Implementing Probabilistic Connections for Bidirectional Path Tracing in the Mitsuba Renderer

Nikola Dodik*
Supervised by: Hiroyuki Sakai

Institute of Visual Computing & Human-Centered Technology
TU Wien
Vienna / Austria

Abstract

Light transport simulation algorithms are remarkably adept at recreating a 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. *Probabilistic connections for bidirectional path tracing* (PCBPT) represents one such recent algorithm. Unfortunately, no implementation of PCBPT was publicly available, which is why we implemented it into the open-source renderer Mitsuba [1]. We evaluate the algorithm against standard bidirectional path tracing on a variety of different scenes. Our comparisons provide insight into the type of scenes for which PCBPT can provide an improvement and we examine cases where the additional computational cost presents too much overhead.

Keywords: bidirectional path tracing, importance sampling, Mitsuba renderer

1 Introduction

Practical photo-realistic image synthesis is one of the long-standing goals of computer graphics research. Its applications range from the entertainment industry to architectural and engineering visualizations. Light transport simulation algorithms simulate how light behaves within a simplified mathematical model to this end. A popular subclass of such algorithms, jointly referred to as Monte Carlo rendering, uses sampling-based statistical methods to estimate the light distribution in a scene, and with it the final image. The most common procedure for creating samples involves shooting rays into the scene, starting either from the camera sensor or from the light emitter.

In practice, getting a good-looking image often implies taking many samples. 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) [7, 10] approaches this problem by combining the strengths of path tracing and light tracing using *multiple importance sampling* (MIS). BDPT reuses a pair of emitter and sensor paths to produce more than one sample by connecting the two paths at different locations building multiple full paths.

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.

Probabilistic connections for bidirectional path tracing (PCBPT) by Popov et al. [9] aims to mitigate these shortcomings. It is able to choose the subpaths to which we connect from a set of possible options, therefore making it possible to concentrate the work where it really matters while reusing subpaths and potentially generating more samples than standard BDPT.

At the time of writing, the original PCBPT implementation by Popov et al. [9] 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 paper, we implement PCBPT into the Mitsuba renderer [6], as detailed in Section 4. The Mitsuba renderer implements many state-of-the-art methods, facilitating easier comparisons with them. Furthermore, Mitsuba has a plugin architecture, as well as a framework for bidirectional methods, which both make it a prime tool for researchers.

In Section 5, we 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 when using PCBPT on difficult scenes compared to standard BDPT. We also noticed a decrease in the root-mean-square error (RMSE) of up to 11%.

*dodiknikola at gmail dot com

[1]The code is available at `https://www.cg.tuwien.ac.at/research/publications/2017/dodik-2017-pcbpt/`

2 Review of Bidirectional Path Tracing

The basic idea behind the bidirectional path tracing 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.

Assuming we have traced a sensor path $\bar{z}$ with vertices $(\mathbf{z}_1, \mathbf{z}_2, \dots)$, and an emitter path $\bar{y}$ with vertices $(\mathbf{y}_1, \mathbf{y}_2, \dots)$, to generate path of length l, we simply prepend the vertices of the emitter subpath of length s to the vertices of the sensor subpath of length t, $\bar{x} = (\mathbf{z}_1, \dots, \mathbf{z}_t, \mathbf{y}_s, \dots, \mathbf{y}_1)$. BDPT uses this idea to generate many full path contributions by combining each emitter to 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 values as if they were evaluated from the reverse direction. The final contribution of a full path then evaluates to

$$\begin{aligned} f(\bar{x}) =& L_e(\mathbf{y}_1 \to \mathbf{y}_2)\Pi_s(\bar{y}) f^{\mathrm{conn}}(\bar{y}, \bar{z}) \qquad (1)\\ & \Pi_t(\bar{z}) W(\mathbf{z}_2 \to \mathbf{z}_1), \\ f^{\mathrm{conn}}(\bar{y}, \bar{z}) =& \rho(\mathbf{y}_{s-1} \to \mathbf{y}_s \to \mathbf{z}_t) G(\mathbf{y}_s \leftrightarrow \mathbf{z}_t) \qquad (2)\\ & \rho(\mathbf{y}_s \to \mathbf{z}_t \to \mathbf{z}_{t-1}), \end{aligned}$$

where $L_e(\mathbf{y}_1 \to \mathbf{y}_2)$ represents the radiance emitted from point $\mathbf{y}_1$ to point $\mathbf{y}_2$. $W(\mathbf{z}_2 \to \mathbf{z}_1)$ determines the sensor's sensitivity for a given light ray, $G(\mathbf{y}_s \leftrightarrow \mathbf{z}_t)$ is the generalized geometric term, ρ is the BSDF of the material, and $\Pi_s(\bar{y})$ and $\Pi_t(\bar{z})$ are the emitter and sensor path throughputs, respectively.

