Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2018)
T. Hachisuka and W. Jakob (Editors)

Primary Sample Space Path Guiding

Jerry Jinfeng Guo[†1], Pablo Bauszat[‡1], Jacco Bikker[§2], and Elmar Eisemann[¶1]

[1]Delft University of Technology, the Netherlands
[2]Utrecht University, the Netherlands

Abstract

Guiding path tracing in light transport simulation has been one of the practical choices for variance reduction in production rendering. For this purpose, typically structures in the spatial-directional domain are built. We present a novel scheme for unbiased path guiding. Different from existing methods, we work in primary sample space. We collect records of primary samples as well as the luminance that the resulting path contributes and build a multiple dimensional structure, from which we derive random numbers that are fed into the path tracer. This scheme is executed completely outside the rendering kernel. We demonstrate that this method is practical and efficient. We manage to reduce variance and zero radiance paths by only working in the primary sample space.

CCS Concepts

•***Computing methodologies*** → *Ray tracing;*

1. Introduction

Simulating light transport for global illumination effects has been a standard approach for offline rendering. The ultimate goal is to capture both *direct illumination* and *indirect illumination*. Light transport is simulated in such a way that a path, possibly composed of multiple scattering events, connects a light source and the sensor. Direct and indirect illumination are both collected along the path and finally contribute to a pixel, possibly with non-zero luminance. This process is modeled by the *rendering equation* [Kaj86]:

$$L(\omega_o \leftarrow \mathrm{x}) = L_e(\omega_o \leftarrow \mathrm{x}) + \int_\Omega L(\mathrm{x} \leftarrow \omega_i) f_s(\omega_o \leftarrow \mathrm{x} \leftarrow \omega_i) G(\mathrm{x}, \omega_i) d\omega_i. \quad (1)$$

As the integral of luminance in the rendering equation is high dimensional and generally impossible to solve analytically, *Monte Carlo* methods are usually employed.

Despite the many advantages of Monte Carlo methods, the noise and the slow convergence rate are two major pitfalls. Various *importance sampling* techniques have been developed to reduce variance and accelerate convergence. The common goal is to find an "optimal" distribution that can approximate the true integral. In practice, a distribution that resembles the optimal distribution is known to have good performance [Vea97].

Two of the many open problems addressed in this work are:

- too many samples are wasted on paths that do not contribute to the final result;
- inappropriate probabilities are assigned to paths that do not match their actual contribution.

One approach to solve the above problems is to model light paths as *states* in *Markov chains* (MC) and the rendering problem is transformed into a *Markov chain Monte Carlo* (MCMC) process. This approach is also called the *Metropolis light transport* (MLT) [VG97]. The core idea of MLT is to mutate existing paths by either altering them a little bit (small mutation) or by completely sampling a new path (large mutation). The major advantage of MLT is that once a path with significant luminance is found, the corresponding region in the path space will be explored extensively, making MLT very suitable for scenes with difficult lighting configurations. The scheme of doing both small and large mutations also guarantees ergodicity. However, MLT suffers from many disadvantages, including start-up bias, poor stratification, and unstable convergence. This prevents its adoption in production especially for animation [VKŠ*14, HEV*16].

One new trend is to build distributions based on the results of a few pilot runs and use these distributions for rendering. Most *path guiding* methods use this technique. Existing path guiding schemes usually work in the spatial domain. One intuitive approach is to cache a sparsely distributed set of records in the scene. To make use of them later on, firstly new entries are queried in the spatial structure and then queried results are interpolated to give a final record.

† J.Guo-3@tudelft.nl
‡ P.Bauszat@tudelft.nl
§ J.Bikker@uu.nl
¶ E.Eisemann@tudelft.nl

We propose to perform path guiding in a different domain: the *primary sample space* (PSS), i.e., the space of random numbers that are used to generate paths. Our scheme works in two phases: a collecting phase comprising of one or more iterations and a rendering phase. During the collecting phase, we take as input a the random numbers used by a few generated paths together with corresponding luminance values returned by these paths and build a structure that will encode these combinations. During the rendering phase, we sample the previously built structure and use the resulting samples to feed the path tracer. By working in the primary sample space, we are able to build distributions and sample from these distributions outside the path tracing kernel. This makes it easy to add our method to existing solutions while still enjoying the many benefits of being an unbiased Monte Carlo method.

2. Related Work

Importance Sampling Variance reduction is the goal for almost all Monte Carlo optimization methods. For a simplified version of Equation 1: $L = L_e + \int L(\omega) f_s(\omega) G(\omega) d\omega$, the corresponding Monte Carlo estimator can be denoted as

$$\hat{L} = L_e + \frac{1}{N} \sum_{i=1}^{N} \frac{L(\omega_i) f_s(\omega_i) G}{p(\omega_i)}, \tag{2}$$

where L denotes luminance, f_s denotes the *bi-directional scattering distribution fuction* (BSDF), G denotes the geometry term and p is the probability associated with the current sample ω_i.

The goal of importance sampling is to make the shape of $p(\omega)$ as close to the numerator as possible [DBB06]. The original integrand is composed of three terms: L, f_s and G, one of which in practice is considered the dominant factor. Sampling according to each term has their own advantages. For instance, sampling G favors light coming from above over light coming from the horizon, which complies with the fact that irradiance is cosine weighted. In practice, this cosine weighted sampling is used as the core method to generate an incoming direction for many BSDF implementations, as for most of them there is no easy analytic inversion [PJH16]. Highly glossy materials generally benefit from sampling f_s, as the reflection cone is usually concentrated around a certain direction. Sampling according to L usually works better for diffuse surfaces since the BSDF is not peaky but it requires global knowledge, which is typically difficult for scenes with unstructured lights and complex visibility.

