Eurographics Symposium on Rendering 2017
P. Sander and M. Zwicker
(Guest Editors)

Volume 36 (2017), Number 4

Practical Path Guiding for Efficient Light-Transport Simulation

Thomas Müller[1,2] Markus Gross[1,2] Jan Novák[2]

[1]ETH Zürich
[2]Disney Research

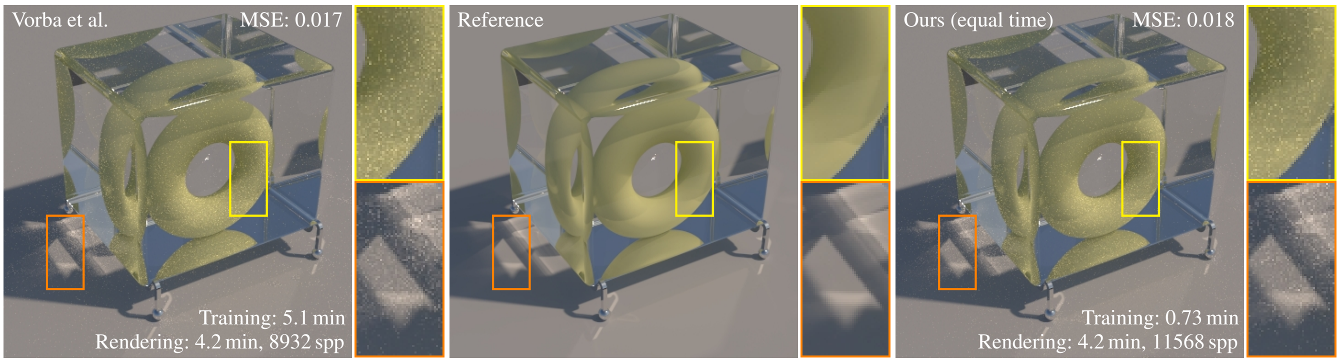

Figure 1: *Our method allows efficient guiding of path-tracing algorithms as demonstrated in the* TORUS *scene. We compare equal-time (4.2 min) renderings of our method (right) to the current state-of-the-art [VKv*14, VK16] (left). Our algorithm automatically estimates how much training time is optimal, displays a rendering preview during training, and requires no parameter tuning. Despite being fully unidirectional, our method achieves similar MSE values compared to Vorba et al.'s method, which trains bidirectionally.*

Abstract

We present a robust, unbiased technique for intelligent light-path construction in path-tracing algorithms. Inspired by existing path-guiding *algorithms, our method learns an approximate representation of the scene's spatio-directional radiance field in an unbiased and iterative manner. To that end, we propose an adaptive spatio-directional hybrid data structure, referred to as* SD-tree, *for storing and sampling incident radiance. The SD-tree consists of an upper part—a binary tree that partitions the 3D spatial domain of the light field—and a lower part—a quadtree that partitions the 2D directional domain. We further present a principled way to automatically budget training and rendering computations to minimize the variance of the final image. Our method does not require tuning hyperparameters, although we allow limiting the memory footprint of the SD-tree. The aforementioned properties, its ease of implementation, and its stable performance make our method compatible with production environments. We demonstrate the merits of our method on scenes with difficult visibility, detailed geometry, and complex specular-glossy light transport, achieving better performance than previous state-of-the-art algorithms.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—; I.3.3 [Computer Graphics]: Picture/Image Generation—

1. Introduction

One of the remaining challenges of realistic image synthesis is an efficient exploration of the space of paths that light can take to reach the sensor. The types of scenes that we nowadays encounter in product visualization, architectural design, and movie production often test the limits of simple algorithms, such as path tracing, leading to excessive render times. A large body of research has thus been devoted to developing sophisticated methods for constructing high-energy light paths [LW93, VG97] or offsetting the inefficiency by reusing computation [Jen01, Kel97, GKDS12, HPJ12, KMA*15]. While these algorithms perform well in certain scenarios, they tend to under-perform in others. Furthermore, integrating them into heavily constrained production environments may still present a significant challenge for retaining efficiency, flexibility, and artistic control.

Our goal is to allow path-tracing algorithms to iteratively "learn" how to construct high-energy light paths while keeping the algorithm simple, yet robust and capable of simulating complex transport phenomena, such as caustics. Our approach is inspired by existing *path-guiding* algorithms that demonstrate that unidirectional path construction can be nearly as efficient as more sophisticated algorithms when properly guided by the adjoint quantity. Instead of using spatially cached histograms [Jen95], cones [HP02], or gaussian mixtures [VKv*14], we store a discrete approximation of the scene's 5D light field using an adaptive spatio-directional tree (SD-tree; see Figure 2). The SD-tree consists of an upper part—a binary tree that partitions the 3D spatial domain of the light field–and a lower part—a quadtree that partitions the 2D directional domain.

Our SD-trees adapt well to both low- and high-frequency light fields without excessive memory requirements and compare favorably to the previously used approaches for path guiding. They also permit fast importance sampling and can be constructed quickly, hence enabling trivial, fast adaptation to new information. Additionally, we present a refined iterative training scheme and provide a principled way for finding the sweet spot between training and rendering when a fixed time (or sample) budget is given. Lastly, the ease of implementation, support of progressive rendering, and the absence of tunable hyper-parameters—the only parameter is the maximum memory footprint—make our approach well-suited for production environments.

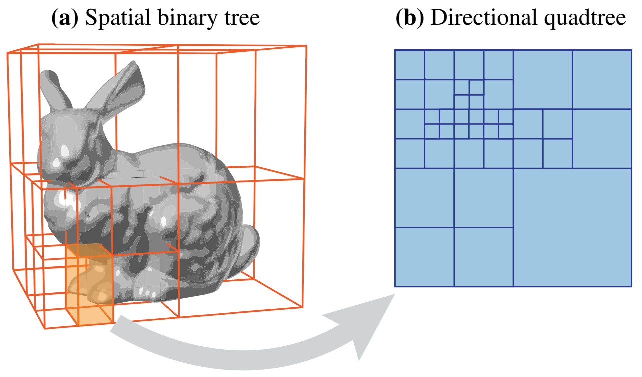

Figure 2: *The spatio-directional subdivision scheme of our SD-tree. Space is adaptively partitioned by a binary tree* **(a)** *that alternates between splitting the x, y, and z dimension in half. Each leaf node of the spatial binary tree contains a quadtree* **(b)**, *which approximates the spherical radiance field as an adaptively refined piecewise-constant function.*

2. Problem Statement and Related Work

The amount of radiance $L_o(\mathbf{x}, \vec{\omega}_o)$ leaving point $\mathbf{x}$ in direction $\vec{\omega}_o$ is quantified by the rendering equation [Kaj86]

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

where $L_e(\mathbf{x}, \vec{\omega}_o)$ is radiance emitted from $\mathbf{x}$ in $\vec{\omega}_o$, $L(\mathbf{x}, \vec{\omega})$ is radiance *incident* at $\mathbf{x}$ from $\vec{\omega}$, and f_s is the bidirectional scattering distribution function. The reflection integral L_r can be estimated numerically using N samples with the following Monte Carlo (MC) estimator

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

The variance of the estimator $V[\langle L_r \rangle]$ is proportional to $1/N$ and can be reduced by drawing $\vec{\omega}_j$ from a probability density function (PDF) $p(\vec{\omega}_j | \mathbf{x}, \vec{\omega}_o)$ that resembles the shape of the numerator. In fact, if the PDF differs from the numerator only by a scaling factor, then $V[\langle L_r \rangle] = 0$. While the BSDF and the cosine term can be approximated relatively well even in the general case, finding efficient means to quantify and represent the incident radiance field, and using it for importance sampling, has been a long-sought goal of MC image synthesis.

