Lab Report:
Practical Path Guiding for Efficient Light-Transport Simulation

Takkasila Saichol
Mat. Nr.: -

February 28, 2023

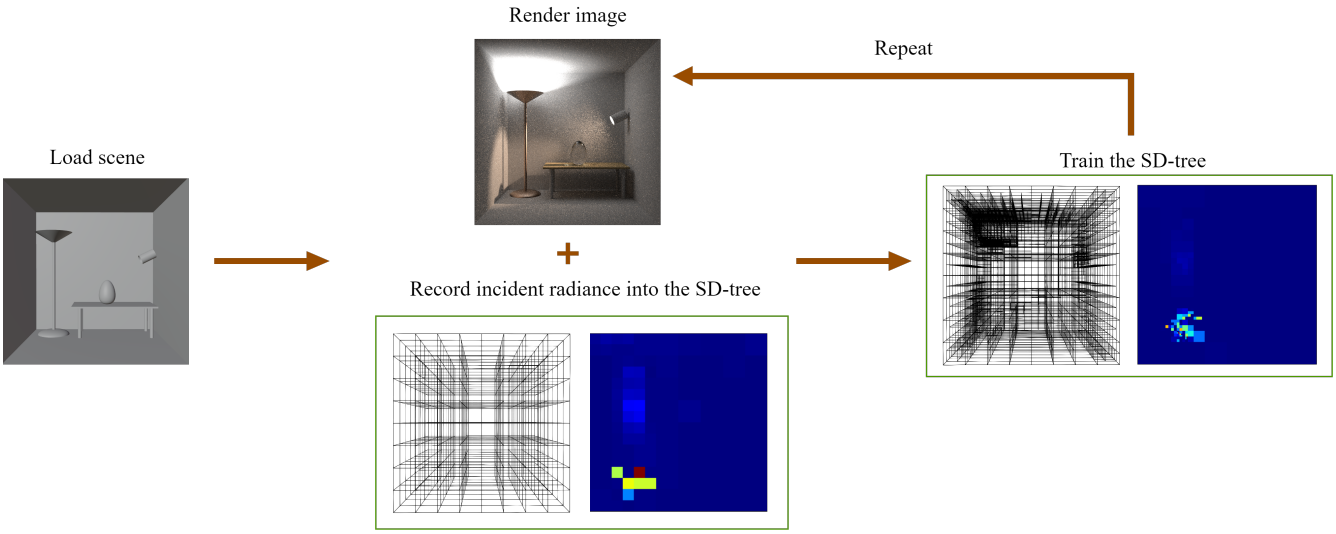


Figure 1: A diagram depicting the overall pipeline of rendering and training the spatio-directional tree (SD-tree) iteratively. An empty SD-tree is first construct to store the incident radiance from the rendering pass, then train to better approximate the scene's radiance field. After finished training, the tree is used to help guide the incident radiance along with the surface BSDF via multiple importance sampling. Repeat the process until used all the computation budget or achieved the desired result.

Abstract

In this report, we explore a robust light-path construction technique (Müller et al., 2017) for the path-tracing algorithm. The proposed technique aims toward improve solving of the rendering equation by using an approximate representation of the scene's radiance field referred to as SD-tree. The SD-tree is an adaptive spatio-directional hybrid data structure that is composed of two components: a 3D binary tree that partition the 3D spatial domain, and a 2D quadtree that partition the *2D* directional domain. The SD-tree is built by iteratively rendering and training from the rendered data in an unbiased manner. The technique does not require tuning hyperparameters, although it is allowed to set the memory footprint limit.

A budgeting scheme is also proposed to automatically split a given budget (time, or samples per pixel) between training and rendering operations in order to minimize the variance of the final image.

The resulting technique is suitable for the production environment for its ease of implementation, robustness, and an ability to integrate into other rendering technique. We implement the technique into the Mitsuba 3 framework (Jakob et al., 2022) and compare the results against the standard Monte Carlo rendering on equal budget.

1 Introduction

Efficient exploration of the space of paths that light can travel to the sensors has always been one of the most important challenge in realistic image synthesis. The vast variety of scenes found across many disciplines: movie production, gaming industry, or product design often exceed the capabilities of simple photo-realistic rendering algorithms such as path tracing, leading to long render times. Therefore, a significant

amount of research as been dedicated to creating sophisticated methods for constructing high-energy light paths (Lafortune and Willems, 1993), (Veach and Guibas, 1997) or reusing computation to reduce the inefficiency (Jensen, 2001), (Keller, 1997), (Georgiev et al., 2012), (Hachisuka et al., 2012), (Kettunen et al., 2015). While these algorithm performs well in some scenarios, they are often not very robust and can under-perform in other scenarios. Furthermore, integrating these methods into heaviliy constrained production environments can be challenging to maintain efficiency, flexibility of the implementation, workflow, and artistic control.

The goal of the proposed method (Müller et al., 2017) is to enable path-tracing algorithms to learn how to construct high-energy light paths iteratively. This approach aims to keep the algorithm simple, yet robust enough to simulate ecomplex transport phenomena such as caustics. The method is inspired by existing path-guiding algorithms that show how unidirectional path consutrction can be nearly as effective as more advanced algorithms when guided by the nearby quantity. Instead of using spatially cached histograms (Jensen, 1995), cones (Hey and Purgathofer, 2002), or gaussian mixtures (Vorba, Karlík, et al., 2014)—the proposed method stores a discrete approximation of the scene's 5D light field using an adaptive spatio-directional tree (SD-tree). The SD-tree consists of an upper part, which is a binary tree that partitions the 3D spatial domain of the light field, and a lower part, which is a quadtree that partitions the 2D directional domain.

