x}' \leftarrow$ TraceRay$((\mathbf{x}, \omega_i))$
      $m \leftarrow 0$
      vertices $\leftarrow$ []
      while $(L_e(\mathbf{x}' \rightarrow \mathbf{x}) = 0) \wedge (m < M)$ **do**
        $\mathbf{x} \leftarrow \mathbf{x}'$
        $\omega_o \leftarrow -\omega_i$
        $\omega_i \sim S_{\mathcal{T}^k_{\text{sampling}}}$
        $\mathfrak{f} \leftarrow \mathfrak{f} \times \frac{f_s(x, \omega_i \rightarrow \omega_o) \times |\mathbf{n}(x) \cdot \omega_i|}{p_{\mathcal{T}^k_{\text{sampling}}}(\omega_i)}$
        Append(vertices, $(x, \omega_i, \mathfrak{f})$)
        $\mathbf{x}' \leftarrow$ TraceRay$((\mathbf{x}, \omega_i))$
        $m \leftarrow m + 1$
      end while
      $L_j \leftarrow L_j + L_e(\mathbf{x}' \rightarrow \mathbf{x}) \times \mathfrak{f}$
      if $L_e(\mathbf{x}' \rightarrow \mathbf{x}) > 0$ **then**
        Record$(\mathcal{T}^k_{\text{building}}, L_e(\mathbf{x}' \rightarrow \mathbf{x})$, vertices$)$
      end if
    end for
    $\langle I_j \rangle^N \leftarrow \frac{L_j}{N}$
  end for
  $k \leftarrow k + 1$
end for

2.5.5 Extending Practical Path Guiding for Product Importance Sampling

One common shortcoming of previous guiding methods is that in essence they only approximate the incident radiance L_i and importance sample the PDF derived from that approximation. Informally this means to sample, at a point in the scene, directions where most of the light is coming from more frequently. This is perfectly viable where that point is on a surface with view-independent, Lambertian BSDF. But if that point is on a surface with view-dependent, glossy BSDF, depending on the view direction, the direction sampled according only to L_i may cause the BSDF to take a zero value, making the contribution of the path to be zero. We emphasize this problem and develop a path guiding method that considers view-dependent BSDFs in addition to the incident radiance to make product importance sampling possible.

Path guiding methods we have reviewed in previous sections try to get around this problem typically by employing MIS. While rendering, they sample directions with a predefined probability α using the PDF proportional to the BSDF, and with probability $1-\alpha$ using the learned PDF that is approximately proportional to incident radiance. These two sampling techniques are then combined through MIS.

Previously, only a few attempts have been made to guide paths according to the product of incident radiance and the BSDF. Steinhurst and Lastra [21] extend the histogram based guiding method of Jensen [7], and Herholz et al. [5] extend the GMM based guiding method of Vorba et al. [27] to enable product importance sampling. Unlike our method, both of these methods preprocess BSDFs. Guo et al. [3] adaptively partition the primary sample space with a k-d tree using the energy carried by the paths constructed from primary sample vectors, and generate primary samples according to this tree. Recently, the use of generative neural networks to approximate a PDF proportional to the integrand of LTE is explored in [19].

For our approach, we have considered several design decisions. Specifically, our method (1) most importantly should handle view-dependent sampling according to $L_i \times f_s$ and therefore should make product importance sampling possible, (2) should not have a

significant performance or memory overhead over the previous work in scenes with mainly diffuse surfaces, and (3) should be easy to implement.

We build our method upon the path guiding method of Müller et al. reviewed in §2.5.4.2. From now on we will refer to this method as PPG. Our method can be seen as an extended version of PPG that takes the aforementioned design decisions into account. Other extensions to PPG have been presented in [17, 28]. These are orthogonal to our contribution and thus can be used in conjunction with our method.

2.5.5.1 View-dependent SD-tree

To record the light field in the $S^2 \times S^2$ domain and consequently enable view-dependent product importance sampling, one may replace the directional quadtree that covers only the two dimensional spherical domain S^2 with a four dimensional tree that is an analogue of the quadtree. The four dimensional tree naturally has a relatively high branching factor of $O(2^{4d})$ where d is the depth of the tree. This increases the memory consumption substantially and also uses unnecessary memory in the scenes where the majority of the surface materials are diffuse. A quadtree is enough to capture the spherical domain of incident radiance on diffuse surfaces, and it learns the light field more robustly in this case, since in 4D tree, the data are scattered into more branches.