Jensen [Jen95] and Lafortune and Willems [LW95] first introduced the concept of guiding camera paths by leveraging information of previously traced paths. Jensen [Jen95] populates the scene with sampling points, each equipped with a hemispherical histogram from an a-priori traced set of photons. The histograms are then used to sample directions when building camera paths. Lafortune and Willems [LW95] propose to rasterize incident radiance in a 5D tree for later usage as a control variate and for importance sampling. Steinhurst and Lastra [SL06] refine Jensen's method by introducing product importance sampling with a discretized BSDF, and Budge et al. [BAJ08] apply a specialized form of Jensen's technique to improve sampling of caustics. Hey and Purgathofer [HP02] recognize that regular histograms are ill-suited for this type of density estimation and instead propose to average cones of adaptive width centered around the incident directions of photons. Recently, Vorba et al. [VKv*14] proposed to instead perform density estimation via a parametric gaussian-mixture model and an iterative on-line reinforcement learning algorithm. Follow-up works added approximate product importance sampling with the BSDF [HEV*16], and adjoint-based Russian roulette [VK16] to the method. Dahm and Keller [DK17] combine rendering and training into the same algorithm that requires a careful sample weighting to diminish in the limit.

In our work, we adopt adjoint-based Russian roulette and progressive reinforcement learning, while also fusing the rendering and learning algorithms into one. Furthermore, we split the learning procedure into distinct passes—each guided by the previous pass—in such a way that each pass remains unbiased.

The main distinction of our approach, however, is the data structure used for representing the incident radiance field. We employ a combination of a binary tree for adaptive spatial partitioning, and a directional quadtree for adaptive binning in the angular domain. Such hybrid data structures, e.g. volume-surface trees [BHGS06] or rasterized BVHs [ND12], were successfully applied to geometry processing and rendering. In particular, Gassenbauer et al. [GKB09] used a spatial octree and a directional kd-tree to record individual radiance samples for the similar problem of *radiance caching*. We opt for a binary spatial tree due to its smaller branching factor that is better suited for our progressive reinforcement learning described in Section 3.4. We use a quadtree for the directional domain to circumvent the downsides of regular binning [Jen95] and to facilitate a straightforward construction (see Section 3.3) and robust performance, which is harder to achieve with kd-trees [GKB09] and gaussian-mixture models [VKv*14, HEV*16], respectively.

3. Path Guiding with SD-Trees

We use reinforcement learning to construct a discrete approximation of the incident radiance field, further denoted as $\hat{L}$. The field $\hat{L}$ is represented by an SD-tree and iteratively improved with a geometrically increasing compute budget; we double the number of samples across iterations. We always maintain two SD-trees: one for *guiding* the construction of light paths and another for collecting MC estimates of incident radiance. More precisely, in iteration k we importance-sample incident radiance using the previously populated $\hat{L}^{k-1}$ and splat estimates of $L(\mathbf{x},\vec{\omega})$ into $\hat{L}^k$. When iteration k is completed, we drop $\hat{L}^{k-1}$ and use the information in $\hat{L}^k$ to prepare an empty SD-tree, $\hat{L}^{k+1}$ for collecting estimates in the next iteration.

3.1. Collecting Estimates of L

When a complete path is formed, we iterate over all its vertices and splat MC estimate of incident radiance into $\hat{L}^k$. For vertex v with radiance estimate $L(\mathbf{x}_v,\vec{\omega}_v)$, we first perform a *spatial* search by descending through the binary tree to find the leaf node that contains position $\mathbf{x}_v$. The leaf node stores a reference to a quadtree. The two dimensions of the quadtree parameterize the full sphere of directions; we use world-space cylindrical coordinates to preserve area ratios when transforming between the primary and directional domain. We continue the traversal in the *directional* domain by descending through the quadtree. We enter only nodes that contain $\vec{\omega}_v$ and deposit the estimate $L(\mathbf{x}_v,\vec{\omega}_v)$ in all nodes visited during the descent.

When all radiance estimates in the current iteration were deposited, the nodes of the quadtrees estimate the total incident radiance arriving through the spherical region corresponding to their respective quad. Note that the directional distribution is averaged over all spatial positions that were sampled within the binary-tree leaf associated with the quadtree. In addition to guiding the path construction in the next iteration, this information is also used to adapt the structure of the SD-tree for collecting future estimates; we describe the adaptation of the spatial and directional components of the SD-tree in the next two sections.

3.2. Adaptive Spatial Binary Tree

The depth and structure of the binary tree determines how refined and adaptive the approximation of the spatial component of the radiance field is. To keep the refinement straightforward, we alternate the x, y, and z axes and always split the node in the middle. The decision whether to split is driven only by the number of path vertices that were recorded in the volume of the node in the previous iteration; we maintain a counter for each leaf node during path tracing. Specifically, we split a node if there have been at least $c \cdot \sqrt{2^k}$ path vertices, where 2^k is proportional to the amount of traced paths in the k-th iteration (Section 3.4) and c is derived from the resolution of the quadtrees; we detail this in Section 5.3. Post subdivision, all leafs contain roughly $c \cdot \sqrt{2^k}$ path vertices. Therefore, the total amount of leaf nodes is proportional to $\frac{2^k}{c \cdot \sqrt{2^k}} = \frac{\sqrt{2^k}}{c}$. Effectively, our threshold ensures that the total number of leaf nodes and the amount of samples in each leaf both grow at the same rate $\sqrt{2^k}$ across iterations. The constant c trades off convergence of the directional quadtrees with spatial resolution of the binary tree.

While refining the tree based only on the number of samples may seem rudimentary, it performs remarkably well since the iteratively learned distributions guide paths into regions with high contributions to the image; these thus get refined more aggressively than low-contribution regions. Having a coarser radiance-function approximation in regions that receive fewer paths is tolerable, because increase in relative noise is generally counteracted by the smaller contribution of such paths.

3.3. Adaptive Directional Quadtree

In addition to splitting the binary tree, we also rebuild all of the quadtrees after each iteration to better reflect the learned directional distribution of radiance; these new quadtrees will be used for collecting the estimates in the next iteration. The structure of each new quadtree is driven by the directional distribution of flux collected in the last iteration. To this end, we first copy either the leaf's old quadtree, or the quadtree of its parent if the leaf node is new.

Our goal is to adapt the subdivision of the copied quadtree so that each leaf contains no more than 1% of flux collected in the old quadtree. We descend through the copied quadtree and subdivide its nodes only if the fraction of collected flux flowing through the node is larger than $\rho = 0.01$; i.e. if $\Phi_n/\Phi > \rho$, where Φ is the total flux flowing through the quadtree, and Φ_n is the flux flowing through the node in question. When subdividing a node, we assign a quarter of its flux to the newly created children and we recursively apply our subdivision criterion. At the same time, we evaluate the subdivision criterion at every existing interior node, and we prune its children if the criterion is not met. Spherical regions with high incident flux are thus represented with higher resolution. The proposed subdivision scheme yields a roughly equi-energy partitioning of the directional domain.

Rebuilding the quadtrees after each iteration ensures that the data structure adapts to the newly gained information about the radiance field, and that memory is used efficiently. The threshold ρ effectively controls how much memory is used, since the amount of nodes in the quadtree is proportional to $1/\rho$. We provide a more thorough analysis of the memory usage in Section 5.2.