The SD-trees are designed to be able to adapt well to both low- and high frequency details without requiring excessive memory and compared favorably to the previously mentioned path-guiding approaches. They also allow for fast importance sampling and can be consturcted quickly, enabling fast adaptation to new information. Furthermore, the authors also present an improved iterative training method and offer systematic approach to finding the optimal balance between training and rendering when given a fixed time or sample budget. It also does not require tuning of hyper-parameters, although the user can choose to limit memory footprint by limiting size of the SD-tree. The resulting method is straightforward to implement and support progressive rendering—make it suitable for production environment.

2 Problem Statement and Related Work

From the rendering equation (Kajiya, 1986), the amount of radiance $L_o(\boldsymbol{x}, \vec{\omega}_o)$ leaving point $\boldsymbol{x}$ in direction $\vec{\omega}_o$ is

$$L_o(\boldsymbol{x}, \vec{\omega}_o) = L_e(\boldsymbol{x}, \vec{\omega}_o) + \int_\Omega L(\boldsymbol{x}, \vec{\omega}) f_s(\boldsymbol{x}, \vec{\omega}_o, \vec{\omega}) \cos\theta d\vec{\omega} \tag{1}$$

where $L_e(\boldsymbol{x}, \vec{\omega}_o)$ is radiance emitted from $\boldsymbol{x}$ in $\vec{\omega}_0$, $L(\boldsymbol{x}, \vec{\omega})$ is the incident radiance at $\boldsymbol{x}$ from $\vec{\omega}$, and f_s is the bidirectional scattering distribution function. Using the Monte Carlo method, the reflection integral L_r can be estimated numerically

$$\langle L_r \rangle = \frac{1}{N} \sum_{j=1}^{N} \frac{L(\boldsymbol{x}, \vec{\omega}_j) f_s(\boldsymbol{x}, \vec{\omega}_o, \vec{\omega}_j) \cos\theta_j}{p(\vec{\omega}_j | \boldsymbol{x}, \vec{\omega}_o)} \tag{2}$$

The variance of the estimator $V[\langle L_r \rangle]$ is proportional to $\frac{1}{N}$ and can be reduced by sampling $\vec{\omega}_j$ from a probability density function (PDF) $p(\vec{\omega}_j | \boldsymbol{x}, \vec{\omega}_o)$ that resembles the shape of the numerator. While both the BSDF and the cosine term can be approximated quite well in general, approximating the incident radiance field to use it from importance sampling has always been a challenging part of MC rendering method.

The concept of guiding camera paths by utilizing information from previously traced paths was initially introduced by Jensen, 1995 and Lafortune and Willems, 1995. Jensen, 1995 populates the scene with sampling points, each having a hemispherical histogram from a set of photons that were traced prio. The histograms are then used to sample directions while constructing camera paths. On the other hand Lafortune and Willems, 1995 suggest rasterizing incident radiance in a 5D tree for later use as a control variate and for importance sampling. Steinhurst and Lastra, 2006 improve Jensen's method by introducing product importance sampling with a discretized BSDF, and Budge et al., 2008 apply a specialized form of Jensen's technique to improve caustics sampling. Hey and Purgathofer, 2002 recognize that regular histograms are not suited for this type of density estimation and instead proposee to average cones of adaptive width centered around the photon's incident direction. Afterward, Vorba, Karlík, et al., 2014 proposed to perform density estimation using a parametric gaussian-mixture model and an iterative on-line reinforcement learning algorithm, which is a basis for improvement of this paper (Müller et al., 2017). There are more follow-up work of Vorba, Karlík, et al., 2014 by adding approximate product importance sampling with the BSDF (Herholz et al., 2016), and adjoint-based Russian roulette (Vorba and Křivánek, 2016) Dahm and Keller, 2017 combine rendering and training into the same algorithm that requires a careful sample weighting to diminish in the limit.

The proposed method adopt both adjoint-based Russian roulette and progressive reinforcement learning, while also combine the rendering and learning algorithm into one. Furthermore, they also split the learning procedure into distinct passes—each pass guided by the previous pass,

while also remaining unbiased.

The key difference between the proposed method and the previous is the data structure that is used for representing the incident radiance field. The author use a hybrid data structure consisting of a binary tree for adaptive spatial partitioning, and a directional quadtree for adaptive binning in the angular domain. This type of hybrid data structure has been succesfully used in geometry processing and rendering, such as volume-surface trees (Boubekeur et al., 2006), or rasterized BVHs (Novák and Dachsbacher, 2012). The binary spatial tree is prefered due to its smaller branching factor, which is better suited for their progressive reinforcement learning approach. Additionally with a quadtree for the directional domain to avoid the downsides of regular binning (Jensen, 1995) and to enable straightforward construction and robust performanc, which is harder to achieve with kd-trees (Gassenbauer et al., 2009) and gaussian-mixture models (Vorba, Karlík, et al., 2014), (Herholz et al., 2016).

