FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Implementing Probabilistic Connections for Bidirectional Path Tracing in the Mitsuba Renderer

BACHELORARBEIT

zur Erlangung des akademischen Grades

**Bachelor of Science**

im Rahmen des Studiums

**Software and Information Engineering**

eingereicht von

**Nikola Dodik**
Matrikelnummer 1328690

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Hiroyuki Sakai

Wien, 3. September 2017

| Nikola Dodik | Michael Wimmer |
|---|---|

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Implementing Probabilistic Connections for Bidirectional Path Tracing in the Mitsuba Renderer

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Bachelor of Science**

in

**Software and Information Engineering**

by

**Nikola Dodik**
Registration Number 1328690

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Hiroyuki Sakai

Vienna, 3rd September, 2017

| Nikola Dodik | Michael Wimmer |
|---|---|

# Erklärung zur Verfassung der Arbeit

Nikola Dodik
Petra Radjenovica 25,
Banja Luka,
Bosnia and Herzegowina

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. September 2017

Nikola Dodik

# Acknowledgements

I would like to thank my mentor, Hiroyuki Sakai, for going above and beyond in his duties. Thank you for letting me explore my interests and keeping my passion for science alive.

I would also like to thank my professor, Michael Wimmer, for helping me achieve my goals.

Lastly, I would like to thank Dr. Stefan Popov for his endless patience with replying to the many questions I had.

# Kurzfassung

Lichtsimulations-Algorithmen haben sich bei der Nachstellung vieler Effekte, welche in der Natur vorkommen, als effektiv erwiesen, weshalb sie in der Industrie großflächig eingesetzt werden. Dies machte die Entwicklung effizienter und robuster Algorithmen notwendig. PCBPT baut auf dem klassischem BDPT Algorithmus auf. In BPDT wird ein Strahl sowohl vom Sensor als auch vom Emitter verfolgt. Die beiden Strahlen werden dann verbunden, um den Beitrag zu Bildpixeln zu errechnen. PCBPT erweitert diese Idee um das Verbinden mehrerer Emitter Pfade mit einem Subpfad des Sensors und verwendet Importance Sampling um die geeignetsten Emitter Pfade zu wählen. Wir implementierten PCBPT im quelloffenen Mitsuba Renderer und evaluierten und vergleichen den Algorithmus mit BDPT in verschiedenen Szenen.

# Abstract

Light transport simulation algorithms are remarkably adept at recreating a large variety of light phenomena which occur in nature. As such they have seen widespread adoption across the industry, which made it paramount to create efficient and robust algorithms. One recent algorithm which tries to deal with this problem is known as Probabilistic Connections for Bidirectional Path Tracing (PCBPT). It builds upon the classical Bidirectional Path Tracing (BDPT) algorithm. In Bidirectional Path Tracing, a ray is traced from the sensor as well as from the emitter. The two rays are then connected to calculate the light contribution to image pixels. PCBPT extends this idea to support connecting multiple emitter paths to one sensor subpath, and introduces importance sampling as a way of choosing the most suitable emitter paths. Unfortunately, there was no implementation of PCBPT publically available, which is why we implemented it into the open-source Mitsuba renderer. We evaluate the algorithm against standard BDPT on a variety of different scenes. Our comparisons provide insight into what type of scenes PCBPT can help improve and where the additional computational cost presents too much of an overhead.

# Contents

CHAPTER 1

# Introduction

Practical photo-realistic image synthesis, also referred to as physically based rendering, is one of the long-standing goals of computer graphics research. Its applications range from the entertainment industry to architectural and engineering visualizations. As such, it is important to have robust methods which are capable of reproducing a wide variety of light phenomena that occur in nature. Light transport simulation algorithms aim to provide a solution to this problem. As their name suggests, these algorithms simulate how light behaves within a simplified mathematical model. Due to their generality, as well as recent hardware advancements, such algorithms have seen large-scale adoption in the industry during the last few decades.

For a simulation to exactly calculate the light intensity of a light ray leaving a point in a scene, it would have to take into account all of the rays of light incident at that point. Furthermore, in order to know the intensity of any single one of these rays, the simulation would need to take into account the infinitely many rays incident at the points where these rays originated. This implies evaluating infinitely many incoming rays at each point in space. We can intuitively see that the complete calculation of the light distribution is an intractable problem for arbitrary scenes.

A popular class of algorithms, jointly referred to as Monte Carlo rendering, uses statistical methods to estimate the light distribution in a scene in order to estimate how the final image should look. They rely on the more general framework of Monte Carlo integration, which we explain in Section 2.2. At their core, instead of fully calculating a pixel value, they take a finite number of samples from the light distribution and average them. A sample in this sense represents a path generated by a ray of light as it bounces around the scene. In practice, the most common procedure for creating samples involves shooting rays into the scene, starting either from the camera sensor or from the light emitter. Note that both of these procedures are equivalent, as demonstrated by Sen et al. [SCG+05]. Once the ray intersects with an object, we take into account the effect of the object's material properties on the color estimate, and calculate in which direction the ray should

bounce next. We repeat this procedure until some criteria is met. In case that we are shooting rays starting from the camera sensor, the described procedure is known as path tracing [Kaj86]. If rays start from the emitter, the algorithm is referred to as light tracing [DW95].

As the number of samples in these algorithms goes to infinity, the process provably converges to the exact pixel value, and the variance of the pixel value estimator goes to zero. In practice this often implies that we need to take many samples to get a good looking result. As taking samples from the distribution is computationally expensive, certain images often require extensive resources and hundreds of hours of processing time. This makes designing smarter algorithms which, in various ways, reduce the running time while remaining robust and converging to the correct result an important and open problem.

Bidirectional path tracing (BDPT) [LW93, Vea98] approaches this problem by combining path tracing with light tracing. It relies on a method called *multiple importance sampling* in order to optimally combine the strengths of both algorithms. BDPT produces more than one sample from two paths generated with path tracing and light tracing by connecting the two paths at different locations to build new complete paths. In other words, it reuses different sections of each of the two paths to build multiple full paths.

Probabilistic Connections for Bidirectional Path Tracing (PCBPT), as proposed by Popov et al. [PRDD15], builds upon the ideas of BDPT. It extends them to allow for additional reuse of both sensor and eye paths, introduces *importance sampling* to the algorithm as another technique for improving the convergence speed, and modifies multiple importance sampling (MIS) to better suit the problem at hand.

Many algorithms also attempt to accelerate the rendering process by reusing already generated paths to produce multiple samples. For example, even the standard path tracing algorithm [Kaj86] reuses the path used for the last sample when generating the next sample. Combinatorial Bidirectional Path Tracing [PBPP11] relies on the idea of connecting a sensor path to multiple emitter paths and is, in many ways, the predecessor to PCBPT. We discuss the related work in more detail in Section 2.4.

Importance sampling, in the general sense, is a variance-reduction technique which allows us to utilize assumptions about which paths are more "important" to explore. Given that our assumptions were correct, it improves the performance of the algorithm. In the context of path tracing, most importance-sampling methods attempt to solve the same problem – given a vertex in a scene and the incoming direction of the ray to that vertex, decide which direction we should follow next. Importance in this sense is determined by how large the light intensity contribution of a path to the final estimate is.

To give better intuition on the issue, we will shortly describe two common importance sampling strategies found in regular path tracing. In one strategy, we would prefer sampling the ray bounces in directions where the material throughput is high, as we assume a larger total light contribution that way. Another strategy assumes that the contribution will be large if it samples towards the light emitters. We provide the

mathematical background for importance sampling in Section 2.2.1, and in Section 2.3.1, we show how importance sampling can be utilized in the context of path tracing.

In comparison, PCBPT approaches importance sampling from a different angle. For a given sensor path and a set of emitter paths, it tries to choose which emitter paths it should connect the sensor path to, in such a way that the final contribution of the generated path is high. We touch upon this method in more detail in Section 3.2.

Given that we can have many strategies similar to the two in the previous example, it would be beneficial if we could combine more of them to get a superior result than we would have by using any of them individually. Multiple importance sampling (MIS) gives us a solution to this problem. It allows us to decide whether a sample is "important" after it has been created.

MIS plays a significant role in BDPT [VG95, Vea98]. In his seminal work, Veach presented a specific method for MIS, named the balance heuristic, which has strong guarantees on the quality of the result. Popov et al. [PRDD15] build upon their method and solve certain issues which arise due to the reuse of emitter paths. Notably, their extensions can serve as a useful tool not only in the context of PCBPT, but also potentially in any other algorithms which rely on path reuse. We discuss the issues which arise when reusing paths in more detail in Section 2.2.3, and explain how PCBPT solves them in Section 3.3.

At the time of writing, the original PCBPT implementation of Popov et al. [PRDD15] remains closed source and is not available to the public. Implementing the method in a well-established open-source renderer would enable comparisons with other existing methods. It would also enable potential future work to build upon the implementation.

In this thesis, we implement PCBPT into the Mitsuba renderer [Jak10]. The Mitsuba renderer implements many of today's state-of-the-art methods, facilitating easier comparisons with them. Its plugin architecture enables the addition of modules in a non-intrusive manner. Furthermore, Mitsuba provides a framework for bidirectional methods, which abstracts away many of the low-level implementation details. The parallelization abstraction layer also allows us to do comparisons across many different types of rendering setups.

We implement PCBPT into Mitsuba, as detailed in Chapter 4, and provide comparisons with other methods to show how this approach handles a number of complex scenes and scenarios. In our tests, we noticed significant speed-ups of up when using PCBPT on difficult scenes compared to standard BDPT. We also noticed a decrease in the RMSE metric of up to 11% times. We present our results in Section 5.

CHAPTER 2

# Background

## 2.1 Light Transport Model

### 2.1.1 Radiometry

Light transport algorithms are formalized through the rendering equation first published by Kajiya et al. [Kaj86]. To understand the rendering equation, in the next few paragraphs we will shortly touch upon what it is trying to measure.

In simple terms, we are interested in somehow measuring the light intensity coming to a pixel on the camera lens. In order to define what "light" actually means in more formal terms, we rely on a mathematical model called radiometry.

A quantity of interest within this model is the *radiant flux*. The radiant flux is defined as the total amount of energy passing through a region of space in an instant of time. The measured flux changes if we scale the region of space which we are observing. Instead, a somewhat more suitable measurement would be the *irradiance*. Irradiance is the density of the flux arriving at a surface of some defined area, or, in other words, the average energy per unit area. More precisely, we are interested at the area density of flux at every single point in the scene. To this extent, we formally define the irradiance at a point $p$ as the differential flux per differential unit area $E(p) = \frac{d\Phi(p)}{dA}$. Note that the same measure describes not only the energy density arriving at a point, but also the energy leaving a point. This related quantity is named the *radiant exitance*.

However, irradiance and radiant exitance are still unsuitable for our purpose. The problem is that these quantities do not take into account the incoming direction of the light. For example, imagine looking at a point and trying to measure how bright it is with a camera sensor. If if that camera sensor was to measure the radiant exitance at that point, it would be taking into account all of the light leaving that point, even that which is not going towards it. Imagine the situation where the radiant exitance is high, but all

of the light is headed in some direction other than where we are standing. Our sensor would register this spot as bright, even though we would actually see it as completely black.

The quantity we are really interested in is called *radiance*. Intuitively, radiance is the energy incoming at a point from a certain direction. More formally, radiance is defined as the radiant flux, per unit projected area, per unit solid angle, or in other words, the irradiance or radiant exitance per unit solid angle,

$$E(p) = \frac{d\Phi(p)}{d\omega dA^{\perp}}. \tag{2.1}$$

This quantity is quite useful, since it represents the smallest building block of our model. All other quantities can be derived once the radiance is known. Furthermore, assuming a pixel is a mathematical point, calculating its value is exactly equivalent to calculating the incoming radiance at that point. Another useful property is that the radiance exiting a point and traveling down an imaginary light ray until it hits some other object will be the same at the beginning and at the end of that light ray. For a more in-depth discussion on the topic of radiometry, refer to *Physically Based Rendering: From Theory to Practice* (PBRT) [PJH16], section 5.4.

At this point, we know that we need to be able to calculated the radiance coming to a point to be able to calculate a pixel's value. We also know that the radiance incoming to the pixel is equivalent to the radiance leaving the point in the scene "in front" of that pixel. As we will see in Section 2.1.2, to know the radiance leaving a point, we would need to know all of the radiance incoming at that point. The rendering equation allows us to formalize all of these ideas, which is, in turn, the first step towards solving this problem.

