10 "Practical Path Guiding" in Production

Thomas Müller, *NVIDIA*§

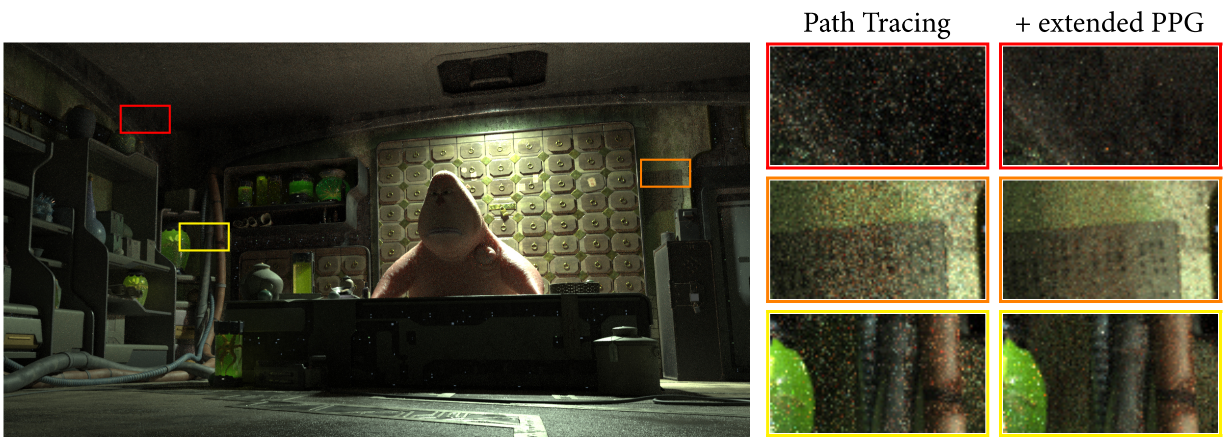

Figure 11: The "Practical Path Guiding" algorithm with our extensions reduces noise of indirect illumination in a scene lit and rendered with production assets. ©Walt Disney Animation Studios

10.1 Introduction

What is a "Practical" Path-Guiding Algorithm? One of the main selling points of path guiding is its ability to simulate complex light transport using a simple, low-overhead, unidirectional algorithm. This is very benefitial for production environments, because unidirectional tracing is preferred over the bidirectional methods that are typically required to simulate similar levels of complexity. However, production rendering does not always need the ability to handle complicated illumination. Many scenes encountered in production actually have quite simple light transport—in part because artists are accustomed to coping with the limitations of unidirectional tracing—which challenges the practicality of path guiding: a truly practical production-path-guiding algorithm must not only robustly handle difficult illumination, but it must also remain performant under simple illumination without introducing additional noise.

How Practical is "Practical Path Guiding"? The "Practical Path Guiding" (PPG) algorithm [Müller et al., 2017] was conceived with the goal of addressing several formerly existing practicality issues. It succeeded on some fronts, such as eliminating an expensive precomputation by instead learning to guide on-line during rendering from unidirectional camera paths (concurrently with Dahm and Keller [2018]), but on other fronts PPG was still limited: for example, its spatio-directional data structure—the SD-tree—had trouble adapting to local, high-frequency illumination, the algorithm discarded up to *half* of the total number of samples, and PPG blindly performed incident-radiance guiding everywhere, even in situations where this did not make sense.

Although these limitations did not prevent PPG from performing well under difficult illumination, they made its performance worse than that of unguided path tracing in simpler settings. Because of this, PPG clearly needed more work.

§The presented work was conducted while the author was employed at ETH Zürich and Disney Research.

Making "Practical Path Guiding" More Practical. We came up with three extensions to the original PPG algorithm that address some of the aforementioned limitations. These extensions are

1. an inverse-variance-weighted sample combination scheme that uses *most* samples as opposed to discarding up to half of them,
2. spatio-directional filtering for improving the quality and robustness of the SD-tree, and
3. on-line learning of the selection probability between guiding and BSDF sampling to avoid introducing extra noise in situations where incident-radiance guiding is suboptimal.

While there is still additional room for improvement—we do not outperform unguided path tracing in *all* situations—our extensions helped in making PPG a useful tool for movie production within Disney's Hyperion renderer [Burley et al., 2018].

In the following, we will briefly describe the original PPG algorithm and then introduce each of our extensions in detail. After that, we will discuss a number of subtle but important implementation details and conclude by stating remaining issues and open problems.

10.2 The PPG Algorithm

The ultimate goal of path guiding is to importance sample the scattering integral

$$L_\mathrm{s}(\mathbf{x}, \omega_\mathrm{o}) = \int_\mathcal{S} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) f_s(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o}) \cos \gamma_\mathrm{i} \, \mathrm{d}\omega_\mathrm{i} \,, \tag{3}$$

where L_i is the spatio-directional incident radiance, f_s is the BSDF, and $\cos \gamma_\mathrm{i}$ is the foreshortening term. The PPG algorithm learns an approximation of incident radiance, denoted $\tilde{L}_\mathrm{i}$, and subsequently samples ω_i proportional to this approximation. Since this approach neglects learning the BSDF and the foreshortening term, it is paramount to combine it with BSDF sampling via multiple importance sampling (MIS) [Veach and Guibas, 1995] to attain low variance.

At PPG's core is an iterative learning scheme: the rendering process is split into M distinct passes, called "iterations", each learning a new, more powerful incident-radiance approximation $\tilde{L}_\mathrm{i}^k$ while being guided by the approximation that was learned in the preceeding iteration $\tilde{L}_\mathrm{i}^{k-1}$. This scheme not only allows guiding early on in the rendering process—as soon as the first iteration finishes—but it also avoids an expensive precomputation that would stand in the way of fast artist workflows.

10.2.1 The SD-tree for Storing Incident Radiance

PPG approximates incident radiance as a piecewise-constant function with adaptive resolution that is represented by a spatio-directional tree (SD-tree; see Figure 12). The SD-tree has an upper part—a binary tree that subdivides the spatial domain—and a lower part—a quadtree that subdivides the directional domain. This split into two parts captures the notion that the spatial and directional domain are used in fundamentally different ways for guiding: *given* a position in space the goal is to *sample* a direction. In other words, the guiding probability distribution is *conditioned* on space and *normalized* across directions.

Recording Radiance. During path tracing, the current SD-tree $\tilde{L}_\mathrm{i}^k$ is populated with radiance estimates from paths that are guided by $\tilde{L}_\mathrm{i}^{k-1}$ as follows. Whenever a path is completed (e.g. via next-event estimation), the radiance arriving at each of its vertices v is recorded in the SD-tree leaf that contains the vertex position $\mathbf{x}_v$ and direction ω_v. This amounts to nearest-neighbor filtering of the samples as they are splatted into the tree. One of our extensions (Section 10.4) replaces this nearest-neighbor filter with a more sophisticated volume-overlap filter that significantly improves the approximation quality of the SD-tree.