Multiple importance sampling (MIS) is a technique that combines different sampling strategies [VG95b]. This is done by evaluating the current configuration by using each strategy and weighing the final contribution by balancing all involved pdfs using a certain heuristic. The advantage of MIS is that it automatically gives a distribution that uses all involved strategies, thus making it less likely to yield inappropriate probabilities. One direct application is in *bi-directional path tracing* (BDPT) [VG95a], where for each sample two pre-traced sub-paths are connected vertex-wise. The core efficiency therein is the massive reuse of existing path vertices and MIS is the tool that makes it practical [HPJ12].

Many works try to importance sample one or more terms. Few early BSDF models such as *Phong* can be explicitly inverted and consequently be directly importance sampled [Shi91, War92, LFTG97]. Sampling unstructured light sources, known as *next event estimation* (NEE), is common practice [DBB06]. Similarly, sampling structured light sources (such as an environment map) has been investigated in [Deb08]. All of the above methods treat one term in the equation as the dominant factor and provide good results in respective applicable scenarios. Sampling according to the product of multiple terms is studied in [CJAMJ05], where a structured environment map is pre-convolved with BSDFs using wavelets.

Primary Sample Space In the original MLT, mutations are done in the path space, a space $\mathcal{P}$ of paths with all possible lengths:

$$\mathcal{P} = \bigcup_{i=2}^{\infty} \mathcal{P}_i, \tag{3}$$

$$\mathcal{P}_i = \{\bar{\mathrm{x}}_i | \bar{\mathrm{x}}_i = \{\mathrm{x}_0, \mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \ldots, \mathrm{x}_i\}\}, \tag{4}$$

where x_i is a path vertex in the spatial domain. However, mutating in path space is difficult: a slight mutation of one vertex could lead to a different path, possibly with different lengths and visibility configurations. Although several methods such as *Manifold exploration* [JM12] solve the path mutation problems for specular surfaces, in practice mutating in path space is still difficult.

Primary Sample Space Metropolis Light Transport (PSSMLT) offers a simplified version of MLT, where mutations are done by changing random numbers in PSS instead of transforming vertices in the original path space [KSKAC02]. The high dimensional vector of random numbers is then transformed into actual light paths for final transport simulation. Many MLT variants are based on PSSMLT. *Replica Exchange Light Transport* [KKK09] introduces jumping between mutation strategies. *Multiplexed Metropolis Light Transport* (MMLT) [HKD14] improves efficiency by fixing path length when mutating samples.

The core advantage of PSSMLT is that it transforms an ultra high dimensional problem in path space into a reasonably high-dimensional problem in PSS. This simplification makes MLT simpler to implement and offers another advantage, which is not by design: by altering random numbers only, we are still able to perform local importance sampling at certain locations. Our method is based on this foundation and works in the same space.

Path Guiding Since we are simulating light transport, applying importance sampling according to a certain distribution at a scattering event to get an incoming direction translates to guiding a random walk in path tracing.

The *Photon map* was first used to guide path tracing [JC95], where at each bounce a 2D histogram over the directional domain of surrounding flux information is used to sample an incoming direction. Density estimation of photons is used to approximate the L term. This achieves faster convergence than regular path tracing at the expense of huge memory consumption and slow preprocessing. Instead of using a fixed-resolution histogram, one can also use cone-shaped splats on the hemisphere to represent directional distributions [HP02]. The latter gives better results in terms of variance reduction than a fixed-resolution histogram at the expense of

significantly more computation. Both above methods are particle-based and require caching of certain information, which can be expensive.

A *Gaussian mixture model* (GMM) is used in [VKŠ*14, HEV*16] to represent incoming luminance. Their method features an online learning scheme and the introduction of GMMs gives superior performance over histogram or cone-shaped particle footprints. However, for scenes with small light sources or complex visibility, GMMs tend to be unstable and perform unpredictably. Light transport is modeled as a *Q-learning* process in [DK17] and previous samples are used to guide successive samples in an iterative manner. This reinforced light-transport scheme, using results from all samples, is thus biased but consistent [MGN17]. Recently a simple path guiding scheme is introduced in [MGN17], where an adaptive quad tree is used to represent a directional domain, while each record is associated with a leaf node in a k-d tree of the spatial domain. This method benefits greatly from averaging incoming luminance over the entire leaf node, which guarantees an abundance of information gathered in each iteration. This method also features a reinforced iterative scheme, with which samples in each iteration are solely guided by the distribution built in the last iteration. This not only guarantees unbiasedness but also reinforces the distribution with minimum effort.

2.1. Distinction

In previous methods for path guiding, records of directional representation are stored sparsely in the spatial domain. The main distinctive feature of our method is that we work in the primary sample space and we do not explicitly build any spatial-directional structure.

3. Method

3.1. Formulation

Equation 1 can be re-formulated as a series:

$$L = \beta_0 E_1 + \beta_0\beta_1 E_2 + \ldots + \prod_{i=0}^{n-1} \beta_i E_n + \ldots \tag{5}$$

where β_i is the unit throughput $f_s G/p$ at the i-th bounce. E_i denotes the emission of the i-th intersected surface. Note that in this formulation, direct illumination sampling is considered another path.

With reference to Figure 1, where a path of length 4 is illustrated, we can model rendering as a function of random number inputs, i.e.:

$$\begin{aligned} L &= f_{\text{path}}(\bar{\mathrm{x}}) \\ \bar{\mathrm{x}} &= T(\{u\}^d) \\ \{u\}^d &= \{u_0, u_1, u_2, u_3, \ldots\}, \end{aligned} \tag{6}$$

where $T(\{u\}^d)$ is a function that transforms certain dimensions of random numbers $\{u\}^d$ into a path $\bar{\mathrm{x}}$. By building a multi-dimensional distribution over this space, we approximate the full integral implicitly, i.e., Lf_sG.