### 2.1.2 The Path Space Rendering Equation

At its core, the rendering equation allows us to calculate the radiance leaving a point in the scene and going along a certain direction. Hence, we can theoretically use the rendering equation to calculate the incoming radiance to each pixel.

We will here discuss the path-space form of this equation as presented by Veach [Vea98], since *Bidirectional Path Tracing* (BDPT) and consequently *Probabilistic Connections for Bidirectional Path Tracing* (PCBPT) derive from it more naturally. As previously mentioned, the rendering equation says that the outgoing radiance at a point in some direction depends on all of the incoming radiance from all possible directions to that point. This means that the rendering equation needs to consider the distribution of radiance in the entire scene.

To model the radiance in a scene, we also need to introduce the concept of *paths*. A path of length $l$, $\bar{x}_l$, is a tuple of points in 3D space, $\bar{x}_l = \boldsymbol{x}_0\boldsymbol{x}_1 \dots \boldsymbol{x}_l$, which lay on surfaces in the scene. The set of all paths of length $l$ is symbolized with $\Omega_l$. Hence, $\Omega_l$ represents

the $l$-dimensional Cartesian product over the surfaces in the scene. Consequently, *path space* is defined as the set of all paths in a scene,

$$\Omega = \bigcup_{l=1}^{\infty} \Omega_l. \tag{2.2}$$

Figure 2.1 shows an example of a path, $\bar{x}_3 \in \Omega_3$ in a 2D scene.

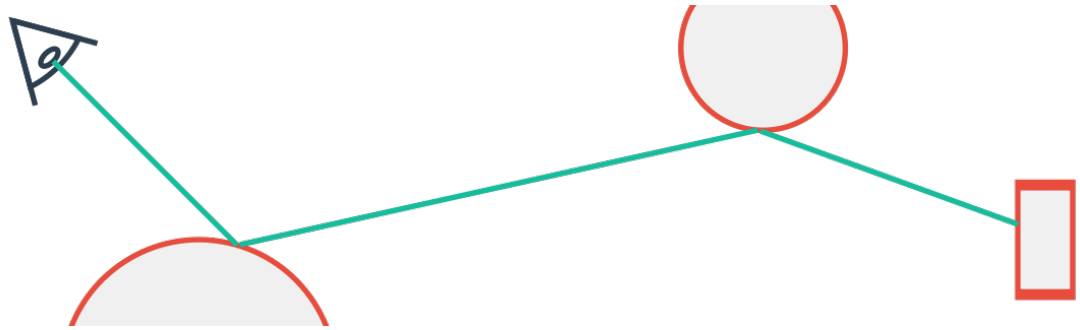


Figure 2.1: A path of length 3 starting at the sensor (symbolized by the eye), visualized using PathGraph [Dar].

Each path carries a certain amount of radiance, $I$, also known as the path contribution. We, therefore, want to know the value of $I$ coming to a pixel from some point in the scene. This means that the final value is a result of accumulating the contributions of all paths which have these two points in common. From there, we can see that the total radiance coming to a pixel intuitively equals the integral of the contributions over the subset of path space containing these two points. More formally, this can be written as a sum of total contributions of paths of increasing length,

$$I = \sum_{l=1}^{\infty} I_l, \tag{2.3}$$

where $I_l$ represents the contribution of all paths of length $l$. We can further define the total contribution of all paths of length $l$ which contribute to $I$ as

$$I_l = \int_{\Omega_l} f(\bar{x}) \mathrm{d}\mu_l(\bar{x}), \tag{2.4}$$

where $f(\bar{x})$ abstracts the contribution of a single path $\bar{x}$, and the *area-product measure* is defined as $\mathrm{d}\mu_l(\bar{x}) = dA(\boldsymbol{x}_0) \dots dA(\boldsymbol{x}_l)$.

Here, we will provide a basic definition of $f(\bar{x})$ for completeness, and refer the interested reader to PBRT [PJH16], section 14.4. Following Popov et al. [PRDD15], we define $f(\bar{x})$ as

$$f(\bar{x}) = L_e(\boldsymbol{x}_1 \to \boldsymbol{x}_2)\Pi_l(\bar{x})G(\boldsymbol{x}_{l-1} \leftrightarrow \boldsymbol{x}_l)W(\boldsymbol{x}_{l-1} \to \boldsymbol{x}_l). \tag{2.5}$$

$L_e(\boldsymbol{x}_1 \to \boldsymbol{x}_2)$ represents the emittance, i.e. the radiance emitted from point $\boldsymbol{x}_0$ to point $\boldsymbol{x}_1$. $W(\boldsymbol{x}_{l-1} \to \boldsymbol{x}_l)$ is the importance function [Vea98] determining the sensor's sensitivity for a given light ray. $G(\boldsymbol{x}_x \leftrightarrow \boldsymbol{x}_y)$ is the generalized geometric term, which includes the geometric relations as well as the visibility between two points.

The path throughput is defined as

$$\Pi_l(\bar{x}) = \prod_{i=0}^{l-3} G(\boldsymbol{x}_i \leftrightarrow \boldsymbol{x}_{i+1})\rho(\boldsymbol{x}_i \to \boldsymbol{x}_{i+1} \to \boldsymbol{x}_{i+2}), \tag{2.6}$$

with $\rho$ being the bidirectional scattering distribution function (BSDF) of the material. The BSDF of a given material determines the probability of a light ray scattering in a certain direction at a point, given some incoming angle for the light ray.

As it turns out, calculating a solution to this equation is an immensely difficult problem. The mathematical background for the most commonly used approach is known as Monte Carlo Integration. In the next few sections, we will explain the theoretical background for this method, as well as a few ways of improving it. Finally, we show how it can be used to give us an estimate for the solution of the rendering equation.

## 2.2 Monte Carlo Integration

The rendering equation is, in general, not known to be analytically solvable. Due to this fact, we have to turn to numerical methods instead. Typically, we employ a statistical method known as Monte Carlo (MC) integration. The result of this method is an estimate of the true value of a pixel.

Let $\tilde{I}$ be the Monte Carlo estimate of the integral $I$ of a function $f(x)$. Let $X_i, i \in [1, \ldots, N]$ be $N$ independent and identically distributed (i.i.d.) random variables and $x_i = X_i$ their realizations. Let their probability distribution function (PDF) be $p(x_i) = P(x_i = X_i)$. Then, the Monte Carlo estimate of the integral of the function is

$$\tilde{I} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)} \approx \int_D f(x)\mathrm{d}x. \tag{2.7}$$

Estimating an integral with this method in practice means that we first generate random samples $x_i$ from some distribution $p(X_i)$ on the domain $D$ over which we want to integrate. Then we evaluate the function at the sampled point $f(x_i)$ and calculate the ratio $\frac{f(x_i)}{p(x_i)}$. To get the final estimate, we do this $N$ times and average the results.

We will first provide some mathematical intuition on how and why Monte Carlo integration works. First, let's look at one sample and the expected value of the ratio itself. This

ratio is in itself an unbiased estimator of the actual value of the integral. This is easily provable as well,

$$\mathbb{E}\left[\frac{f(X)}{p(X)}\right] = \int \frac{f(x)}{p(x)} p(x)\,\mathrm{d}x = \int f(x)\,\mathrm{d}x = I. \tag{2.8}$$

However, the error here is potentially extremely large since the sampled value could be far from the expected value. It would be useful to have a way of making this estimate converge to the correct value.

To this extent, Monte Carlo integration relies on the Law of Large Numbers. For a sequence of i.i.d. random variables $X_i$, $i \in [1, \ldots, \infty)$, with the expected value $\mathbb{E}[X_i] = \mu$ and the variance $\mathrm{Var}[X_i] = \sigma^2 < \infty$, the Law of Large Numbers asserts that the sample mean of the first $N$ elements of the sequence, $\bar{X}_N = \frac{1}{N}\sum_1^N X_i$, converges in the probability to the actual expected value, $\bar{X}_N \xrightarrow{P} \mu$.

Combining this with the result above, it is visible that the Monte Carlo estimate of the integral converges in probability to the true result as the sample size tends to infinity,

$$\lim_{N\to\infty} \mathrm{P}(|I - \tilde{I}| < \epsilon) = 1. \tag{2.9}$$

Monte Carlo integration has a major advantages over many other numerical integration methods. In Monte Carlo integration, the rate of convergence of the method remains constant as the dimensionality of the integration domain increases. This is not the case for most other numerical integration algorithms, which often suffer from the curse of dimensionality, where the number of function evaluation increases exponentially with the number of dimensions. In rendering, we are dealing with integrals over highly dimensional spaces, where the dimensionality of $\Omega_l$ tends to infinity as $l \to \infty$.

However, Monte Carlo integration also comes with a downside, namely the fact that the number of samples required to evaluate an integral and get a satisfying result can be very high. This, in turn, requires extensive computational resources. To help mitigate these issues, different variance-reduction techniques were devised. One such technique which allows us to rely on certain heuristics to choose a "better" sampling distribution is called importance sampling.

### 2.2.1 Importance Sampling

To understand the usefulness of importance sampling, we need to understand the problems it tries to solve. The first issue with MC integration, which we touched upon, is that of the speed of convergence. Additionally, cases where Monte Carlo integration fails to converge at any reasonable rate occur often, even in simple scenarios. Furthermore, as we will see in this section, the choice of the sampling PDF matters a lot, and a bad choice can lead to even worse results than simply using a uniform distribution.

An example of a simple scenario which is difficult for MC integration is a function with a very high, narrow, peak, but a very wide integration domain. Figure 2.2 demonstrates evaluating such an example with a uniform PDF. Notice that most of the samples are distributed across the entire domain, where the function value is very low. This means that the value of the integral will be underestimated until we eventually "hit" the peak.

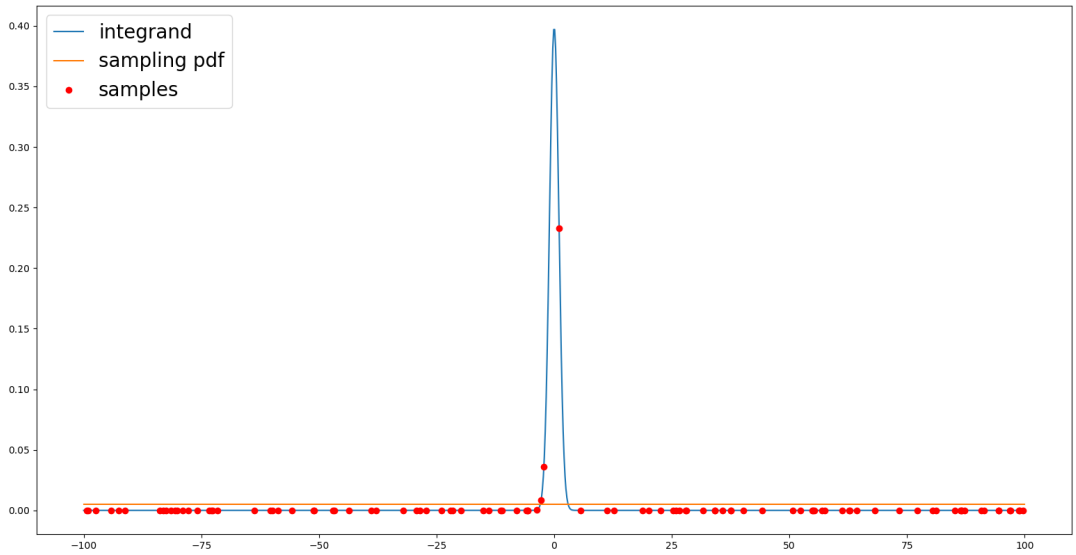



Figure 2.2: Using a uniform sampling distribution can often lead to poor performance. Only few out of the 100 samples managed to take into account the peak of the function.

Instead, we would like to prefer our sampling scheme to somehow adapt to the shape of the integrand. Intuitively, we want to make it so that each section of the function is represented in the samples proportional its contribution. This would make sure we sample the peaks more often, while giving less significance to the low contribution areas. As it turns out, this is also the optimal strategy to choosing a sampling distribution from a variance-reduction point of view.