3.4. Unbiased Iterative Learning and Rendering

In order to accelerate learning, we use an iterative scheme similar to the one proposed by Vorba et al. [VKv*14]. We train a sequence $\hat{L}^1, \hat{L}^2, ..., \hat{L}^M$ where $\hat{L}^1$ is estimated with just BSDF sampling, and for all $k > 1$, $\hat{L}^k$ is estimated by combining samples of $\hat{L}^{k-1}$ and the BSDF via multiple importance sampling. Sampling from the previously learned distribution $\hat{L}^{k-1}$ to estimate $\hat{L}^k$ often drastically accelerates the convergence compared to naïve Monte Carlo estimation.

Given a path vertex v, we sample a direction from $\hat{L}^{k-1}$ as follows. First, we descend *spatially* through the binary tree to find the leaf node containing the vertex position $\mathbf{x}_v$. Next, we sample the direction $\vec{\omega}_v$ from the quadtree contained in the spatial leaf node via hierarchical sample warping as described by McCool and Harwood [MH97]. We provide pseudocode in the supplementary material.

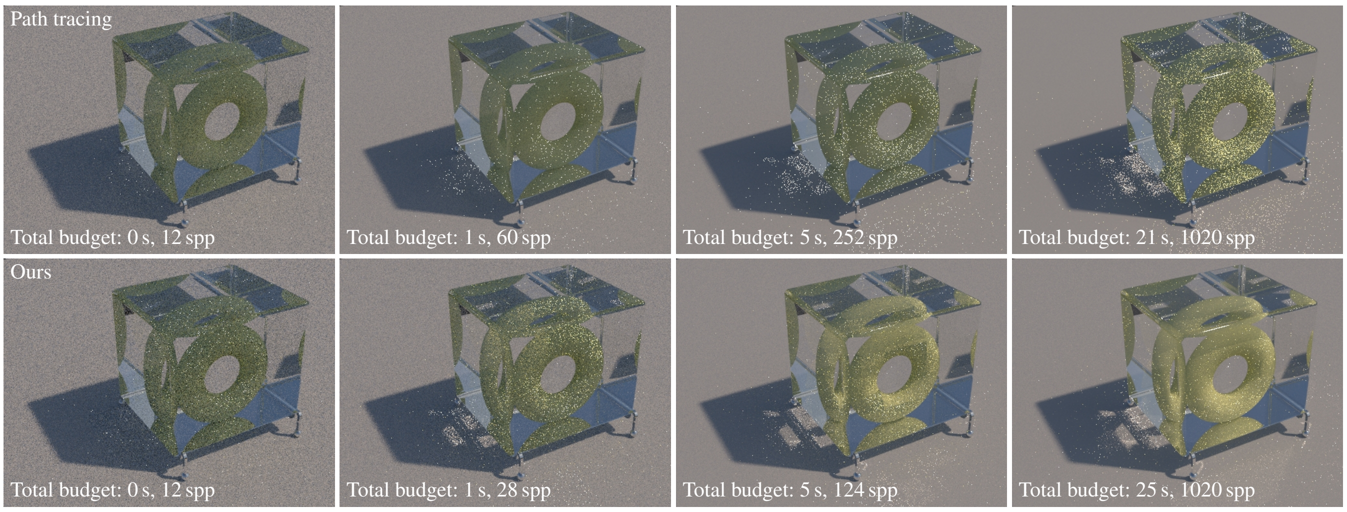
Path tracing
Total budget: 0 s, 12 spp
Total budget: 1 s, 60 spp
Total budget: 5 s, 252 spp
Total budget: 21 s, 1020 spp
Ours
Total budget: 0 s, 12 spp
Total budget: 1 s, 28 spp
Total budget: 5 s, 124 spp
Total budget: 25 s, 1020 spp

Figure 3: *Our method converges most rapidly in the beginning of rendering, yielding a reasonable preview of the final image much more quickly than regular path tracing. Except for Dahm and Keller [DK17], existing path guiding approaches perform training in a separate pass, neglecting the rendering preview.*

Exponential Sample Count. If we used an equal amount of path samples in each iteration, then only a small (the last) fraction of samples would contribute to the image directly, with the (preceding) majority being used "just" for learning the incident radiance field. This would not be a problem if the learned distributions were proportional to the full numerator in Equation (2), in which case a single sample would theoretically suffice for finding the scaling factor between the numerator and the PDF. Our distributions, however, only approximate the incident radiance requiring us to still integrate the full product over the hemisphere. We thus propose to increase the number of path samples in each iteration *geometrically*. More precisely, we use twice as many samples to learn $\hat{L}^k$ than to learn $\hat{L}^{k-1}$. Learning $\hat{L}^k$ then takes approximately twice longer than learning $\hat{L}^{k-1}$, but $\hat{L}^k$ has roughly half the variance. In practice, the variance reduction is typically much higher due to the positive effects of the iterated importance sampling scheme. Nevertheless, in the worst-case of iterative learning *not* improving the convergence, only half of the samples is "wasted" on learning the distributions.

Another important property of doubling the sample count in each iteration surfaces when considering our spatial subdivision scheme. Since spatial subdivision of a binary-tree leaf node halves its volume, doubling the amount of samples ensures that approximately the same number of samples reaches both new leaf nodes. Therefore, even locally, $\hat{L}^k$ generally does not become noisier than $\hat{L}^{k-1}$.

Online Rendering Preview. To provide the user with a quick visual feedback, we progressively display images synthesized using the path samples of the current iteration. As long as we do not fuse path samples across iterations, the image will be unbiased since all path samples within the same iteration are mutually independent. Since each iteration starts rendering the image from "scratch", naïvely displaying the latest result would lead to sudden quality degradations whenever a new iteration starts. We avoid this by switching to the image of the current iteration only once it accumulated more samples than the previous iteration. In Figure 3 we demonstrate the rendering preview of the TORUS scene comparing against a path tracer. In the supplementary video we compare the preview on additional scenes against other methods.

3.5. Balancing Learning and Rendering

In this section, we describe how to split a given compute budget B, which can be defined either as time or number of samples, between learning and rendering such that the variance of the final image is minimized. For iteration k, we define the *budget to unit variance* $\tau_k = V_k \cdot B_k$, i.e. the product of variance of image I_k computed using paths traced in iteration k, and the budget B_k spent on constructing these paths. Variance V_k is computed as the *mean* variance of pixels in I_k. Assuming we keep using $\hat{L}^k$ for guiding the paths until we reach B, we can estimate the variance of the *final* image as

$$\hat{V}_k = \frac{\tau_k}{\hat{B}_k}\,, \tag{3}$$

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

Our goal is to find the optimal iteration $\hat{k}$ that minimizes the final-image variance, i.e. $\hat{k} = \arg\min_k \hat{V}_k$. To that end, we assume that training has monotonically diminishing returns; more precisely, the sequence τ_k is monotonically decreasing and convex. It follows that $\hat{V}_k$ is also convex (see Appendix A). We can then find $\hat{k}$ as the smallest k for which $\hat{V}_{k+1} > \hat{V}_k$ holds. Since we need to evaluate $\hat{V}_{k+1}$, we need to perform one more iteration than would be optimal, but the wasted computation is greatly outweighed by the variance reduction due to our automatic budgeting mechanism.