Calculating the probability of a path is straightforward. We use $p^{\to}(\bar{x})$ 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, $p^{\to}(\bar{y})$ 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 $\mathbf{y}_s$ with $\mathbf{z}_t$ equals one. This makes the probability of sampling the full path simply equal to $p(\bar{y})p(\bar{z})$.

The big improvement to BDPT was brought about by Veach and Guibas [11] after they introduced MIS to the algorithm. MIS is based on the simple idea that, under certain conditions, we can separate the Monte Carlo estimator into a weighted sum of multiple estimators. Each of these estimators uses a different *sampling strategy* (i.e. a different PDF) for sampling the integrand. After we generate a sample from a strategy, we can assign it a weight based on the PDFs of the other strategies. This technique allows us to retroactively downweigh the negative effects of an inappropriate sampling strategy.

The MIS 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, $(s = 0, t = l), (s = 1, t = l - 1), \dots, (s = l, t = 0)$. In this setting, $t = 0$ represents the case where the emitter path intersects the camera directly, and $s = 0$ corresponds to standard path tracing. Note that we can evaluate the probability of path having been generated by any of these strategies by calculating what the probability of sampling the vertices would have been, had they been sampled from the other direction.

Denoting the BDPT sampling strategy by the length of the sensor subpath before the deterministic connection, t, we define the MIS contribution of the j^{th} sampled path $\bar{x}_j$ as

$$\tilde{I}_t(\bar{x}_j) = w_t(\bar{x}_j) \frac{f(\bar{x}_j)}{p_t(\bar{x}_j)}, \qquad (3)$$

where $w_t(\bar{x}_j)$ represents the MIS weight of strategy t for $\bar{x}_j$, and $p_t(\bar{x}_j)$ the probability of $\bar{x}_j$ having been sampled. The full MIS estimator for paths of length l is then given by

$$\tilde{I}_l = \sum_{t=0}^{l} \frac{1}{N_t} \sum_{j=1}^{N_t} \tilde{I}_t(\bar{x}_j), \qquad (4)$$

where N_t is the total number of samples made from strategy t. The standard balance heuristic for paths, as presented by Veach and Guibas [11], is given as

$$w_t(\bar{x}_j) = \frac{p_t(\bar{x}_j)}{\sum_{i=1}^{l} p_i(\bar{x}_j)}. \qquad (5)$$

3 Probabilistic Connections for Bidirectional Path Tracing

PCBPT extends BDPT to allow for additional reuse of both sensor and eye paths, and introduces *importance sampling* of the sensor and eye path connections. Additionally, it modifies multiple importance sampling (MIS) to better suit the problem at hand.

In each iteration, for each pixel, PCBPT generates an emitter and a sensor path of infinite length [2]. The first M emitter paths are stored in a cache. Then, for each sensor subpath PCBPT importance samples K emitter subpaths from the cache to which it can connect. We present one iteration of the high-level PCBPT algorithm in Algorithm 1, and discuss the theoretical details behind this in Section 3.1.

In order to sample the emitter subpaths, PCBPT uses an importance sampling scheme similar to the one presented by Georgiev et al. [4], which we discuss in more detail in Section 3.1.3.

While path reuse can save on computational resources, the downside is that it potentially introduces correlation between two sampled paths, which in turn invalidates the

[2]Naturally, in practice the paths either get truncated or attenuated to zero by Russian roulette.