Importance sampling is based on the idea that the variance of the Monte Carlo estimator is reduced when the sampling distribution is similar to the integrand. Less variance implies less noise, which in turn implies faster convergence. Let's assume that we know this perfect distribution $p(x_i)$ and that $p(x_i) = cf(x_i)$. Since $p(x_i)$ needs to integrate to 1, $c$ needs to equal

$$c = \frac{1}{\int f(x_i)\mathrm{d}x}. \tag{2.10}$$

Simple algebraic manipulation can show that the variance of the estimator from equation 2.8 when using such a PDF, always equals 0, no matter what the actual sample is, since

the ratio always evaluates exactly to the integral,

$$\frac{f(x_i)}{p(x_i)} = \frac{1}{c} = \int f(x_i)\mathrm{d}x. \tag{2.11}$$

While this works in theory, in practice we don't know the exact PDF, since knowing it would also imply knowing the integral. However, even if the distribution is not a perfect match for the integrand, the rate of convergence will still increase if they are similar. This means that we can rely on certain assumptions about the shape of the integrand and, as long as they are not wrong, the variance will be decreased.

In their essence, all importance-sampling methods are based on this seemingly straightforward idea. However, choosing the right heuristic can be a difficult problem in itself. When it comes to the rendering equation, the integrand usually depends on many factors, such as material properties, the distribution and shape of lights in the scene and the occlusions generated by objects in the scene. There are usually multiple strategies to choose from and no way to know which one is better for the current situation. In the next section, we will discuss multiple importance sampling as a method for combining multiple different strategies, to produce a result superior to any of them individually.

### 2.2.2 Multiple Importance Sampling

As already mentioned, choosing the right sampling distribution is a difficult problem, as choosing an unsuitable one can lead to an increase in variance. Figure 2.3 shows a simplified illustration of this problem using a 1D function. In this example, we assume that we do not know the actual shape of our integrand, shown in blue. However, we do have two heuristics, from here on referred to as strategies, on which we can rely. Each of the strategies makes valid assumptions about where one of the peaks is located, and shapes its PDF in such a way that it perfectly importance samples that peak.

However, if we were to use the PDF in the left image to sample the integrand and end up producing a sample inside the peak on the right-hand side, we would get a very large sample value, which would increase the variance of our estimator. For illustrative purposes, we can assume that the probability for sampling that point equaled 0.01, whereas the sample value equaled 3. We can calculate the MC estimate (with $N = 1$), $\frac{f(x_i)}{p(x_i)} = \frac{3}{0.01} = 300$, which is relatively large compared to the true value of the integral in the image which equals 20. This shows how having a low probability of sampling a high contribution region can pose a problem, even though the PDF approximates another part of the integrand rather well.

Let us now look at this problem more formally. Without loss of generality let us assume only two sampling strategies, $A$ and $B$, with their corresponding sampling PDFs, $p_A(x)$ and $p_B(x)$. Our goal is to avoid an increase in variance for the cases where the probability of sampling a highly contributing part of the integrand is low.

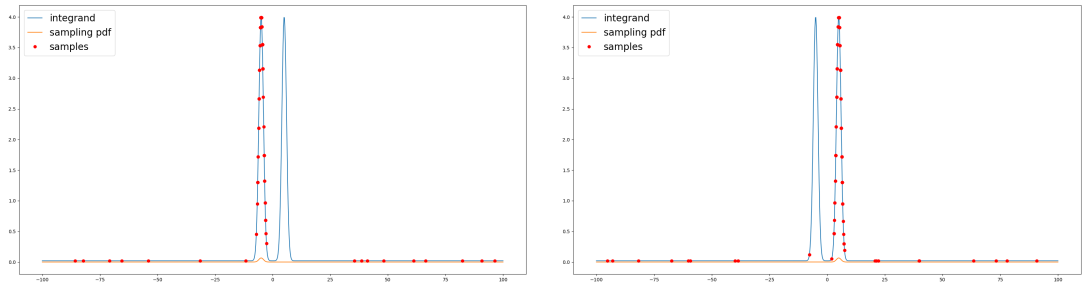



Figure 2.3: Sampling with two different distributions, each approximating one one the peaks. It is visible that both strategies undersample one of the peaks.

A way of achieving this is by relying on multiple importance sampling (MIS) [Vea98]. MIS allows us to retroactively mitigate the negative effect of an inappropriate sampling strategy. In this context, retroactively means that we first generate a sample from each of the strategies, and then assign a weight to each sample with the goal to decrease overall variance. We will refer to the sample weights for strategies $A$ and $B$ as $w_A(x)$ and $w_B(x)$ respectively.

Mathematically, MIS is based on the simple idea that we can separate our estimator into a weighted sum of two different estimators with different PDFs. The only conditions imposed are that every non-zero point on the integrand must have a non-zero probability of being sampled by at least one of the PDFs, and that the weights of the estimators for sampling the sample point sum up to one. This also allows either of the PDFs to be zero in some parts of the function.

Therefore, we can turn the single value estimator in equation 2.8 into an MIS estimator, by asserting the conditions that $w_A(x) + w_B(x) = 1$, wherever $f(x) \neq 0$, and $w_A(x) = 0$ and $w_B(x) = 0$ where $p_A(x) = 0$ and $p_B(x) = 0$. Assuming those conditions hold, and assuming we have made two samples, $a$ from $p_A(x)$ and $b$ from $p_B(x)$, it can be shown that the following is an unbiased estimator of the integral:

$$\tilde{I} = w_A(a)\frac{f(a)}{p_A(a)} + w_B(b)\frac{f(b)}{p_B(b)}. \tag{2.12}$$

The conditions can be justified by looking at what would happen to the estimate in a region where the two sampling strategies overlap as the number of samples tends to infinity. At infinity, the estimate for the integral within the sampled domain will converge to the true value for both sampling strategies. If the weights did not equal one when added together, the final estimate would be inaccurate.

To offer some intuition on why this works, we will show that the expected value of this estimate is the integral itself, in a similar manner to equation 2.8,

$$\begin{aligned}
\mathbb{E}[\tilde{I}] &= \mathbb{E}\left[w_A(x)\frac{f(x)}{p_A(x)} + w_B(x)\frac{f(x)}{p_B(x)}\right] \\
&= \int w_A(x)f(x)\mathrm{d}x + \int w_B(x)f(x)\mathrm{d}x \\
&= \int w_A(x)f(x) + w_B(x)f(x)\mathrm{d}x \\
&= \int (w_A(x) + w_B(x))f(x)\mathrm{d}x \\
&= \int f(x)\mathrm{d}x \\
&= I.
\end{aligned} \tag{2.13}$$

Now that we have defined the general MIS framework, we are still left with the question on how to effectively combine the samples, i.e. how we should calculate our weights. This problem is solved in a provably optimal way for i.i.d. samples, by a method known as the *balance heuristic*.

To understand how the balance heuristic works, let us observe sample $a$, generated using strategy $A$. The balance heuristic makes its judgment by looking at what the probability of drawing that sample would have been had it been generated by using the other strategy as well. In other words, it not only looks at $p_A(a)$, but also observes $p_B(a)$, when calculating the weights. In our two-strategies, one-sample example from equation 2.12, the formula for the balance heuristic weight of the first estimator would be defined as

$$w_A(a) = \frac{p_A(a)}{p_A(a) + p_B(a)}. \tag{2.14}$$

The assumption here is that at least one of our strategies is high where the integrand is high. When this assumption holds, the balance heuristic prevents high variance, since the low-probability high-contribution samples get weighted down. This also implies that samples with a high probability relative to both sampling distributions get weighted more.

In our example, if $a$ had been sampled from an area where $p_A(a) = cf(a)$, $c \in \mathbb{R}$, and $p_B(a)$ was low, the ratio from equation 2.14 would be close to one, which is what we want. If, however, $a$ was sampled in an area where $p_A(a)$ was low, but $f(a)$ was high and $p_B(a) = cf(a)$, a situation which would normally lead to a variance increase, the weight $w_A(a)$ would be close to 0, reducing the potential negative effects of such a sample.

It is straightforward to extend these ideas to a multi-sample estimator which relies on many different strategies instead of just two. Here we only present the equation, but all of the intuition as well as the arguments apply just as easily to this context. Assuming

$l$ different sampling strategies, where $t$ is the current strategy and $x_j$ is the j$^{\text{th}}$ sample taken from that strategy, we define the contribution of that sample to the final estimate as

$$\tilde{I}_t(x_j) = w_t(x_j)\frac{f(x_j)}{p_t(x_j)} \tag{2.15}$$

The balance heuristic would then be defined as

$$w_t(x_j) = \frac{N_t p_t(x_j)}{\sum_{i=1}^{l} N_i p_i(x_j)}, \tag{2.16}$$

where $N_t$ and $N_i$ represent the number of samples in strategies $t$ and $i$, respectively. The full MIS estimator is then given by,

$$\tilde{I} = \sum_{t=1}^{l} \frac{1}{N_t} \sum_{j=1}^{N_t} \tilde{I}_t(x_j) \tag{2.17}$$

As we have seen, the MIS estimator is an important variance-reduction technique. Combining it with Bidirectional Path Tracing, as was done by Veach [VG95], makes for a powerful and robust rendering algorithm. However, as already noted, the balance heuristic is only optimal under the assumption that the samples come from i.i.d. random variables. In the next section we will look at what happens to the error produced by the estimator if the samples are not independent.

### 2.2.3 Sample Correlation

So far we have discussed the MC estimator in the context of samples from i.i.d. random variables. To see why the samples being independent is important, we need to look at the variance of the MC estimator. The variance is given by

$$\operatorname{Var}\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}\right] = \frac{1}{N^2}\sum_{i=1}^{N}\operatorname{Var}\left[\frac{f(X_i)}{p(X_i)}\right] + \frac{1}{N^2}\sum_{\substack{i,j=1 \\ i\neq j}}^{N}\operatorname{Cov}\left[\frac{f(X_i)}{p(X_i)}, \frac{f(X_j)}{p(X_j)}\right] \tag{2.18}$$

If two random variables $X_i$ and $X_j$ are independent, the covariance between them is zero. This means that for i.i.d. samples, the second term is already minimized and the entirety of the error comes from the first term.

The balance heuristic tries to mathematically minimize the variance of the estimator, and to do that, it assumes that the covariance is zero. In the original paper, Popov et

al. [PRDD15] note that these variance-reduction guarantees no longer hold when the covariance term is present.

It should be noted that sometimes, correlated samples are desirable. In some scenarios, allowing some covariance to be present can enable other improvements which in turn reduce the overall error. These improvements could either come from reducing the variance to such a degree that the covariance term is negligible, or from simply allowing for quicker evaluation, i.e., making more samples possible in the same amount of time. In fact, many Monte Carlo integration techniques, such as stratified sampling or the Metropolis-Hastings algorithm rely, at their core, on generating correlated samples.

In the following sections, we introduce the path tracing algorithm as a way of using MC integration to estimate the rendering equation, and show how importance sampling and multiple importance sampling can be integrated into the algorithm. We also discuss the Bidirectional Path Tracing algorithm as the basis for Probabilistic Connections for Bidirectional Path Tracing, and discuss how MIS can be used to improve the original algorithm as presented by Lafortune and Willems [LW93].

## 2.3 Solution techniques

### 2.3.1 Path Tracing

Path tracing is an algorithm which tries to estimate the rendering equation (as seen in equations 2.3 and 2.4) by relying on Monte Carlo integration. On a higher level, the idea is to produce Monte Carlo samples by sampling the path space in order to evaluate the incoming radiance at a pixel. As with the general MC estimator, the algorithm averages all of the contributions. The MC estimate for the contribution of paths of length $l$ is therefore equal to

$$\tilde{I}_l = \frac{1}{N} \sum_{j=1}^{N} \frac{f(\bar{x}_j)}{p(\bar{x}_j)}, \tag{2.19}$$

where $\bar{x}_j$ is the $j^{\text{th}}$ generated path of length $l$, $f(\bar{x}_j)$ is the contribution of that path, and $p(\bar{x}_j)$ is the probability of sampling that path.

In practice, the naive path tracing algorithm works as follows: To generate a path, we start off by generating a ray starting from a pixel, and following it into the scene until it intersects with an object. This represents a path of length one, i.e. $\bar{x} \in \Omega_1$. If the object we intersected is emissive the contribution will be non-zero and we add it to the estimate.

Next, we generate another ray starting from the intersection point by randomly sampling the ray's direction from a spherical distribution around the point. Once we have calculated the nearest intersection point of this ray, we have generated a path from $\Omega_2$. In theory, we would need to continue increasing the path length $l$ so that $l \to \infty$. In practice, we continue generating paths of increasing length until some breaking condition is met. To

get the final image, the entire procedure is repeated a desired number of times for each pixel.