We can use a similar approach to optimally trade-off training and rendering when aiming for a target variance. In this case, we can estimate the *rendering* budget $\bar{B}_k$ required to reach a target variance

	SPPM	MEMLT	Vorba et al.	Ours	Reference
MSE:	0.936	2067.754	0.150	0.144	—
Samples per pixel:	—	—	4264	5812	—
Minutes (training + rendering):	0 + 7.5	0 + 7.5	5.3 + 6.4	1.1 + 6.4	—

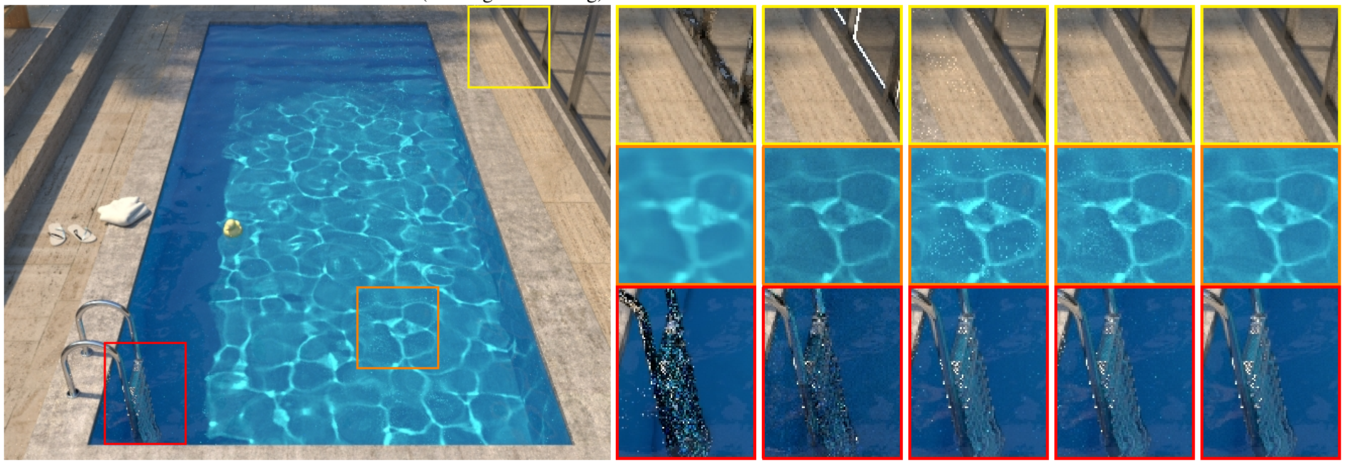

Figure 4: *Equal-time comparison of our method versus previous work on the* POOL *scene; we report training + rendering time in minutes. The caustics inside the pool consist of difficult* specular, diffuse, specular *light transport that varies spatially due to the waves. The computation efficiency of our method allows drawing* 1.36× *more samples than the method by Vorba et al. [VKv*14] at equal time, achieving a slight improvement in overall noise. The optimal training-rendering budgeting is in this case automatically determined to be* 15% *and* 85%*, respectively.*

	PT w/ NEE	BDPT	MEMLT	Vorba et al.	Ours	Reference
MSE:	7.949	2.800	0.742	1.052	0.694	—
Samples per pixel:	3100	2560	—	1104	1812	—
Minutes (training + rendering):	0 + 5.1	0 + 5.1	0 + 4.85	5.5 + 3.9	1.1 + 3.9	—

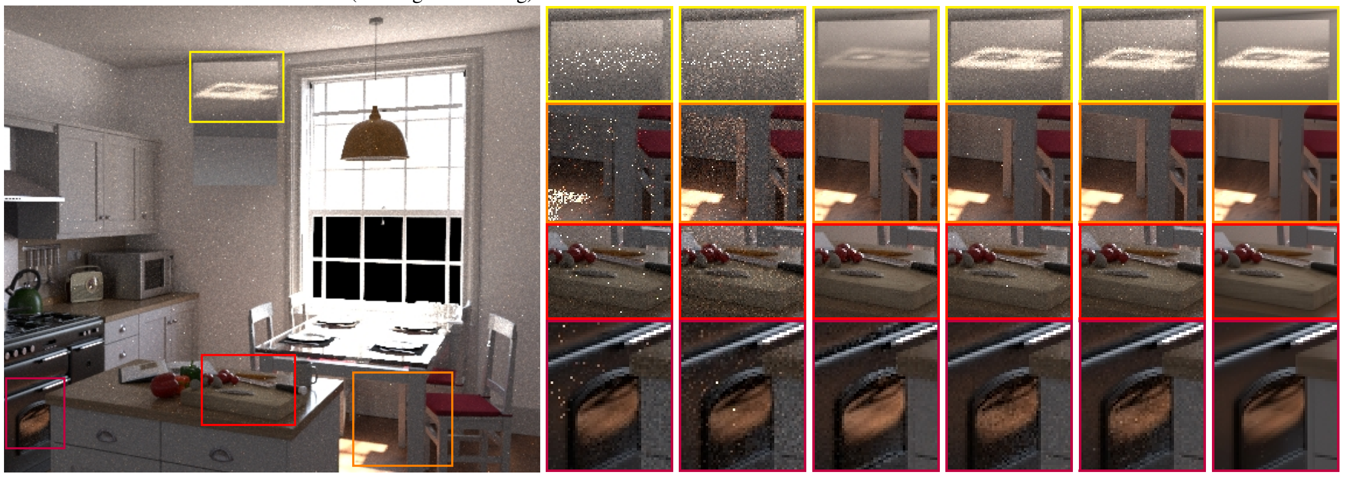

Figure 5: *The glass tabletop and the mirror next to the window in the* KITCHEN *scene challenge the efficiency of most light-transport algorithms. The window is glass-free to permit path-tracing algorithms to perform connections, e.g. for next-event estimation (NEE). Nevertheless, unidirectional path tracing is unable to capture most of the non-trivial indirect-illumination patterns. Bidirectional path tracing struggles with constructing light sub-paths through the window (just like SPPM would). As in the* POOL *scene, MEMLT avoids "fireflies" but has convergence issues. Both guiding methods, even with just a vanilla path tracer without NEE, are capable of finding difficult light paths. However, our approach performs similar compared to Vorba et al.'s despite being purely unidirectional. The scene is a slightly modified version of "Country Kitchen" by Jay-Artist (CC BY 3.0).*

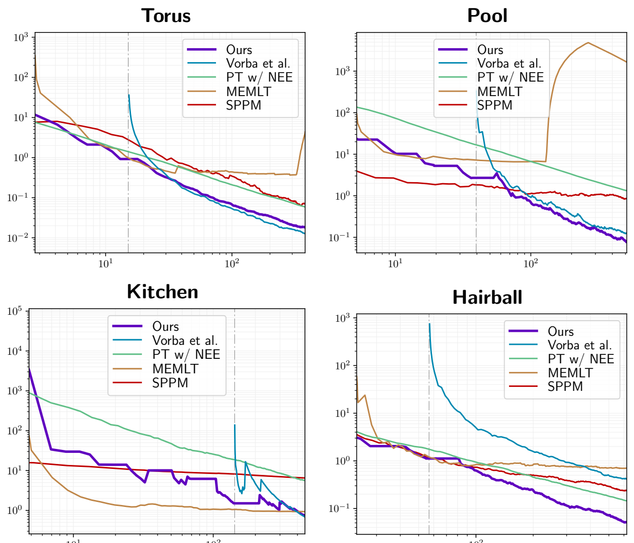

Figure 6: Mean squared error *(MSE) plotted as a function of time demonstrates the convergence of individual algorithms. The dash-dotted vertical lines indicate when our method stops training and switches to render-only mode for the rest of the given budget: 5, 7.5, 5, and 20 min. In these plots the rendering component of Vorba et al.'s algorithm is synchronized with ours.*

