Online path sampling control with progressive spatio-temporal filtering

Jacopo Pantaleoni*
NVIDIA

Abstract
This work introduces progressive spatio-temporal filtering, an efficient method to build all-frequency approximations to the light transport distribution into a scene by filtering individual samples produced by an underlying path sampler, using online, iterative algorithms and data-structures that exploit both the spatial and temporal coherence of the approximated light field. Unlike previous approaches, the proposed method is both more efficient, due to its use of an iterative temporal feedback loop that massively improves convergence to a noise-free approximant, and more flexible, due to its introduction of a spatio-directional hashing representation that allows to encode directional variations like those due to glossy reflections. We then introduce four different methods to employ the resulting approximations to control the underlying path sampler and/or modify its associated estimator, greatly reducing its variance and enhancing its robustness to complex lighting scenarios. The core algorithms are highly scalable and low-overhead, requiring only minor modifications to an existing path tracer.

CR Categories: I.3.2 [Graphics Systems C.2.1, C.2.4, C.3]: Stand-alone systems—; I.3.7 [Three-Dimensional Graphics and Realism]: Color,shading,shadowing, and texture—Raytracing;

Keywords: global illumination, light transport simulation, Markov Chain Monte Carlo

1 Introduction
Light transport simulation can be an arbitrarily challenging problem, due to the fact it requires to numerically estimate millions of pixel integrals whose infinite dimensional inte-

* e-mail: jpantaleoni@nvidia.com
grands may have arbitrarily high variance. Forty years of research have produced a vast plethora of methods to increase the efficiency of this complex estimation problem, mostly based on variants of Monte Carlo integration methods, often tailored to specific scenarios. The vast majority of these propose different strategies for path sampling, the core operation required to numerically sample the pixel integrals. In this category fall many general purpose methods, like bidirectional path tracing and its variants [Veitch 1997], MCMC techniques like Metropolis Light Transport and its descendents [Veitch and Guibas 1997; Kellem et al. 2002; Pantaleoni 2017; Bitterli et al. 2017], as well as more ad-hoc methods such as many-lights sampling, manifold exploration [Jakob and Marschner 2012; Kaplanyan et al. 2014; Hanika et al. 2015], and many others.

Despite the sheer amount of research, the most popular basic method for path sampling remains path tracing. [Kajiya 1986], often augmented by specific techniques to sample particular light transport events. The reason why the most basic technique is also the most successful is to be found both in its simplicity, which leads to higher execution efficiency on modern computing architectures, and to its very high per-sample efficiency on average content, that does not feature extremely complex visibility or rare events such as those due to specular-diffuse-specular transport. In order to improve path tracing, a recent spur of research has focused the attention on path guiding, with the idea of learning custom importance samplers on-the-fly to better guide the samples towards the more important regions of path space [Vorba et al. 2014; Herholz et al. 2016; Mueller et al. 2017; Dahm and Keller 2017; Muller et al. 2019]. All of these methods can be seen as forms of online learning of different spatio-directional approximations to the underlying light field (for example based on Gaussian mixture distributions embedded in a spatial k-d tree in the approaches of Vorba et al. 2014 and Herholz et al. 2016; quad-trees in the approach of Mueller et al. 2017; simple tabulations in the approach of Dahm et al. 2017; and neural networks in neural importance sampling [Muller et al. 2019].

A more limited form of online approximation of the input light field can be found in historical approaches that cached irradiance at specific points in the scene [Chaos Group 2008; Keller et al. 2014]. The path space filtering algorithm by Keller et al. 2014 constructed an approximation of the input irradiance arriving at a given vertex along a path (in the original paper, the first diffuse vertex as seen from the camera). The approximation was built by augmenting a path tracer with a spatial data structure used to average the contributions from all paths whose first diffuse vertex happen to be close-by in space. The averaged contributions would then be used as a replacement for the original unbiased estimator at the specified vertex, resulting in a biased (although potentially consistent) estimator of the diffuse portion of the rendering equation. A similar strategy is described in the documentation of v-ray’s Light Cache [Chaos Group 2008]. The path space filtering algorithm has been later extended to perform this on-the-fly using fast spatial hashing as a spatial data structure [Binder et al. 2018].