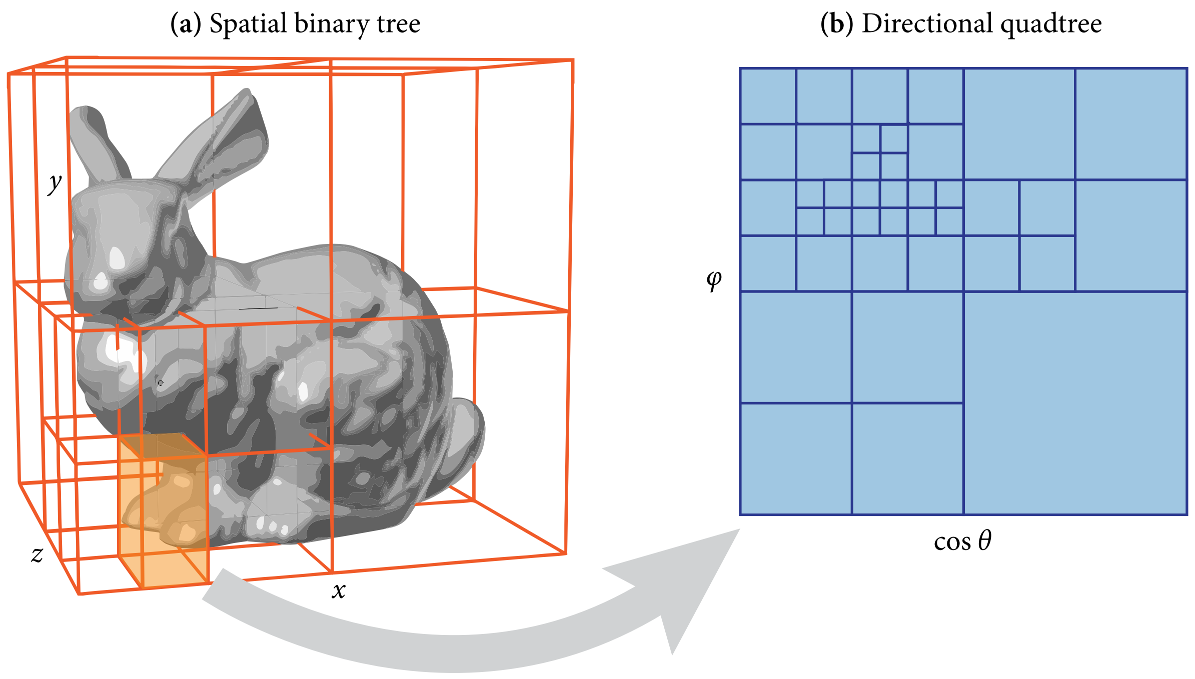

Figure 12: The SD-tree subdivides space using an adaptive binary tree (**a**) that alternates between splitting the x, y, and z axes in half. Each leaf of the spatial tree contains a quadtree (**b**), which approximates the incident radiance as an adaptively refined piecewise-constant function in cylindrical world-space coordinates $(\cos\theta, \varphi)$ with $\cos\theta = \omega^z$ and $\varphi = \operatorname{atan2}(\omega^y, \omega^x)$. Illustration from Müller et al. [2017] with additional labels.

Sampling. When using the SD-tree from the previous iteration $\tilde{L}_\mathrm{i}^{k-1}$ for importance sampling of ω_i at an intersected location $\mathbf{x}$, its spatial component is traversed to the leaf containing $\mathbf{x}$; then the quadtree contained in the leaf is sampled using hierarchical sample warping [McCool and Harwood, 1997]. We provide pseudocode for sampling and for evaluating the corresponding PDF in Algorithm 2.

As mentioned before, to achieve low variance it is essential to combine SD-tree sampling with BSDF sampling via MIS: PPG probabilistically selects either SD-tree or BSDF sampling with a fixed selection probability of 50%. To reduce variance even further, in Section 10.5, we replace the 50% probability with a spatially varying value that is *optimized* jointly with the SD-tree during rendering based on the optimization of Müller et al. [2018].

Subdivision. At the end of each iteration, the newly populated SD-tree $\tilde{L}_\mathrm{i}^{k}$ is used to determine the subdivision of the next SD-tree $\tilde{L}_\mathrm{i}^{k+1}$ in such a way that all *spatial* leaf nodes encounter a roughly equal number of path vertices during rendering, and such that all *directional* leaf nodes of a given quadtree contain roughly the same amount of flux. We omit additional details of this subdivision process, because they are not relevant for the remainder of the text; we refer to Müller et al. [2017] and our public implementation for more details.

10.2.2 Iterative Learning and Rendering

The rendering algorithm is divided into M iterations, each of which produces an image I^k and an SD-tree $\tilde{L}_\mathrm{i}^{k}$. Since the early iterations are guided only by coarse, inaccurate SD-trees, they typically result in much noisier images than the later iterations. Naïvely averaging the images produced by each iteration could thus lead to more noise in the final image than simply discarding the images from early iterations. This motivates PPG's iteration scheme: rather than allocating an equal number of samples to each iteration, PPG *doubles* the number of samples of each iteration. The *total* number of samples is thus $N = 1 + 2 + \cdots + 2^{M-1} = 2^M - 1$, which is approximately twice the number of samples of the final iteration. The

Algorithm 2: Sampling and PDF evaluation of the directional quadtree.

```
function sampleQuadtree(node)
    if isLeaf(node) then
        return cylindricalToDirection(uniformRandomPositionIn(node))
    else
        childNode ← sampleChildByEnergy(node)
        return sampleQuadtree(childNode)
function pdfQuadtree(node, $\omega_\mathrm{i}$)
    if isLeaf(node) then
        return $1/4\pi$
    else
        childNode ← getChild(node, directionToCylindrical($\omega_\mathrm{i}$))
        $\beta$ ← 4 · getFlux(childNode) / getFlux(node)
        return $\beta$ · pdfQuadtree(childNode, $\omega_\mathrm{i}$)
```

algorithm then discards the images produced by all but the final iteration, preventing initial high-variance samples from increasing overall noise while limiting the "wasted" computation to at most half of the total sample count.[5]

While this scheme increases robustness in the worst cases, it is clearly undesirable to throw away half of the computation when the initial iterations already resulted in small variance. Müller et al. [2017] address this problem by adaptively assigning a larger proportion of samples to the final iteration, thereby discarding a smaller proportion, but the heuristic they use for this purpose is brittle in practice. In the next section, we explain an alterative: a principled weighting scheme for combining almost all samples in a robust manner.

10.3 Extension 1: Inverse-Variance-Weighted Sample Combination

Our first extension allows robustly combining the majority of samples by using an inverse-variance-based weighting scheme that acts on the images $I^1, I^2, \dots, I^M$ produced in each iteration.

Theoretically Optimal Sample Combination. Because the images $I^1, I^2, \dots, I^M$ are independent random variables, the *optimal* per-pixel combination weights—i.e. those that result in the least combined variance—are proportional to the *inverse pixel variance* [Graybill and Deal, 1959]

$$I(p) = \frac{1}{\sum_{i=1}^{M} w^k(p)} \sum_{i=1}^{M} w^k(p) I^k(p)\,, \tag{4}$$

$$w^k(p) = \frac{1}{\mathbb{V}[I^k(p)]}\,, \tag{5}$$

where $I(p)$ is the combined pixel value of pixel p.