Instead of using a single tree both for view-dependent and view-independent light transport, we use separate structures for each. We keep the same quadtree structure for Lambertian surfaces from the previous work. In addition to that, to handle view-dependent surfaces, we introduce a hierarchical tree. This tree is comprised of an upper quadtree and lower quadtrees connected to the leaf nodes of the upper quadtree. The upper tree covers the spherical domain of view directions and lower trees record radiance and used for sampling directions. Formally, we will refer to the quadtree inherited from PPG as $\mathcal{T}_1(\mathbf{x})$, to its leaf nodes as $\mathcal{T}_1(\mathbf{x}, \omega_i)$, to the upper quadtree of our hierarchical tree as $\mathcal{T}_2(\mathbf{x})$, to its lower quadtree given a view direction ω_o as $\mathcal{T}_2(\mathbf{x}, \omega_o)$, and to the leaf node of that lower quadtree as $\mathcal{T}_2(\mathbf{x}, \omega_o, \omega_i)$.

2.5.5.2 Training

The training proceeds in iterations with increasing number of samples as described in §2.5.4.2. Once a full path with nonzero contribution is traced, for each path vertex $\mathbf{x}_l$ we record radiance in the appropriate quadtree depending on the BSDF at $\mathbf{x}_l$. If $\mathbf{x}_l$ is on a diffuse surface, then the incident radiance is recorded in the respective nodes in quadtree $\mathcal{T}^k_{1,\text{building}}(\mathbf{x}_l)$ as in PPG. If $\mathbf{x}_l$ is on a glossy surface, then the incident radiance is recorded, depending on the view direction ω_o, in the quadtree leaf node $\mathcal{T}^k_{2,\text{building}}(\mathbf{x}_l, \omega_o, \omega_i)$ and in its parent nodes. This way, depending on the material properties of the scene, our modified SD-tree does not use too much unnecessary memory where there are no view-dependent BSDFs.

In PPG, during the refinement phase of SD-tree, each leaf node of the spatial binary tree is recursively split whenever the number of path vertex data recorded through that leaf node is above the threshold $c \cdot \sqrt{2^k}$. This threshold increases with each iteration k, proportional to the number of paths traced per pixel in that iteration, and c is chosen to be 12000 by default. We keep the same threshold for the binary tree and also use it as the subdivision threshold of our upper quadtree. We set $c = 4000$ for our upper quadtree, and because of the memory consumption concerns, upper quadtree leaf nodes that are eligible for refinement are split only once per iteration.

2.5.5.3 Sampling

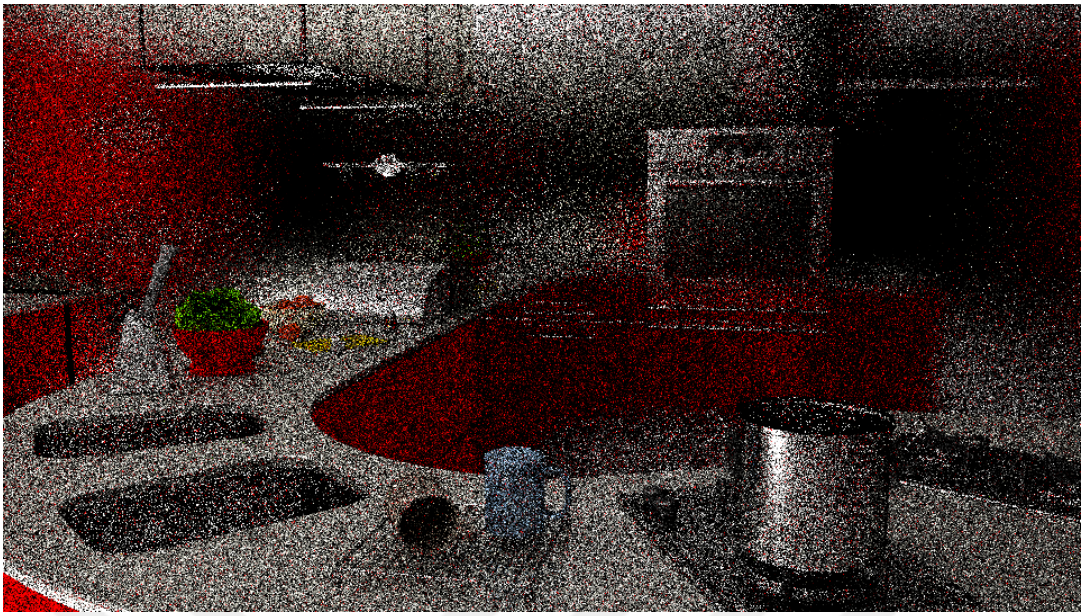
Sampling is very trivial as we reuse the same quadtree structure to sample directions. While tracing a path, at each path vertex $\mathbf{x}_l$ we sample a new direction ω_i through $\mathcal{T}^k_{1,\text{sampling}}(\mathbf{x}_l)$ if the BSDF at $\mathbf{x}_l$ is view-independent as in PPG, and through $\mathcal{T}^k_{2,\text{sampling}}(\mathbf{x}_l, \omega_o)$ if the BSDF at $\mathbf{x}_l$ is view-dependent.