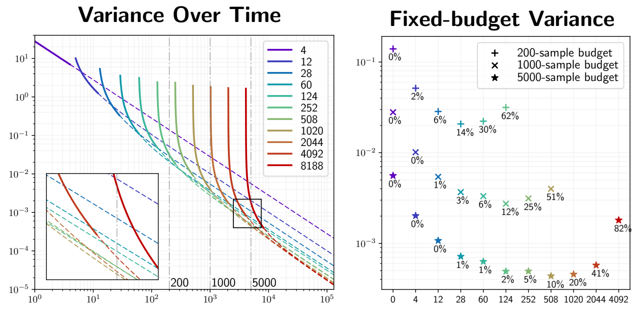

Figure 7: *Left: variance as a function of time plotted for subsequent training iterations; the legend shows the number of samples invested into training the guiding distributions. The dashed extension lines indicate convergence if we continued rendering, i.e. we did not switch to the next training iteration. Right: intersections of the convergence curves with three dash-dotted lines representing three distinct sample budgets. For each sample budget, the intersections form a convex sequence when plotted according to the total number of samples. The annotations show the percentage of the sample budget that was spent on training the distributions.*

$\bar{V}$ via $\bar{B}_k = \tau_k/\bar{V}$, and training is stopped whenever the *total* budget $\tilde{B}_k > \tilde{B}_{k-1}$, where

$$\tilde{B}_k = \bar{B}_k + \sum_{i=1}^{k-1} B_i \,. \tag{4}$$

This successfully finds $\hat{k} = \arg\min_k \tilde{B}_k$, because the sequence $\tilde{B}_k$ is convex whenever B_k is monotonically increasing, which is the case with the exponential sample count.

4. Results

We integrated our algorithm into the Mitsuba renderer [Jak10]. Pseudocode of the main parts is included in the supplementary material. We compare our technique to several other approaches, including bidirectional path tracing [VG94], stochastic progressive photon mapping (SPPM) [HJ09], manifold exploration metropolis light transport (MEMLT) [JM12], and, closely related to ours, the technique by Vorba et al. [VKv*14, VK16], which represents incident radiance using gaussian-mixture models (GMM); we use the authors' implementation for comparison. To ensure the path-guiding GMMs are properly trained, we always use 30 pre-training passes, 300000 importons and photons, adaptive environment sampling, and we leave all other parameters at their default values. In all comparisons images were rendered with an equal time budget. For our method, training happens *within* its time budget. We *do not* count pre-training of the GMMs as part of their budget; we give GMMs as much *rendering* time as our method uses.

Both path-guiding methods—Vorba et al.'s and ours—render with unidirectional path tracing *without next event estimation* (NEE) to emphasize the difference in guiding distributions. Application to more sophisticated algorithms such as BDPT or VCM [GKDS12] would only mask the shortcomings of path guiding and obscure the comparisons. Lastly, none of the methods perform product importance sampling, since its benefits are orthogonal (and complementary) to path guiding. Extending our work to perform product importance sampling is discussed in Section 7. All results presented onwards can be inspected using an interactive viewer in the supplementary material.

The TORUS scene contains very long chains of specular interactions and a significant amount of *specular-diffuse-specular* (SDS) light transport, which is notoriously difficult to simulate with most unbiased algorithms. Path guiding approaches are able to learn and importance sample the high-frequency transport as long as it can be discovered by the underlying algorithm. In Figure 1 we render the scene with our method comparing at equal-time to the method by Vorba et al. [VKv*14]. The GMMs struggle at learning the correct distribution on the torus consistently, manifesting as uneven convergence; see Figure 9. Our method achieves a slightly worse MSE as Vorba et al.'s method, while our automatic budgeting mechanism assigned 44 s to training out of the total 298 s compute time.

The POOL scene features difficult SDS light transport in a realistic scenario: rendering under-water caustics caused by waves. Standard (unguided) path tracing performs very poorly on this scene; the reference image in Figure 4 (right) took 18 h to render and still exhibits residual noise in some regions. By the nature of density estimation the SPPM algorithm can handle such scenes without bright pixels ("fireflies"), but it still struggles with preserving the sharpness of caustics and produces splotchy artifacts on the window frames. The manifold-walk-enhanced MLT (MEMLT) preserves the caustics' sharpness, but its uneven visual convergence manifests on the over-exposed window frame and darker pool ladder. Both guided unidirectional path tracers suffer from occasional outliers,

Table 1: *Statistics of Vorba et al.'s method and ours on all scenes presented in this paper.*

Scene	Method	Memory	Training	Rendering	SPP	MSE
TORUS	Vorba et al.	2.5 MB	5.1 min	4.2 min	8932	0.017
	Ours	4.2 MB	0.73 min	4.2 min	11568	0.018
HAIRBALL	Vorba et al.	511.2 MB	60 min	17.3 min	1492	0.246
		7.2 MB	6.7 min	17.3 min	476	1.250
	Ours	6.1 MB	1.7 min	17.3 min	20596	0.022
POOL	Vorba et al.	12.8 MB	5.3 min	6.4 min	4264	0.150
	Ours	8.0 MB	1.1 min	6.4 min	5812	0.144
KITCHEN	Vorba et al.	23.8 MB	5.5 min	3.9 min	1104	1.052
	Ours	15.6 MB	1.1 min	3.9 min	1812	0.694

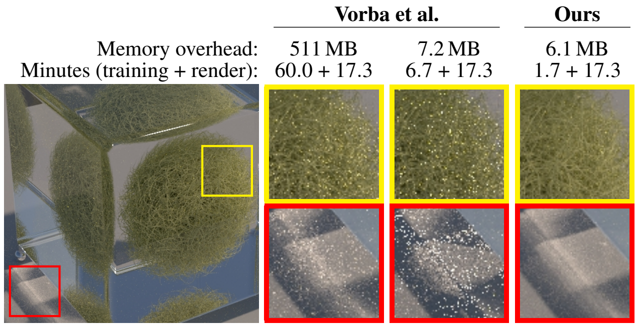

Figure 8: *Comparison of our method and the method by Vorba et al. [VKv*14] on a heavily tesselated hairball inside a glass cube. With its default parameters, Vorba et al.'s algorithm (left) exhibits worse performance and 83× higher memory consumption than ours (right). The benefits of our approach become even more prominent in a roughly equal-memory comparison (middle vs right).*

but estimate the overall brightness more reliably than MEMLT and without the bias of SPPM. Compared to the method by Vorba et al., our SD-trees slightly reduce the number of "fireflies" as well as the average noise at roughly two thirds of the memory; see Table 1.

The KITCHEN scene in Figure 5 consists of various glossy materials and complex geometries that are lit by sunlight entering through a glass-free window and being reflected by a glass tabletop. The reflection of the sun on the ceiling viewed through the mirror—an SDS interaction depicted in the yellow inset—poses a challenge for most algorithms, including MEMLT. When properly guided, a simple unidirectional path tracer *without* NEE is capable of efficiently sampling these narrow, high-energy regions of path space even without Markov-chain mutations. Despite learning the incident light field only unidirectionally, our method performs the best in terms of MSE thanks to the ability to render faster than Vorba et al.'s method. Our technique also requires slightly less memory in this scene.

5. Analysis

In this section, we analyze the convergence and memory usage and discuss the threshold for subdividing the binary tree.

5.1. Convergence

Comparison to Other Methods. Figure 6 compares the convergence of our algorithm to several competing methods. Pre-training time of Vorba et al.'s algorithm is not included in these plots, but instead its rendering phase is synchronized with ours. In the beginning of the rendering process, MEMLT and SPPM often outperform our method. However, the inherently unpredictable nature of MEMLT and the asymptotically slower convergence of SPPM allow our method to eventually overtake. The "flat" regions in the training phase of our algorithm are due to delayed switching of the online preview to the result of the current iteration, as described in Section 3.4. We provide a visualization of the progressive rendering process in the supplementary video.