Robust Sample Combination. Unfortunately, it is unrealistic to assume accurate knowledge of the variance of each individual pixel, which makes the aforementioned optimal scheme difficult to apply. While it *is* possible to estimate the pixel variance from the samples themselves, such estimates often are highly inaccurate and would therefore lead to suboptimal weights if they were used. But even worse: estimating the variance from the samples introduces *correlation* between the image I^k and the variance-estimate-based weights $w^k(p)$, ultimately leading to bias.

We want to reduce this bias and increase stability while retaining the core idea behind the inverse-variance sample combination. To this end, we propose to average the per-pixel variance estimates over

[5]The algorithm can also handle non-power-of-two sample counts by *lengthening* the last iteration.

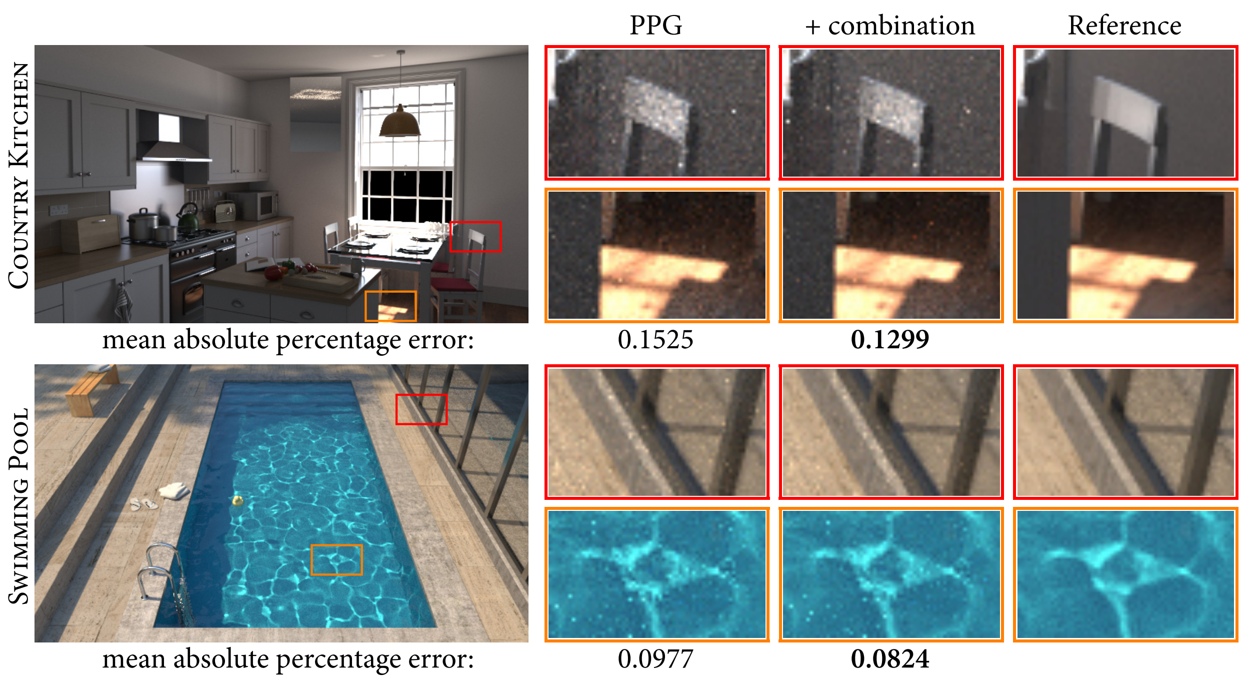

Figure 13: Combining the images produced by each iteration weighted by their mean pixel variance reduces the number of “wasted” samples and thereby slightly subdues noise. We quantify the noise as *mean absolute percentage error* averaged over each image.

the image plane

$$\bar{w}^k = \sum_p w^k(p)\,, \tag{6}$$

leading to the less biased but also less optimal whole-image weighting scheme

$$I = \frac{1}{\sum_{i=1}^{M} \bar{w}^k} \sum_{i=1}^{M} \bar{w}^k I^k\,. \tag{7}$$

In our experiments, we were unable to actually observe any bias using this scheme, both visually and numerically. Furthermore, the resulting images consistently had less noise compared to images produced by the original PPG algorithm; we show two examples in Figure 13.

Importantly, the proposed weighting scheme is *consistent*. As rendering progresses, the variance estimate is built using an increasing number of samples, approaching the true variance of the pixel value. This higher accuracy of the variance estimate leads to vanishing correlation between the weights and the pixel values—because the true variance does not depend on any specific samples—and thereby to vanishing bias.

Lastly, to further improve robustness, we only combine the *last* 4 images. While this amounts to always discarding slightly fewer than the first 6.25% of the samples, these initial samples are the noisiest and can—in the most difficult cases—otherwise cause unstable weights.

10.4 Extension 2: Spatio-directional Splatting into the SD-Tree

While testing the SD-tree on a large number of scenes, we observed distracting artifacts under spatio-directionally narrow illumination. In Figure 14, we created a contrived situation that demonstrates the problem: we render a Cornell Box that is illuminated by a tiny quad light with *disabled* next-event estimation. Although PPG dramatically improves overall variance over an unguided path tracer (left), residual noise manifests non-uniformly along the spatial structure of the SD-tree (middle).

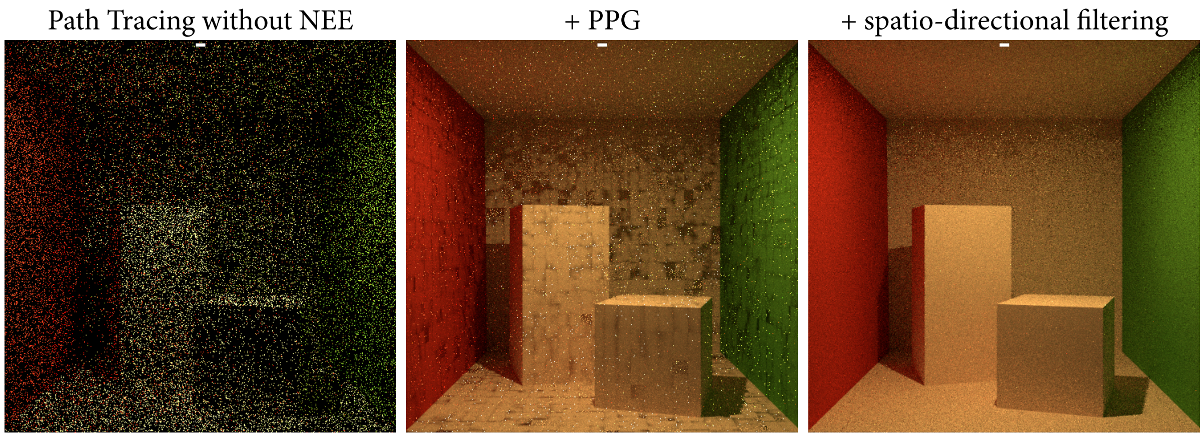

Figure 14: Improved handling of narrow illumination by spatio-directional filtering of radiance estimates that are splatted into the SD-tree. We illustrate a contrived situation: a small light source that is *not* sampled with next-event estimation (NEE). Although PPG (middle) dramatically improves overall variance compared to an unguided path tracer (left), residual noise is distributed non-uniformly along the spatial structure of the SD-tree, which is visually unpleasing. Spatio-directional filtering (right) addresses the structured noise by distributing it evenly across the scene and at the same time reducing it overall.