3. RESULTS AND DISCUSSION

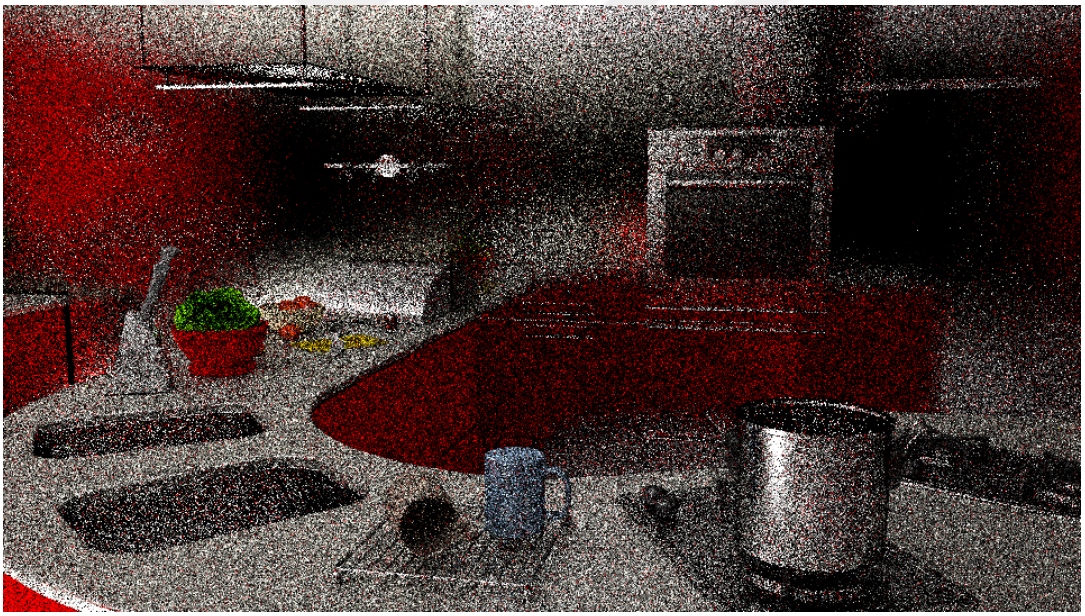
We implemented our method by modifying the openly available PPG plugin for Mitsuba renderer [6]. All tests are done using a computer equipped with Intel i7-4700MQ CPU. We evaluate our method by considering three cases; (1) a scene with many glossy surfaces, (2) a scene with glossy and diffuse surfaces and (3) a scene with only diffuse surfaces.

Glossy scene. We first evaluate our method against PPG in a scene with many glossy surfaces and thus with many glossy-to-glossy light transport. In addition to RMSE of produced images, we also evaluate the ratio of number of sampled paths with nonzero contribution to total number of sampled paths. This ratio is correlated to variance and thus an indicator of how noisy the image produced will be. In the glossy scene, most of the paths are terminated because their BSDF values become zero at some vertex. Furthermore, we also compare memory usages. We run both our method and PPG with a budget of 1024 samples per pixel. In both methods, next event estimation is enabled only for the first few passes (passes 1-5 in tables below, pass 0 omitted). Default parameters are used for PPG unless stated otherwise. By default, PPG uses $\alpha = 0.5$ as BSDF sampling probability. We also make comparisons by setting this probability to a lower value.