The mentioned breaking condition could either be a manually chosen limit on the path length, which would make the estimator biased, or it could be based on a technique named Russian Roulette. For more information on this technique, we refer the reader to section 2.7.2 of Veach's PhD thesis [Vea98].

In order to evaluate the equation 2.19, we need to divide the contribution of the path, $f(\bar{x})$, defined in 2.5 and 2.6, by the probability of sampling the entire path. We define the probability of generating the path $\bar{x}$ of length $l$, with the vertices $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_l)$, as

$$p(\bar{x}) = \sum_{i=1}^{l} p(\boldsymbol{x}_i), \tag{2.20}$$

where $p(\boldsymbol{x}_i)$ represents the probability of sampling the vertex $\boldsymbol{x}_i$.

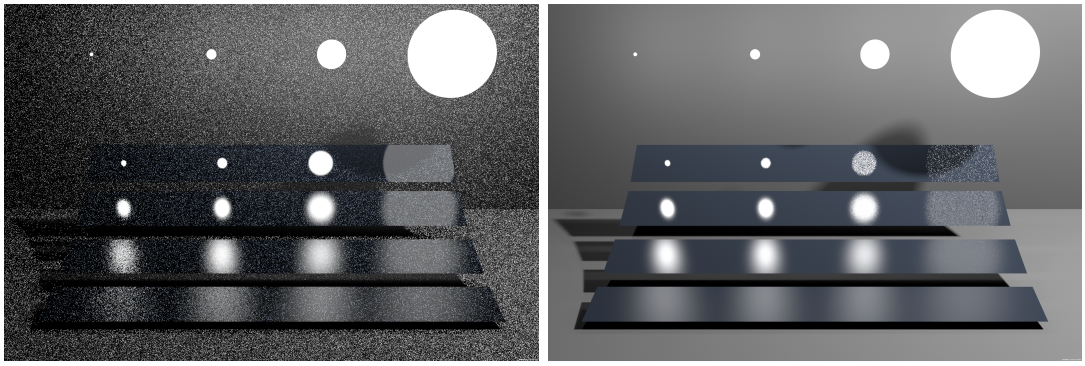
One thing to note is that we have a lot of freedom in choosing the sampling distribution from which we generate directions for rays in paths longer than one. This is where importance sampling comes into play in the context of rendering. Instead of generating a uniform point on a sphere, it might be beneficial to sample the ray in a direction which points towards a light source. If the material properties are suitable (for example, if we are sampling from a point on a diffuse material), we will always generate a path with a non-zero contribution this way.

On the other hand, highly reflective materials generate zero contributions for all but a tiny subset of possible directions. Another way of putting this is that for such materials, $\rho$ from equation 2.6 equals zero for almost all points in the scene. In this case, it would make little sense to sample the light source, as the light contribution would almost certainly end up being attenuated to zero by the material properties. Instead, we would like to have a high probability of sampling along the subset of directions which generate non-zero contributions.

These two importance-sampling schemes are often used in conjunction within the path tracing algorithm. The first strategy, named light sampling, is based on the heuristic that the contribution will be high in directions pointing towards the light sources. The second strategy, called BSDF sampling, assumes that the contribution will be high in directions with low material attenuation.

In Figure 2.4, we see the results of using these sampling strategies, in an example analogous to the that in Figure 2.3. It is clear from the images that using either of them alone leads to suboptimal results.

As with our example from Section 2.17, using the balance heuristic greatly improves the convergence in this scenario as well. To use MIS in this context, we would perform path tracing by generating samples from both of the strategies and using the balance heuristic to weight them. Figure 2.5 demonstrates the benefits of this approach.



(a) Sampling the BSDF. (b) Sampling the emitters.

Figure 2.4: Comparison of two importance sampling-strategies. Notice that they both have different deficiencies. The smaller spheres are better sampled by light sampling (right), whereas the highly reflective planes are better served by sampling the BSDF (left). The scene was originally presented by Veach and Guibas, [VG95], here rendered using Mitsuba [Jak10].

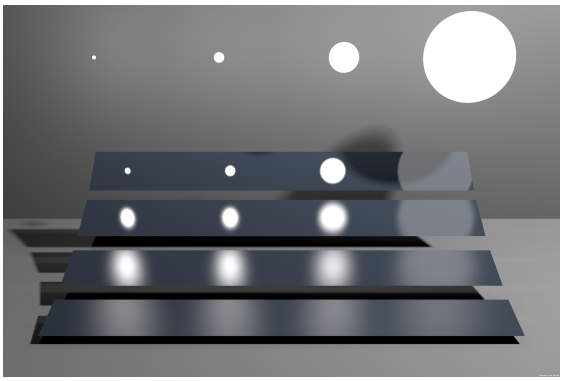


Figure 2.5: Using MIS to combine light sampling and BSDF sampling. Notice that the bright pixels caused by high variance are now gone, and the image has significantly less noise.

### 2.3.2 Bidirectional Path Tracing

The original Bidirectional Path Tracing (BDPT) paper, as presented by Lafortune and Willems [LW93], presents two major advantages over the standard path tracing algorithm. In standard path tracing, if a scene had an occluded emitter, it could be hard for the sensor ray to reach it. This could potentially waste a lot of computational resources as only a small subset of all sensor rays would actually make a non-zero contribution to the pixel value. BDPT tries to deal with this problem by creating an emitter subpath which would often "escape" the occlusion. The second contribution of BDPT is that it reuses the generated sensor and emitter subpaths to generate more full paths than possible in

regular path tracing.

The basic idea behind the algorithm is to start tracing rays not just from the sensor (path tracing), but from the emitter as well (light tracing). Once the tracing procedure is finished, we have generated a set of emitter subpaths of increasing length, as well as a set of sensor subpaths of increasing length. We can connect each sensor subpath to each emitter subpath to create full paths.

For illustrative purposes, imagine having traced two paths of infinite length: a sensor path $\bar{z}$ with vertices $(\boldsymbol{z}_1, \boldsymbol{z}_2, \dots)$, and an emitter path $\bar{y}$ with vertices $(\boldsymbol{y}_1, \boldsymbol{y}_2, \dots)$. In practice, this process actually generates two sets of paths of increasing length, called subpaths. Since they were generated by the same procedure as in path tracing, each subpath includes all of the vertices of the subpath generated before it, plus one additional vertex,

$$\mathcal{S} = \{(\boldsymbol{z}_1), (\boldsymbol{z}_1, \boldsymbol{z}_2), \dots\}, \text{ and} \tag{2.21}$$
$$\mathcal{E} = \{(\boldsymbol{y}_1), (\boldsymbol{y}_1, \boldsymbol{y}_2), \dots\}. \tag{2.22}$$

From here on, we could simply follow the same procedure as in naive path tracing and add the contributions of each of those paths to the estimate. However, a much more powerful technique relies on the idea that we can connect two subpaths to create a longer full path. To generate path of length $l$, such as in equation 2.19, we simply prepend the vertices of the emitter subpath of length $s$ to the vertices of the sensor subpath of length $t$,

$$\bar{x} = (\boldsymbol{z}_1, \dots, \boldsymbol{z}_t, \boldsymbol{y}_s, \dots, \boldsymbol{y}_1). \tag{2.23}$$

BDPT uses this idea to generate many full path contributions. In fact, it does a Cartesian product over the two sets and generates $|\mathcal{S} \times \mathcal{E}|$ full paths by combining each emitter subpath with each sensor subpath.

The contribution of such a path is calculated in the same fashion as the contribution of a path in path tracing. To be able to do this, we need to pay attention while creating the emitter subpaths to keep track of the BSDF function values as if they were evaluated from the reverse direction. The final contribution of a full path then evaluates to

$$f(\bar{x}) = L_e(\boldsymbol{y}_1 \to \boldsymbol{y}_2)\Pi_s(\bar{y})f^{\text{conn}}(\bar{y}, \bar{z})\Pi_t(\bar{z})W(\boldsymbol{z}_2 \to \boldsymbol{z}_1), \tag{2.24}$$
$$f^{\text{conn}}(\bar{y}, \bar{z}) = \rho(\boldsymbol{y}_{s-1} \to \boldsymbol{y}_s \to \boldsymbol{z}_t)G(\boldsymbol{y}_s \leftrightarrow \boldsymbol{z}_t)\rho(\boldsymbol{y}_s \to \boldsymbol{z}_t \to \boldsymbol{z}_{t-1}). \tag{2.25}$$

Calculating the path probability is straightforward. We use $p^{\rightarrow}(\bar{x})$ to symbolize the probability of a path being sampled in the direction away from the emitter, and $p^{\leftarrow}(\bar{x})$

for the direction away from the sensor. The probabilities of sampling the subpaths, $p^{\rightarrow}(\bar{y})$ and $p^{\leftarrow}(\bar{z})$, are calculated as a product of the probabilities of sampling each vertex in the path, in the same way as in regular path tracing. The connection between the subpaths is made deterministically, which means that the probability of connecting $\boldsymbol{y}_s$ with $\boldsymbol{z}_t$ equals one. This makes the probability of sampling the full path simply equal to

$$p(\bar{x}) = p(\bar{y})p(\bar{z}). \tag{2.26}$$

The big improvement to BDPT was brought about by Veach and Guibas [VG95] after they introduced MIS to the algorithm. The strategies in the sense of BDPT are given by the location of the deterministic step. Therefore, a path of length $l = s + t$ could have been generated in $l + 1$ different ways, $(s = 0, t = l), (s = 1, t = l - 1), \ldots, (s = l, t = 0)$, where $t = 0$ is the case where the emitter path intersected the camera directly, and $s = 0$ is the strategy of standard path tracing. Note that we can evaluate the probability of each full path being generated by any of these strategies by also calculating what the probability of sampling the vertices would have been, had they been sampled from the other direction.

Since the sampling strategy depends on the length of the subpaths, we will opt to use $t$, representing the length of the sensor subpath, to indicate the current strategy. Following equation 2.15, we define the MIS contribution of the $j^{\text{th}}$ sampled path $\bar{x}_j$

$$\tilde{I}_t(\bar{x}_j) = w_t(\bar{x}_j)\frac{f(\bar{x}_j)}{p_t(\bar{x}_j)}, \tag{2.27}$$

and the full MIS estimator for paths of length $l$ as

$$\tilde{I}_l = \sum_{t=1}^{l} \frac{1}{N_t} \sum_{j=1}^{N_t} \tilde{I}_t(\bar{x}_j), \tag{2.28}$$

where $N_t$ is the total number of samples made from strategy $t$. We define the balance heuristic for paths analogous to equation 2.16 as

$$w_t(\bar{x}_j) = \frac{p_t(\bar{x}_j)}{\sum_{i=1}^{l} p_i(\bar{x}_j)}. \tag{2.29}$$

In order to make the evaluation more efficient and numerically stable, Veach and Guibas also presented a recursive evaluation scheme, which allows us to calculate the balance heuristic weights in linear complexity, instead of the $O(n^4)$ which would be required by a naive implementation [VG95].

BDPT as presented here is a powerful and robust algorithm which is capable of handling many different scenes. However, some situations still remain solved in a suboptimal

fashion. This is apparent in highly occluded scenes, or scenes with specular-diffuse-specular paths. The first issue is directly addressed by Popov et al. in Probabilistic Connections for Bidirectional Path Tracing [PRDD15]. In the following chapter, we will explore the theoretical background of this algorithm, and how it contributes to solving this problem.

## 2.4 Related Work

Bidirectional path tracing [LW93, VG95] introduced many ideas upon which Probabilistic Connections for Bidirectional Path Tracing (PCBPT) builds. Lafortune et al. [LW93] originally provided with the general framework for BDPT. In BDPT, one sensor and one emitter path are generated, and each pair of subpaths from the two generated paths were connected to create full paths. PCBPT generalizes this idea so that many emitter paths are generated and kept in a cache, and, during the evaluation, we connect to many of them.

Importance sampling introduces the idea of shaping the sampling probability distribution functions in such a way that variance is minimized. There has been a lot of research regarding effective importance-sampling strategies.