3 Path Guiding with SD-Tree

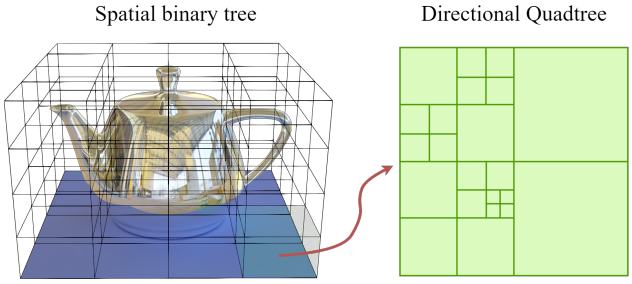


Figure 2: SD-tree diagram. On the left is the spatial bianry tree that subdivides the space by alternately splitting x, y, and z dimension in half in each iteration. Each of the leaf node then contains a quadtree on the right side, which is an approximation of the radiance field in spherical domain (θ, Φ)

In an order to guide the light path, a discrete approximation of the incident radiance field, denoted as $\hat{L}$, is constructed by reinforcement learning. The field $\hat{L}$ is represented by an SD-tree and then get iteratively improve with a geometrically increasing compute budget—basically doubling every each iteration. The strategy is to maintain *two* SD-trees: one for *guiding* the incident radiance, and another for *collecting* the incident radiance from MC estimation. For each iteration k starting from 0, we do an importance sampling of the incident radiance using the previous iteration radiance field $\hat{L}^{k-1}$, collect the MC estimates of $L(\boldsymbol{x}, \vec{\omega})$ and scatter into a skeletal $\hat{L}^k$. The skeletal-$\hat{L}$, -SD-tree means it has the approximate structure of the incident radiance field, but does not have the amount of radiance i.e. a tree that have branches but every leaf node are empty. After iteration k is finished, the skeletal $\hat{L}^k$ is trained with the data that it has received in that iteration, creating a complete $\hat{L}^k$. We then generate a skeletal $\hat{L}^{k+1}$ from $\hat{L}^k$ and continue to repeat the same process in the next iteration. We will talk about the iteration process more in detail in Section 4

3.1 Collecting Estimates of L

During rendering, when a complete light path is form, we compute the incident radiance at every path verticies $L(\boldsymbol{x}_v, \vec{\omega}_v)$. Each incident radiance is then scatter into the skeletal SD-tree by first performing a *spatial* search from the root node of the binary tree to the leaf node that contains the position $\boldsymbol{x}_v$. Each binary-tree leaf node will then have a reference to *a unique* directional quadtree. We continue by performing a *directional* search from the root node of the directional quadtree to the leaf node that contains the direction $\vec{\omega}_v$, while also store the estimate $L(\boldsymbol{x}_v, \vec{\omega}_v)$ along every node that we pass through. The quadtree is a 2D parameterization of the full sphere of direction. It uses the *world-space* cylindrical coordinates to preserve area ratios.

After depositing all radiance estimates in the current iteration, the nodes of the quadtrees are able to estimate the total incident radiance that arrives through the spherical region corresponding to their respective quad. It's important to note that the directional distribution is averaged over all spatial positions that were sampled within the binary-tree leaf associated with the quadtree. For an example, unlike Photon Mapping (Jensen, 1995) where it stores photons individually, we merge average every photon within the same spatial binary tree leaf node. The gathered information is used to guide the path construction in the next iteration but also to adapt the structure of the SD-tree for collecting future estimates. The adaptation of the spatial and directional components of the SD-tree is described in the next two sections 3.2, 3.3.

3.2 Adaptive Spatial Binary Tree

The depth and structure of the binary tree ultimately determines how refined and adaptive the approximation of the radiance field is. To goal is to refine the spatial binary tree such that the number of path verticies in each leaf node is less than a threshold $c \cdot \sqrt{2^k}$. If the number of path verticies is more than the threshold then we alternately split the leaf node along x, y, or z axis into two new leaf node, where each have half of the path verticies of the parent and each hold a unique, exact copy of the parent's quad tree. The

threshold $c \cdot \sqrt{2^k}$ where 2^k is proportional to the amount of traced paths in the k-th iteration and c is derived from the desired resolution of the quadtree. After subdivision, all leafs should contain roughly the same amount of path verticies as the threshold. Therefore, the total amount of leaf nodes is proportional to $\frac{2^k}{c \cdot \sqrt{2^k}} = \frac{\sqrt{2^k}}{c}$. This ensures that the total number of leaf nodes and the amount of samples in eac leaf both grow at the same rate $\sqrt{2^k}$ for every iterations. The constant c trades off convergence of the directional quadtress with spatial resolution of the binary tree. And by learning iteratively, the distribution guide paths into regions with high contribution, which in turn increase the resolution of the learned distribution even more—the area with high contribution gets refine much more aggresively than the area with low contribution.

The constant c is derived from the desired number of samples s that each quadtree leaf node should receive. Every quadtree leaf node have roughly the same amount of irradiance and thus sampled with similar probability during trianing. The average samples a quadtree leaf receives is then $s = \frac{S}{N_l}$ where S is the total amount of samples drawn from a quadtree, and N_l is the amount of leaf nodes. From their experiment, they have found that the quadtree have rougly 300 nodes on average, and using $s = 40$ samples per quadtree results in a resonably converged quadtree. And because we want to subdivide the tree into these thresholds since after the first iteration ($k = 0$), c can be derived as

$$c = \frac{s \cdot N_l}{\sqrt{2^k}} = \frac{40 \cdot 300}{1} = 12000 \tag{3}$$

However, this constant c is not fixed and can be change depending on the nature of the scene. The author has also suggested that $c = 5000$ might even be sufficient enough for simple scene.

Algorithm 1 Refine the binary tree

```
1:  procedure RefineBinaryTree(k)
2:      for all leaf ∈ leafNodes do
3:          if NumSamples(leaf)> c · √(2^k) then
4:              SubDivide(leaf)
5:          end if
6:      end for
7:  end procedure

8:  procedure Subdivide(leaf)
9:      numSamples ← NumSamples(Leaf)
10:     children ← SplitAlternatingAxis(leaf)
11:     SetNumSamples(children, numSamples/2)
12: end procedure
```

3.3 Adaptive Directional Quadtree

After each iteration, we reconstruct every quadtrees to better reflect the distribution of the radiance. The structure of the quadtree is determined by the directional distribution of the flux that was collected in the previous iteration. The goal is to modify the quadtree such that each leaf node contains rougly 1% of flux collected in the old quadtree. There are two operations, depending on the amount of flux Φ_n in the leaf node. If the amount of flux Φ_n is more than 1% of the total flux of that quad tree Φ; i.e. $\frac{\Phi_n}{\Phi} > 0.01$ then we split the node, resulting in 4 new nodes where each have $\frac{1}{4}\Phi_n$. Or if the amount of combined flux of it's children node $\Phi_{\text{child}} = \sum_{i=1}^{4} \Phi_{\text{child}_i}$ is less than 1%, then we merge the children node into parent, resulting in a single node that has roughly 1% of the total flux. Spherical regions with high flux are thus represented with higher resolution. This proposed subdivision scheme yields a rougly equi-energy partitioning of the directional domain.