Our method clearly shows improvement in nonzero path percentages with the expense of memory usage in this extreme case (see Table 3.1). Consequently, this is reflected in the rendered images shown in Fig. 3.1. Difference between our method and PPG becomes even more apparent when we lower the BSDF sampling probability α to 0.1, as can be seen in Fig. 3.2 and from per pass statistics in Table 3.2. This is because unlike our method, PPG only learns the incident radiance and thus has to rely on BSDF sampling more when the surface is glossy.

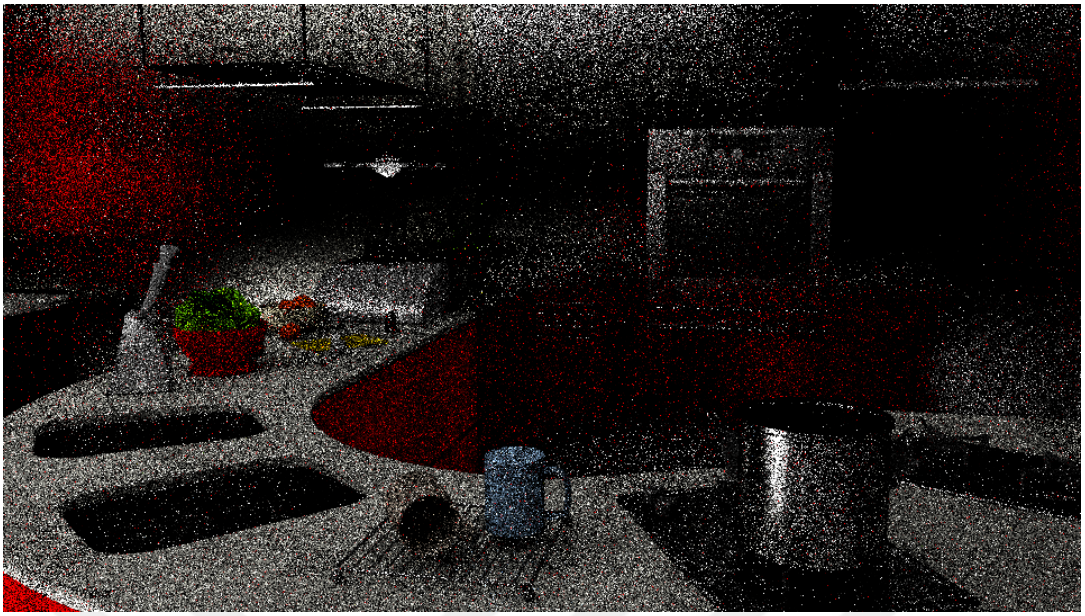


(a) PPG, 1024 spp (24 minutes), RMSE 0.567

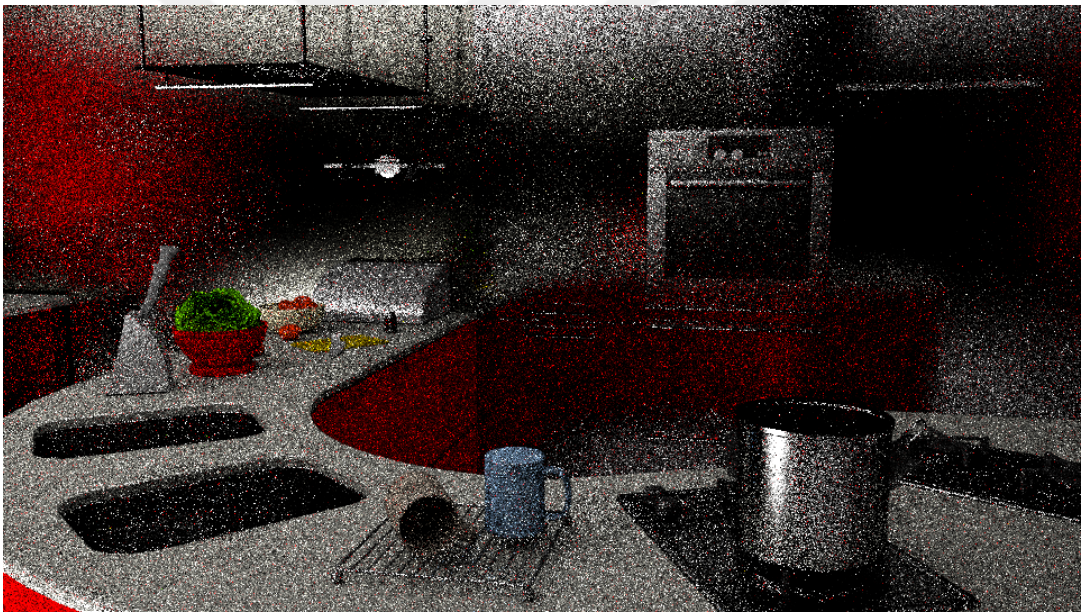


(b) Our, 1024 spp (28 minutes), RMSE 0.504

Figure 3.1: Visual quality comparison of our method and PPG in a scene with many glossy surfaces. $\alpha = 0.5$