The idea was first introduced by Jensen [Jen95], who stored a hemispherical histogram of the particle distribution in a photon map, as viewed from a point in the scene, to approximate the light distribution in the scene. Hey and Purgathofer [HP01] recognized that a fixed bin size histogram solution was suboptimal, and opted for a an adaptive approach, where the histogram bins change in size to allow for a more finely grained probability distribution for sampling. Vorba et al. [VKv+14] use Gaussian Mixture Models to learn the particle distribution while rendering. More recently, Müller introduced a novel data structure known as the SD-tree which is capable of representing an approximation of the light distribution [MGN17].

All of these methods try to create an importance-sampling scheme which tries to best choose the what direction which should be sampled next, given a current path vertex and the incoming direction to that path vertex. Note that PCBPT is orthogonal to any such method, since it only importance samples light subpaths and has no effect on how the paths are generated.

Veach and Guibas [VG95] introduced multiple importance sampling (MIS) in the context of BDPT, and showed that it is an effective way of combining the many different sampling strategies employed by the algorithm. The MIS weights suggested by them perform suboptimally when sample correlation is present. As this is the case with PCBPT, Popov et al. [PRDD15] extend the idea to perform well even with sample correlation present.

Importance Caching [GKPS12] has a number of similar ideas to PCBPT, except in the context of Instant Radiosity [Kel97]. In their work Georgiev et al. cache the probability distribution functions (PDFs) for sampling virtual point lights at a discrete set of importance-cache points. Each PDF optimizes for different lighting conditions

and represents a different sampling strategy. Additionally, they combine the strategies in a multiple importance-sampling scheme. Note that a similar MIS approach could be combined with PCBPT, although this is not mentioned in the original paper.

Other works which sample connections in the context of Instant Radiosity are Bidirectional Instant Radiosity [SIMP06] and Light Cuts [WFA+05]. As noted in the original paper by Popov et al. [PRDD15], these approaches rely on intrinsic properties of Instant Radiosity and are inherently limited compared to full global illumination solutions.

Combinatorial BDPT [PBPP11] introduced the idea of connecting multiple emitter subpaths with multiple sensor subpaths. No importance sampling of the connections was done and the connections were made on the GPU. In Bidirectional Light Cuts, Walter et al. [WKB12] also connect to multiple emitter subpaths. Similar to PCBPT, they try to keep the number of connections low while preserving the quality of the results. However, their algorithm produces biased results, whereas PCBPT does not.

At its core, PCBPT tries to solve the problem of highly occluded scenes. It would therefore be practical if it were easily extensible to include methods which try to handle other difficult scenarios. Indeed, the fact that PCBPT only samples connections and makes no assumptions about how the paths are generated makes it compatible with many other state-of-the-art rendering techniques. For example, it would be straightforward to combine it with Vertex Connection and Merging (VCM) [GKDS12] as the two methods have similarities in their implementation. VCM tries to improve the handling of specular-diffuse-specular (SDS) paths, a notoriously difficult problem in rendering, by merging BDPT with stochastic progressive photon mapping [HJ09]. It would also, theoretically, be possible to combine PCBPT with Markov Chain Monte Carlo methods, such as Metropolis Light Transport [VG97], or more recently with a work by Šik et al. [vOHK16], which combines VCM with Metropolis Light Transport

The Mitsuba renderer [Jak10] was designed to be modular, performant and research oriented. It concentrates on implementing various techniques which might be of interest to researchers, but which are not implemented in most production-oriented renderers. This allowed us to provide comparisons to both classical and state-of-the-art algorithms. It is open sourced under the GPL licence, enabling anybody to see and modify the existing algorithms. In our work we use Mitsuba's BDPT implementation as a starting point. Mitsuba also comes with a out-of-the-box parallel rendering support, which we utilize in chapter 5 to test our results on a commodity laptop, as well as a 20 CPU cloud virtual machine. Lastly, due to Mitsuba's plugin architecture, adding our code to an existing installation should require minimum changes to the rest of the codebase.

CHAPTER 3

# Methodology

## 3.1 Algorithm Overview

Bidirectional Path Tracing (BDPT) is a powerful algorithm, capable of handling many difficult scenes efficiently. However, even though each vertex in the two subpaths can be importance sampled using standard methods, the connections between the two subpaths are made in a deterministic fashion. Therefore, BDPT cannot "sample" the connection towards a direction where the contribution would be large. BDPT also connects each emitter subpath to each sensor subpath, even if the connections might not produce a large contribution. Furthermore, BDPT discards the two paths after their contribution has been accounted for. These factors present potential sources of inefficiency as BDPT potentially wastes computational resources on low-contribution paths.

If we were instead able to choose which subpaths we connect from a set of possible options, we could then concentrate the work where it really matters while reusing subpaths to potentially generate more samples than would be possible with standard BDPT. This is the idea behind Probabilistic Connections for Bidirectional Path Tracing (PCBPT) by Popov et al. [PRDD15].

In each iteration, PCBPT first generates $W \times H$ emitter and sensor paths of infinite length, where $W$ is the image width and $H$ the image height. It takes the first $M$ emitter paths and stores them in a cache. Then, for each pair of paths, it first evaluates the standard BDPT contribution for subpaths where either $s < 2$ or $t < 2$. For each sensor subpath of length $t \geq 2$, PCBPT importance samples $K$ emitter subpaths to which it can connect. We discuss the exact theoretical details behind this in Section 3.2.

The subpath-length threshold between BDPT and PCBPT, here set to 2, could be changed to an arbitrary number larger than 2. In practice we want to utilize the benefits PCBPT offers, and therefore set the threshold as low as possible. Note that it would not be possible to use PCBPT for paths where $t = 0$ or $s = 0$ since these paths represent the

case where the sensor and the emitter are intersected during the tracing procedure. For paths where $t = 1$ or $s = 1$, we follow standard BDPT practice and use direct sampling of the sensor or emitters, respectively.

In order to sample the emitter subpaths, PCBPT uses an importance-sampling scheme similar to the one presented by Georgiev et al. [GKPS12]. As we show in Section 3.2.3, generating the perfect probability mass function (PMF) for importance sampling the connections implies evaluating all of the connections. In order to avoid this cost, PCBPT stores the perfect PMF only at a small subset of importance-cache records, created at the vertices of $C$ sensor paths. In order to get the final PMF for sampling emitter subpaths given a sensor vertex, PCBPT interpolates the 6 spatially nearest importance cache points. We present one iteration of the high level PCBPT algorithm in Algorithm 3.1.

**Algorithm 3.1:** One iteration of the PCBPT algorithm.

```
Generate the emitter paths.
Generate the sensor paths.
Evaluate the path tracing and light tracing contributions.
for i ← 1 to C do
    Generate sensor path z̄^c.
    for z_t^c ∈ z̄^c, where t = 2 to l_z̄^c do
        Generate the PMF for z_t^c.
        Create an importance-cache record from z_t^c.
    end
end
for i ← 1 to number of pixels in the image do
    Generate a sensor path, z̄.
    Find the closest importance-cache records.
    Interpolate the PMF.
    Sample the interpolated PMF.
    Add the contribution to the i^th pixel estimate.
end
```

We can see that PCBPT potentially reuses the same emitter subpaths when connecting to a sensor subpath. While this can save on computational resources, the downside is that it also potentially introduces correlation between two sampled paths. As we saw in Section 2.2.3, using correlated Monte Carlo samples increases the total variance, and the variance-reduction guarantees given by the balance heuristic no longer hold. To help mitigate this, Popov et al. [PRDD15] also present a novel derivation of the balance heuristic for correlated samples. We give the definition of the modified balance heuristic in Section 3.3.

## 3.2 Importance Sampling Emitter Paths

### 3.2.1 Connecting to Multiple Emitter Subpaths

The PCBPT algorithm is constituted of a number of building blocks, the most basic of which only involves connecting a sensor subpath to multiple emitter subpaths, similar to [PBPP11]. The part of the algorithm dealing with probabilistically connecting to a subset of these connections can be thought of as an additional optimization. In the following sections, we ignore the connections between subpaths which are handled by standard BDPT methods, and concentrate on subpaths where $s \geq 2$ and $t \geq 2$. As in the original paper, we will also limit our discussion to the evaluation of the integral over paths of length $l$, $I_l$, without loss of generality. Therefore, we will be observing the connections between a sensor subpath of length $t$ and $M$ emitter subpaths of length $s$, such that $s + t = l$.

As previously mentioned, PCBPT caches $M$ emitter paths of infinite length, or, in other words, $M$ emitter subpaths for each possible length of the emitter subpath. It then connects each sensor subpath of length $t$ to each of the $M$ emitter subpaths of length $s$, such that $s + t = l$. This can also be seen as taking $M$ different samples from $\Omega_l$. Following the definitions from equations 2.27 and 2.28, our MIS MC estimator can be written as

$$\tilde{I}_l = \sum_{t=1}^{l} \frac{1}{M} \sum_{j=1}^{M} \tilde{I}_t(\bar{x}_j). \tag{3.1}$$

In PCBPT the path $\bar{x}_j$ is generated by connecting the sensor subpath of length $t$ with the $\mathrm{j}^{\mathrm{th}}$ emitter subpath of length $s$. To generate $N_t = M$ samples, this procedure is repeated for all cached emitter subpaths of length $s$.

While this facilitates path reuse and could save some computational resources, it still does not allow us to importance sample the connections. To do that, Popov et al. [PRDD15] introduce the concept of probabilistic connections.

### 3.2.2 Probabilistic Connections

In order to importance sample emitter subpaths, Popov et al. [PRDD15] suggest using another MC estimator over the inner sum. In other words, they suggest evaluating a Monte Carlo estimate of

$$S_t = \sum_{j=1}^{M} \tilde{I}_t(\bar{x}_j), \tag{3.2}$$

by importance sampling an interpolated PMF to generate $K$ samples of $\tilde{I}_t(\bar{x}_j)$ .

Monte Carlo estimation of sums works analogous to Monte Carlo integration, except that all of the domain being sampled is discrete, instead of continuous. In our case, the sample space is the emitter cache and the PMF is given by the interpolated cache records. The MC estimator of the above sum is then given by

$$\tilde{S}_t = \frac{1}{K} \sum_{\bar{x}_k \in \mathcal{K}} \frac{\tilde{I}_t(\bar{x}_k)}{\mathrm{pmf}(\bar{x}_k)}, \tag{3.3}$$

where $\mathcal{K}$ represents the set of sampled paths. The full PCBPT estimate of the contribution of paths of length $l$ is therefore given by

$$\tilde{I}_l = \sum_{t=1}^{l} \frac{1}{M} \frac{1}{K} \sum_{\bar{x}_k \in \mathcal{K}} \frac{\tilde{I}_t(\bar{x}_k)}{\mathrm{pmf}(\bar{x}_k)}. \tag{3.4}$$

### 3.2.3 Probability Mass Function Caching

In order to importance sample the emitter cache, we need a way of generating a sampling PMF which will prefer subpaths with a higher contribution. To do this, we could evaluate each of the full path contributions for each sensor vertex, and create the PMF such that the probability of sampling an emitter subpath is proportional to the contribution of the path which would be generated by connecting to that subpath. This would create the perfect sampling PMF for that vertex and the variance of the estimator $\tilde{S}_t$ would equal 0.

Assuming the sensor subpath $\bar{z}$, and assuming cache contains $M$ emitter subpaths, $\bar{y}^{(1)}$, $\bar{y}^{(2)}$, $\ldots \bar{y}^{(M)}$, the probability of connecting to the $\mathrm{j}^{\mathrm{th}}$ emitter subpath is given by normalizing the contribution of the full path $\bar{x}^{(j)}$ by dividing it with $S_t$ so that the PMF would sum up to one,

$$p^{conn}(j) = \frac{\frac{f(\bar{x}^{(j)})}{p(\bar{x}^{(j)})}}{\sum_{k=0}^{M} \frac{f(\bar{x}^{(k)})}{p(\bar{x}^{(k)})}}. \tag{3.5}$$

However, creating such a PMF incurs the same computational cost as evaluating the full sum. Thankfully, the value of $p^{conn}$ only depends on the emitter subpath, and the last vertex in the sensor subpath, as can be shown,

$$p^{conn}(j) = \frac{\frac{f(\bar{z})}{p(\bar{z})} \frac{f^{\mathrm{conn}}(\bar{y}^{(j)},\bar{z}) f(\bar{y}^{(j)})}{p(\bar{y}^{(j)})}}{\frac{f(\bar{z})}{p(\bar{z})} \sum_{k=0}^{M} \frac{f^{\mathrm{conn}}(\bar{y}^{(k)},\bar{z}) f(\bar{y}^{(k)})}{p(\bar{y}^{(k)})}} = \frac{\frac{f^{\mathrm{conn}}(\bar{y}^{(j)},\bar{z}) f(\bar{y}^{(j)})}{p(\bar{y}^{(j)})}}{\sum_{k=0}^{M} \frac{f^{\mathrm{conn}}(\bar{y}^{(k)},\bar{z}) f(\bar{y}^{(k)})}{p(\bar{y}^{(k)})}} \tag{3.6}$$

Due to this, we can instead only calculate the PMFs at a small set of importance-cache records in the scene and then interpolate them for other vertices. The interpolation

scheme is described in the supplemental material to the original paper by Popov et al. [PRDD15].