It is worth pointing out that PSS is still constrained by the dimensionality. That is, we can only process sample vectors with a limited length. In practice this makes sense, as shorter paths are more likely to bring back luminance than longer ones [AK90]. To keep the method unbiased, when extracting the random numbers of a path from this structure, we do not stop the path on the position defined by the last random numbers, but continue tracing the path in a Monte-Carlo fashion.

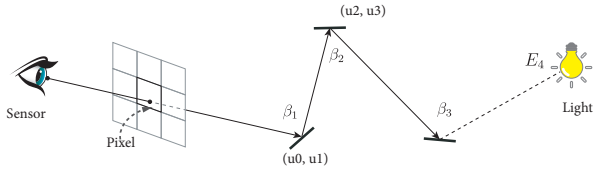

Figure 1: *A path of length* 4*, where the light source for the last bounce is connected by either NEE or BSDF sampling. Suppose all three bounces hit non-emissive surfaces. Then the final luminance brought back by this path is* $L = \beta_1\beta_2\beta_3 E_4$*. In this example, each bounce takes two random numbers to generate an incoming direction. For the first two bounces, the corresponding path record is* $\{\{u_0, u_1, u_2, u_3\}, L\}$*.*

With this formulation, we build a structure with input gathered from a path tracer $\{\{u\}^d, L\}$ so that the distribution contained in our structure is proportional to $\prod_{i=0}^{n-1} \beta_i E_n$. ($d = n \times m$ where n is the bounces we work on, and m is the amount of random numbers used per bounce). In practice, in our examples, we have $n = 2$ and $m = 2$.

3.2. Overview

An overview of the general process is given in Figure 2. In a regular path-tracing solution, *sampler*s are used to generate uniformly-distributed random numbers. The main role of samplers is to maintain a certain level of stratification across dimensions, which is crucial for exploration. In our method, we will use a structure to influence the random number generation. With proper and careful transformation of the sample distribution and probability, we can distribute samples in each dimension such that they can better contribute to the final result.

Our method works in two phases: collecting and rendering, which are iterated. In the first phase, records are collected to build a structure. In the rendering phase, instead of using uniformly-distributed random numbers to generate paths, we sample the structure to subsequently generate paths. Following the scheme as in [MGN17], we fuse the collecting and rendering within one single rendering budget, i.e., total samples.

More specifically, in the collecting phase, we use all path records from the rendering stage (u, L), where u is the random number vector, which corresponds to the path, and L is the brought-back luminance, add them to the structure and will refine it accordingly. Initially, in the first iteration, the structure is empty and will simply return uniformly distributed random numbers.

During the rendering phase, the successively sampled random numbers will be based on the structure, which is influenced by the path records collected in the previous iteration. This process can be

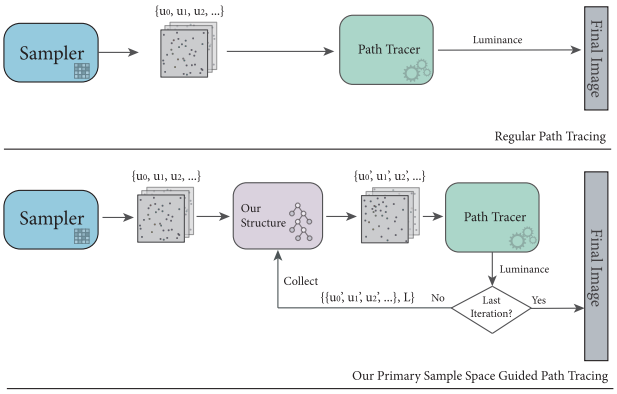

Figure 2: *A flow chart view of regular path tracing (above) and our method (below). Notice how our method is inserted into an existing pipeline and not inside the kernel.*

seen as working in the primary sample space (instead of the spatial domain plus the directional domain). Hereby, we are automatically freed from the trouble of building sparsely distributed records and the interpolations incurred.

The rest of the rendering process remains largely untouched; only the random numbers that are used to feed the path tracer are altered. Multiple importance sampling is used to combine samples from our structure and regular uniform samples to ensure that samples cover corresponding domains properly without being stuck in local maxima.

Within one iteration, all samples are generated from the same unbiased distribution, thus results from one iteration remain unbiased. However, when we combine results across iterations, they would be no longer unbiased [DK17]. Thus, we adopt the scheme as in [MGN17] and only use the last iteration when outputting results to the final image. This process is demonstrated in Algorithm 1.

3.3. k-d tree

In this part, we will detail our structure and how to use it to influence the random-number generation process.

Traditional path guiding methods work in a $5D$ domain where 3 dimensions are used for spatial domain and the remaining for directional domain. In our case, the more bounces we take into account, the higher the dimension. As a common practice in implementation, each scattering event usually takes a fixed amount of random numbers [PJH16]. Recent methods from Metropolis Light Transport also assume fixed dimensions of random numbers when inverting paths to primary samples [Pan17]. It is therefore reasonable to assume that each bounce consumes the same amount of random numbers. We will denote n the amount of bounces and m the involved random numbers, thus our k-d tree spans $n \times m$ dimensions.

For the high dimensional problem that we are facing, one proper structure is the *multiple dimensional binary tree* [Ben75], also known as the *k-d tree*. The advantage of using a k-d tree instead of a hyper grid or any other structure is that k-d trees have $O(N)$ memory efficiency and $O(\log N)$ query efficiency. We use a k-d tree that switches axis in a round robin fashion and always splits in the center of current node range.