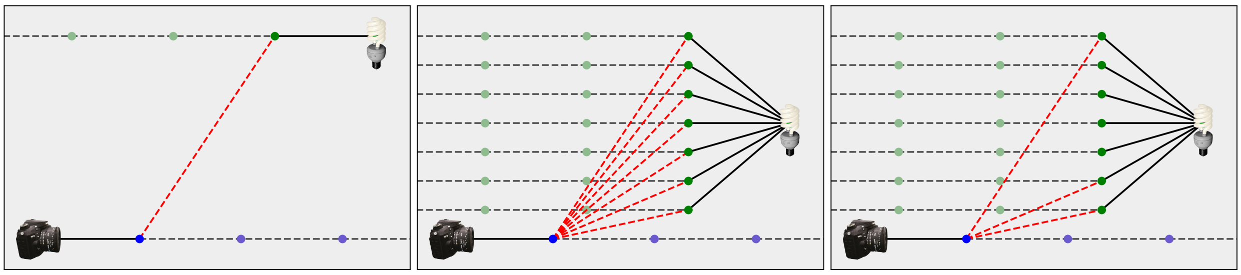


Figure 1: Different approaches to sampling from the $t = 2$ strategy for paths of length $l = 4$. Left: In each iteration, BDPT generates only one sample for this strategy, i.e., $N_t = 1$ in Equation 4. Note that the sample is made deterministically. Middle: An alternative approach would be to cache M emitter subpaths, and connect to all of them, i.e., $N_t = M$. This method corresponds to *combinatorial bidirectional path tracing* [8]. Right: Instead of evaluating all M cached connections, PCBPT importance samples $K \ll M$ of them. This way, we only connect to the high-contribution subpaths, and fewer connections need to be evaluated.

variance-reduction guarantees given by the balance heuristic. To mitigate this, Popov et al. [9] also present a novel derivation of the balance heuristic for correlated samples. We provide the definition of the modified balance heuristic in Section 3.2.

It should be noted that PCBPT evaluates the standard BDPT contribution for any pair of subpaths where the number of vertices in either subpaths is less than some threshold. 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. The subpath-length threshold between BDPT and PCBPT, here set to 2, could be changed to an arbitrary number larger than 2. However, in practice we want to utilize the benefits PCBPT offers, and therefore set the threshold as low as possible.

Algorithm 1: One iteration of the PCBPT algorithm.

```
1  Generate the emitter paths.
2  Generate the sensor paths.
3  Evaluate the path tracing and light tracing
     contributions.
4  for $i \leftarrow 1$ to *number of importance cache paths.* do
5      Generate sensor path $\bar{z}^c$.
6      for $\mathbf{z}_t^c \in \bar{z}^c$, where $t = 2$ to $l_{\bar{z}}^c$ do
7          Generate the PMF for $\mathbf{z}_t^c$.
8          Create an importance-cache record from $\mathbf{z}_t^c$.
9      end
10 end
11 for $i \leftarrow 1$ to *number of pixels in the image* do
12     Generate a sensor path, $\bar{z}$.
13     Find the closest importance-cache records.
14     Interpolate the PMF.
15     Sample the interpolated PMF.
16     Add the contribution to the $i^{\text{th}}$ pixel estimate.
17 end
```

3.1 Importance Sampling Emitter Paths

3.1.1 Connecting to Multiple Emitter Subpaths

The PCBPT algorithm is constituted by a number of building blocks, the most basic of which only involves connecting a sensor subpath to multiple emitter subpaths, similar to *combinatorial bidirectional path tracing* [8]. The part of the algorithm dealing with probabilistically connecting to a subset of these subpaths can be thought of as an additional optimization. As in the original paper, in following sections, we ignore the connections between subpaths which are handled by standard BDPT methods, and limit our discussion to the evaluation of the integral over paths of length l, I_l, without loss of generality.

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 $s+t = l$. Following the definitions from Equations 3 and 4, our MIS MC estimator can be written as

$$\tilde{I}_l = \sum_{t=0}^{l} \frac{1}{M} \sum_{j=1}^{M} \tilde{I}_t(\bar{x}_j). \tag{6}$$

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

3.1.2 Probabilistic Connections

In order to importance sample emitter subpaths, Popov et al. [9] suggest using another MC estimator over the inner sum. In other words, they suggest evaluating a Monte Carlo estimate of $S_t = \sum_{j=1}^{M} \tilde{I}_t(\bar{x}_j)$ by importance sampling an interpolated probability mass function (PMF) to generate K samples of $\tilde{I}_t(\bar{x}_j)$. 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 inner sum is then given by

$$\tilde{S}_t = \frac{1}{K} \sum_{\bar{x}_k \in \mathscr{K}} \frac{\tilde{I}_t(\bar{x}_k)}{\mathrm{pmf}(\bar{x}_k)}, \tag{7}$$

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

$$\tilde{I}_l = \sum_{t=0}^{l} \frac{1}{M} \frac{1}{K} \sum_{\bar{x}_k \in \mathscr{K}} \frac{\tilde{I}_t(\bar{x}_k)}{\mathrm{pmf}(\bar{x}_k)}. \tag{8}$$

3.1.3 Probability Mass Function Caching

The perfect PMF for a sensor vertex, s.t. $\mathrm{Var}[\tilde{S}_t] = 0$, is achieved when the probability of sampling an emitter subpath is proportional to the contribution of the path which would be generated by connecting to that subpath.