(a) PPG, 1024 spp (16 minutes), RMSE 0.614



(b) Our, 1024 spp (22 minutes), RMSE 0.506

Figure 3.2: Visual quality comparison of our method and PPG in a scene with many glossy surfaces. $\alpha = 0.1$

Table 3.1: Nonzero path percentage and memory usage statistics of our method and PPG in a scene with many glossy surfaces. $\alpha = 0.5$

	Nonzero Path %		Memory Usage (MB)	
Pass #	PPG	Our	PPG	Our
1	3.558	3.949	2.195	4.57
2	6.526	9.716	3.912	8.855
3	16.828	22.733	5.307	13.522
4	21.71	27.473	6.549	27.04
5	25.029	32.196	10.053	45.964
6	0.14	0.203	14.727	76.725
7	0.275	0.536	36.592	228.602

Table 3.2: Nonzero path percentage and memory usage statistics of our method and PPG in a scene with many glossy surfaces. $\alpha = 0.1$

	Nonzero Path %		Memory Usage (MB)	
Pass #	PPG	Our	PPG	Our
1	1.264	1.804	2.195	4.57
2	3.255	7.887	3.352	7.122
3	10.351	21.1	3.834	10.342
4	14.021	29.406	4.923	17.628
5	20.956	36.266	5.325	32.625
6	0.089	0.33	8.578	55.911
7	0.285	1.126	20.836	156.522

Mixed scene. In this case we repeat the previous experiment on a scene with many large diffuse surfaces and some relatively smaller glossy surfaces. As can be seen in Fig. 3.3 and Table 3.3, our method did not bring any improvement over PPG, albeit consuming

more memory.

(a) PPG, 1024 spp, RMSE 0.0336

(b) Our, 1024 spp, RMSE 0.0334

Figure 3.3: Visual quality comparison of our method and PPG in a scene with relatively lesser number of glossy surfaces. $\alpha = 0.5$

Table 3.3: Nonzero path percentage and memory usage statistics of our method and PPG in a scene with relatively lesser number of glossy surfaces. $\alpha = 0.5$

	Nonzero Path %		Memory Usage (MB)	
Pass #	PPG	Our	PPG	Our
1	41.862	42.078	2.195	9.141
2	64.731	66.128	3.781	14.069
3	68.362	69.468	6.13	19.259
4	72.291	73.279	8.99	21.997
5	73.968	74.457	11.879	30.287
6	49.216	49.693	17.22	42.83
7	51.489	51.991	34.03	84.168

Diffuse scene. Finally we test our method in a scene that has only diffuse surfaces. The main purpose of this scenario is to evaluate the memory overhead of our method in a case where it is not designed for and to see whether it can perform similar to PPG. In theory, the memory and runtime overhead of our method should be negligible in comparison with PPG, as the view-dependent tree is not used and no data is recorded in it in this case. Indeed, as shown in Table 3.4, memory usage of our tree is only slightly higher than the previous work.

Table 3.4: Memory usages of our method and PPG in a diffuse-only scene.

	Memory Usage (MB)	
Pass #	PPG	Our
1	1.0976	2.2851
2	2.0265	3.7051
3	3.2196	6.2974
4	5.3281	10.3316
5	8.2614	15.8805
6	11.4196	22.2428

4. CONCLUSIONS AND FUTURE WORK

In this work, we presented adaptive rendering techniques, focusing on path guiding methods. We stressed that most of the path guiding methods do not consider the product importance sampling of BSDF and incident radiance terms in the integrand of LTE, approximating only the incident radiance and emulating product importance sampling through MIS. We attempted to solve this problem with a simple extension to the PPG method. While our contribution did not solve the problem completely, it gave promising results in scenes where the majority of light transport happens between glossy surfaces.