These artifacts are primarily caused by three problems of the SD-tree: first, the spatial subdivision scheme assumes that splitting a node in half causes the newly created nodes to receive a roughly equal number of samples, which is not the case when path vertices are distributed anisotropically, e.g. along geometric edges. Second, even *if* the path vertices were evenly distributed across spatial leaf nodes, their sample variance is disregarded by the algorithm. And lastly, during path tracing, incident radiance estimates at path vertices are recorded within their *nearest* spatial leaves, which causes the learned approximation to be better in the center of nodes than at their edges, leading to visible darkening at leaf boundaries.

All the above problems result in undesired *non-uniform* learning of the SD-tree, giving rise to the artifacts in Figure 14 (middle). A popular and effective approach to mitigate such non-uniformity is filtering. We therefore introduce a spatio-directional filter to the splatting of radiance estimates into the SD-tree: instead of recording radiance estimates $\langle L_\mathrm{i} \rangle$ from vertices v within the leaf node that contains the vertex position $\mathbf{x}_v$ and direction ω_v, we record $\langle L_\mathrm{i} \rangle$ within those leaf nodes that fall into a *spatial neighborhood* around $\mathbf{x}_v$ and a *directional neighborhood* around ω_v.

More concretely, given $\langle L_\mathrm{i} \rangle$ at a vertex v, we begin by traversing the spatial tree to obtain the *spatial footprint* and corresponding volume V of the leaf containing $\mathbf{x}_v$; this footprint determines the filter radius and thereby the size of the spatial neighborhood. We then traverse the spatial tree again, this time visiting each node that has non-zero volume overlap with the spatial footprint from before, centered around $\mathbf{x}_v$. For each spatial leaf with non-zero volume overlap V_o that we find this way, we record the radiance estimate weighted by the fraction of overlapped volume $\langle L_\mathrm{i} \rangle \cdot V_o / V$.

Directional filtering works analogously, only that we perform area-based filtering over the cylindrical domain as opposed to spatial filtering over space.

Stochastic Filtering.

When implemented as described above, the filtering operation comes with significant computational cost. This cost can be mostly avoided at the expense of slightly worse quality by replacing the deterministic filtering that traverses entire sub-trees with stochastic filtering that traverses only towards a single leaf: after obtaining the spatial footprint, the position $\mathbf{x}_v$ is *jittered* (i.e. positioned uniformly at random) within the footprint's volume—again, centered around $\mathbf{x}_v$—and then recorded in the SD-tree as done in the original algorithm without weighting the radiance by V_o/V.

To find the sweet-spot between quality and computational cost, it is important to consider the compounding effect of enabling deterministic spatial *and* directional filtering at the same time: since deterministic spatial filtering increases the number of visited quadtrees (from one to many) and directional

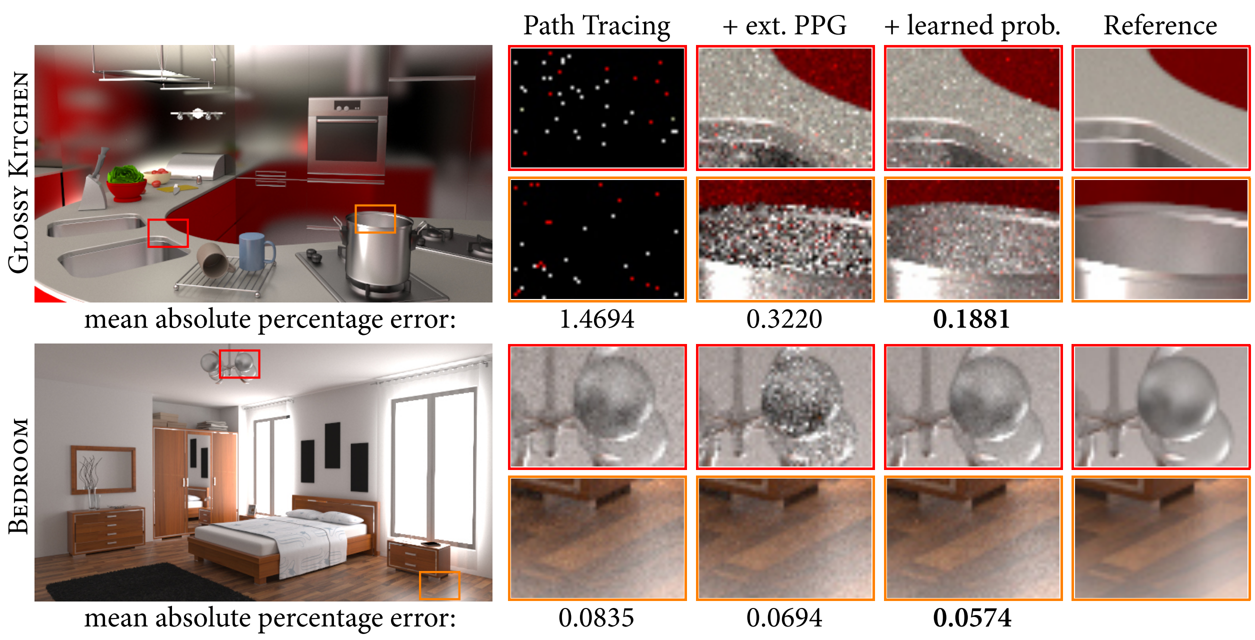

Figure 15: Learning the selection probability of multiple importance sampling on top of our other extensions reduces noise in difficult glossy light transport (top) and prevents path guiding from producing much worse results in places that are simple to render using BSDF sampling alone (bottom). We use 1024 samples per pixel and we quantify noise using *mean absolute percentage error* averaged over each image.

filtering increases the number of visited leaf quads *for each visited quadtree* (again, from one to many), the total computational cost is roughly proportional to the *product* of the spatial filtering overhead and the directional filtering overhead. It follows, that negating the overhead via stochastic filtering on only *one* of the two filtering operations provides *most* of the possible speedup.

Additionally, since spatial filtering operates on 3 dimensions whereas directional filtering operates only on 2, the overhead caused by spatial filtering is bigger.

Because of these two reasons, we perform spatial filtering stochastically and use deterministic filtering only in the directional domain. The computational overhead of this combined approach over vanilla PPG is around 20% in a Mitsuba-rendered Cornell Box and below 10% in Hyperion-rendered production scenes. The smaller overhead in Hyperion is a consequence of more computation being devoted to rendering significantly more complex scenes.

Subdivision Criterion. The original PPG algorithm subdivides spatial leaf nodes when they receive more than $c = 12000 \cdot 2^{k/2}$ samples. Due to the increased robustness from filtering, we are able to reduce this number to $c = 4000 \cdot 2^{k/2}$, thereby leading to a finer subdivision and therefore not only a more robust but also a more fine-grained fit; see Figure 14.