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 j^{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. As the sensor subpath is reused for all connections, its contributions cancel out, making the PMF depend only on the emitter subpaths and the last vertex of the sensor subpath,

$$p^{\mathrm{conn}}(j) = \frac{\frac{f(\bar{x}^{(j)})}{p(\bar{x}^{(j)})}}{\sum_{k=0}^{M} \frac{f(\bar{x}^{(k)})}{p(\bar{x}^{(k)})}} = \frac{\frac{f^{\mathrm{conn}}(\bar{y}^{(j)},\bar{z}) f(\bar{y}^{(j)})}{p(\bar{y}^{(j)})}}{\sum_{k=0}^{M} \frac{f^{\mathrm{conn}}(\bar{y}^{(k)},\bar{z}) f(\bar{y}^{(k)})}{p(\bar{y}^{(k)})}}. \tag{9}$$

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 for the original paper by Popov et al. [9].

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

3.2 Multiple Importance Sampling for Correlated Paths

Reusing paths leads to sample correlation, which in turn leads to an overall increase in variance. Popov et al. [9] 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 for their paper. The final modified MIS weights are given by

$$w_t(\bar{x}) = \frac{N_t p_t(\bar{x})}{\sum_{i \in S_u} N_i p_i(\bar{x}) + \sum_{i \in S_c} p_i(\bar{x})}, \quad t \in S_u \tag{10}$$

$$w_t(\bar{x}) = \frac{p_t(\bar{x})}{\sum_{i \in S_u} N_i p_i(\bar{x}) + \sum_{i \in S_c} p_i(\bar{x})}, \quad t \in S_c, \tag{11}$$

where S_u represents the set of uncorrelated and S_c the set of correlated samples, N_i represents the number of samples in strategy i. 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.

Intuitively, since one "bad" emitter subpath which produces high-variance samples when connected to can influence many pixels, the modified balance heuristic tries to decrease the weights assigned to these samples.

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 Section 4.2 we describe our implementation in detail, along with comprehensive pseudo-code in Algorithms 2 and 3. 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 Section 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. [9], 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 the variance-reduction guarantees of the modified balance heuristic still hold in this case.

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 would be a subpath 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 $0.004 \times W \times H$ 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 $\mathbf{z}^c$. For each $\mathbf{z}^c$, 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 kD tree [1], 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* [3]. 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 2.

The function *EvalBDPT* from Algorithm 2 represents the standard BDPT subpath connection algorithm, except that in this case, it only connects subpaths where $s < 2$

Algorithm 2: One iteration of the PCBPT algorithm.

```
1  for $i \leftarrow 1$ to $M$ do
2      Generate and cache an emitter path.
3  end
4  for $i \leftarrow 1$ to number of importance cache paths. do
5      Generate sensor path $\bar{z}^c$.
6      for $\mathbf{z}^c_t \in \bar{z}^c$, where $t = 2$ to $l^c_{\bar{z}}$ do
7          Generate the PMF for $\mathbf{z}^c_t$.
8          Add $\mathbf{z}^c_t$ to a $k$D tree as an importance-cache
             record.
9      end
10 end
11 for $i \leftarrow 1$ to number of pixels in the image do
12     Generate a sensor path, $\bar{z}$.
13     if $i < M$ then
14         $\bar{y}^{\mathrm{BDPT}} \leftarrow i^{\mathrm{th}}$ emitter path from the emitter
             cache.
15     end
16     else
17         $\bar{y}^{\mathrm{BDPT}} \leftarrow$ generate a new emitter path.
18     end
19     $contribution \leftarrow$
         $EvalBDPT(\bar{z}, \bar{y}^{\mathrm{BDPT}}) + EvalPCBPT(\bar{z})$;
20     Add contribution to the $i^{\mathrm{th}}$ pixel estimate.
21 end
```

or $t < 2$. The function *EvalPCBPT* evaluates the PCBPT contribution and is shown in more detail in Algorithm 3

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.