Our work work is divided into two main parts. In the first, we introduce a general purpose method that can be used to build similar approximations of full rank incoming and outgoing light fields, as well as their products with local brdfs, with greatly increased convergence speed. Our methods are based on a rigorous discretization of the involved scalar light fields, and the application of efficient transport simulation methods derived from a novel combination of path-tracing and radiosity-style finite element solvers. This part also introduces a novel spatio-directional hashing scheme allowing to compactly encode the resulting high-dimensional fields. In the second, we study many different uses of the resulting approximants to improve the underlying path sampling estimators, not restricted to simple path guiding. In particular, we will show that there are simpler and more efficient unbiased estimators than those used for path guiding that can be obtained by using the obtained light-field approximation as a control variate, and that by introducing some bias we can bridge the gap between unbiased estimation and biased techniques that directly use the approximation as a lighting cache. Using control variates for path tracing had already been attempted by Lafortune and Willems [2016], by employing a 5d tree based approximation of radiance built on-the-fly. Our work shares many similarities with theirs, although we have built it on a more formal framework and faster algorithms for computing such approximations, and focused on novel algorithms and data structures geared towards a real-time implementation.

While the theoretical contributions we are introducing have general validity, our work explicitly targets real-time settings which have not been previously addressed by other path guiding methods. Contrary to previous approaches, all of our methods are designed to be efficiently mapped to GPUs, exploit all the available parallelism and be effective even at the low sample counts typically available in real-time ray-tracing scenarios.

2 Progressive Spatio-Temporal Filtering

In order to describe our key algorithm, let’s first consider a hypothetical discretization \( \mathcal{H} = \{ h : h \in \{1, \cdots, N\} \times \{1, \cdots, M\} \} \) of the 5-dimensional light field, seen as the tensor product of \( N \) spatial basis functions and \( M \) directional basis functions. Our key insight is that we can see the construction of our desired approximation as a finite-element solver for the rendering equation using our discretization \( \mathcal{H} \) as a basis. In order to build an efficient solver, we can draw a parallel to and get inspiration from so-called radiosity methods. In fact, while radiosity solvers have been soon discarded in favour of the more flexible Monte Carlo methods, which proved to allow for much greater realism due to their capacity to model arbitrarily high frequency effects without the restrictions imposed by a finite-element basis, many methods developed for radiosity were nearly optimal for the finite-element setting. In this setting, we can view the solution of the discretized rendering equation as the solution of:

\[
L_o = TL_o + L_e \quad (1)
\]

where \( T \) is the transport operator [Veitch 1997], and \( L_e \) denotes projection on the basis functions. Our approach to solving it efficiently is a hybrid between progressive radiosity and Monte Carlo path tracing.

2.1 Basic path tracing solver

Since we want to obtain an online learning algorithm that reuses the samples we generate by the underlying path sampler to build the finite element approximation, we start by considering a path-tracing based solver of the discretized equation.

The first thing to notice is that each generated path will touch as many finite-elements as it has vertices: as a consequence, we can use each sample path to update all the finite-elements it lands upon.
In the following, we will assume we may have several path sampling techniques, each associated with a sampling probability \( p \) and a corresponding multiple importance sampling weight \( w \) (where we omit the dependence on the technique for improved readability). Given a sample path \( \mathbf{x} \) with \( n + 1 \) vertices \( \mathbf{x} = x_0 x_2 \ldots x_n \), and assuming its probability and multiple importance sampling weight decompose into products of the form:

\[
p(\mathbf{x}) = p(x_0) \cdot p(x_1|x_0) \cdots p(x_n|x_{n-1})
\]
\[
w(\mathbf{x}) = w(x_0) \cdot w(x_1|x_0) \cdots w(x_n|x_{n-1})
\]

we can update the solution at the finite elements touched by vertex \( x_i \) using an unbiased estimator provided by the tail of the path \( x_i \cdots x_n \). Let’s denote with \( \hat{L}_o(x_i, \omega) \) the quantity:

\[
\begin{align*}
\hat{L}_o(x_i, \omega_i^o) &= L_o(x_i, \omega_i^o) \\
&= \sum_{i \leq j < n} L_o(x_j, \omega_j^o) \prod_k f_k(\omega_k, \omega_k^o) G(x_k, x_{k+1}) \frac{w(x_{k+1}|x_k)}{p(x_{k+1}|x_k)},
\end{align*}
\]