Algorithm 1 Pseudo code for PSS path guiding. The only add-ons to a PBRT style render kernel are highlighted in red.

```
1:  procedure RENDER(scene, camera)
2:      ...
3:      for pixel in tile do
4:          kdTree.Init()
5:          for each iteration do
6:              for each sample in iteration do
7:                  ray ← camera.generate(pixel)
8:                  {u}^d ← uniform random numbers
9:                  {u'}^d ← kdTree.SampleTree({u}^d)
10:                 L ← Li(scene, ray, {u'}^d)
11:                 kdTree.Collect({{u'}^d, L})
12:             end for
13:             kdTree.Update()
14:         end for
15:     end for
16:     ...
17: end procedure
```

3.3.1. Data Structure Design

We use a pointer-less indexed tree structure, so the left child is always next to the parent in memory and a child pointer indicates the right child. We use a compact node structure of only 8 bytes, the child offset for inner nodes, which is used as a sample counter in leafs, is encoded on 31 bits, a leaf flag one bit, and the remaining bits are used to encode a node value.

3.4. Tree Construction

Our method follows in spirit three dimensional binary trees in the spatial domain [MGN17]. Whenever a path is complete, all path vertices are traversed and luminance from a vertex onward is accumulated to the leaf node where the vertex is located. Within the spatial tree leaf, all accumulated information is averaged and added to an associated directional quad-tree in world space. Similarly, we use all path records within a tile of pixels to construct our sample-space tree. We also adopt an iterative scheme, where samples from the current iteration are sampling results from the tree built in the previous iteration. Using samples from the same distribution defined by this tree gives unbiased results. For the initial iteration, with an empty tree, the output remains uniform. Figure 3 shows a visualization of two selected trees across iterations.

3.5. Tree Sampling

We maintain two trees: one for collecting and the other for sampling. After each iteration, we copy the collection tree to the sampling tree. We descend the k-d tree as in [MH97] from the root node and select a child according to the values contained therein

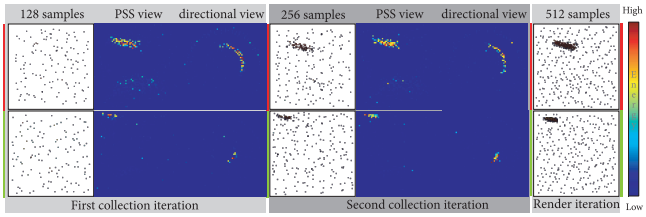

Figure 3: *Visualizations of the first two dimensions in k-d trees in two regions. In this case we use a two dimensional tree. Notice in the first iteration, samples are uniformly distributed. In the second iteration and the third iteration, samples are warped. Other regions remain covered properly thanks to MIS. Notice also that the directional views are merely for visualization, all collection and sampling are done in PSS.*

compared to a random number. In this way, we change the sampling process to follow previously recorded contributing samples.

Specifically, for node i, with a value recorded as v_i, children nodes j and k, the naïve probability of selecting either one is set to:

$$p_j{}' = \frac{v_j}{v_i} = \frac{v_j}{v_j + v_k} \tag{7}$$

$$p_k{}' = \frac{v_k}{v_i} = \frac{v_k}{v_j + v_k}. \tag{8}$$

Since we are dividing node i in half, the correct probability should include this scaling factor. Thus the correct probability is;

$$p_j = \frac{p_j{}'}{1/2} = \frac{2v_j}{v_j + v_k} \tag{9}$$

$$p_k = \frac{p_k{}'}{1/2} = \frac{2v_k}{v_j + v_k}. \tag{10}$$

The denominator in the expression of the current depth will cancel all the way to the root node and the probability of a node i being sampled will depend on node depth and node value:

$$p_i = \frac{2^{d_i} v_i}{v_{\text{root}}}, \tag{11}$$

where d_i is the depth of node i.

Once at the leaf node, the leaf node represents a $n \times m$-dimensional range (the range covered by the k-d tree leaf). We then sample a uniformly distributed vector $u \in [0,1)^{n \times m}$ and multiply it with the ranges to derive our new random variables to be used to define a path. To support paths of arbitrary lengths, beyond n bounces, we can simply feed the path tracing kernel with uniform random numbers after having used the components of u.

To be able to weigh the contribution of the chosen path correctly according to its probability, we keep track of the probability of the ranges that are represented in the leaf in form of a vector $(pdf_0^0, \ldots, pdf_{m-1}^0, \ldots pdf_{m-1}^{n-1})$. If all values in the tree are one, these probabilities would correspond to the size of the k-d tree cell. With the values in the nodes, these probabilities need to be adapted accordingly during the descent.

3.6. Rendering

Having derived random numbers to produce a path, we need to properly weigh its contribution. In our case, for bounce n, the sample probability p^n_{sample} is comprised of two parts; the conditional probabilities of previous dimensions and the probabilities of the current bounce:

$$p^n_{\text{sample}} = p(p^n | p^{n-1}_{\text{sample}}) = \prod_{i=0}^{n-1} p^i_{\text{sample}} \prod_{j=0}^{m-1} pdf_j^n \tag{12}$$

where p^i_{sample} is the sample probability of bounce i, p_j is the probability of the j-th random number for bounce n. This sample probability is multiplied with the Jacobian $||J||$ of the BSDF sampling method, e.g., for cosine weighted sampling where the sample is transformed into a direction ω:

$$p(\omega) = \frac{1}{||J||} \times p_{\text{sample}} = \frac{\cos\theta}{\pi} \times p_{\text{sample}}. \tag{13}$$

To avoid over exploration of certain region in PSS due to our reinforced scheme, MIS is used to combine samples from our structure and the original uniform samples. A guide probability is used to balance between the two. In all our tests, we use a guide probability of 0.5.