By rebuilding the quadtrees after each iteration, the SD-tree can adapt to new information about the radiance field and utilize memory more efficiently. The threshold $\rho = 0.01$ determines the amount of memory used, as the number of nodes in the quadtree is inversely proportional to ρ.

Algorithm 2 Refine the a quadtree

```
1:  procedure RefineQuadtree(quadtree)
2:      Φ ← Flux(quadtree)
3:      for all node ∈ quadtree do
4:          if Flux(node)/Φ ≤ ρ then
5:              PruneChildren(node)
6:          else if IsLeaf(node) then
7:              Subdivide(node)
8:          end if
9:      end for
10: end procedure

11: procedure Subdivide(leaf)
12:     Φ_n ← Flux(leaf)
13:     children ← SplitQuadTree(leaf)
14:     SetFlux(children, Φ_n/4)
15: end procedure
```

3.4 Sampling the SD-tree

In each iteration k, we sample the incident radiance direction $\vec{\omega}$ at the position $\boldsymbol{x}$ and the probability of that direction $p(\vec{\omega}|\boldsymbol{x})$ from the previous iteration SD-tree $\hat{L}^{k-1}$. The sampling method is a hierarchical sample warping, described by McCool and Harwood, 1997. First, we traverse through

the spatial binary tree to find the corresponding leaf node that contains the position x. From that spatial binary tree leaf node we can get its corresponding directional quadtree. We then sample a direction ω from the corresponding quadtree by traversing through the quadtree. If the current quadtree node is a leaf node then we sample a uniform, random position withing it's bounding box. If not, we sample a child node from it's four children based on the amount of flux, then continue traversing through that child node. After receiving the sampled direction ω, we can calculate the probability $p(\vec{\omega}|\boldsymbol{x})$ by, again traversing the quadtree. If the current quadtree node is a leaf node then we return a probability of uniform sampling a random direction on a full sphere $\frac{1}{4\pi}$. Otherwise, we calculate the probability based on the amount of flux of that node as

$$p(\vec{\omega}|\boldsymbol{x}) = \frac{\frac{\Phi_{\text{child}}}{A_{\text{child}}}}{\frac{\Phi_{\text{parent}}}{A_{\text{parent}}}} = \frac{\frac{\Phi_{\text{child}}}{A_{\text{child}}}}{\frac{\Phi_{\text{parent}}}{4 \cdot A_{\text{child}}}} = \frac{4 \cdot \Phi_{\text{child}}}{\Phi_{\text{parent}}} \tag{4}$$

where Φ_{parent} is a flux of a parent node, Φ_{child} is a flux of the parent's child node, A_{parent} is the surface area of the parent node, and A_{child} is the surface area of the child node.

Algorithm 3 Sample from the SD-tree

```
procedure SampleSDTree(position)
    binaryTreeLeafNode ← FindBinaryTreeLeafNode(position)
    quadTree ← GetQuadTree(binaryTreeLeafNode)
    direction ← SampleQuadtree(quadTree)
    pdf ← PdfQuadtree(quadTree)
    return direction, pdf
end procedure

procedure SampleQuadtree(node)
    if IsLeaf(node) then
        return UniformRandomPositionIn(node)
    else
        child ← SampleChildByEnergy(child)
        return SampleQuadtree(child)
    end if
end procedure

procedure PdfQuadtree(node)
    if IsLeaf(node) then
        return 1/(4π)
    else
        child ← GetChild(ω⃗)
        α ← 4 · Flux(child)/Flux(node)
        return α· PdfQuadtree(child)
    end if
end procedure
```

4 Unbiased Iterative Learning and Rendering

The proposed method used a similar learning scheme as Vorba, Karlík, et al., 2014 by training a sequence $\hat{L}^1, \hat{L}^2, ..., \hat{L}^M$ where $\hat{L}^1$ is estimated with just BSDF sampling (because there is no $\hat{L}^0$ to sample from), and then for all $k > 1$, $\hat{L}^k$ is estimated by combining samples from both the BSDF and $\hat{L}^{k-1}$ via multiple importance sampling. Sampling from the previously learned distribution $\hat{L}^{k-1}$ to estimate $\hat{L}^k$ often drastically accelerates the convergence compare to a naïve Monte Carlo estimation.