where \( G \) denotes the geometric throughput between two vertices, \( f_k \) denotes the bidirectional scattering distribution function at vertex \( k \), and \( \omega_k \) and \( \omega_k^o \) denote the incoming and outgoing directions at vertex \( k \). A single-sample unbiased estimator of our approximation \( \hat{L}_{o,h} = \hat{L}_o, b_h \) could now theoretically be obtained as:

\[
\hat{L}_{o,h} \approx \frac{\hat{L}_o(x_i, \omega_i^o) b_h(x_i, \omega_i^o)}{P_T(x_i, \omega_i^o)}.
\]

where \( P_T(x, \omega) \) is the total throughput measure probability of sampling a path which lands on the point \( x \) from direction \( -\omega \). Unfortunately, as we show in the Appendix, this factor is itself a marginal probability, whose computation would involve integrating over all of path space.

However, if the spatial and angular support of the basis functions is small, and we can neglect variations inside the support, a slightly biased but practical density estimator can be obtained by shooting \( N \) paths \( \mathbf{x}_p \), with \( p \in 1 \cdots N \) and keeping track of a weighted sum of the number of vertices \( c_h \) that fall on each basis function \( b_h \):

\[
c_h = \sum_{p=1}^N \sum_i b_h(x_{p,i}, \omega_{p,i}^o) \cdot w(x_{p,i}|x_{p,i-1})
\]

and using the formula:

\[
\hat{L}_{o,h} \approx \frac{\sum_{p=1}^N \hat{L}_o(x_{p,i}, \omega_{p,i}^o) b_h(x_{p,i}, \omega_{p,i}^o) w(x_{p,i}|x_{p,i-1})}{c_h}.
\]

Notice how this is similar in principle to what was proposed in path space filtering [Keller et al. 2014], except it is extended to update an approximation of the light field at all path vertices, using arbitrary basis functions that span both the spatial and the directional domain, and using multiple importance sampling.

### 2.2 Progressive solver

In the previous section we saw how the sample paths obtained by a regular path sampler can be used to estimate the projection over the outgoing light field on a finite element basis. The resulting method is unbiased, but has the same convergence speed as ordinary Monte Carlo path tracing. Much faster convergence can obtained looking at solutions inspired by the radiosity literature. Recall that the solution of the rendering equation can be written as:

\[
L_o = L_e + TL_e + T^2L_e + T^3L_e + \cdots
\]

In other words, the equilibrium radiance distribution is the sum of emitted radiance transported once, twice, three times, and so on. We can exploit this fact by replacing our unbiased estimator of equation (4) with an estimator that reuses the current projection estimate at each basis function. This is similar to the application of Jacobi iteration for the solution of a linear system, or so called progressive radiosity algorithms.

Let’s call \( \hat{L}_{o,h}^{new} \) our current estimate for the outgoing radiance projected over the basis \( b_h \), and let’s redefine our estimator as:

\[
\hat{L}_{o,h}^{new} \approx \sum_{p=1}^N \hat{L}_o^{new}(x_{p,i}, \omega_{p,i}^o) \frac{b_h(x_{p,i}, \omega_{p,i}^o) w(x_{p,i}|x_{p,i-1})}{c_h}
\]

with:

\[
\hat{L}_o^{new}(x_i, \omega_i^o) = L_e(x_i, \omega_i^o) + \hat{L}_{o,h}^{corr}(x_{i+1}, \omega_{i+1}^o f_i(\omega_i^o) G(x_i, x_{i+1}) \frac{w(x_{i+1}|x_i)}{p(x_{i+1}|x_i)}
\]

Notice that even though this definition applies the transport operator only once, since it transports our current estimate of \( \hat{L}_{o,h}^{corr} \), its iterative application will lead to the full solution of equation (1) and (7).

In practice, in order to apply this technique, we can cast paths in waves, for example by sampling one path per pixel per frame, and performing the updates of equation (8) and (9) using the approximation corresponding to the previous frame.

### 2.3 Progressive hierarchical solver

The last step for obtaining even faster convergence is the use of a hierarchical solver. In order to do this, we have to...
assume a hierarchy of finite elements $H_I$ where $I$ represents the level of detail. Once we have that, we can simply replace the use of $\bar{L}^{\text{curr}}_{o,h,i}$ in equation (9) with a suitably selected hierarchy level $\bar{L}^{\text{curr}}_{o,h,i}$. In our implementation, we choose the appropriate level by tracking approximate path footprints, using the heuristic described by Bekkers et al. [2003]. A conceptual visualization of the final algorithm is sketched in Figure 2 whereas pseudo-code for the basic path-tracing skeleton is given in Algorithm 1 [1]. The left side of Figure 1 shows an approximation built using the above algorithm.