As explained in Section 4.1, we follow Mitsuba's practice and assume that the number of uncorrelated samples per iteration equals one. Setting these numbers in Equation 10, we see that the separation between the correlated and uncorrelated samples becomes irrelevant, and that the

Algorithm 3: *EvalPCBPT*

```
Input: Sensor path $\bar{z}$
Output: PCBPT Contribution to the pixel estimate
1  $contribution \leftarrow 0$
2  for $\mathbf{z}_t \in \bar{z}$, where $t = 2$ to $l_{\bar{z}}$ do
3      $\bar{z}_t \leftarrow$ subpath ending in vertex $\mathbf{z}_t$.
4      Query 6 importance-cache points which are
         spatially closest to $\mathbf{z}_t$.
5      for $k \leftarrow 1$ to $K$ do
6          Use binary search to lazily inversion sample
             the interpolated PMF and choose an emitter
             subpath $\bar{y}_s^{(j)}$.
7          $s \leftarrow$ length of the subpath $\bar{y}_s^{(j)}$.
8          $p^{\mathrm{conn}}(j) \leftarrow$ probability of having sampled $\bar{y}_s^{(j)}$
             from the PMF.
9          $\bar{x} \leftarrow (\bar{z}_t, \bar{y}_s^{(j)})$.
10         $contribution \leftarrow$
             $contribution + w_s(\bar{x}) \frac{f(\bar{x})}{p^{\mathrm{conn}}(j)p(\bar{x})}$.
11     end
12 end
13 $contribution \leftarrow \frac{1}{KM} contribution$
14 return $contribution$
```

final MIS weight becomes simply

$$w_t(\bar{x}) = \frac{p_t(\bar{x})}{\sum_{i \in S_u \cup S_c} p_i(\bar{x})}, \tag{12}$$

for all t.

Given that Mitsuba assumes that the number of samples equals one for all strategies, our implementation simply reuses Mitsuba's MIS functionality. 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.

4.3 From Theory to Practice

During implementation, we found it useful to separate the algorithm into smaller testable pieces. Accordingly, we have developed a potentially useful path for taking PCBPT from theory to practice. Here, we present the steps which we took in order to arrive at the end result.

1. First, we created a new integrator, paying attention to the fact that PCBPT cannot be parallelized on tile-by-tile basis. Instead each thread performs one iteration of the algorithm and renders an entire image, with a fraction of the total samples per pixel.

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 6. Note that the algorithm at this point equates to *combinatorial bidirectional path tracing* [8], 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 8, replacing Equation 6.

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.

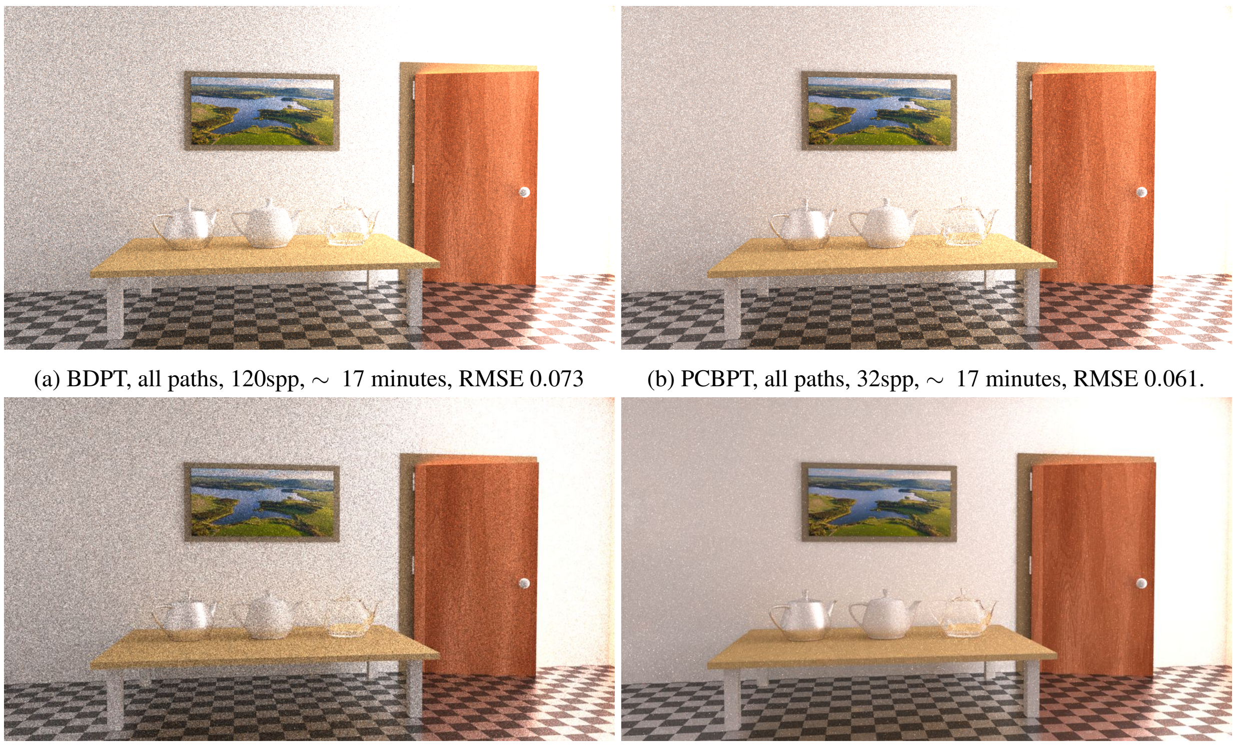
7. Finally, we implemented the lazy-evaluation binary search on the 6 nearest neighbors to sample from the CDF.