3.6.1. Record Collection

We start with an empty tree that has a uniform output. After the first iteration, to insert samples from the rendering stage, we take a path with random numbers u and luminance L. We traverse the tree according to u. At inner nodes, we simply add the path record value and then check into which child to descend. At leaf nodes, we not only add a path record value, but also increase the count variable in the corresponding leaf. When a threshold for the count is reached and increasing depth is still permissible, we split the current leaf node into two and update index information accordingly.

4. Implementation and Results

4.1. Implementation

We implemented our method in PBRT [PJH16] as a guided unidirectional path tracing integrator. The k-d tree is implemented as an acceleration structure. We build a k-d tree for each tile, as this requires minimum parallel implementation effort and distant pixels from other tiles are less likely to share correlated illumination condition.

Memory-wise, due to our scheme of working within tiles, the maximum memory is bounded by the number of threads times maximum memory for one tree. In our test where we use an eight core machine and a k-d tree max depth of 20, memory is in theory bounded by 128 MB. However in our low dimensional tests, we never observe higher memory consumption than 32 MB for all k-d trees during the whole rendering process. Since in our scheme memory is bounded and not an issue, we focus our verifications on other criterias.

4.2. Comparing Standards

In all our tests, we compare our results with non-guided results. Since the main objective of this work is to explore the possibility of guiding paths in PSS, we do not compare our results with existing path-guiding methods that work in the spatial-directional space.

Two comparing criteria corresponding to the two problems mentioned in section 1 are:

- **Zero-radiance path ratio** refers to the ratio of full paths that do not contribute to final results, as well as the NEE sampled direct-illumination subpaths that are occluded.
- **Mean square error** (MSE) is a common tool to evaluate convergence rate. Typically, the lower the MSE, the lower the variance.

To better visualize the difference between our method and standard path tracing, we off NEE at the first bounce. We leave NEE on from the second bounce onwards to improve finding non-zero light paths.

We test our method in four closed scenes with complex illumination configurations:

- **Ajar door scene**: most of the image is lit by indirect illumination
- **Dinning room scene**: featuring complex visibility because of the window blinds
- **Staircase scene**: most of the image is lit by an area ceiling light, regions under the staircase have difficult light configuration
- **Kitchen scene**: complexity visibility

All scenes are rendered at a resolution of 800×450 and a tile size of 16×16.

4.3. Results

We render four scenes with different sample budget (collection plus rendering) and compare them against a regular path tracer with the same sample budget. The reference images are rendered with the regular path tracing integrator using around $10k$ samples per pixel. In all our guided renderings, we use $1/8$ total samples for the collection phase and the remaining $7/8$ samples for the rendering phase. For sample rate above 1024, we use two iterations for the collecting phase, each using an equal amount of samples.

Visual results are shown in Figure 4, Figure 5, Figure 6 and Figure 7. In all our figures, we use the notation of "collection" samples + "rendering" samples, e.g, $64+448$ means 64 samples are collection samples and the remaining 448 are rendering samples. Convergence rates (MSE-spp) of all tests are plotted in Figure 8. Zero radiance paths statistics are given in Table 1.

Observations Comparing our guided results with non-guided results at the same budget, we have the following observations:

- In all our tests, our guided results show better overall convergence in terms of MSE, where in the best case we managed to reduce costs by 39.5% in Figure 7.

- In regions where the first bounces make a major difference, our results show obvious improvement over non-guided results. See, e.g., red region in the *dining room* scene in Figure 5. In regions, where more than two bounces are needed to connect the light source to the sensor, the improvement is minor or none.

- In all our tests, we managed to reduce the ratio of non-contributing paths. In the best case by 12.77% in the *Ajar* scene.

- With our iterated scheme, this ratio drops further, as the sample budget increases. While for the non-guided tests, this ratio stays at the same level. This is to be expected, since samples in later iterations are guided towards lit regions.

5. Discussion

Our method reduces non-contributing paths and overall variance, showing the effectiveness of the primary sample space-path guiding. When comparing same sample rate quality, especially in the *dinning room* scene and the *kitchen* scene, our method significantly outperforms unguided results.

In all our tests with building distributions in primary sample space, results show that this approach is not only viable, but also efficient. Within a fixed rendering budget, even $1/8$-th of the samples will result in a proper distribution that improves the result. It may not work in all cases, but the bottom line is that it does not introduce variance. One of the core advantages of our approach is that we are working completely outside the rendering kernel, all we need to do is modify samplers so that their outputs are no longer uniform.

Higher dimensions In our tests, we are only working with two bounces and use two random numbers for each bounce (hence, $n = m = 2$). The result we get in the *staircase* scene shows that for regions that can reach the light source via a two-bounce path, the improvement is evident. However, for regions that require more bounces, our method provides little help for them. These regions remain noisy, similar to the result of an un-guided path tracer.

Adaptive structures In our implementation, we use one k-d tree per tile and average information gathered in this tile. It can be helpful to make our structure adaptive. The fact that most of the obviously improved regions in our tests are flat and smooth surfaces also complies with this point. To this extent, we could define tiles according to scene geometry, e.g., the surface normal difference.

Combination with existing methods The independence of the rendering kernel and the spatial domain, makes it easy to combine our scheme with existing path-guiding methods using directional records in spatial structures. One possible combination is to work in a "relay race" mode, in which the duty of path guiding is handed over to the spatial-directional structure from our primary sample space structure after a few bounces.