10.5 Extension 3: Optimization of MIS Selection Probability

Our third extension aims at on-line learning of the multiple-importance-sampling selection probability between SD-tree and BSDF sampling such that variance is minimized. This addresses the problem of overusing the SD-tree on glossy surfaces and when it approximates incident radiance poorly. We demonstrate the benefits of this extension in Figure 15.

Objective Definition. Recall that in PPG, radiance sampling is combined with BSDF sampling using the *one-sample* MIS model [Veach and Guibas, 1995]. Mathematically, this amounts to sampling according to a linear combination of the SD-tree PDF $q \propto \tilde{L}_\mathrm{i}$ and the BSDF PDF $p_{f_s} \propto f_s$

$$\hat{q}(\omega_\mathrm{i}|\mathbf{x}, \omega_\mathrm{o}; \alpha) = \alpha \cdot p_{f_s}(\omega_\mathrm{i}|\mathbf{x}, \omega_\mathrm{o}) + \big(1 - \alpha\big) \cdot q(\omega_\mathrm{i}|\mathbf{x}) \,, \tag{8}$$

where α is the *selection probability* that determines how frequently each of the two strategies is sampled from. A value of 0 amounts to always sampling from the SD-tree, whereas a value of 1 corresponds to always using BSDF sampling.

Many radiance-based guiding approaches use a fixed value of $\alpha = 0.5$ [Müller et al., 2017, Vorba et al., 2014], whereas others that learn the product are typically more aggressive: Herholz et al. [2018, 2016] use $\alpha = 0.1$. However, it would be better to let α vary spatially to account for the fact that BSDF sampling may be more suitable in some scene regions, whereas guiding may be more appropriate in others.

With this motivation in mind, our objective is to replace the fixed selection probability α with a *learned* function $\alpha(\mathbf{x})$. To derive a learning algorithm for $\alpha(\mathbf{x})$, we begin by formalizing the desired form that we would like the combined PDF $\hat{q}$ to take.

Zero variance can only be attained when sampling proportional to the product of incident radiance and the BSDF, i.e. according to the "ideal" PDF $p(\omega_\mathrm{i}|\mathbf{x}, \omega_\mathrm{o}) \propto L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) f_s(\mathbf{x}, \omega_\mathrm{i}) \cos \gamma_\mathrm{i}$. Although it is impossible to *perfectly* match the ideal PDF simply by varying $\alpha(\mathbf{x})$, our goal is to optimize $\alpha(\mathbf{x})$ such that the combined PDF $\hat{q}$ at least approximates the ideal PDF p as closely as possible.

Since the goal of approximating $p(\omega_\mathrm{i}|\mathbf{x}, \omega_\mathrm{o})$ with $\hat{q}(\omega_\mathrm{i}|\mathbf{x}, \omega_\mathrm{o})$ can be accomplished independently for any spatio-directional coordinate $(\mathbf{x}, \omega_\mathrm{o})$, we will omit $\mathbf{x}$ and ω_o from the following derivations. The objective is then to approximate $p(\omega_\mathrm{i})$ with $\hat{q}(\omega_\mathrm{i}; \alpha) = \alpha \cdot p_{f_s}(\omega_\mathrm{i}) + (1 - \alpha) \cdot q(\omega_\mathrm{i})$ by varying α. We can formalize this as an optimization problem: we try to minimize a suitably chosen *distance* $D(p \,\|\, \hat{q}; \alpha)$ between the PDFs p and $\hat{q}$, referred to as "objective function". The optimal selection probability is then the one that minimizes the objective function

$$\hat{\alpha} = \operatorname*{arg\,min}_{\alpha \in [0,1]} D(p \,\|\, \hat{q}; \alpha) \,. \tag{9}$$

Choice of Objective Function. Ideally, we would set the objective function D to the Monte Carlo variance, such that the above equation directly corresponds to minimizing variance. Such an optimization *is* actually feasible, but it is numerically unstable which leads to worse results compared to alternative objectives [Müller et al., 2018]. We therefore use another, more numerically stable function to capture the difference between p and $\hat{q}$: the Kullback-Leibler (KL) divergence.

The KL divergence between the ideal PDF p and the learned PDF $\hat{q}$ is defined as

$$D_\mathrm{KL}(p \,\|\, \hat{q}; \alpha) = \int_\mathcal{S} p(\omega_\mathrm{i}) \log \frac{p(\omega_\mathrm{i})}{\hat{q}(\omega_\mathrm{i}; \alpha)} \, \mathrm{d}\omega_\mathrm{i} \,. \tag{10}$$

It is a good surrogate for the variance, because—like the variance—it attains a value of 0 if and only if $p = \hat{q}$ and because it approaches infinity as $\hat{q}$ undersamples the integrand, i.e. when $\hat{q}(\omega_\mathrm{i}; \alpha)$ approaches zero for directions where $p(\omega_\mathrm{i}) > 0$.

10.5.1 Minimizing the Kullback-Leibler Divergence

Müller et al. [2018] showed that the selection probability α can be optimized such that the KL divergence $D_\mathrm{KL}(p \,\|\, \hat{q}; \alpha)$ is minimized when α is the output of a neural network. In this section, we use an adapted approach that does the same *without* involving a neural network.

We minimize the KL divergence by setting its gradient to zero. Ideally, we would do this in closed form, which is unfortunately not possible because of the difficult integral. Such difficult minimization problems are well studied in the field of machine learning and are often addressed using algorithms that build on "stochastic gradient descent". One of the most successful of such approaches is "Adam" [Kingma and Ba, 2014], which is remarkably simple to implement. For this reason, and for another reason that we will mention shortly, we use Adam to optimize the selection probability α.

The Adam algorithm takes stochastic estimates of the "loss gradient" (in our case: the gradient of the KL divergence) as input and is guaranteed to converge to a (locally) optimal value if these estimates are *unbiased*. Fortunately, it *is* possible to obtain unbiased KL-divergence gradient estimates: Müller et al. [2018] showed, that the gradient of the KL divergence can be written as an expectation over samples drawn from an arbitrary PDF q_s.[6]

$$\nabla_\alpha D_{\mathrm{KL}}(p \,\|\, \hat{q}; \alpha) = \mathop{\mathbb{E}}_{\omega_i \sim q_s}\left[-\frac{p(\omega_i)}{q_s(\omega_i)} \nabla_\alpha \log \hat{q}(\omega_i; \alpha) \right], \tag{11}$$

which yields the desired unbiased gradient as the inner part of the above expectation for directions ω_i drawn from q_s.