Convergence as a Function of Training Budget. Figure 7 demonstrates the convergence as a function of how many samples are used for training the guiding distributions. The curves represent the convergence of subsequent, geometrically growing iterations. Later iterations converge faster to a lower error thanks to their better-trained guiding distributions. When a fixed compute budget is given, our balancing scheme, as described in Section 3.5, automatically chooses when to stop the iterative training and continue rendering until the budget is exhausted. In the scatter plot on the right, we show the estimated final variance for the increasingly-trained distributions for the target budget of 200, 1000, and 5000 samples. The three series empirically demonstrate the convexity of this optimization problem (the small deviations from a perfectly convex shape are due to the discrete refinement decisions). Our balancing scheme always correctly predicted the optimum.

5.2. Memory Usage

The memory usage of our directional quadtrees can be estimated using the subdivision threshold ρ: with the energy-based refinement from Section 3.3 there can be no fewer than $1/\rho$ leaf nodes. The maximum number is unbounded for arbitrarily narrow energy distributions (e.g. a delta light source). We remedy this by limiting the maximum depth of the quadtree to 20, which is sufficient for guiding towards extremely narrow radiance sources without precision issues. The worst case of refining $1/\rho$ quads to the maximum depth results in the upper bound of $4 \cdot 20/\rho$ on the amount of nodes.

We found $\rho = 0.01$ to work well, which in practice results in an average of roughly 300 nodes per quadtree. The maximum amount of nodes across all quadtrees and results we tested is 792; well below the theoretical maximum of $4 \cdot 20/\rho = 8000$. Our quadtree nodes require 5 bytes of memory each, translating to an average footprint of 1.5 kb, and an upper bound of 40 kb, for each quadtree. A single mixture model with 8 gaussians requires 273 bytes.

According to the iterative training and rendering scheme, only two SD-trees $\hat{L}^{k-1}$ and $\hat{L}^k$ have to be kept in memory at the same time. However, because the spatial binary tree of $\hat{L}^k$ is merely a more refined version of the spatial tree of $\hat{L}^{k-1}$, it is straightforward to use the *same* spatial tree for both distributions, where each leaf contains *two* directional quadtrees; one for $\hat{L}^{k-1}$ and one for $\hat{L}^k$.

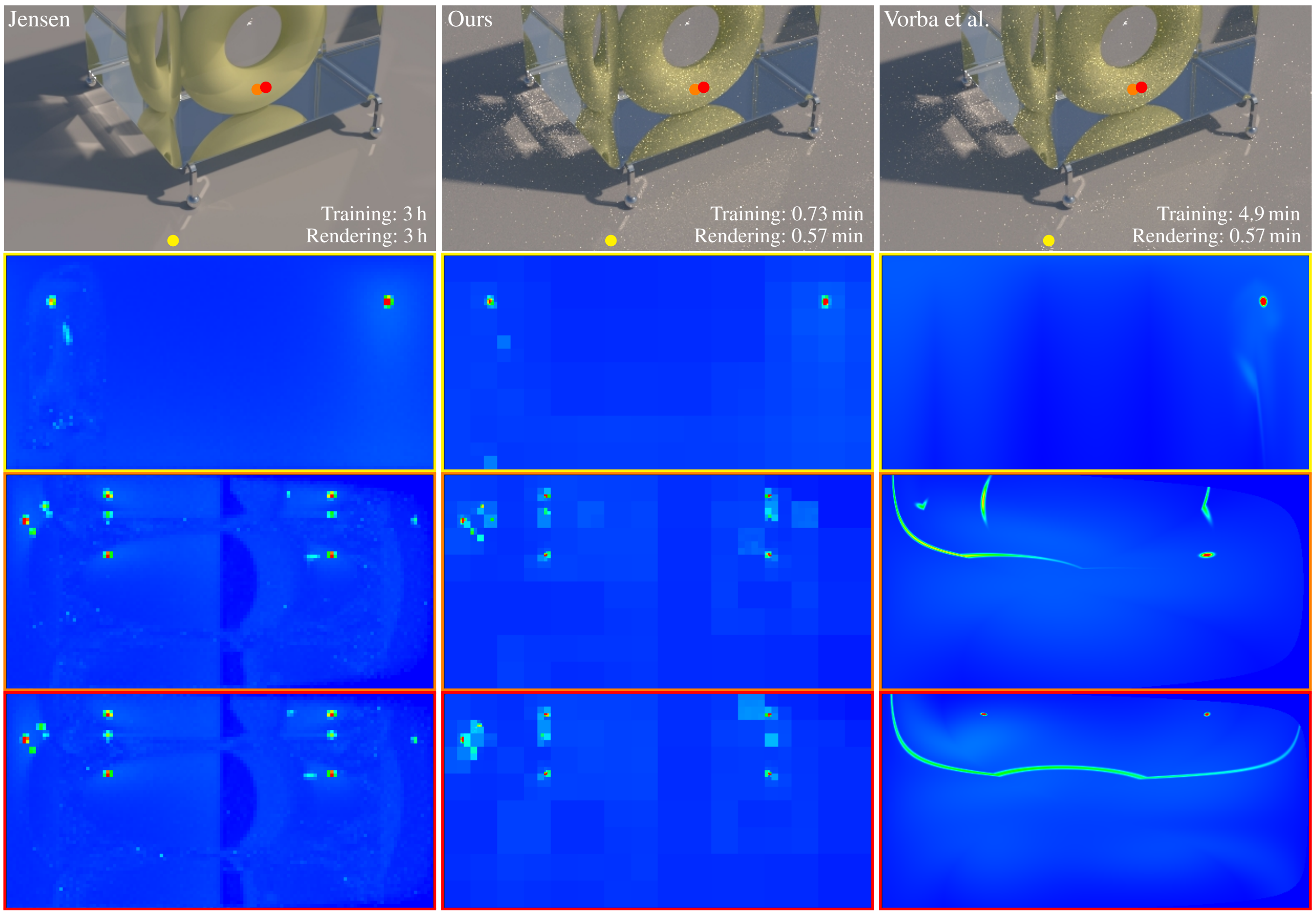

Figure 9: *We show directional radiance distributions discretized in Jensen's [Jen95] datastructure (left), our quadtrees (middle), and Vorba et al.'s [VKv*14] gaussian mixture model (GMM; right). We used the same spatial locations to train all three directional distribution models. The left column was trained and rendered for 6 h to provide a reasonably converged reference. Our and Vorba et al.'s distributions have each been computed within 5 min. At the location marked in yellow, Vorba et al.'s method exhibits high variance as the GMM failed to represent the light reflected by the glass cube (right; yellow inset). The distributions at the locations marked in orange and red demonstrate the instability of the GMM under similar incident illumination, and its struggle to capture multiple modes robustly. Our quadtrees reliably capture all 5 important modes with high precision. We provide additional visualizations in the supplementary video.*

This means, that only a single spatial binary tree has to be traversed for each path vertex.

In all our scenes, the full guiding distribution never exceeded 20 mb (see Table 1) without anyhow limiting the subdivision of the spatial binary tree. However, we allow specifying an upper bound of nodes in the spatial tree to ensure memory requirements are kept. Once the limit is reached, the spatial tree is not subdivided further.