We can intuitively see why this PMF calculation performs well. It only samples visible vertices due to the visibility calculation in $f^{\text{conn}}$, it prefers subpaths with larger throughputs, and importantly, importance samples based on the throughput of the connections, also contained in $f^{\text{conn}}$.

It should be noted that, whereas in regular importance sampling, we handcraft strategies for which we believe will produce paths with high contributions, here, we are circumventing assumptions about strategies, and simply sampling based on the approximated contributions. The only assumption we are making in this scenario is that the interpolated PMFs approximate the perfect PMF sufficiently.

## 3.3 Multiple Importance Sampling for Correlated Paths

The strength of PCBPT lies in the fact that light subpaths are reused, meaning we have to trace fewer new paths. On the other hand, reusing paths leads to sample correlation, which, as we have seen in Section 2.2.3, leads to an overall increase in variance.

Popov et al. [PRDD15] provide a modified balance heuristic which minimizes an upper bound on the variance, given the presence of correlated samples. They offer the exact derivation in the supplemental material to their paper. To achieve this, they observe the sets of uncorrelated samples, $S_u$, and the set of correlated samples, $S_c$ separately. This separation is necessary as the full PCBPT algorithm contains both correlated and uncorrelated samples. The uncorrelated samples are the ones handled by standard BDPT, i.e. samples where either $t < 2$ or $s < 2$. All of the samples which are generated using PCBPT are treated as correlated.

The final modified MIS weights, as originally derived by Popov et al. [PRDD15], are given by

$$w_t(\bar{x}) = \frac{N_t p_t(\bar{x})}{\sum_{i \in S_u} N_i p_i(\bar{x}) + \sum_{i \in S_c} p_i(\bar{x})}, t \in S_u \tag{3.7}$$

$$w_t(\bar{x}) = \frac{p_t(\bar{x})}{\sum_{i \in S_u} N_i p_i(\bar{x}) + \sum_{i \in S_c} p_i(\bar{x})}, t \in S_c. \tag{3.8}$$

Comparing this to the standard balance heuristic from equation 2.16, we notice that the $N_i$ and $N_t$ terms are missing for the correlated samples. Intuitively, due to the path reuse present in PCBPT, one "bad" emitter subpath which produces high-variance samples when connected to can influence many pixels. In order to mitigate the increase in covariance, the modified balance heuristic decreases the weights assigned to these samples. Therefore, the modified balance heuristic reduces the negative effects caused by path correlation.

CHAPTER 4

# Implementation

In this section we will discuss in more detail how we implemented the Probabilistic Connections for Bidirectional Path Tracing (PCBPT) algorithm into the Mitsuba renderer. Our practical implementation of PCBPT differs somewhat from the theoretical explanation presented in the previous chapter. We discuss these differences and provide justifications for them in Section 4.1. In Algorithm 4.1, we show the detailed pseudo-code for our implementation. In Section 4.3, we give a short summary of the path we found useful when taking PCBPT from theory to implementation.

## 4.1 Modifications

The first modification we made concerns the memory consumption of the algorithm presented in 3.1. Directly implementing this pseudo-code would require us to store $W \times H$ emitter and sensor subpaths. We opted for a more optimized implementation, which, at any point only requires saving the $M = 100$ emitter paths, and the importance-cache records with their respective PMFs.

The second necessary modification is due to the fact that Mitsuba features sensors that can be intersected by rays. In the original paper by Popov et al. [PRDD15], the renderer only supported pinhole cameras, and connections where $t = 0$ were impossible. Therefore, our algorithm differs from the original implementation in that it supports these connections, and treats them with standard BDPT methods.

The third modification is due to the way Mitsuba handles light-tracing contributions. In the standard BDPT algorithm, the number of light rays coming to a pixel potentially equals the total number of rays emitted into the scene. Due to this, the number of samples for strategies $t = 0$ and $t = 1$ in one iteration would typically be treated as $N_0 = N_1 = W \times H$. However, we were unable to verify that this is the case in Mitsuba, as Mitsuba saves these contributions in a separate buffer, named the *light image*.

Furthermore, when calculating the balance heuristic weights, Mitsuba assumes that the number of connections for these samples equals one, i.e., $N_0 = N_1 = 1$. Due to these differences, the variance of the estimator might differ from the standard MIS estimator. We were unable to verify whether variance-reduction guarantees of the modified balance heuristic still hold assuming $N_0 = N_1 = 1$ within Mitsuba.

## 4.2 Algorithm

In our implementation, we start off by generating and saving the $M$ emitter paths. Note that it is not necessary to store the emitter subpaths which would have a zero probability of being sampled from any possible importance cache point. This includes emitter subpaths with zero throughput, and those shorter than two. Subpaths with zero throughput include vertices where the throughput has been attenuated to zero by Russian Roulette, as well as subpaths to which we can never connect to by random chance such that we produce a non-zero contribution. One example of such a subpaths includes those where the last subpath vertex is located on a material with a Dirac delta BSDF (i.e., a perfectly specular material). We also do not need to reserve space in the cached PMFs for subpaths shorter than two, as those are not handled by PCBPT, and the probability of sampling them needs to equal zero. This seemingly small optimization decreases the memory consumption to about one third in our tests. We also noticed a slight decrease in execution times, which we attribute to improved cache locality.

To build the importance cache, we generate $C = 0.04 \times W \times H$ sensor paths, following the original paper's suggestion. At each vertex of each path, $\bar{z}^c$ of length $l^c_{\bar{z}}$, we create an importance-cache record, $\boldsymbol{z}^c$. For each $\boldsymbol{z}^c$, we iterate over all of the cached emitter subpaths, and store the luminance of its contribution in the PMF. We chose the luminance as creating a multi-dimensional PMF based on the spectral contribution would make little sense in practice.

We store these cache records in a $k$D tree, which is already implemented in Mitsuba. This allows us to perform fast $k$-nearest-neighbor queries when searching for the nearest importance-cache points to use for interpolation. We then normalize the PMF and do a prefix sum over the values to create a cumulative distribution function (CDF).

The original paper suggests using low discrepancy sampling to generate importance cache paths uniformly over the screen. During our implementation in Mitsuba, we realized that there is no way to explicitly seed the low discrepancy sampler plugin. Due to this, we use the uniform random integer sampling capabilities offered by the *Boost library* [Boo17]. As Boost is already distributed with Mitsuba, we introduce no new dependencies to the build.

After the CDFs have been generated, we start with the evaluation of the path contributions. In order to save some computational resources, we reuse the $M$ emitter paths for the first $M$ BDPT connections as well. After that, we generate a new emitter path for each pixel to use for the BDPT contribution. This entire procedure is detailed in Algorithm 4.1.

**Algorithm 4.1:** One iteration of the PCBPT algorithm.

```
for i ← 1 to M do
    Generate and cache an emitter path.
end
for i ← 1 to C do
    Generate sensor path z̄^c.
    for z_t^c ∈ z̄^c, where t = 2 to l_z̄^c do
        Generate the PMF for z_t^c.
        Add z_t^c to a kD tree as an importance-cache record.
    end
end
for i ← 1 to number of pixels in the image do
    Generate a sensor path, z̄.
    if i < M then
        ȳ^BDPT ← i^th emitter path from the emitter cache.
    end
    else
        ȳ^BDPT ← generate a new emitter path.
    end
    contribution ← EvalBDPT(z̄, ȳ^BDPT) + EvalPCBPT(z̄);
    Add contribution to the i^th pixel estimate.
end
```

The function $EvalBDPT$ from Algorithm 4.1 represents the standard BDPT subpath connection algorithm, except that, in this case, it only connects subpaths where $s < 2$ or $t < 2$. The function $EvalPCBPT$ evaluates the PCBPT contribution and is shown in more detail in Algorithm 4.2

To evaluate the PCBPT contribution, we start by iterating over the vertices of the sensor path. For each sensor vertex, in order to accumulate the contribution of $K$ subpaths, we repeat the following procedure $K$ times. First, we query its 6 nearest importance-cache points by using Mitsuba's $k$-nearest-neighbors functionality. At no point we explicitly interpolate the entire CDFs. Rather, we use a lazy evaluation scheme.

To sample from a CDF, we first generate a uniform random sample between 0 and 1. As in standard inversion sampling practice, we find the lower bound for the uniform sample within our CDF, i.e. the largest CDF value which is smaller than our sample. We can rely on using the binary-search algorithm to find the lower bound since the CDF is monotonically increasing per definition. We proceed in standard binary-search fashion with an index pointing to the middle of the CDF arrays. We interpolate the 6 CDF values only at that index, and based on that information, move either to the right or to the left of that index. After the emitter subpath has been sampled, we accumulate its contribution to the estimate.

**Algorithm 4.2:** *EvalPCBPT*

**Input:** Sensor path $\bar{z}$
**Output:** PCBPT Contribution to the pixel estimate

$contribution \leftarrow 0$
**for** $\boldsymbol{z}_t \in \bar{z}$, **where** $t = 2$ **to** $l_{\bar{z}}$ **do**
  $\bar{z}_t \leftarrow$ subpath ending in vertex $\boldsymbol{z}_t$.
  Query 6 importance-cache points which are spatially closest to $\boldsymbol{z}_t$.
  **for** $k \leftarrow 1$ **to** $K$ **do**
    Use binary search to lazily inversion sample the interpolated PMF and choose an emitter subpath $\bar{y}_s^{(j)}$.
    $s \leftarrow$ length of the subpath $\bar{y}_s^{(j)}$.
    $p^{conn}(j) \leftarrow$ probability of having sampled $\bar{y}_s^{(j)}$ from the PMF.
    $\bar{x} \leftarrow (\bar{z}_t, \bar{y}_s^{(j)})$.
    $contribution \leftarrow contribution + w_s(\bar{x}) \frac{f(\bar{x})}{p^{conn}(j) p(\bar{x})}$.
  **end**
**end**
$contribution \leftarrow \frac{1}{KM} contribution$
**return** *contribution*

Following Mitsuba's practice, we assume that the number of uncorrelated samples per iteration equals one. Setting these numbers in equation 3.7, we see that the separation between the correlated and uncorrelated samples becomes irrelevant, and that the final MIS weight becomes simply

$$w_t(\bar{x}) = \frac{p_t(\bar{x})}{\sum_{i \in S_u \cup S_c} p_i(\bar{x})}, \tag{4.1}$$

for all $t$.

Note that the final weights assigned to the correlated samples still get weighted down to account for sample correlation, as the number of samples $N_t$ where $t \in S_c$ equals $M$. Furthermore, Mitsuba provides no way of specifying the number of samples for a strategy during the MIS calculation, and assumes that it equals one for all strategies. Due to this, our implementation simply reuses Mitsuba's MIS functionality.

## 4.3 From Theory to Practice

When implementing the algorithm, we found it useful to separate the algorithm into smaller testable pieces. In our implementation we have discovered a potentially useful path for taking PCBPT from theory to practice. We present here the steps which we took in order to arrive at the end result.

1. Firstly, we created the Mitsuba boiler-plate code necessary for a new integrator plugin. Note that the PCBPT integrator is not tile-based, like the BDPT one, but rather iteration based. This means that each thread is rendering an entire image, instead of just a tile inside of a larger image.

2. We then created a function which connects two paths and evaluates their contribution. This function can be reused to calculate the final sample value for both BDPT and PCBPT contributions.

3. Next, we implemented the generation of $M$ emitter paths and allocated the memory for the PMFs accordingly. We save the PMFs in contiguous memory and only save iterators to the beginning of the PMF at each importance-cache point.