Evaluating this unbiased gradient is difficult, because we cannot evaluate the ideal PDF $p(\omega_i)$ in closed form. We can, however, replace the ideal PDF with an unnormalized unbiased estimate of it: the product of incident radiance, the BSDF, and the foreshortening term. This replacement introduces a constant factor but otherwise leaves the unbiasedness of the gradient intact because of the linearity of expectations. Slightly abusing notation and denoting $\langle X \rangle$ as an *unbiased estimate* of X, we have

$$c \cdot \langle \nabla_\alpha D_{\mathrm{KL}}(p \,\|\, \hat{q}; \alpha) \rangle = -\frac{\langle L_i(\omega_i) \rangle f_s(\omega_i) \cos \gamma_i}{q_s(\omega_i)} \nabla_\alpha \log \hat{q}(\omega_i; \alpha). \tag{12}$$

Here lies the second reason behind our choice of using the Adam optimizer: Adam automatically compensates for the constant factor c, so we ignore the factor and express the gradient only up to proportionality.

We can further expand the gradient estimate by applying the chain rule, finally leading to an expression that can be evaluated within a renderer

$$\begin{aligned} \langle \nabla_\alpha D_{\mathrm{KL}}(p \,\|\, \hat{q}; \alpha) \rangle &\propto -\frac{\langle L_i(\omega_i) \rangle f_s(\omega_i) \cos \gamma_i}{q_s(\omega_i)} \nabla_\alpha \log \hat{q}(\omega_i; \alpha) \\ &= -\frac{\langle L_i(\omega_i) \rangle f_s(\omega_i) \cos \gamma_i}{q_s(\omega_i)} \frac{\nabla_\alpha \hat{q}(\omega_i; \alpha)}{\hat{q}(\omega_i; \alpha)} \\ &= -\frac{\langle L_i(\omega_i) \rangle f_s(\omega_i) \cos \gamma_i}{q_s(\omega_i)} \frac{p_{f_s}(\omega_i) - q(\omega_i)}{\hat{q}(\omega_i; \alpha)}. \end{aligned} \tag{13}$$

It may be tempting to directly feed this gradient into the Adam algorithm to optimize the selection probability α, but there is a remaining problem we must solve before we can do so: the selection probability is only valid in the range $[0, 1]$, but the above optimization is unaware of this constraint. The optimization could therefore produce invalid probabilities outside of $[0, 1]$. We *enforce* probabilities that lie within $[0, 1]$ by modeling α in terms of an auxiliary "latent" variable $\theta \in \mathbb{R}$ that can take *any* real value without constraint, using the logistic sigmoid function $\sigma : \mathbb{R} \rightarrow [0, 1]$

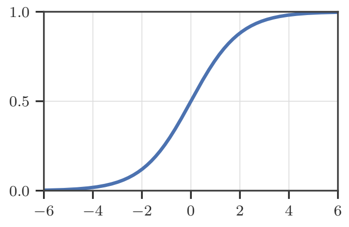

Figure 16: Logistic sigmoid.

$$\alpha(\theta) \equiv \sigma(\theta) = \frac{1}{1 + e^{-\theta}}. \tag{14}$$

Since the sigmoid enforces valid probabilities α, we can optimize θ using Adam without worrying about the range of α. To perform this optimization, we need an unbiased gradient estimate w.r.t. θ instead of α, which can be obtained using the chain rule

$$\begin{aligned} \langle \nabla_\theta D_{\mathrm{KL}}(p \,\|\, \hat{q}; \alpha) \rangle &= \langle \nabla_\alpha D_{\mathrm{KL}}(p \,\|\, \hat{q}; \alpha) \rangle \cdot \nabla_\theta \, \alpha(\theta) \\ &= \langle \nabla_\alpha D_{\mathrm{KL}}(p \,\|\, \hat{q}; \alpha) \rangle \cdot \alpha(\theta)\big(1 - \alpha(\theta)\big) \\ &\propto -\frac{\langle L_i(\omega_i) \rangle f_s(\omega_i) \cos \gamma_i}{q_s(\omega_i)} \frac{p_{f_s}(\omega_i) - q(\omega_i)}{\hat{q}(\omega_i; \alpha)} \alpha(\theta)\big(1 - \alpha(\theta)\big). \end{aligned} \tag{15}$$

[6]In practice, q_s is chosen to equal $\hat{q}$, but implementation details of multithreaded optimization necessitate their distinction.

Algorithm 3: Optimization of the MIS selection probability. One MIS optimization step is executed whenever a radiance estimate is splatted into a spatial leaf of the SD-tree.

```
class SpatialLeaf()
    $t, m, v, \theta \leftarrow 0$                                                    // Initialize state
    $\beta_1 \leftarrow 0.9,\ \beta_2 \leftarrow 0.999,\ \varepsilon \leftarrow 10^{-8}$, learningRate $\leftarrow 0.01$, regularization $\leftarrow 0.01$    // Hyperparameters
    function adamStep($\nabla_\theta$)
        $t \leftarrow t + 1$                                                          // Increment iteration counter
        $l \leftarrow$ learningRate $\cdot \sqrt{1 - \beta_2^t}/(1 - \beta_1^t)$      // Compute de-biased learning rate
        $m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla_\theta$            // Update first moment
        $v \leftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot \nabla_\theta \cdot \nabla_\theta$    // Update second moment
        $\theta \leftarrow \theta - l \cdot m/(\sqrt{v} + \varepsilon)$               // Update parameter
    function misOptimizationStep($\mathbf{x}$, $\omega_\mathrm{i}$, $\omega_\mathrm{o}$, radianceEstimate, samplePdf)
        productEstimate $\leftarrow$ radianceEstimate $\cdot f_s(\omega_\mathrm{i}) \cos \gamma_\mathrm{i}$
        bsdfPdf $\leftarrow p_{f_s}(\omega_\mathrm{i}|\omega_\mathrm{o}, \mathbf{x})$
        learnedPdf $\leftarrow$ isDiscrete($f_s(\omega_\mathrm{i})$) ? 0 : $q(\omega_\mathrm{i}|\mathbf{x})$
        spin lock (this)                                    // Ensure $\theta$ is only optimized by one thread at a time
            $\alpha \leftarrow$ selectionProbability()
            combinedPdf $\leftarrow \alpha \cdot$ bsdfPdf $+ (1 - \alpha) \cdot$ learnedPdf                  // Equation 8
            $\nabla_\alpha \leftarrow -$productEstimate $\cdot$ (bsdfPdf $-$ learnedPdf)/(samplePdf $\cdot$ combinedPdf)  // Equation 13
            $\nabla_\theta \leftarrow \nabla_\alpha \cdot \alpha(1 - \alpha)$                                   // Chain rule
            regGradient $\leftarrow$ regularization $\cdot\ \theta$                // L2 regularization to avoid sigmoid saturation
            adamStep($\nabla_\theta +$ regGradient)
    function selectionProbability()                         // Called by the path tracer to use the learned probability
        return $1/(1 + e^{-\theta})$                                                 // Sigmoid as in Equation 14
```

10.5.2 Integration of MIS Optimization into PPG

In this section, we describe how we implemented the selection-probability optimization using Adam within the PPG algorithm.

Spatial Discretization. Our goal is to optimize the selection probability α spatially and jointly with the learning of the SD-tree. To this end, we utilize the spatial subdivision of the SD-tree: in each spatial leaf node, we not only store a directional quadtree, but also the latent variable θ controlling the selection probability α. During rendering, whenever we splat a radiance estimate into a spatial leaf node, we not only record it within the corresponding quadtree, but we also execute an Adam optimization step.