We also found it important to be able to display PCBPT connections separately from the BDPT connection during development and testing. It was also useful to have scenes with highly occluded emitters since this is where PCBPT has the largest effect.

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 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. 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 [2]. The rendering was done on a commodity Lenovo Y50-70 laptop with an Intel i7-4710HQ processor operating at 2.50GHz


(a) BDPT, all paths, 120spp, $\sim$ 17 minutes, RMSE 0.073 (b) PCBPT, all paths, 32spp, $\sim$ 17 minutes, RMSE 0.061.

(c) BDPT, inner paths, 120spp, $\sim$ 15 minutes, RMSE 0.088. (d) PCBPT, inner paths, 32spp, $\sim$ 15 minutes, RMSE 0.0040.

Figure 2: *Veach Ajar Door* scene equal-time comparison.

and 16GB of RAM. All of the scenes were rendered with $M = 100$ cached paths, and $K = 10$ connections.

In order to verify our algorithm, we rendered the *Cornell Box* scene using both BDPT and PCBPT and compared the RMSE values for both images. We have noticed a major performance penalty for such scenes when using PCBPT, since this is 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.

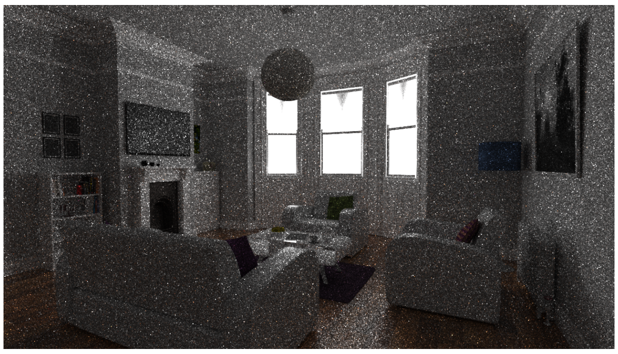
5.1 Comparisons

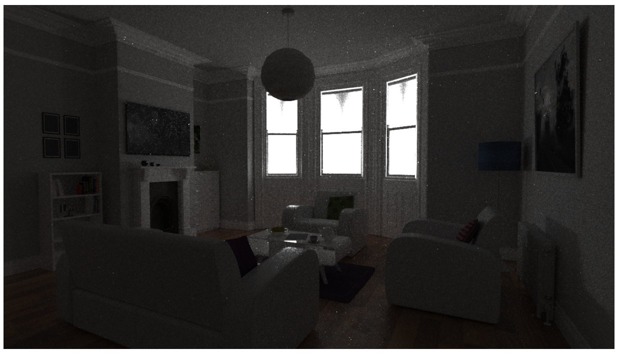
We created equal-time comparisons between BDPT and PCBPT on a number of difficult scenes. For these scenes, we rendered the ground-truth images using a large number of samples per pixel, against which we calculated the normalized RMSE. We present the ground-truth images as well as the results in Section 5.2.

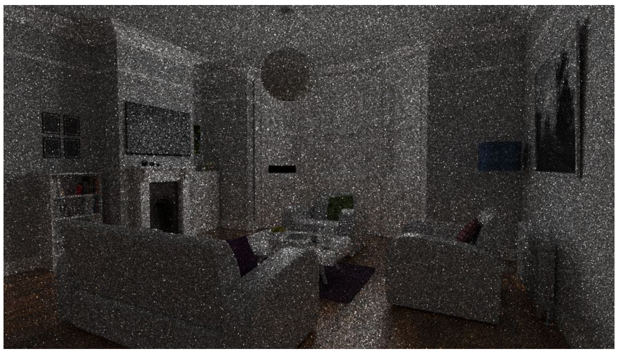
The *Veach Ajar Door* represents a scene with an occluded emitter. More precisely, the emitter is located directly behind a slightly ajar door, emitting light into the room through the opening. This scene is potentially difficult for classical path tracing, as the sensor paths are unlikely to find the emitter. Figure 2 shows the comparison between the two images rendered using all paths, i.e., both the inner as well as the outer connections, and the comparison of only the inner paths.