4. We implemented an intermediary step, where the $t \geq 2$ subpaths were connected to each vertex of all $M$ emitter paths, and calculated the contribution as given in equation 3.1. Note that the algorithm at this point equates to Combinatorial Bidirectional Path Tracing [PBPP11], without the offloading of connections to the GPU.

5. We continued by including probabilistic connections. At first, we used a uniform PMF to sample $K$ connections and evaluate equation 3.4, replacing equation 3.1.

6. Next, we replaced the uniform PMF with the PMF of the first nearest neighbor of the sensor path vertex. The results of PCBPT should start being visible.

7. Finally, we implemented the lazy-evaluation binary search on the 6 nearest neighbors to sample from the CDF.

We also found it important to be able to display PCBPT connections separately from the BDPT connection during the development and during the testing. It was also useful to have scenes with highly occluded emitters since this is where PCBPT has the largest effect. In Chapter 5, we show how large this effect is, by evaluating the algorithm in a number of different scenarios.

CHAPTER 5

# Results

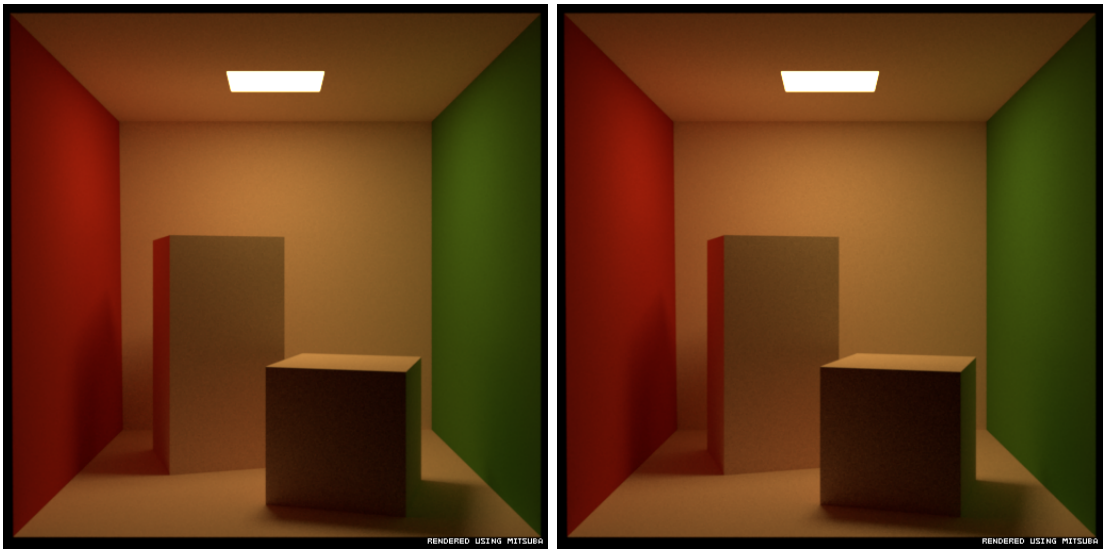
It is clear that PCBPT induces the same path-generation cost as BDPT since both algorithms generate $W \times H$ paths. PCBPT, however, has additional cost for generating the caches, as well as calculating and interpolating the PMFs. This usually means that BDPT will manage to produce many additional new paths in the same time compared to PCBPT. However, the benefits of PCBPT become visible when most of the contribution is caused by inner paths, i.e., paths where either $s \geq 2$ and $t \geq 2$. In such cases, the computational cost is made up for by the faster convergence of inner paths caused by importance sampling the connections.

Based on this knowledge, we designed our tests to cover a number of different scenarios. First, we verify that our algorithm converges to the correct result using a very simple scene, namely the *Cornell Box*. Furthermore, we create equal-time comparisons with BDPT on three difficult scenes – the *Veach Ajar Door* scene, *The Breakfast Room* scene, and our own modification of the *The White Room* scene. For each of the scenes, we render the images once using all paths for both BDPT and PCBPT, and once using only the inner paths. We also present the reference images, as well as the normalized Root Mean Square Error (RMSE) of the equal-time comparisons. All of the used scenes were obtained from Bitterli's Rendering Resources page [Bit16]. The rendering was done on a commodity Lenovo Y50-70 laptop, with an Intel i7-4710HQ processor operating at 2.50GHz, and 16GB of RAM. All of the scenes were rendered with $M = 100$ cached paths, and $K = 10$ connections.

## 5.1 Verification

In order to verify our algorithm, we rendered the *Cornell Box* scene using both BDPT and PCBPT. The results are presented in Figure 5.1. Note that the images look nearly identical, as corroborated by the RMSE values. During the development, we have noticed a major performance penalty for such scenes when using PCBPT. This is to be expected

as the *Cornell Box* represents a very simple scene with little geometry and a large visible emitter. As such, most BDPT connections do actually make non-zero contributions, and the cache-generation overhead of PCBPT offers no benefits in return.




(a) BDPT, 200 samples per pixel, RMSE 0.004. (b) PCBPT, 100 samples per pixel, RMSE 0.005.

Figure 5.1: The *Cornell Box* scene which we used for verifying our implementation. RMSE measured against a reference image rendered at 1000 samples per pixel, RMSE .

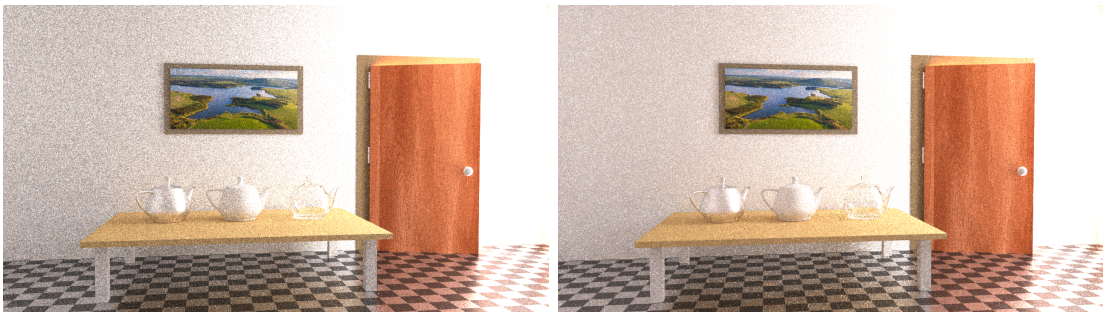
## 5.2 Comparisons

We created equal-time comparisons between PCBPT and BDPT on a number of difficult scenes. For these scenes, we rendered the ground-truth images using a large number of samples per pixel, against which we calculated the normalized RMSE. We present the ground-truth images as well as the results in Section 5.3.

The *Veach Ajar Door* represents a scene with an occluded emitter. More precisely, the emitter is located directly behind a slightly ajar door, emitting light into the room through the opening. This scene is potentially difficult for classical path tracing, as the sensor paths are unlikely to find the emitter. Figure 5.2 shows the comparison between the two images rendered using all paths, i.e., both the inner as well as the outer connections, whereas Figure 5.3 shows the comparison of only the inner paths.

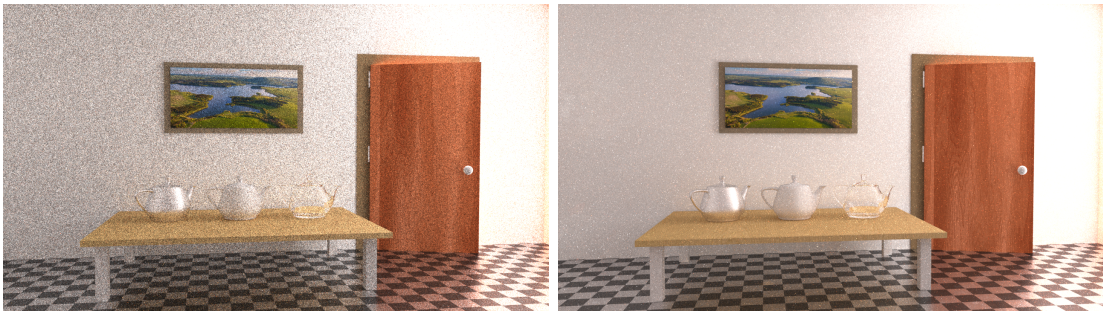
BDPT and PCBPT had a comparable error on the comparison where all paths were taken into account, with PCBPT being slightly better than BDPT. This is justified by the fact that even though the emitter is occluded, due to its position, most of the rays coming from it still reach the sensor. This means that light tracing actually has a significant contribution to the final estimate. In Figure 5.3, we can see that the inner paths for PCBPT do converge a lot faster than for BDPT. For the inner paths, RMSE

improved by around 4% for the inner paths, compared to the 1% for the image rendered using all paths. This leads to the conclusion that most of the noise in the image rendered using PCBPT comes from the outer paths, and that BDPT is able to create more outer subpath contributions in the same time to overcome the lack of quality in the inner paths.



(a) BDPT, 120 samples per pixel, RMSE 0.073 (b) PCBPT, 32 samples per pixel, RMSE 0.061.

Figure 5.2: *Veach Ajar Door* all paths comparison.

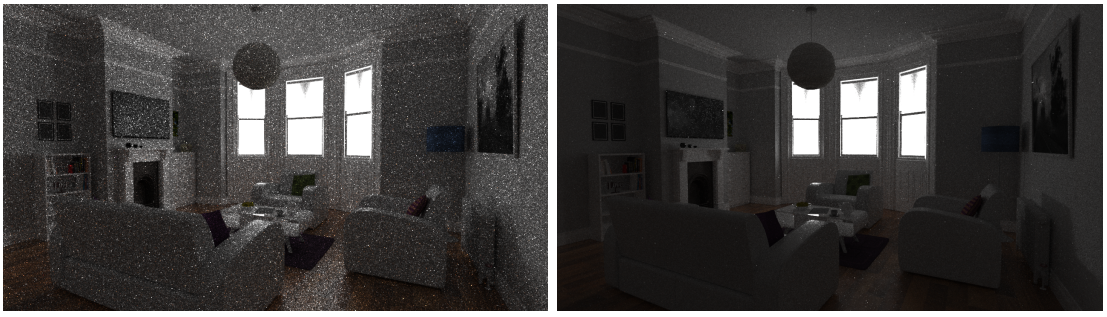


(a) BDPT, 120 samples per pixel, RMSE 0.088. (b) PCBPT, 32 samples per pixel, RMSE 0.040.

Figure 5.3: *Veach Ajar Door* inner paths comparison.

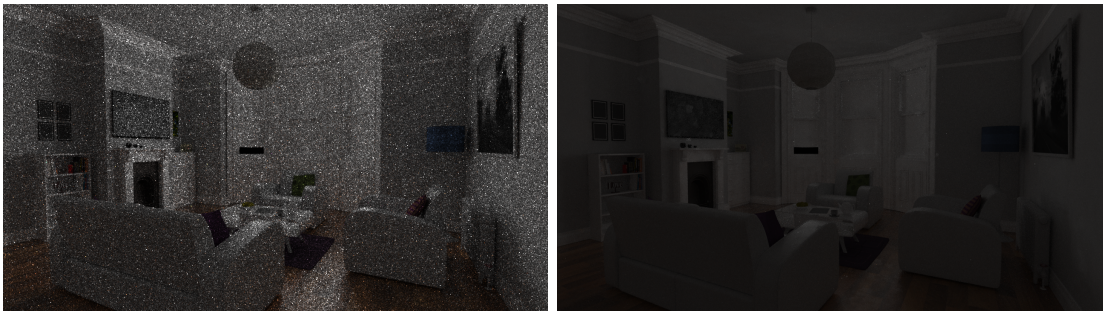
We modified *The White Room* scene to produce a more difficult lighting condition, by resizing the blinds covering the windows such that most of the illumination coming into the room is due to a small opening in the middle window. We notice significant improvements for this scene when using PCBPT. The RMSE error for the images where all of the paths were accounted for improved by 11%, and by 13% for images where only inner paths were used. This is to be expected as only a small subset of rays actually makes it into the scene. Therefore, it makes sense to importance sample the rays to which we can connect.

*The Dining Room* scene represents a failure case. It contains a large emitter on the side of the room, behind a partially occluded window. However, given that the emitter is only somewhat occluded, most of the paths from the camera can actually directly sample the emitter and vice versa. Furthermore, most of the vertices are connectable as they are all located inside of the room. This makes this scene highly suitable for BDPT, as is



(a) BDPT, 120 samples per pixel, RMSE 0.130. (b) PCBPT, 32 samples per pixel, RMSE 0.022.