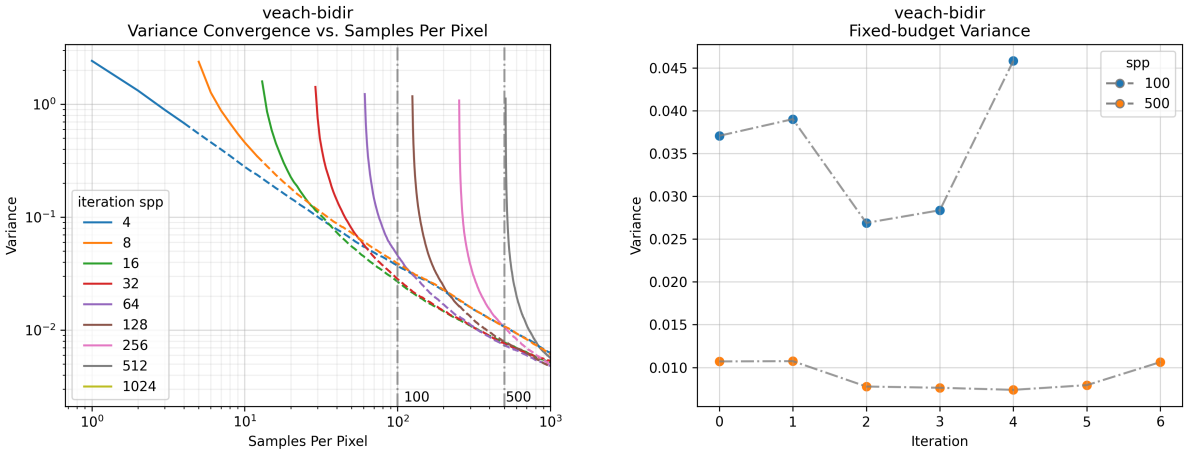
Becuase the learned distribution can only approximate the incident radiance field, if we use closely the same amount in every iteration then the majority of samples will be used toward learning and only a small amount of samples would contribute directly toward the image. The author thus proposed to double the amount of samples every iteration starting from 4 samples per pixel—precisely $\text{SPP}_k = 2^{k+2}$. Learning $\hat{L}^k$ then takes approximately twice as long as learning $\hat{L}^{k-1}$, and also has roughly half the variance, or in the worst case that the learning did not improve the convergence, only half of the samples is wasted and we stop the learning process (Section 4.1). Another important property of doubling the amount of samples is that it ensures the amount of samples in each spatial binary tree node to be the same when it gets subdivided.

Since we are training and rendering in an iterative manner, it is also possible to synthesize an image using the path sampled from that iteration. As long as we do not fuse path samples across iterations, the image is unbiased and we can progressive display feedback every iteration for visual feedback.

4.1 Balancing Learning and Rendering

Another major contribution proposed in the paper (Müller et al., 2017) is an automatic budgeting system that balance between learning and rendering in order to minimize the final image variance. In iteration k, we define the budget to unit variance $\tau_k = V_k \cdot B_k$, where V_k is the variance of image I_k computed by paths traced in the iteration k, and B_k is the budget spent on constructing these paths. The variance V_k is computed as a mean variance of pixels in image I_k. If we were to continue using $\hat{L}^k$ to guide paths until reaching a given budget B, which could be either time of number of samples, we can then estimate the variance of the final image as

$$\hat{V}_k = \frac{\tau_k}{\hat{B}_k} \tag{5}$$



(a) Convergence of variance over samples of each iteration

(b) Variance from a given fixed-budget of each iteration

Figure 3: Left: variance as a function of samples plotted for each iterationi; the legend shows the number of samples invested into training the SD-tree. The dashed extension lines indicate convergence if we continued rendering using the current learnedd distribution. Right: intersections of the convergence curves with the dash-dotted lines, representing two distinct sample budgets, 100 and 500 spp. For each sample budget, the intersections should form a convex sequence when plotted according to the total number of samples—essentially depicting the optimal iteration $\hat{k}$ where $\hat{V}_{k+1} > \hat{V}_k$ for the first time.

where $\hat{B}_k$ is the remaining budget from the start of the k-th iteration before rendering the image I_k: $\hat{B}_k = B - \sum_{i=1}^{k-1} B_i$

The goal is to find $\hat{k}$ such that the final image variance is optimized: $\hat{k} = \text{argmin}_k \hat{V}_k$. The author then proposed to assume that the benefit of training (τ) is monotonically decreasing every iteration and is convex, thus $\hat{V}_k$ is also convex. Which means we can find $\hat{k}$ for the smallest k which $\hat{V}_{k+1} < \hat{V}_k$ holds. Once we reach the iteration $\hat{k}$, the training stops and we use the rest of the budget toward rendering the final image. Since we need to evaluate $\hat{V}_{k+1}$, we need to perform one more iteration than what would be optimal, however, the benefit of the additional variance reduction greatly outweight the extra budget spent.

The same approach can be applied when aiming for a target variance $\bar{V}$ instead of a given budget B. In this case, we can estimate the rendering budget $\bar{B}_k$ required to reach the target variance $\bar{V}$ via $\bar{B}_k = \frac{\tau_k}{\bar{V}}$. The training is then stops when $\tilde{B}_k > \tilde{B}_{k-1}$, where $\tilde{B}_k = \bar{B}_k + \sum_{i=1}^{k-1} B_i$. Then we follow the same argument for $\tilde{B}_k$ is convex becacuse B is monotonically increasing, thus we can the find $\hat{k} = \text{argmin}_k \tilde{B}_k$.

5 Evaluation

We will now evaluate the proposed algorithm by comparing to the standard path tracing method with next event estimation. Both of the algorithm has been implement using the Mitsuba 3 framework (Jakob et al., 2022), which we will discuss about the implementation details in Section 5.3. The specific details of hardwares using to evaluate are: CPU: 11th Gen Intel(R) Core(TM) i9-11900H @ 2.50GHz, RAM: 32 GB, GPU: NVIDIA(R) GeForce(R) RTX 3060 Laptop GPU, VRAM: 6 GB. For evaluation, we use Cornell-Box scene for a standard evaluation. Torus scene for long chain specular iteraction evaluation. And complex scenes Veach-Bidir, Veach-Ajar, and Kitchen for general case evaluation.