2.4 Temporal averaging

In order to accommodate for dynamic scene updates, we employ a non-linear temporal averaging scheme that allows to give more weight to new samples than older ones. In practice we do this by keeping track of two counters $c_h^{\text{old}}$ and $c_h^{\text{new}}$ for each basis function, corresponding to the cumulative counters up to the old frame, and new counters for the new frame only. When we apply equation (8), we then use the following weighted average:

$$\bar{L}^{\text{new}}_{o,h,i} = (1 - \alpha^{\text{new}}) \cdot \bar{L}^{\text{old}}_{o,h,i} + \alpha^{\text{new}} \cdot \sum_{p=1}^{N} \bar{L}^{\text{new}}_i (x_{p,i}, \omega_{p,i}^o) \cdot b_h(x_{p,i}, \omega_{p,i}) \cdot w(x_{p,i}) / c_h^{\text{new}}$$

(10)

where the blending coefficient $\alpha^{\text{new}}$ is computed as:

$$\alpha^{\text{new}} = \max \left( \sqrt{c_h^{\text{old}} / (c_h^{\text{old}} + c_h^{\text{new}})}, T_{\text{max}}^{-1} \right)$$

(11)

and $T_{\text{max}}$ is a user-defined constant useful to limit the size of the temporal window. Notice that the presence of the square root inside the blending coefficient makes the weighted average a hybrid between linear accumulation (which would be obtained without the square root), and exponential weighting, which would be obtained with a constant $\alpha^{\text{new}}$. The counters $c_h^{\text{old}}$ can additionally be zeroed either locally, and on demand, according to custom heuristics designed to detect local changes, or globally, in the presence of large structural changes to geometry or illumination.

The reason why such a non-linear, non-exponential hybrid is desirable is to be found in the fact that, in a static setting, linear averaging corresponds to calculating the optimal Monte Carlo sample average, whereas exponential averaging gives exponentially diminishing weight to older samples, and hence discards information at an exponential rate. We found the ability to limit the loss of temporal information to be very useful, especially as some latency in the changes in indirect illumination is typically not very noticeable.

2.5 Incoming radiance and other fields

So far we have discussed representations that span the incoming radiance field only, without directly encoding the incoming radiance distribution. Some of the applications we will describe in the following sections require approximations of the product of incoming radiance and the local brdf. Extending the representation to also account for the incoming radiance distribution would require minor modifications to the update equations. It is enough to recall that the outgoing and incoming radiance are related by:

$$L_i = G L_o$$

where $G$ is the propagation operator [Veach 1997]. The basic update equations for the incoming radiance would then be:

$$\bar{L}^{\text{new}}_{i,h} \approx \sum_{p=1}^{N} \bar{L}^{\text{new}}_i (x_{p,i}, \omega_{p,i}^o) \cdot b_h(x_{p,i}, \omega_{p,i}) \cdot w(x_{p,i}) / c_h^{\text{new}}$$

(12)

$$\bar{L}^{\text{new}}_i (x_i, \omega_i^o) = \bar{L}^{\text{curr}}_{o,h,i} (x_{i+1}, \omega_i^o) G(x_i, x_{i+1})$$

(13)

If we instead want to encode the product of incoming radiance and the local brdf, which we denote by $\bar{f} L_i$, we get:

$$\bar{f} L^{\text{new}}_i (x_i, \omega_i^o) = \sum_{p=1}^{N} \bar{f} L^{\text{new}}_i (x_{p,i}, \omega_{p,i}^o) / c_h^{\text{new}}$$

(14)

$$\bar{f} L^{\text{new}}_i (x_i, \omega_i^o, \omega_i^b) = \bar{L}^{\text{curr}}_{o,h,i} (x_{i+1}, \omega_i^o)$$

$$\cdot f_i (\omega_i^o, \omega_i^b) G(x_i, x_{i+1})$$

$$\cdot w(x_{i+1}, \omega_i^b) / p(x_{i+1}, \omega_i^b)$$

(15)

Yet another field that might be useful to approximate is $\bar{L}_{o,e} = L_o - L_e$ (corresponding to all radiance transported at least once). In order to learn the corresponding projection $\bar{L}_{o,e}$ it is enough to omit the $L_e$ term from equation (9). Pseudo-code for tracking this field is given in Algorithm 2.