As future work, we would like to combine a proper reinforcement learning technique such as Q-learning with our method, and explore different approaches to make our path guiding method learn faster and more robustly. Our method also inherits some of the disadvantages of PPG. The main issue is that it wastes a lot of samples, as it updates the trees only at the end of each pass, and at the beginning of each pass, starts to render the image from scratch. The question remains as to how it can be turned into a method that works continuously without passes.

REFERENCES

[1] Robert L. Cook and Kenneth E. Torrance. “A Reflectance Model for Computer Graphics”. In: 15.3 (Aug. 1981), pp. 307–316. ISSN: 0097-8930. DOI: 10/br5ps6.

[2] Ken Dahm and Alexander Keller. “Learning Light Transport the Reinforced Way”. In: (Los Angeles, California). 2017, 73:1–73:2. ISBN: 978-1-4503-5008-2. DOI: 10/gfzsm4.

[3] Jerry Guo et al. “Primary sample space path guiding”. In: *Eurographics Symposium on Rendering*. Vol. 2018. The Eurographics Association. 2018, pp. 73–82.

[4] Wilfred K. Hastings. “Monte Carlo Sampling Methods Using Markov Chains and Their Applications”. In: 57.1 (Apr. 1, 1970), pp. 97–109. ISSN: 0006-3444. DOI: 10/dkbmcf.

[5] Sebastian Herholz et al. “Product Importance Sampling for Light Transport Path Guiding”. In: (2016). ISSN: 1467-8659. DOI: 10/f842dt.

[6] Wenzel Jakob. *Mitsuba Renderer*. 2013. URL: http://www.mitsuba-renderer.org.

[7] Henrik Wann Jensen. “Importance Driven Path Tracing Using the Photon Map”. In: ed. by Patrick M. Hanrahan and Werner Purgathofer. 1995, pp. 326–335. ISBN: 978-3-7091-9430-0. DOI: 10/gf2hcr.

[8] James T. Kajiya. “The Rendering Equation”. In: 20.4 (Aug. 1986), pp. 143–150. ISSN: 0097-8930. DOI: 10/cvf53j.

[9] Csaba Kelemen et al. “A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm”. In: 21.3 (Sept. 1, 2002), pp. 531–540. ISSN: 0167-7055. DOI: 10/bfrsqn.

[10] Eric P. Lafortune and Yves D. Willems. “A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing”. In: ed. by Patrick M. Hanrahan and Werner Purgathofer. NY, June 1995, pp. 11–20. ISBN: 978-3-7091-9430-0. DOI: 10/gfz5ns.

[11] Eric P. Lafortune and Yves D. Willems. “Bi-Directional Path Tracing”. In: (Alvor, Portugal). Vol. 93. Alvor, Portugal, Dec. 1993, pp. 145–153.

[12] Tzu-Mao Li et al. “Anisotropic Gaussian Mutations for Metropolis Light Transport through Hessian-Hamiltonian Dynamics”. In: 34.6 (Oct. 2015), 209:1–209:13. ISSN: 0730-0301. DOI: 10/f7wrcs.

[13] Percy Liang and Dan Klein. “Online EM for unsupervised models”. In: *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*. 2009, pp. 611–619.

[14] Nicholas Metropolis and Stanisław M. Ulam. “The Monte Carlo Method”. In: 44.247 (Sept. 1, 1949), pp. 335–341. ISSN: 01621459. DOI: 10/dvn2n8.

[15] Nicholas Metropolis et al. “Equation of State Calculations by Fast Computing Machines”. In: 21.6 (June 1, 1953), pp. 1087–1092. ISSN: 0021-9606. DOI: 10/ds736f.

[16] Todd K Moon. “The expectation-maximization algorithm”. In: *IEEE Signal processing magazine* 13.6 (1996), pp. 47–60.

[17] Thomas Müller. ““Practical Path Guiding” in Production”. In: *ACM SIGGRAPH Courses: Path Guiding in Production, Chapter 10*. Los Angeles, California: ACM, 2019, 18:35–18:48. DOI: 10.1145/3305366.3328091.

[18] Thomas Müller, Markus Gross, and Jan Novák. “Practical Path Guiding for Efficient Light-Transport Simulation”. In: 36.4 (June 2017), pp. 91–100. DOI: 10/gbnvrs.

[19] Thomas Müller et al. “Neural Importance Sampling”. In: (Aug. 11, 2018). arXiv: 1808.03856 [cs, stat].