5.1 Qualitative Evaluation

In Figure 4. we shows comparisions between the normal path tracing method with next event estimation (PTw/NEE) and the proposed path guiding method (PG) with varying SPP as a budget. The budget SPP has been choosed such that it alings with the training iteration criteria of the path guiding algorithm. Since $\text{SPP}_k = 2^{k+2}$, total SPP to reach and complete iteration k is $\text{BudgetSPP}_k = \sum_{i=0}^{k} \text{SPP}_i$, i.e. 4, 12, 28, 60, ... From the figure, there are several things we can observe.

First, in the Cornell-Box scene, we can see that the results from both method are identical. This is because doesn't have any specular surface. Thus, sampling $\vec{\omega}$ from a PDF, which in this case is the BRDF, will have the exact shape of the numerator and lead to zero variance $V[\langle L_r \rangle] = 0$ (Section 2.).

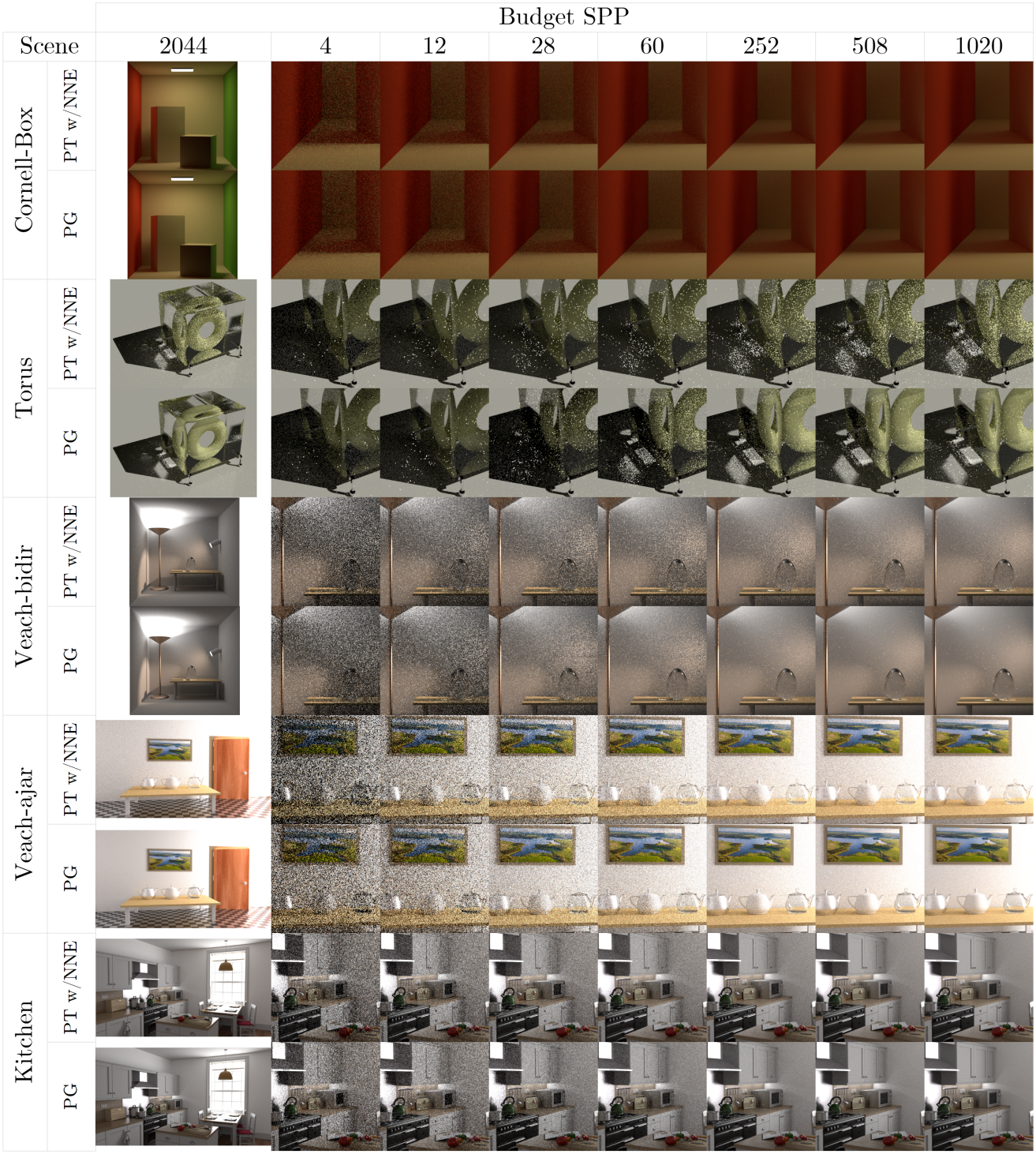


Figure 4: Rendering result comparision between normal Path Tracing with Next Event Estimation (PT w/NEE) and the proposed Path Guiding (PG) with varying SPP as a budget. After training for 2 to 3 iterations (column 28, 60 SPP respectively), the path guiding method outperform the normal path-tracing with NEE—especially in the torus scene which has long chains of specular iteraction and Specular-Diffuse-Specular (SDS). Further details in Section 5.1.