BDPT and PCBPT have a comparable error in the images where all paths are 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 2, we can see that the inner paths do converge a lot faster with PCBPT than with BDPT. For the inner paths, RMSE improved by around 4%, while only an 1% improvement was achieved 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 for the inner paths.

We modified *The White Room* scene 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 in order to produce a more difficult lighting condition. We notice significant improvements for this scene when using PCBPT (Figure 3). 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. Note that the image rendered using standard BDPT samples high-contribution paths with low probability, resulting in firefly artifacts, which require many samples to disappear (compare with Figure 6).


(a) BDPT, all paths, 120spp, $\sim$ 16 minutes, RMSE 0.130.


(b) PCBPT, all paths, 32spp, $\sim$ 16 minutes, RMSE 0.022.


(c) BDPT, inner paths, 120spp, $\sim$ 11 minutes, RMSE 0.147.

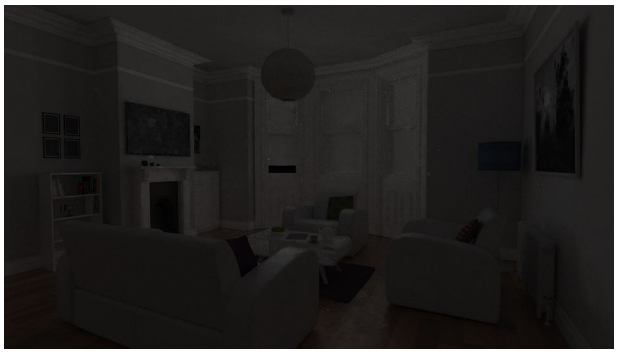

(d) PCBPT, inner paths, 32spp, $\sim$ 11 minutes, RMSE 0.013.

Figure 3: *The White Room* scene equal-time comparison.

The Dining Room scene represents a failure case (Figure 4). 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, which is 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. Note that we had to artificially adjust the brightness and contrast of the inner path images, in order for them to be visible in the printed version of the paper [3].

5.2 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, 6, and 7. For an easier overview, we show all RMSE values in Table 1, as well as the ratio between the RMSE for the BDPT image, E_B, and the RMSE for the PCBPT image, E_P.

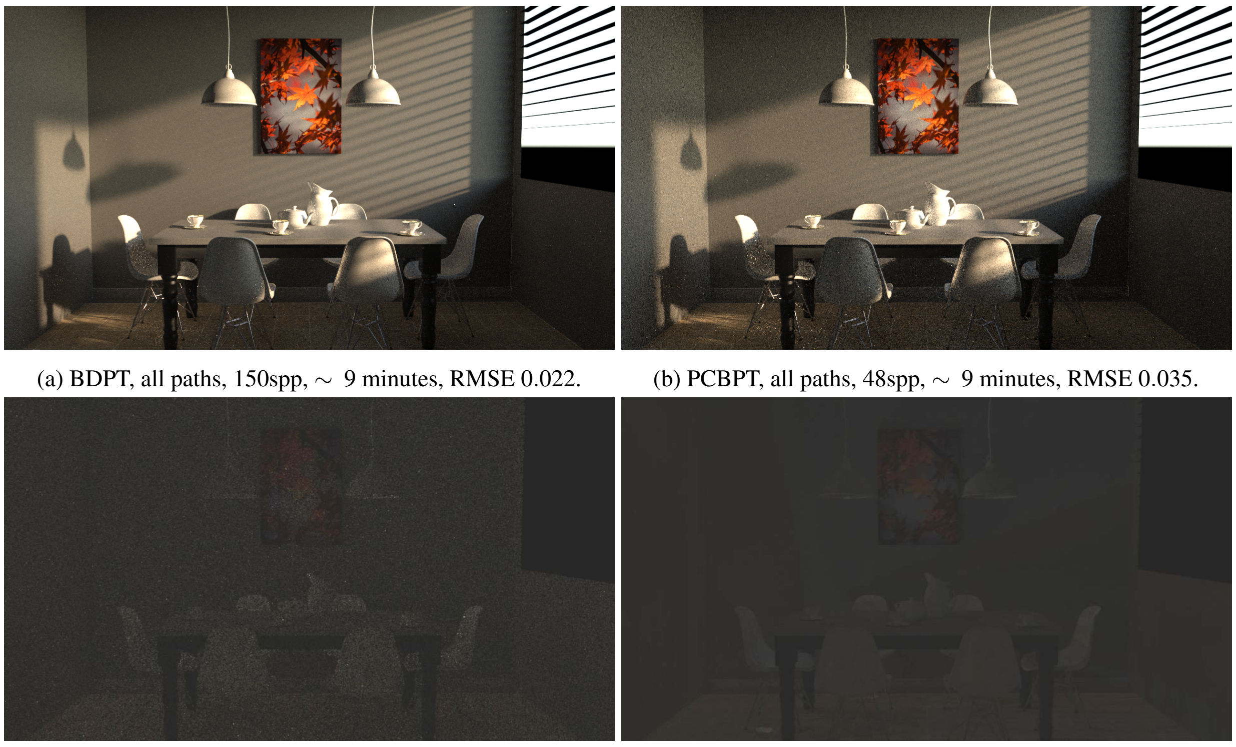
All Paths	BDPT	PCBPT	$\frac{E_B}{E_P}$
Veach Ajar Door	0.073	**0.061**	1.20
The White Room	0.130	**0.022**	5.90
The Dining Room	**0.022**	0.035	0.63
Average	0.075	0.040	2.60
Inner Paths	BDPT	PCBPT	$\frac{E_B}{E_P}$
Veach Ajar Door	0.088	**0.040**	2.20
The White Room	0.147	**0.013**	11.31
The Dining Room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