[20] Christophe Schlick. “An inexpensive BRDF model for physically-based rendering”. In: *Computer graphics forum*. Vol. 13. 3. Wiley Online Library. 1994, pp. 233–246.

[21] Joshua Steinhurst and Anselmo Lastra. “Global importance sampling of glossy surfaces using the photon map”. In: *2006 IEEE Symposium on Interactive Ray Tracing*. IEEE. 2006, pp. 133–138.

[22] Richard S Sutton and Andrew G Barto. “Reinforcement learning: an introduction MIT Press”. In: *Cambridge, MA* (1998).

[23] Eric Veach. “Robust Monte Carlo Methods for Light Transport Simulation”. Ph.D. Thesis. Stanford University, Dec. 1997.

[24] Eric Veach and Leonidas J. Guibas. “Bidirectional Estimators for Light Transport”. In: 1995, pp. 145–167. ISBN: 978-3-642-87825-1. DOI: 10/gfznbh.

[25] Eric Veach and Leonidas J. Guibas. “Metropolis Light Transport”. In: vol. 31. Aug. 1997, pp. 65–76. ISBN: 978-0-89791-896-1. DOI: 10/bkjqj4.

[26] Eric Veach and Leonidas J. Guibas. “Optimally Combining Sampling Techniques for Monte Carlo Rendering”. In: vol. 29. Aug. 1995, pp. 419–428. ISBN: 978-0-89791-701-8. DOI: 10/d7b6n4.

[27] Jiří Vorba et al. “On-Line Learning of Parametric Mixture Models for Light Transport Simulation”. In: 33.4 (Aug. 2014), 101:1–101:11. ISSN: 0730-0301. DOI: 10/f6c2cp.

[28] Jiří Vorba et al. “Path Guiding in Production”. In: *ACM SIGGRAPH Courses*. Los Angeles, California: ACM, 2019, 18:1–18:77. DOI: 10.1145/3305366.3328091.

[29] Bruce Walter et al. “Microfacet Models for Refraction through Rough Surfaces”. In: (Grenoble, France). June 2007, pp. 195–206. ISBN: 978-3-905673-52-4. DOI: 10/gfz4kg.

[30] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

M. Sencer Çavuş

Personal Data

email: scavus@protonmail.com
gitlab: https://gitlab.com/scavus

Research Interests

Main Interests:	Offline & Real-time Rendering, Machine Learning
Other Misc. Interests:	Physics-based Animation, Differential Geometry & Topology, Geometric Mechanics, Category Theory & Functional Programming

Education

Sept. 2016 – Feb. 2020 — Master of Science in Computer Science, **Marmara University**, Istanbul

Thesis: "Adaptive Methods for Photorealistic Image Synthesis"

Researched methods that apply machine learning techniques to enable more robust, adaptive sampling of paths in light transport simulation. Based upon a previously published tree-based path guiding method, developed a simple extension that improves the sampling quality in scenes dominated by glossy surfaces.

Classes taken: Machine Learning, Reinforcement Learning, Digital Image Processing, Computer Vision, Pattern Recognition, Parallel Processing, Evolutionary Computing

GPA: 3.93/4.0

Sept. 2011 – Oct. 2015 — Bachelor of Science in Computer Science, **Marmara University**, Istanbul

Final year project: "A GPU-Accelerated Physically Based Renderer"

Developed a GPU-based uni-directional path tracer (UDPT) to learn; (1) the theoretical foundations and implementation details of physically based rendering and (2) GPU programming by writing code optimized for it. Specifically, implemented a naïve version of UDPT for the GPU and a GPU-optimized version of it, namely wavefront path tracing, and compared their performances.

Link to Code: https://gitlab.com/scavus/Wisard

GPA: 2.75/4.0

Work Experience

Summer 2013 — Intern at SigmaRD, Istanbul

Developed and documented introductory tutorials for depth sensors utilizing OpenNI and Kinect SDK.

`C++, OpenGL, OpenNI, Kinect`

Technical Skills

Programming Languages:	Proficient in C, C++98, Python, Julia Familiarity with Haskell, F#, OCaml, Rust
Frameworks & APIs:	OpenGL 3.3, Direct3D 11, Vulkan, CUDA, OpenCL, MPI
Operating Systems:	Linux, Windows

Languages

English:	C1
German:	B1 (Deutsches Sprachdiplom der Kultusministerkonferenz (DSD I) – 2010)
Turkish:	Native