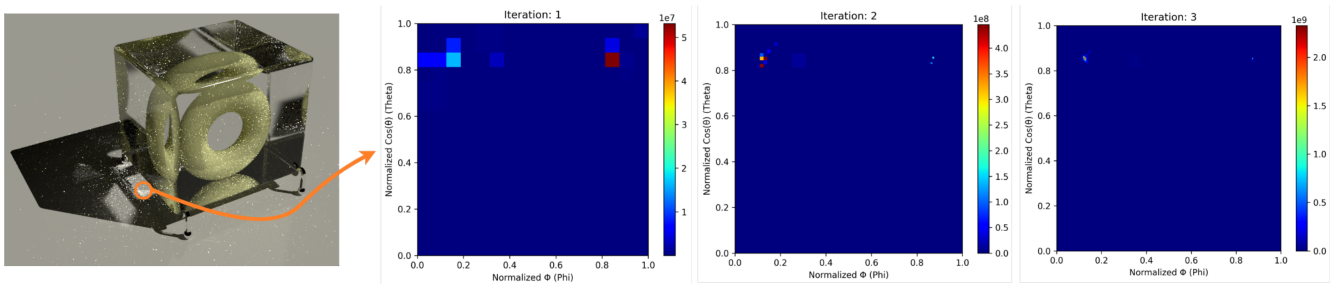


Figure 5: Torus's Quadtree. The figure displays improvement of the learned scene's radiance field by showing the evolution of a directional quadtree at the bright shadow spot highlighted by the orange circle across training iteration. After the second training iteration ($k = 1$), it is able to rougly approximate the radiance field of that area. And after two more iterations, it converged to almost a point (in the top left in the graph) indicating the direction of highest radiance.

Second, in the 12 SPP budget column across every scene, we can see that the results from path guiding method does not shows an improvement from the PTw/NEE, and in face will be a little bit worse. This is because in the second iteration of path guiding, 4 SPP are spent training the radiance field, which is not yet a good approximation of the scene's radiance field, and 8 SPP are then spent on rendering in that iteration; contrary to PTw/NEE where it used all 12 SPP on rendering (Figure 6). However, once the SD-tree has been trained for a 2 to 3 iterations, it is able to learned the distribution and can closely approximate scene's radiance field (Figure 5). Making sampling from the SD-tree leads to a faster convergence than the PTw/NEE. For example, in the Torus scene there is a big difference between budget SPP 28 and 60. The SD-tree is able to learn this complex long-chain scattering path, which leads to a much faster convergence than PTw/NEE.

5.2 Quantitative Evaluation

The variance is computed as a mean variance of pixels in the image with respect to the ground truth. We then compare variance between the two methods: PT w/ NEE, and PG on the same scenes as in Section 5.1. From the Figure 6, in the top row we compare variance between the two methods against samples per pixel. In this case we can see that the path guiding algorithm produces lower variance than the path tracing w/ NEE in every test scenes, or atleast equally in the kitchen scene. This means that the path guiding algorithm is always more efficient with a given budget. However, in the bottom row of the figure where we plotted against time, we see a different story. In every test scenes, the path guiding algorithm produces higher variance at almost any given time compare to the path tracing w/ NEE algorithm. Or atmost equally, in the torus scene. This could means that the path guiding algorithm does not execute fast enough for the benefit of guiding to outweight the extra computation time. Since the paper have stated that they found the path guiding to out perform other algorithms in a given time. This is most likely mean that our implementation is not efficient enough.

5.3 Implementation Evaluation

The implementation of path tracing with next event estimation and the proposed path guiding algorithm has been implement using the Mitsuba 3 rendering framework (Jakob et al., 2022). Since the major part of the path guiding algorithm is the SD-tree, it is crucial that the tree data structure has to be implemented in a way that supports the parallel-computing nature of rendering algorithms to gain the best performance. In order to do so, we choosed to implement the tree datastructure in a Data-Oriented Design (DOD), where a node is represented as an index number, and the relating values can be accessed by accessing through the shared array of the specified index. For example, for the directional binary tree there are `bbox` which represents the bounding box volume of the node, `vert_count` counts the number of light path verticies inside the node, `quadtree_index` stores the unique index of the quadtree node, and `child_left_index` and `child_right_index` for storing leaf and right children's node index respectively. The same goes for the directional quadtree datastructure.

In the rendering process, we render every pixels in a normal parallel fashion while storing incident radiance at each light light path verticies at the same time. After the rendering pass is finished, we scatter every recorded radiance simultaneously into the SD-tree by traversing to the corresponding quadtree leaf node and increase radiance of passthrough nodes along the way. Then we refine the tree in an iterative manner by refining every nodes that are

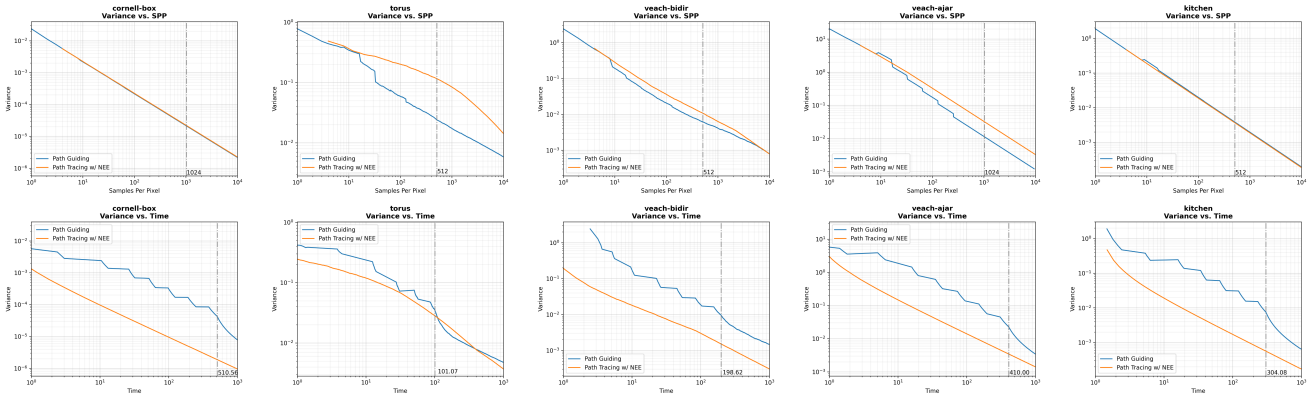


Figure 6: Variance comparision between path tracing with next event estimation and path guiding algorithm. Top row: variance plotted as a function of SPP. Bottom row: variance plotted as a function of time. The comparision against SPP demonstrated that the path guiding algorithm always produce lower variance than the normal path tracing with NEE in every tested scenes. However, in the comparision against time, the path guiding algorithm produces higher variance in almost scenarios. This could be a result of poor implementation on our side.

on the same depth (starting from root), which can then get merged or splitted until every node statisfy the conditions (Algorithm 1, 2).