Table 1: Normalized Root Mean Square Error for the images from Section 5.1. A lower value means a better result.

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.

[3] The unaltered version of the images is available in the thesis version of this work, available at `https://www.cg.tuwien.ac.at/research/publications/2017/dodik-2017-pcbpt/`


(a) BDPT, all paths, 150spp, $\sim$ 9 minutes, RMSE 0.022. (b) PCBPT, all paths, 48spp, $\sim$ 9 minutes, RMSE 0.035.

(c) BDPT, inner paths, 150spp, $\sim$ 5 minutes, RMSE 0.021. (d) PCBPT, inner paths, 48spp, $\sim$ 5 minutes, RMSE 0.011.

Figure 4: *The Dining Room* scene equal-time comparison.

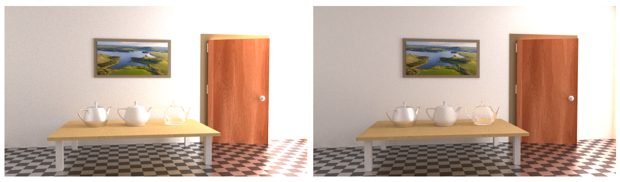

(a) All Paths. (b) Inner paths.

Figure 5: *Veach Ajar Door* ground-truth images.

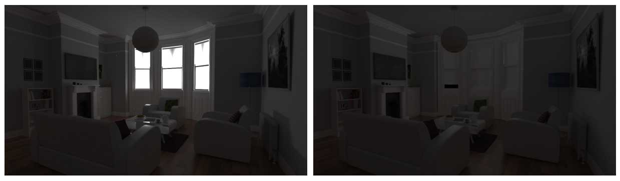

(a) All paths. (b) Inner paths.

Figure 6: *The White Room* ground-truth images.

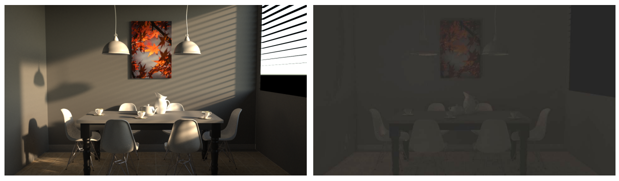

(a) All Paths. (b) Inner paths.

Figure 7: *The Dining Room* ground-truth images.

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. [9]. 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 the parameters. We were unable to test our implementation on the scenes they used as they were not publicly available.

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. PCBPT is orthogonal to methods which improve performance for paths where $s < 2$ or $t < 2$, as well as methods which help with specular-diffuse-specular paths, such as *vertex connection and merging* [5], or *Metropolis light transport* [12]. 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 by 11% compared to BDPT when taking into account both inner and outer paths, and by 13% for just the inner paths.

Acknowledgements. I would like to thank Dr. Stefan Popov for his endless patience with replying to the many questions I had, as well as my mentor, Hiroyuki Sakai, for going above and beyond in his duties.

References

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

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

[3] Boost. Boost c++ libraries. `http://www.boost.org/`, 2017. Last accessed 2017-08-29.

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

[5] Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.*, 31(6):192:1–192:10, November 2012.

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

[7] 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.

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

[9] 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.

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

[11] 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.

[12] 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.