Finally, while so far we have assumed that all available sampling techniques might be used to update these fields, for path guiding applications it might in fact be beneficial to exclude some - for example because we would like to focus guided samples to areas that are not already covered by other techniques such as next-event estimation [Karlik et al. 2019]. This would result in a down-weighted field, including only one or some of the multiple importance sampling components (and hence with weights not summing up to one).

2.6 Basis functions and data structures: spatio-directional hashing

The choice of basis functions and data structures is orthogonal to the methods described in this work. However, in our implementation we have chosen a representation based on the efficient 5d spatial hashing scheme described by Binder et al. [2018], with one crucial modification: for the outgoing radiance fields $L_o$ and $L_{o,e}$, instead of using the surface normal at each path vertex to create a 5d hash as proposed in the original paper, we employ the outgoing direction - thus matching the representation needed to encode our 5d light field. Similarly, for the incoming radiance field $\bar{L}_i$ we employ a 5d hash over the position and the incoming direction.

Thus, our basis functions are essentially the product of incoming radiance and the local brdf, which we denote by $\bar{L}^{\text{curr}}_{o,h,i}$. If we instead want to encode the product of incoming radiance and the local brdf, which we denote by $\bar{f} L_i$, we get:

$$\bar{f} L^{\text{new}}_i (x_i, \omega_i^o) = \sum_{p=1}^{N} \bar{f} L^{\text{new}}_i (x_{p,i}, \omega_{p,i}^o) / c_h^{\text{new}}$$

(14)

$$\bar{f} L^{\text{new}}_i (x_i, \omega_i^o, \omega_i^b) = \bar{L}^{\text{curr}}_{o,h,i} (x_{i+1}, \omega_i^o)$$

$$\cdot f_i (\omega_i^o, \omega_i^b) G(x_i, x_{i+1})$$

$$\cdot w(x_{i+1}, \omega_i^b) / p(x_{i+1}, \omega_i^b)$$

(15)
the incoming and outgoing directions, such a data-structure would not be practical for importance sampling due to its sparsity in the incoming directional domain. Hence, we have also experimented combining spatial hashing for the 3d spatial component and the 2d outgoing direction with three different dense representations for the 2d incoming direction domain: regular grids, k-d trees, and spherical gaussian mixture models (GMMs). While regular grids represent an orthonormal basis, making their update straightforward (even on parallel architectures), k-d trees and gaussian mixtures do not, and require custom update methods.

2.6.1 Spherical GMMs

Vorba et al. [2014] proposed using spherical Gaussian mixture models to learn the incoming radiance distribution, using an algorithm dubbed stepwise-EM. We use the same algorithm to learn the product of incoming radiance and the BRDF slice tied to the given cone of output directions associated to a spatial hash cell.

The original algorithm was designed to be executed independently for each CPU thread, each working on a different GMM. This execution model scales poorly to massively parallel GPU hardware: partly because of lack of parallelism (typically the number of cells/GMMs to update is measured in thousands to tens of thousands per frame), partly because each GMM requires significant amounts of memory (with 6 floats per component, plus 8 more for the sufficient statistics), which cannot easily fit in on-chip memory and hence would require heavy longer-latency memory traffic.

Hence we developed two different parallel adaptations. The first and simplest is a plain SIMT adaptation that uses one SIMT lane per component. Recall that the original algorithm is divided into two broad phases, the E- and the M-steps, which are executed, respectively, for every new sample and every N samples. Focusing on a single GMM, given the sufficient statistics $u_i$ at step i, represented as a matrix with $C$ rows and 8 columns, where $C$ is the number of mixture components, and a new sample $s_i = (s_x, s_y)$ with weight $w_i$, the E-step can be written as:

$$ u_i[c] = a_i \cdot u_{i-1}[c] + b_i \cdot v_i $$

where:

$$ v_i = (1, s_x, s_y, s_z \cdot s_y, s_y \cdot s_x, s_z \cdot s_y, s_x \cdot s_y, s_z) / \gamma_c, 1 / b_i) $$

$$ a_i = (1 - i^{-\alpha}) $$

$$ b_i = i^{-\alpha} \cdot w_i \cdot \gamma_c $$

and $\gamma_c$ is the responsibility of the c-th component of the current GMM for the point $s_i$:

$$ \gamma_c = \frac{p_c(s_i)}{\sum_j p_j(s_i)} $$

Notice how the update equations [16] can be trivially parallelized across components; the only computation that needs special care is the calculation of $\gamma_c$, which requires all threads to participate in the computation of the denominator, essentially computing a parallel reduction.