Figure 5.4: *The White Room* all paths comparison.



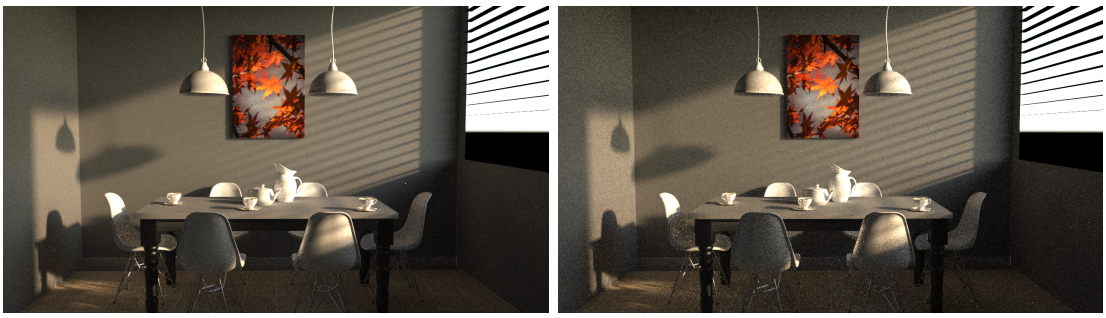
(a) BDPT, 120 samples per pixel, RMSE 0.147. (b) PCBPT, 32 samples per pixel, RMSE 0.013.

Figure 5.5: *The White Room* inner paths comparison.

also visible from the results. PCBPT does show a much lower RMSE on the inner path comparisons, but this is overshadowed by the many outer path contributions BDPT is able to evaluate. Furthermore, in the PCBPT image, more noise is visible on the chair, where the BSDF is mostly specular. As it is impossible to connect to a vertex on a specular surface, PCBPT can offer no improvement in this situation, and most of the contributions need to come from standard path tracing.

## 5.3 Ground-Truth Images and The Root Mean Square Errors

In order to provide RMSE values, we generated a number of ground-truth images, shown in this section. The images are shown in Figures 5.8, 5.9, and 5.10. For an easier overview, we show all of the RMSE in Table 5.1, as well as the ratio between the RMSE for the BDPT image, $E_B$, and the RMSE for the PCBPT image, $E_P$.



(a) BDPT, 150 samples per pixel, RMSE 0.022. (b) PCBPT, 48 samples per pixel, RMSE 0.035.

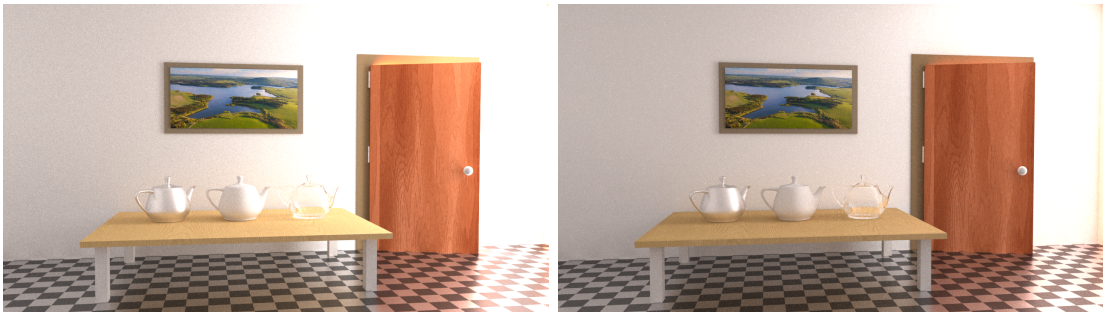
Figure 5.6: *The Dining Room* all paths comparison.

(a) BDPT, 150 samples per pixel, RMSE 0.021. (b) PCBPT, 48 samples per pixel, RMSE 0.011.

Figure 5.7: *The Dining Room* inner paths comparison.

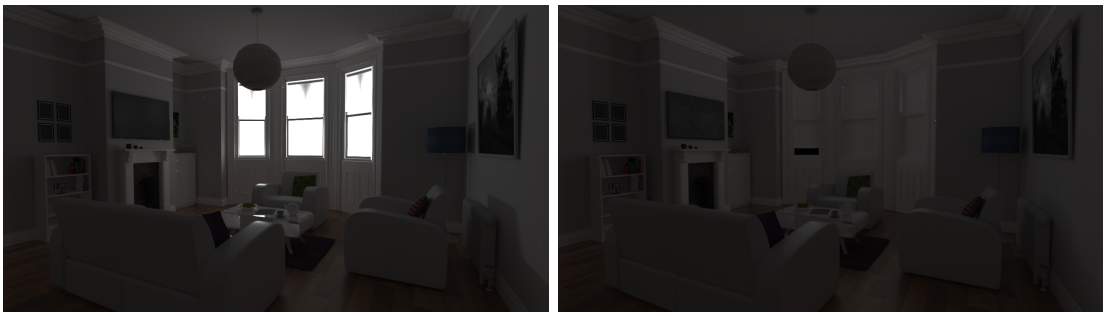
| | All Paths | | | Inner Paths | | |
|---|---|---|---|---|---|---|
| | BDPT | PCBPT | $\frac{E_B}{E_P}$ | BDPT | PCBPT | $\frac{E_B}{E_P}$ |
| *Veach Ajar Door* | 0.073 | 0.061 | 1.20 | 0.088 | 0.040 | 2.20 |
| *The White Room* | 0.130 | 0.022 | 5.90 | 0.147 | 0.013 | 11.31 |
| *The Dining Room* | 0.022 | 0.035 | 0.63 | 0.021 | 0.011 | 1.91 |
| Average | 0.075 | 0.040 | 2.60 | 0.085 | 0.021 | 5.14 |

Table 5.1: Normalized Root Mean Square Error for the images from Section 5.2. A lower value means a better result.



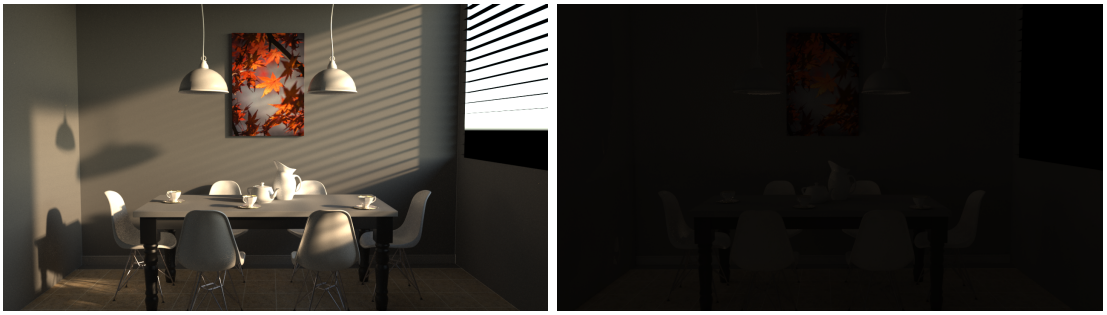
(a) All Paths. (b) Inner paths.

Figure 5.8: *Veach Ajar Door* ground-truth images.



(a) All paths. (b) Inner paths.

Figure 5.9: *The White Room* ground-truth images.



(a) All Paths. (b) Inner paths.

Figure 5.10: *The Dining Room* ground-truth images.

CHAPTER 6

# Conclusion

We implemented Probabilistic Connections for Bidirectional Path Tracing, and showed that it significantly improves the convergence of inner paths, as well as the overall convergence of scenes with very difficult lighting conditions. However, in scenes where the emitter was easily reachable by the sensor rays and vice versa, PCBPT incurred an overhead.

In this sense, our results are in line with those of Popov et al. [PRDD15]. They reported an average ratio of 6.4 between the L1 errors of BDPT and PCBPT for images rendered using the inner paths, whereas, for our tests, the average ratio of the RMSE equaled 5.4. It should be noted that all of our tests were done with the same value for $K$. We are confident that further improvements can be attained by fine-tuning of the parameters. We were unable to test our implementation on the scenes they used as they were not available to the general public.

The method could be improved by automating the procedure of choosing the values for $M$ and $K$, such that there is less overhead for relatively simple scenes. Improving paths where $s < 2$ or $t < 2$ would potentially provide a large improvement, as PCBPT makes no advances in this regard. Another improvement could be made regarding specular-diffuse-specular paths. Currently, PCBPT does nothing to help with these, and is, in theory, orthogonal to methods such as Vertex Connection and Merging [GKDS12], or Metropolis Light Transport [VG97]. Other PMF sampling strategies could be devised and joined together with MIS. Regarding our implementation, future work would need to verify the correctness of the MIS weights. Further experimentation could be done with the parameters, as all of our tests were done using the same values for $M$ and $K$.

As shown in our results, PCBPT is a simple, yet powerful method for variance reduction of contributions produced by inner paths. For the most difficult scene, we were able to improve the RMSE metric by 11% compared to BDPT when taking into account both inner and outer paths, and by 13% for just the inner paths.

# List of Figures

# List of Tables

# List of Algorithms

# Bibliography


[Bit16] Benedikt Bitterli. Rendering resources, 2016. https://benedikt-bitterli.me/resources/.

[Boo17] Boost. Boost C++ Libraries. `http://www.boost.org/`, 2017. Last accessed 2017-08-29.

[Dar] Dario Seyb. Path graph.

[DW95] Philip Dutré and Yves D. Willems. Importance-driven monte carlo light tracing. In *5th Eurographics Workshop on Rendering*, pages 185–197. Springer Berlin Heidelberg, Darmstadt, Germany, 1995.

[GKDS12] Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.*, 31:XXX:1–XXX:10, December 2012. SIGGRAPH Asia 2012.

[GKPS12] Iliyan Georgiev, Jaroslav Křivánek, Stefan Popov, and Philipp Slusallek. Importance caching for complex illumination. *Computer Graphics Forum*, 31(2), 2012. EUROGRAPHICS 2012.

[HJ09] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 141:1–141:8, New York, NY, USA, 2009. ACM.

[HP01] Heinrich Hey and Werner Purgathofer. Importance sampling with hemispherical particle footprints. Technical Report TR-186-2-01-05, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, January 2001. human contact: technical-report@cg.tuwien.ac.at.

[Jak10] Wenzel Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[Jen95] Henrik Wann Jensen. *Importance Driven Path Tracing using the Photon Map*, pages 326–335. Springer Vienna, Vienna, 1995.

[Kaj86] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM.

[Kel97] Alexander Keller. Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[LW93] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, December 1993.

[MGN17] Thomas Müller, Markus Gross, and Jan Novák. Practical path guiding for efficient light-transport simulation. *Computer Graphics Forum*, 36(4):91–100, June 2017.

[PBPP11] Anthony Pajot, Loïc Barthe, Mathias Paulin, and Pierre Poulin. Combinatorial bidirectional path-tracing for efficient hybrid cpu/gpu rendering. *Computer Graphics Forum*, 30(2):315–324, 2011.

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2016.

[PRDD15] Stefan Popov, Ravi Ramamoorthi, Frédo Durand, and George Drettakis. Probabilistic connections for bidirectional path tracing. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 34(4), 2015.

[SCG+05] Pradeep Sen, Billy Chen, Gaurav Garg, Stephen R. Marschner, Mark Horowitz, Marc Levoy, and Hendrik P. A. Lensch. Dual photography. *ACM Trans. Graph.*, 24(3):745–755, July 2005.

[SIMP06] B. Segovia, J. C. Iehl, R. Mitanchey, and B. Péroche. Bidirectional instant radiosity. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, EGSR '06, pages 389–397, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[Vea98] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation.* PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[VG95] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 419–428, New York, NY, USA, 1995. ACM.

[VG97] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[VKv+14] Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. On-line learning of parametric mixture models for light transport simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)*, 33(4), aug 2014.

[vOHK16] Martin Šik, Hisanari Otsu, Toshiya Hachisuka, and Jaroslav Křivánek. Robust light transport simulation via metropolised bidirectional estimators. *ACM Trans. Graph.*, 35(6):245:1–245:12, November 2016.

[WFA+05] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: A scalable approach to illumination. *ACM Trans. Graph.*, 24(3):1098–1107, July 2005.

[WKB12] Bruce Walter, Pramook Khungurn, and Kavita Bala. Bidirectional lightcuts. *ACM Trans. Graph.*, 31(4):59:1–59:11, July 2012.