Multi-Threading. The gradient computation and Adam optimization are not thread-safe, so mutual exclusivity must be guaranteed when two threads attempt to perform an optimization step within the same spatial leaf concurrenty. Fortunately, such thread collisions are rare, because the spatial binary tree has a resolution that approximately matches the spatial density of path vertices. We are therefore able to use an inexpensive spin lock, with which we observed near-perfect linear performance scaling up to 48 threads, which is the maximum number we could test on.

Regularization. Recall, that we express the selection probability as the sigmoid of a latent variable $\alpha(\theta) \equiv \sigma(\theta)$. This sigmoid levels off as θ approaches positive or negative infinity (see Figure 16), i.e. its gradient approaches zero. Unfortunately, small gradients make gradient-based optimizers such as Adam less effective, which is known in machine learning as the "vanishing-gradient" problem.

We limit the vanishing-gradient problem by introducing L^2 regularization to our latent variable θ, which "encourages" the optimization to prefer values of θ that are closer to zero, i.e. values of α that are closer to 0.5. This amounts to modifying our objective function (the KL divergence) to include an additive term $\lambda\theta^2$, where λ controls the strength of the regularization. The modified gradient estimate is then

$$\langle \nabla_\theta (D_{\mathrm{KL}}(p \,\|\, \hat{q}; \alpha) + \lambda\theta^2) \rangle = \langle \nabla_\theta D_{\mathrm{KL}}(p \,\|\, \hat{q}; \alpha) \rangle + \nabla_\theta \lambda\theta^2 = \langle \nabla_\theta D_{\mathrm{KL}}(p \,\|\, \hat{q}; \alpha \rangle + 2\lambda\theta \,. \tag{16}$$

We pick a weak regularization $\lambda = 0.005$, allowing the optimization to produce selection probabilities close to 0 or 1 (e.g. 0.01 or 0.99), but not close enough that the optimization suffers.

Discrete-Smooth Hybrid BSDFs. The above optimization can handle BSDFs that are hybrids of smooth and discrete reflectances. Abusing mathematics a little bit, whenever a discrete component is sampled, the smooth SD-tree PDF $q(\omega_\mathrm{i})$ must be treated as zero[7] (see Algorithm 3), similar to how smooth BSDF components are usually treated as zero in such cases.

Code. We provide pseudocode of the *full* MIS optimization—including Adam—in Algorithm 3. The pseudocode also includes the implementation details that we discussed in the preceeding paragraphs. For an actual implementation within the Mitsuba renderer, we refer to our public code at https://github.com/Tom94/practical-path-guiding.

10.6 Adjoint-Driven Russian Roulette and Splitting

Russian roulette is an important building block for making almost any path tracer efficient. Unfortunately, path-throughput-based russian roulette typically has detrimental effects on guided path tracers, because the throughput does not account for incident illumination (the adjoint) encountered upon path completion. While it is possible to avoid such detrimental effects by entirely disabling russian roulette, this comes with potentially significant extra computational cost.

Adjoint-driven russian roulette (and splitting) [Vorba and Křivánek, 2016] solves this issue by incorporating an estimate of the adjoint—the learned incident-radiance approximation $\tilde{L}_\mathrm{i}$—into the russian-roulette decision. We use this scheme because of its increased efficiency and because of its easy implementation within a path-guiding algorithm that learns incident radiance. For details on adjoint-driven Russian roulette and splitting we refer to Sec. 7.7.

10.7 Miscellaneous Implementation Details

In the following, we discuss miscellaneous implementation details of the PPG algorithm that are easy to overlook but greatly improve its effectiveness.

Next-Event Estimation. Although path guiding is able to sample complex *general* illumination, it often performs worse than next-event estimation (NEE) on *direct* illumination, especially when NEE uses an importance cache such as the one used in Hyperion [Burley et al., 2018]. Because of this, we enable NEE and do not train the SD-tree with direct illumination from light sources that are sampled by NEE. The SD-tree is therefore trained using *all* indirect illumination and only direct illumination from light sources that are *not* sampled by NEE (e.g. emissive volumes).

Parameterization of Directional Distribution. There are a number of possible ways to parameterize the directional guiding distribution. We use *world-space-aligned cylindrical coordinates* for two practical reasons. First, world-space alignment—as opposed to surface alignment—allows the usage of the same distribution of incident radiance when there is high-frequency normal variation. Second, we use cylindrical coordinates—as opposed to spherical coordinates—due to their area-preserving correspondence to the surface of the solid sphere.

[7]Most guiding methods—including PPG—do not learn discrete components. However, if a discrete component *was* learned, then it should *not* be treated as zero in this situation.

Rapid Adaptation of the SD-tree to Large Scenes. It is common in production to encounter vast scenes of which only a small portion is visible to the camera. For a path-guiding data structure to be effective in such situations, it must rapidly adapt to the distribution of paths being traced. Although the SD-tree *does* continually adapt to the distribution of paths, the amount of adaptation *within each iteration* is relatively small. To address this problem, we perform a small number of 1-sample-per-pixel iterations (e.g. 8) at the beginning of rendering. This has the effect of "initializing" the subdivision of the SD-tree to match the camera frustum before it is used for path guiding in the remaining rendering process.

10.8 Remaining Issues and Future Work

In this section, we mention several known issues of PPG that we were not able to address or did not have time to investigate yet. These remaining issues provide interesting directions of future work, both within the context of PPG as well as in the pursuit of replacing PPG (or components of it) with better alternatives.

Subdivision of the SD-tree. In Section 10.4, we proposed to combat non-uniform learning of the SD-tree using spatio-directional filtering. While this approach is effective at reducing the symptoms of the problem—i.e. a bad, non-uniform incident-radiance approximation—it does not tackle its fundamental causes, which are rooted in PPG's suboptimal subdivision scheme. In the future it is worth investigating alternative data structures such as the BSP-tree proposed by Herholz et al. [2019] (described in detail in Section 11.5.1) or neural networks that internally learn a spatio-directional representation [Müller et al., 2018].

Product Guiding. The SD-tree can only learn to guide according to incident radiance, which limits its practicality compared to alternative approaches that can guide according to the full product [Herholz et al., 2018, 2016, 2019, Müller et al., 2018]. Although it *is* possible to sample from the product of the SD-tree and a BSDF represented by Haar wavelets [Clarberg et al., 2005] or spherical harmonics [Jarosz et al., 2009], such representations are difficult to obtain for rich, parametric BSDFs such as the Disney BSDF [Burley, 2012] which is used in the Hyperion renderer.

Temporal Guiding. The guiding of motion-blur effects is difficult with PPG, because the motion can cause large incident-radiance variation that is not captured by the SD-tree. As Müller et al. [2017] already suggested, it may be possible to simply add the temportal dimension to the spatial binary tree (making it a 4-D spatio-temporal binary tree), but this is yet to be tested.

Volumetric Guiding. PPG performs poorly when used to guide volumetric path tracing. This is because of a number of reasons such as the lack of product guiding with the phase function and the lack of guided distance sampling. Progress towards volumetric path guiding has been made by Herholz et al. [2019] using a spatial data structure similar to our binary tree in combination with a directional parametric mixture model. Sebastian Herholz will discuss more details in Section 11.5.