This one-thread-per-component mapping is significantly faster than the trivial one-thread-per-GMM mapping, especially as it allows each thread to only keep one component worth of data in registers. However, load-balancing might still be an issue, as some hash cells/GMMs might receive many more samples than others, requiring an uneven number of E-step iterations.

Hence, we devised an even broader parallelization strategy that uses one thread per sample. In fact, while the recurrent relation [16] seems to impose complex dependencies that do not allow parallelizing across samples, expanding the recurrent relation one can obtain:

$$ u_i[c] = b_i \cdot v_i + \sum_{1 \leq j < i} b_j \cdot v_j \cdot \prod_{j < k \leq i} (1 - k^{-\alpha}) $$

that is to say: the sufficient statistics for the i-th sample are obtained by summing up the contributions $v_j$ of all the samples preceding it, weighted by a product term of the form:

$$ g(j) = \prod_{j < k \leq i} (1 - k^{-\alpha}) $$

The main observation behind our parallelization strategy is that we can compute the logarithm of $g(j)$ with a parallel prefix-sum:

$$ \log(g(j)) = \sum_{j < k \leq i} (1 - k^{-\alpha}) $$

Hence, we first sort the samples recorded during a frame by GMM; then we proceed by calculating the coefficients $\log(g(j))$ for each sample contributing to each GMM with a segmented parallel prefix-sum (where each GMM defines a segment) and finally, we evaluate equation [19] using another segmented parallel reduction.

2.6.2 Spherical k-d trees

In order to develop scalable algorithms to efficiently access and update tens of thousands of spherical k-d trees in a massively parallel setting, we opted for a simple and compact representation, constraining each k-d tree to possess the same number of leaves, while freely adapting both their topology and the sampling probabilities assigned to their leaves using statistics collected during each frame.

Constraining each tree to possess the same number of leaves $L$, and consequently the same number of nodes $2L - 1$, allows to store the trees compactly in deterministic order and avoids random memory allocation, striking a careful balance between memory-access efficiency and representational flexibility.

Our k-d trees span a 2d domain $[0, 1]^2$, representing a uniform parameterization of the sphere. Starting from uniformly split k-d trees (essentially representing uniform grids), we update the probability of each leaf according to the sum of the contributions of the samples falling within it - so as to keep the sampling probability of each leaf proportional to the integral of the incoming radiance times the BRDF (remembering that each k-d tree is tied to a given cone of output directions).

Finally, we update each tree’s topology using the split-collapse algorithm employed by Pantaleoni for reinforcement light-cuts learning [2019]: at the end of each frame, for each k-d tree we look at the leaf with highest probability $l_{\text{max}}$, and the parent with lowest probability $p_{\text{min}}$: if the probability of the former is higher than that of the latter times a constant $T$, i.e. $P(l_{\text{max}}) > T \cdot P(p_{\text{min}})$, we split $l_{\text{max}}$ and collapse $p_{\text{min}}$. Similarly to the original implementation of split-collapse, we launch one thread block per k-d tree, and parallelize all phases of the algorithm.

During rendering, sampling from each k-d tree itself is performed using hierarchical sample warping [Clarberg et al. 2005]. For each sample, we store its primary sample space coordinate together with its MIS weighted contribution in order to update the leaf probabilities at the end of the frame.
3 Relation to previous work

Besides the similarities and the differences to path space filtering [Keller et al. 2014] [Binder et al. 2018] already mentioned at the end of section 2.1, the overall structure of our update scheme shares some similarities to that used for the Q-table creation in the reinforcement learning approach from Dahm and Keller [2017]. Here, however, the creation of all our approximators is entirely decoupled from path guiding and reinforcement learning and simply embedded in a more general approximation framework tied to arbitrary path sampling schemes, and extended to represent outgoing radiance, incoming radiance and the product of incoming radiance with the local BRDF, as opposed to a pdf (the Q-table) approximating incoming radiance only (limiting the technique from Dahm et al [2017] to only handle path guiding for diffuse materials); this new, more flexible framework is further extended to explicitly keep track of multiple importance sampling and weighted distributions.