5.3. Binary Tree Subdivision Threshold

As described in Section 3.3, we subdivide a leaf node of the binary tree whenever it records more then $c \cdot \sqrt{2^k}$ path vertices during the previous iteration. We derive c from the desired number of samples s that each *quadtree leaf node* should receive. All quadtree leaves carry a roughly equal amount of energy and are thus sampled with similar probability during training. The average number of samples a quadtree leaf receives is then $s = S/N_l$, where S is the total amount of samples drawn from a quadtree, and N_l is the amount of leaf nodes in it. Our quadtrees have 300 nodes on average (see Section 5.2), and almost never more than that. We found, that $s = 40$ samples per quadtree leaf node results in a reasonably converged quadtree, and we thus choose c such that binary tree leaf nodes get subdivided after the first iteration ($k = 0$) when this number of samples is reached. We thus obtain

$$c = \frac{s \cdot N_l}{\sqrt{2^k}} = \frac{40 \cdot 300}{1} = 12000\,. \tag{5}$$

6. Discussion and Future Work

Spherical vs. Hemispherical Domain. Our directional quadtree distributions cover the entire sphere of directions parameterized using world-space-aligned cylindrical coordinates. This has two key benefits compared to most previous work [Jen95, VKv*14, HEV*16] that covers only the upper (oriented) hemisphere. Firstly, we do not need to discriminate distributions according to their angular distance to the normal at the shading point. This simplifies the search to merely selecting the spatially nearest distribution and avoids the need for rotating the distribution to prevent overblurring in the directional domain. Secondly, spherical distributions naturally generalize to volumetric path tracing and typically perform better on organic structures, such as foliage and hair. We demonstrate this on the HAIRBALL scene in Figure 8 consisting of cylindrical hairs inside a glass cube. We compare our orientation-independent cylindrical parametrization to the hemispherical parameterization of Vorba et al. [VKv*14]. To remain accurate, Vorba et al.'s method has to densely populate the hairball with hemispherical distributions, leading to a significant memory and performance cost. The cylindrical parametrization is decoupled from geometric properties, and results in almost 83× lower memory consumption and significantly improved convergence rate. Adjusting the parameters of Vorba et al.'s method to approximately match our memory consumption yields significantly slower convergence.

Quadtree vs. Gaussian Mixture Model. The main advantage of quadtrees over gaussian-mixture models is the increased robustness. The expectation-maximization algorithm utilized by Vorba et al. [VKv*14] is not guaranteed to find the global optimum, and the distribution can vary dramatically across nearby spatial locations; see Figure 9. In contrast, our quadtrees adapt to the energy distribution hierarchically, top-down and adapt the resolution such that each leaf contains roughly the same amount of energy. The convergence of the rendering algorithm (within one iteration) is thus more stable.

Geometric Iterations vs. Moving Average. The final image could be estimated using an exponential moving average of *all* path samples. Since in this case earlier samples are drawn from a suboptimal distribution their variance can be very large, potentially resulting in higher overall variance. Our geometrically growing iterations ensure that the last iteration has a sufficient number of high-quality samples to provide an accurate estimate. Samples in previous iterations are invested only into constructing path-guiding distributions.

Temporal path guiding. We described an algorithm for guiding light transport in a static scene. For time-dependent effects, e.g. motion blur, it would be beneficial to refine the SD-tree also over the temporal domain. We suspect that adding a fourth dimension t to our spatial binary tree and including it in the dimension-alternating subdivision scheme is a straightforward extension which is capable of handling temporal effects.

Multiple and Product Importance Sampling. Sampling scattering directions purely from our quadtrees can lead to arbitrarily large variance in regions where where the incident radiance is approximated badly. We avoid this problem by combining directions from our quadtrees with BSDF samples via multiple importance sampling, which guarantees that the variance can not be much worse compared to pure BSDF sampling. The performance of sampling with our quadtrees could be further improved by ignoring quads in the bottom hemisphere. This can be done explicitly with a rectifying parameterization by Bitterli et al. [BNJ15], though the mapping would no longer preserve areas. Another possible solution is to generate several samples from the BSDF and the quadtree, and to then perform importance resampling [TCE05] towards their product. This technique never generates samples in the wrong hemisphere and approaches product importance sampling as the number of candidate samples for resampling approaches infinity.

A potentially more fruitful avenue for future research would be to directly sample the product of the BSDF and the incident radiance. Herholz et al. [HEV*16] demonstrated the benefits of product importance sampling with GMMs. We could similarly extend our work by adopting the approach of Jarosz et al. [JCJ09]. This would require converting the incident radiance distributions into Haar wavelets and representing BSDFs in spherical harmonics. Because our quadtrees only approximate the incident radiance, however, it is advisable to still combine the product with the pure BSDF via multiple importance sampling to bound the error.

Guiding sophisticated sampling algorithms. We presented results with guiding applied to a unidirectional path tracer. The deliberate choice of using a simple path tracer was made to best highlight the strengths and weaknesses of different guiding approaches; more sophisticated path-construction algorithms would only mask the shortcomings in regions where they already sample the integrand well. In practice, however, we recommend combining path-guiding with NEE via multiple importance sampling due to the ease of implementation and low performance overhead. More sophisticated light-transport algorithms typically also benefit from path guiding as demonstrated by Vorba et al. [VKv*14] for BDPT and VCM.

7. Conclusion

We presented a practical, easy-to-implement, unbiased path-guiding approach with a low-latency rendering preview. Our approach does not require tuning hyper-parameters; $\rho = 0.01$, $c = 12000$ performed well in all our tests. The only parameter to be specified is the maximum memory footprint of the SD-tree. While more sophisticated algorithms may outperform our technique in specific situations, its high performance and the aforementioned advantages of SD-trees make them appealing for production environments. We demonstrated this on a simple unidirectional path tracer that outperformed more sophisticated techniques in all our comparisons. Another benefit is the fairly even spatial convergence, which we expect to increase the attractiveness of our guiding approach for animation rendering, when combined with a suitable technique for removing fireflies. Finally, since path-guiding is largely agnostic to the underlying path-sampling algorithm, SD-trees can be easily applied to BDPT, PPM, VCM, MLT, or other techniques, significantly improving their performance beyond the demonstrated results.

Acknowledgements

We thank Brian McWilliams for helpful discussions and insights into various machine learning topics, and for narrating the supplementary video. We thank Ryusuke Villemin for valuable feedback; specifically, for pointing out the applicability of importance resampling. We are grateful to Marios Papas for proofreading, to Olesya Jakob for remodeling the TORUS scene by Cline et al. [CTE05], to Ondřej Karlík for the POOL scene, to Blend Swap user Jay-Artist for the KITCHEN scene [Bit16] (CC BY 3.0), which we modified slightly, and to Samuli Laine for the Hairball model [McG11].