6. Conclusion and Future Work

In this work, we presented a novel scheme for unbiased path guiding in primary sample space, which we showed to be practical and efficient. Even when sparing only a small portion of the total rendering budget on the collection step the results typically lead to a better outcome than using the entire budgets for rendering alone.

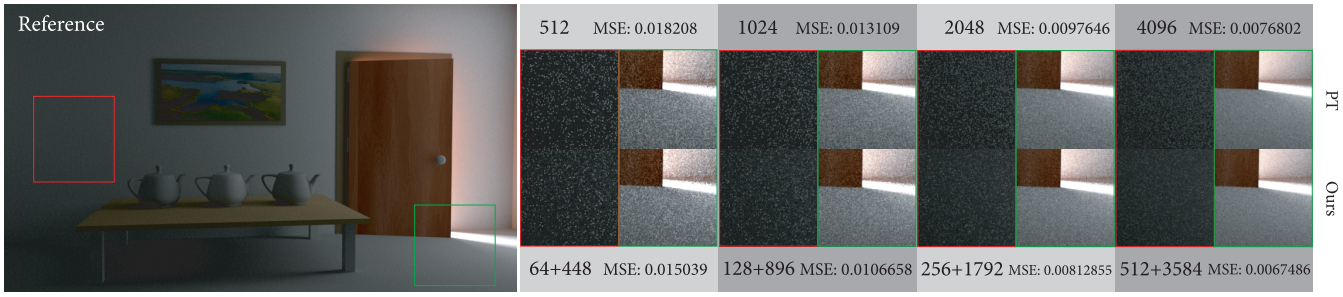

Figure 4: *The* Ajar *scene.*

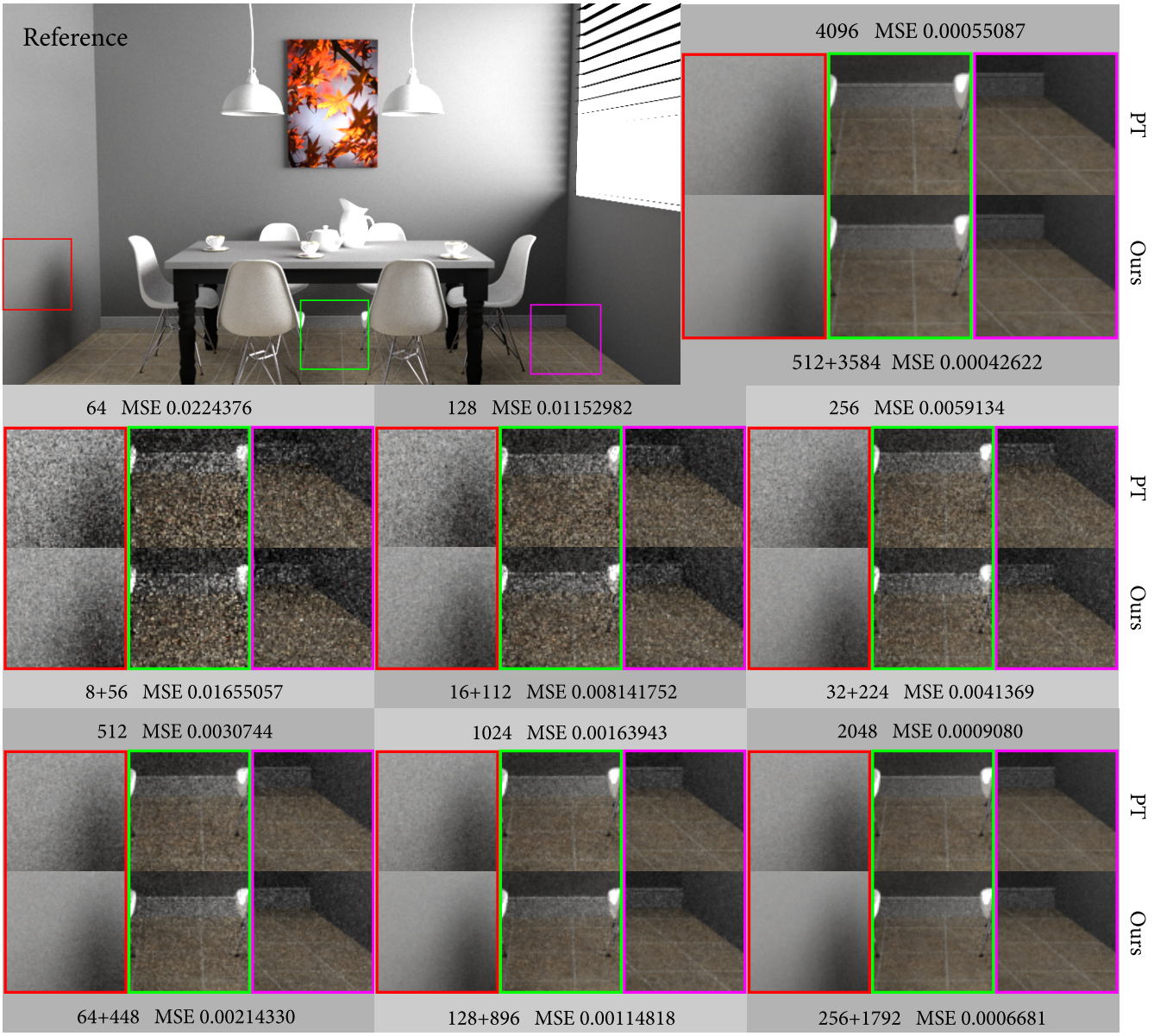

Figure 5: *The* Dining room *scene.*

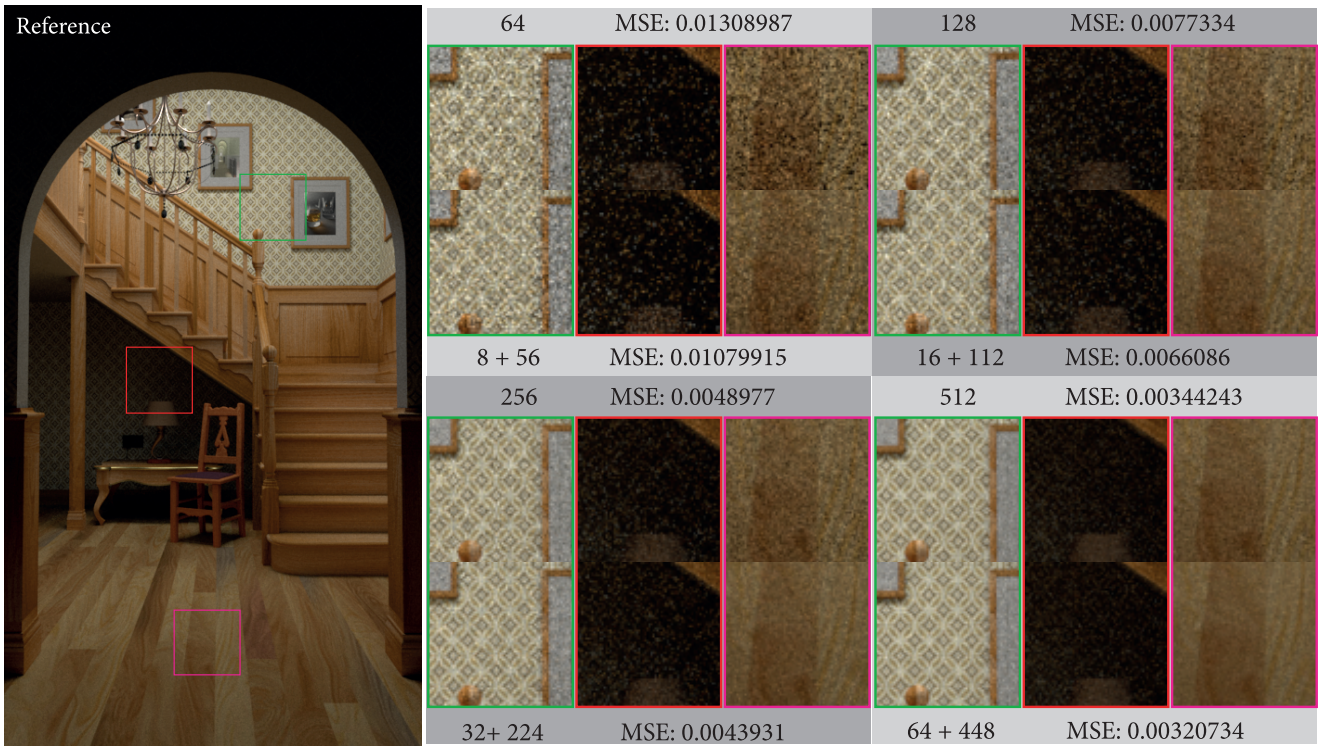

Figure 6: *The* staircase *scene. Notice the improvement in pink regions.*

Figure 7: *The* kitchen *scene.*

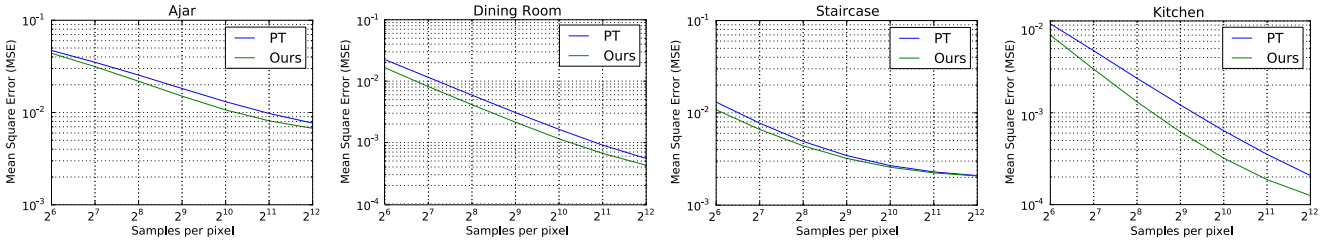

Figure 8: *Convergence plot of all our tests.*

Scene	Method	64 spp	128 spp	256 spp	512 spp	1024 spp	2048 spp	4096 spp
Ajar	PT	96.58%	96.58%	96.59%	96.59%	96.58%	96.58%	96.59%
Ajar	Ours	94.70%	93.68%	92.37%	90.72%	88.71%	86.50%	84.26%
Dining	PT	87.13%	87.13%	87.13%	87.13%	87.13%	87.14%	87.14%
Dining	Ours	87.33%	87.24%	87.08%	86.92%	86.75%	86.62%	86.53%
Kitchen	PT	97.50%	97.50%	97.50%	97.50%	97.50%	97.50%	97.50%
Kitchen	Ours	96.46%	95.52%	94.20%	92.61%	90.94%	89.40%	88.11%
Staircase	PT	66.51%	66.52%	66.52%	66.52%	66.52%	66.52%	66.52%
Staircase	Ours	65.93%	65.38%	64.61%	63.72%	62.74%	61.76%	60.76%

Table 1: *Zero radiance path ratios of all our tests. Notice in all our tests, our method manages to bring down zero radiance paths ratio. Notice also that as more and more samples are used for the collection phase, zero-radiance path rates drops accordingly. While in the non-guided tests, this ratio remains at the same level regardless of the sample rate. This demonstrates the effectiveness of our scheme.*

Our scheme of working in primary sample space is easy to implement and can be comfortably added onto any existing Monte Carlo rendering system.