Simple Light Transport. Lastly, there is still room for improving PPG under easy-to-render illumination. Even though our extensions achieve better results than vanilla PPG in difficult scenes (Figure 15, top) and surpass unguided path tracing in some simple settings (e.g. in Figure 15, bottom), there are remaining cases where unguided path tracing is superior. These are caused mostly by the 10–20% computational overhead of the SD-tree and the discarding of all but the last 4 PPG iterations (around 6.25% of all samples). It is therefore important future work to further improve upon the algorithm and to investigate the possibility of effective guiding of *direct* illumination that complements next-event estimation for better overall efficiency.

10.9 Conclusion

Our goal was to obtain a path-guiding algorithm that not only accelerates rendering of difficult light transport, but also performs no worse than simple unidirectional path tracing under simple illumination.

To this end, we reviewed the "Practical Path Guiding" algorithm [Müller et al., 2017] and introduced three extensions that helped us improve its effectiveness and robustness. First, we avoided discarding large fractions of the samples by weighting them approximately proportional to their inverse variance. Next, we improved the quality of the learned SD-tree by spatio-directional filtering of splatted radiance estimates, and lastly, we showed how the MIS selection probability can be optimized to increase the overall efficiency of guiding.

After that, we discussed a number of important details to consider when implementing PPG with our extensions in a production environment. These include the combination of PPG with next-event estimation, the use of adjoint-driven russian roulette, and various aspects of effectively utilizing SD-trees.

Unfortunately, all of this was not quite enough to make PPG strictly better than an unguided path tracer under simple illumination. In some scenes—for example directly lit outdoor environments—unidirectional path tracing sometimes outperformed our extended PPG by a small margin that is mostly caused by the 10-20% computational overhead of the SD-tree and our discarding of all but the last 4 iterations (around 6.25% of all samples). Nevertheless, our extensions significantly shrunk the gap between both approaches while improving PPG in difficult situations beyond what it was capable of before.

Additionally, our extensions are not inherently limited to PPG. With minor modification, they are also compatible with alternative guiding schemes [Vorba et al., 2014], other incident-radiance representations [Herholz et al., 2019, Müller et al., 2018, Vorba et al., 2014], and even different paradigms, such as guiding in primary-sample space [Guo et al., 2018, Müller et al., 2018, Zheng and Zwicker, 2018] or path space [Simon et al., 2018].

We are excited about the ongoing adoption of path guiding in movie production and look forward to further progress that will be made in the field.

Acknowledgments

We thank Disney's Hyperion team for having been invaluable in integrating PPG into Hyperion and developing the presented extensions; in particular, we thank Laura Lediaev, Joseph Schutte, Peter Kutz, Ralf Habel, Yining Karl Li, and David Adler. We also thank thank Jan Novák for help with the SD-tree illustration and Brent Burley, Dan Teece, Fabrice Rousselle, Jan Novák, Joseph Schutte, and Sebastian Herholz for proofreading and helpful feedback. Lastly, we thank the following entities and people for providing scenes and models that appear in our figures: Benedikt Bitterli [2016], Ondřej Karlík (Swimming Pool), Jay-Artist (Country Kitchen), and SlykDrako (Bedroom).

References

Benedikt Bitterli. 2016. Rendering resources. https://benedikt-bitterli.me/resources/.

Brent Burley. 2012. Physically-based shading at Disney. In *SIGGRAPH Course Notes. Practical physically-based shading in film and game production.*, Vol. 2012. 1–7.

Brent Burley, David Adler, Matt Jen-Yuan Chiang, Hank Driskill, Ralf Habel, Patrick Kelly, Peter Kutz, Yining Karl Li, and Daniel Teece. 2018. The Design and Evolution of Disney's Hyperion Renderer. *ACM Trans. on Graphics* 37, 3, Article 33 (Aug. 2018). https://doi.org/10.1145/3182159

Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. 2005. Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 24, 3 (aug 2005), 1166–1175.

Ken Dahm and Alexander Keller. 2018. Learning Light Transport the Reinforced Way. In *Monte Carlo and Quasi-Monte Carlo Methods. MCQMC 2016. Proceedings in Mathematics & Statistics*, A. Owen and P. Glynn (Eds.), Vol. 241. Springer, 181–195.

Franklin A. Graybill and R. B. Deal. 1959. Combining Unbiased Estimators. *Biometrics* 15, 4 (1959), 543–550.

Jerry Jinfeng Guo, Pablo Bauszat, Jacco Bikker, and Elmar Eisemann. 2018. Primary Sample Space Path Guiding. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*, Wenzel Jakob and Toshiya Hachisuka (Eds.). The Eurographics Association.

Sebastian Herholz, Oskar Elek, Jens Schindel, Jaroslav Křivánek, and Hendrik P. A. Lensch. 2018. A Unified Manifold Framework for Efficient BRDF Sampling based on Parametric Mixture Models. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations*. The Eurographics Association.

Sebastian Herholz, Oskar Elek, Jiří Vorba, Hendrik Lensch, and Jaroslav Křivánek. 2016. Product Importance Sampling for Light Transport Path Guiding. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering)* 35, 4 (2016), 67–77.

Sebastian Herholz, Yangyang Zhao, Oskar Elek, Derek Nowrouzezahrai, Hendrik Lensch, and Jaroslav Křivánek. 2019. Zero-Variance Based Sampling for Volume Path Guiding. *ACM Trans. on Graphics (conditionally accepted)* (2019).

Wojciech Jarosz, Nathan A. Carr, and Henrik Wann Jensen. 2009. Importance Sampling Spherical Harmonics. *Computer Graphics Forum (Proc. of Eurographics)* 28, 2 (April 2009), 577–586.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

Michael D. McCool and Peter K. Harwood. 1997. Probability trees. In *Proceedings of the Graphics Interface 1997 Conference, May 21-23, 1997, Kelowna, BC, Canada*. 37–46.

Thomas Müller, Markus Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering)* 36, 4 (June 2017), 91–100.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2018. Neural Importance Sampling. *CoRR* abs/1808.03856 (2018). arXiv:1808.03856 http://arxiv.org/abs/1808.03856

Florian Simon, Alisa Jung, Johannes Hanika, and Carsten Dachsbacher. 2018. Selective guided sampling with complete light transport paths. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 37, 6 (Dec. 2018). https://doi.org/10.1145/3272127.3275030

Eric Veach and Leonidas Guibas. 1995. Optimally combining sampling techniques for Monte Carlo rendering. *Proc. SIGGRAPH* (1995), 419–428.

Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-line Learning of Parametric Mixture Models for Light Transport Simulation. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 33, 4 (Aug. 2014), 101:1–101:11.

Jiří Vorba and Jaroslav Křivánek. 2016. Adjoint-Driven Russian Roulette and Splitting in Light Transport Simulation. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 35, 4 (July 2016), 1–11.

Quan Zheng and Matthias Zwicker. 2018. Learning to Importance Sample in Primary Sample Space. *arXiv:1808.07840* (Sept. 2018).