6 Discussion and Conclusion

The authors have presented a path guiding algorithm which has been applied to a unidirectional path tracer. We have been able to replicate the results and the effects presented in the original paper. The iterative learning process is able to train the SD-tree to closely approximate the scene's radiance field in just a small number of iterations. The implementation of the algorithm is straightforward and robust enough to suit the production environment. The benefit of path guiding can also be complementary and is able to apply with more sophisticated rendering methods for an even better result.

References

Boubekeur, T., W. Heidrich, X. Granier, and C. Schlick (2006). "Volume-surface trees". In: *Computer Graphics Forum*. Vol. 25. 3. Wiley Online Library, pp. 399–406.

Budge, B. C., J. C. Anderson, and K. I. Joy (2008). "Caustic forecasting: Unbiased estimation of caustic lighting for global illumination". In: *Computer Graphics Forum*. Vol. 27. 7. Wiley Online Library, pp. 1963–1970.

Dahm, K. and A. Keller (2017). "Learning light transport the reinforced way". In: *ACM SIGGRAPH 2017 Talks*, pp. 1–2.

Gassenbauer, V., J. Křivánek, and K. Bouatouch (2009). "Spatial directional radiance caching". In: *Computer Graphics Forum*. Vol. 28. 4. Wiley Online Library, pp. 1189–1198.

Georgiev, I., J. Krivánek, T. Davidovic, and P. Slusallek (2012). "Light transport simulation with vertex connection and merging." In: *ACM Trans. Graph.* 31.6, pp. 192–1.

Hachisuka, T., J. Pantaleoni, and H. W. Jensen (2012). "A path space extension for robust light transport simulation". In: *ACM Transactions on Graphics (TOG)* 31.6, pp. 1–10.

Herholz, S., O. Elek, J. Vorba, H. Lensch, and J. Křivánek (2016). "Product importance sampling for light transport path guiding". In: *Computer Graphics Forum*. Vol. 35. 4. Wiley Online Library, pp. 67–77.

Hey, H. and W. Purgathofer (2002). "Importance sampling with hemispherical particle footprints". In: *Proceedings of the 18th spring conference on Computer graphics*, pp. 107–114.

Jakob, W., S. Speierer, N. Roussel, M. Nimier-David, D. Vicini, T. Zeltner, B. Nicolet, M. Crespo, V. Leroy, and Z. Zhang (2022). *Mitsuba 3 renderer*. Version 3.0.1. https://mitsuba-renderer.org.

Jensen, H. W. (1995). "Importance driven path tracing using the photon map". In: *Rendering Techniques' 95: Proceedings of the Eurographics Workshop in Dublin, Ireland, June 12–14, 1995 6*. Springer, pp. 326–335.

– (2001). *Realistic image synthesis using photon mapping*. Vol. 364. Ak Peters Natick.

Kajiya, J. T. (1986). "The rendering equation". In: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pp. 143–150.

Keller, A. (1997). "Instant Radiosity". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. USA: ACM Press/Addison-Wesley Publishing Co., pp. 49–56. URL: https://doi.org/10.1145/258734.258769.

Kettunen, M., M. Manzi, M. Aittala, J. Lehtinen, F. Durand, and M. Zwicker (2015). "Gradient-domain path tracing". In: *ACM Transactions on Graphics (TOG)* 34.4, pp. 1–13.

Lafortune, E. P. and Y. D. Willems (1995). "A 5D tree to reduce the variance of Monte Carlo ray tracing". In: *Rendering Techniques' 95: Proceedings of the Eurographics Workshop in Dublin, Ireland, June 12–14, 1995 6*. Springer, pp. 11–20.

– (Dec. 1993). "Bi-directional path tracing". In: *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*. Alvor, Portugal, pp. 145–153.

McCool, M. D. and P. K. Harwood (1997). "Probability trees". In: *Graphics Interface*. Vol. 97, pp. 37–46.

Müller, T., M. Gross, and J. Novák (June 2017). "Practical Path Guiding for Efficient Light-Transport Simulation". In: *Computer Graphics Forum (Proceedings of EGSR)* 36.4, pp. 91–100.

Novák, J. and C. Dachsbacher (2012). "Rasterized bounding volume hierarchies". In: *Computer Graphics Forum*. Vol. 31. 2pt2. Wiley Online Library, pp. 403–412.

Steinhurst, J. and A. Lastra (2006). "Global importance sampling of glossy surfaces using the photon map". In: *2006 IEEE Symposium on Interactive Ray Tracing*. IEEE, pp. 133–138.

Veach, E. and L. J. Guibas (1997). "Metropolis light transport". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 65–76.

Vorba, J., O. Karlík, M. Šik, T. Ritschel, and J. Křivánek (Aug. 2014). "On-line Learning of Parametric Mixture Models for Light Transport Simulation". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)* 33.4.

Vorba, J. and J. Křivánek (2016). "Adjoint-driven Russian roulette and splitting in light transport simulation". In: *ACM Transactions on Graphics (TOG)* 35.4, pp. 1–11.