For future work, we would like to extend our idea to higher dimensions and to bi-directional path tracing.

7. Acknowledgments

All the test scenes are provided by [Bit16].

References

[AK90] ARVO J., KIRK D.: Particle transport and image synthesis. *ACM SIGGRAPH Computer Graphics 24*, 4 (1990), 63–66. doi:10.1145/97879.97886. 3

[Ben75] BENTLEY J. L.: Multidimensional binary search trees used for associative searching. *Communications of the ACM 18*, 9 (1975), 509–517. doi:10.1145/361002.361007. 4

[Bit16] BITTERLI B.: Rendering resources, 2016. https://benedikt-bitterli.me/resources/. 9

[CJAMJ05] CLARBERG P., JAROSZ W., AKENINE-MÖLLER T., JENSEN H. W.: Wavelet importance sampling: efficiently evaluating products of complex functions. *ACM Transactions on Graphics (TOG) 24*, 3 (2005), 1166–1175. doi:10.1145/1186822.1073328. 2

[DBB06] DUTRÉ P., BEKAERT P., BALA K.: *Advanced global illumination*. CRC Press, 2006. 2

[Deb08] DEBEVEC P.: Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *ACM SIGGRAPH 2008 classes* (2008), ACM, p. 32. doi:10.1145/1401132.1401175. 2

[DK17] DAHM K., KELLER A.: Learning light transport the reinforced way. In *ACM SIGGRAPH 2017 Talks* (2017), ACM, p. 73. doi:10.1145/3084363.3085032. 3, 4

[HEV*16] HERHOLZ S., ELEK O., VORBA J., LENSCH H., KŘIVÁNEK J.: Product importance sampling for light transport path guiding. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 67–77. doi:10.1111/cgf.12950. 1, 3

[HKD14] HACHISUKA T., KAPLANYAN A. S., DACHSBACHER C.: Multiplexed metropolis light transport. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 100. doi:10.1145/2601097.2601138. 2

[HP02] HEY H., PURGATHOFER W.: Importance sampling with hemispherical particle footprints. In *Proceedings of the 18th spring conference on Computer graphics* (2002), ACM, pp. 107–114. 2

[HPJ12] HACHISUKA T., PANTALEONI J., JENSEN H. W.: A path space extension for robust light transport simulation. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 191. doi:10.1145/2366145.2366210. 2

[JC95] JENSEN H. W., CHRISTENSEN N. J.: Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers & Graphics 19*, 2 (1995), 215–224. doi:10.1016/0097-8493(94)00145-O. 2

[JM12] JAKOB W., MARSCHNER S.: Manifold exploration: a markov chain monte carlo technique for rendering scenes with difficult specular transport. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 58. doi:10.1145/2185520.2185554. 2

[Kaj86] KAJIYA J. T.: The rendering equation. In *ACM Siggraph Computer Graphics* (1986), vol. 20, ACM, pp. 143–150. doi:10.1145/280811.280987. 1

[KKK09] KITAOKA S., KITAMURA Y., KISHINO F.: Replica exchange light transport. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 2330–2342. doi:10.1111/j.1467-8659.2009.01540.x. 2

[KSKAC02] KELEMEN C., SZIRMAY-KALOS L., ANTAL G., CSONKA F.: A simple and robust mutation strategy for the metropolis light transport algorithm. In *Computer Graphics Forum* (2002), vol. 21, Wiley Online Library, pp. 531–540. doi:10.1111/1467-8659.t01-1-00703. 2

[LFTG97] LAFORTUNE E. P., FOO S.-C., TORRANCE K. E., GREENBERG D. P.: Non-linear approximation of reflectance functions. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 117–126. doi:10.1145/258734.258801. 2

[MGN17] MÜLLER T., GROSS M., NOVÁK J.: Practical path guiding for efficient light-transport simulation. In *Proceedings of the Eurographics Symposium on Rendering* (June 2017). doi:10.1111/cgf.13227. 3, 4

[MH97] MCCOOL M. D., HARWOOD P. K.: Probability trees. In *Graphics Interface* (1997), vol. 97, pp. 37–46. 4

[Pan17] PANTALEONI J.: Charted metropolis light transport. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 75. doi:10.1145/3072959.3073677. 4

[PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation (3rd ed.)*, 3rd ed. Morgan Kaufmann Publishers Inc., 2016. 2, 4, 5

[Shi91] SHIRLEY P. S.: *Physically based lighting calculations for computer graphics*. PhD thesis, University of Illinois at Urbana-Champaign, 1991. 2

[Vea97] VEACH E.: *Robust monte carlo methods for light transport simulation*. PhD thesis, Stanford University, 1997. 1

[VG95a] VEACH E., GUIBAS L.: Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*. Springer, 1995, pp. 145–167. doi:10.1007/978-3-642-87825-1_11. 2

[VG95b] VEACH E., GUIBAS L. J.: Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM, pp. 419–428. doi:10.1145/218380.218498. 2

[VG97] VEACH E., GUIBAS L. J.: Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 65–76. doi:10.1145/258734.258775. 1

[VKŠ*14] VORBA J., KARLÍK O., ŠIK M., RITSCHEL T., KŘIVÁNEK J.: On-line learning of parametric mixture models for light transport simulation. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 101. doi:10.1145/2601097.2601203. 1, 3

[War92] WARD G. J.: Measuring and modeling anisotropic reflection. *ACM SIGGRAPH Computer Graphics 26*, 2 (1992), 265–272. doi:10.1145/133994.134078. 2