References

[BAJ08] BUDGE B. C., ANDERSON J. C., JOY K. I.: Caustic Forecasting: Unbiased Estimation of Caustic Lighting for Global Illumination. *Computer Graphics Forum* (2008). 2

[BHGS06] BOUBEKEUR T., HEIDRICH W., GRANIER X., SCHLICK C.: Volume-Surface Trees. *Computer Graphics Forum 25*, 3 (2006), 399–406. 2

[Bit16] BITTERLI B.: Rendering resources, 2016. https://benedikt-bitterli.me/resources/. 10

[BNJ15] BITTERLI B., NOVÁK J., JAROSZ W.: Portal-masked environment map sampling. *Computer Graphics Forum 34*, 4 (June 2015). 9

[CTE05] CLINE D., TALBOT J., EGBERT P.: Energy redistribution path tracing. *ACM Trans. Graph. 24*, 3 (July 2005), 1186–1195. 9

[DK17] DAHM K., KELLER A.: Learning light transport the reinforced way. *CoRR abs/1701.07403* (2017). 2, 4

[GKB09] GASSENBAUER V., KŘIVÁNEK J., BOUATOUCH K.: Spatial directional radiance caching. *Computer Graphics Forum 28*, 4 (2009), 1189–1198. Eurographics Symposium on rendering, EGSR '09. 2

[GKDS12] GEORGIEV I., KŘIVÁNEK J., DAVIDOVIČ T., SLUSALLEK P.: Light transport simulation with vertex connection and merging. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 192:1–192:10. 1, 6

[HEV*16] HERHOLZ S., ELEK O., VORBA J., LENSCH H., KŘIVÁNEK J.: Product Importance Sampling for Light Transport Path Guiding. *Computer Graphics Forum* (2016). 2, 8, 9

[HJ09] HACHISUKA T., JENSEN H. W.: Stochastic progressive photon mapping. *ACM TOG 28*, 5 (Dec. 2009), 141:1–141:8. 6

[HP02] HEY H., PURGATHOFER W.: Importance sampling with hemispherical particle footprints. In *Proceedings of the 18th Spring Conference on Computer Graphics* (New York, NY, USA, 2002), SCCG '02, ACM, pp. 107–114. 2

[HPJ12] HACHISUKA T., PANTALEONI J., JENSEN H. W.: A path space extension for robust light transport simulation. *ACM TOG 31*, 6 (Nov. 2012), 191:1–191:10. 1

[Jak10] JAKOB W.: Mitsuba renderer, 2010. http://www.mitsuba-renderer.org. 6

[JCJ09] JAROSZ W., CARR N. A., JENSEN H. W.: Importance sampling spherical harmonics. *Computer Graphics Forum 28*, 2 (Apr. 2009), 577–586. 9

[Jen95] JENSEN H. W.: Importance driven path tracing using the photon map. In *Rendering Techniques* (Vienna, 1995), Springer Vienna, pp. 326–335. 2, 8

[Jen01] JENSEN H. W.: *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001. 1

[JM12] JAKOB W., MARSCHNER S.: Manifold exploration: A markov chain monte carlo technique for rendering scenes with difficult specular transport. *ACM Trans. Graph. 31*, 4 (July 2012), 58:1–58:13. 6

[Kaj86] KAJIYA J. T.: The rendering equation. *Computer Graphics 20* (1986), 143–150. 2

[Kel97] KELLER A.: Instant radiosity. In *Proc. SIGGRAPH 97* (New York, NY, USA, 1997), Annual Conference Series, ACM Press/Addison-Wesley Publishing Co., pp. 49–56. 1

[KMA*15] KETTUNEN M., MANZI M., AITTALA M., LEHTINEN J., DURAND F., ZWICKER M.: Gradient-domain path tracing. *ACM Trans. Graph. 34*, 4 (July 2015), 123:1–123:13. 1

[LW93] LAFORTUNE E. P., WILLEMS Y. D.: Bi-directional path tracing. In *Compugraphics '93* (1993), pp. 145–153. 1

[LW95] LAFORTUNE E. P., WILLEMS Y. D.: A 5d tree to reduce the variance of monte carlo ray tracing. In *Rendering Techniques '95 (Proc. of the 6th Eurographics Workshop on Rendering)* (1995), pp. 11–20. 2

[McG11] MCGUIRE M.: Computer graphics archive, August 2011. `http://graphics.cs.williams.edu/data`. 10

[MH97] MCCOOL M. D., HARWOOD P. K.: Probability trees. In *Proceedings of the Graphics Interface 1997 Conference, May 21-23, 1997, Kelowna, BC, Canada* (May 1997), pp. 37–46. 3

[ND12] NOVÁK J., DACHSBACHER C.: Rasterized bounding volume hierarchies. *Computer Graphics Forum 31*, 2 (2012), 403–412. 2

[SL06] STEINHURST J., LASTRA A.: Global Importance Sampling of Glossy Surfaces Using the Photon Map. *Symposium on Interactive Ray Tracing 0* (2006), 133–138. 2

[TCE05] TALBOT J. F., CLINE D., EGBERT P.: Importance resampling for global illumination. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2005), EGSR '05, Eurographics Association, pp. 139–146. 9

[VG94] VEACH E., GUIBAS L. J.: Bidirectional estimators for light transport. In *EG Rendering Workshop* (1994). 6

[VG97] VEACH E., GUIBAS L. J.: Metropolis light transport. In *97* (New York, NY, USA, 1997), Annual Conference Series, ACM Press/Addison-Wesley Publishing Co., pp. 65–76. 1

[VK16] VORBA J., KŘIVÁNEK J.: Adjoint-driven russian roulette and splitting in light transport simulation. *ACM TOG 35*, 4 (jul 2016). 1, 2, 6

[VKv*14] VORBA J., KARLÍK O., ŠIK M., RITSCHEL T., KŘIVÁNEK J.: On-line learning of parametric mixture models for light transport simulation. *ACM TOG 33*, 4 (Aug. 2014). 1, 2, 3, 5, 6, 7, 8, 9

Appendix A: Proof of $\hat{V}_k$ Being Convex

We want to show, that $\hat{V}_k$ is convex, i.e.

$$2\hat{V}_k \leq \hat{V}_{k+1} + \hat{V}_{k-1}\,. \tag{6}$$

Proof Let $\forall k : B_k > 0$, $\hat{B}_k > 0$, $\tau_k > 0$. We write Eq. 6 terms of τ as

$$\frac{2\tau_k}{\hat{B}_k} \leq \frac{\tau_{k+1}}{\hat{B}_{k+1}} + \frac{\tau_{k-1}}{\hat{B}_{k-1}}\,, \tag{7}$$

$$2\tau_k \leq \tau_{k+1}\frac{\hat{B}_k}{\hat{B}_{k+1}} + \tau_{k-1}\frac{\hat{B}_k}{\hat{B}_{k-1}}\,. \tag{8}$$

We use the convexity of τ_k to obtain the tighter inequality

$$2\tau_k \leq (2\tau_k - \tau_{k-1})\frac{\hat{B}_k}{\hat{B}_{k+1}} + \tau_{k-1}\frac{\hat{B}_k}{\hat{B}_{k-1}}\,, \tag{9}$$

$$2\tau_k \leq 2\tau_k\frac{\hat{B}_k}{\hat{B}_{k+1}} + \tau_{k-1}\left(\frac{\hat{B}_k}{\hat{B}_{k-1}} - \frac{\hat{B}_k}{\hat{B}_{k+1}}\right)\,. \tag{10}$$

We further tighten the inequality by using $\tau_k < \tau_{k-1}$ as

$$2\tau_k \leq 2\tau_k\frac{\hat{B}_k}{\hat{B}_{k+1}} + \tau_k\left(\frac{\hat{B}_k}{\hat{B}_{k-1}} - \frac{\hat{B}_k}{\hat{B}_{k+1}}\right)\,, \tag{11}$$

$$2 \leq \frac{\hat{B}_k}{\hat{B}_{k+1}} + \frac{\hat{B}_k}{\hat{B}_{k-1}} = \frac{\hat{B}_k}{\hat{B}_k - B_k} + \frac{\hat{B}_k}{\hat{B}_k + B_{k-1}}\,. \tag{12}$$

Through re-arrangement and simplification we obtain

$$B_{k-1} - B_k - \frac{B_{k-1}B_k}{\hat{B}_k} \leq 0\,, \tag{13}$$

which holds, because the sequence B_k is monotonically increasing and always positive (see Section 3). □