Furthermore, we have shown how to enable faster convergence by using hierarchical basis functions (Section 2.3) and an improved, non-exponential handling of temporal averaging (Section 2.4), and we have extended the scope of practical implementations to use a larger set of basis functions that can span the complete 7d field needed to represent product distributions while using acceptable storage. Key to the latter is the use of a representation that is sparse in the outgoing direction, achieved by modifying the sparse spatial hashing scheme of Binder et al [2018] to hash over the outgoing direction as opposed to using the surface normal. Details to normal variation in each cell is instead optionally recaptured using the YCoCg spherical harmonics representation.

To our knowledge our framework is the first that can directly handle product distributions without computing the product of separate approximations of the incoming light field and the local BRDF on-the-fly, as done by Herholz et al [2016], an operation that is rather expensive and that requires BRDF representations that can easily be converted to the target basis functions (again limiting the applicability to complex material models).

Finally, we have provided novel scalable algorithms for efficiently learning GMMs and adaptive k-d trees on massively parallel architectures.

In the next sections we will further show how the resulting approximations can be used to enable a new set of estimators and control methods that go beyond simple path guiding.

4 Unbiased estimation

Once we have an approximation of the outgoing and the incoming light fields $L_o$ and $L_i$ we can directly use them to control our path sampling estimators. While previous research on path guiding methods has already covered using similar approximations for importance sampling, we will show how they can also be very effectively employed as control variates.

4.1 Importance sampling (or path guiding)

All path guiding methods are based on importance sampling from either an approximate representation of the incoming radiance distribution [Vorba et al. 2014] [Müller et al. 2017] [Dahm and Keller 2017] or a representation of the product of incoming radiance and the local BRDF [Herholz et al. 2016] that is learnt on-the-fly. Progressive spatio-temporal filtering allows to build exactly such a representation $fL_i$. In order to make the process unbiased, during each frame we importance sample $fL_i^{\text{old}}$ while updating an entirely separate approximation $fL_i^{\text{new}}$ that is only going to be used in the next frame.

In the following it will be convenient to look at local path sampling as a recursive solution of the rendering equation written in its integral form:

$$L_o(x, \omega^o) = L_o(x, \omega^i) + \int_{\Omega} L_i(x, \omega') f_{x_i}(\omega', \omega^i) \cos(\theta') d\omega'$$

(22)

Given a path vertex $x_j$ and an output direction $\omega^o_j$ we solve for $L_o(x_j, \omega^o_j)$ by sampling a direction $\omega^i_j$ according to some projected solid angle probability $p^i(\omega^i_j | x_j)$ and using the single-sample estimator:

$$L_{\omega^i \omega^o}(x_j, \omega^o_j) \approx L_i(x_j, \omega^i_j) f_{x_j}(\omega^i_j, \omega^o_j) \frac{w(\omega^i_j | x_j)}{p^i(\omega^i_j | x_j)}$$

(23)

This view makes it clear that the changes due to the approximation-based importance sampling technique are simply embedded in the vertex sampling probabilities $p(x_{j+1} | x_j) = p^- (\omega^i_j | x_j) G(x_j, x_{j+1})$, and do not change the form of the final path sampling estimators.

In practice, at each path vertex we combine sampling according to $fL_i$ with a defensive strategy based on the BSDF by means of multiple importance sampling. Similarly, other vertex sampling techniques such as next-event estimation can be easily incorporated.

4.2 Control variates

As anticipated, another estimator can be obtained using our new approximations as a control variate. Suppose we are integrating a function $g(x)$, and have another function $h(x)$ with known integral $I_h$; we can then obtain an unbiased estimator of the integral of $g$ as:

$$E[g] \approx \frac{[g(x) - \beta h(x) + \beta I_h]}{p(x)}$$

(24)

where $\beta$ is a control parameter. The function $h$ is said to be a control variate [Owen and Zhou 2000].

Again, by recalling that at each path vertex $(x_j, \omega^o_j)$ we are locally solving equation (22), we can exploit this fact by using the control variate $h = fL_i(x_j, \omega^o_j)$ with known integral $L_{\omega^i \omega^o}(x_j, \omega^o_j)$. The corresponding estimator will be:

$$L_{\omega^i \omega^o}(x_j, \omega^o_j) \approx [L_i(x_j, \omega^i_j) f_{x_j}(\omega^i_j, \omega^o_j) - \beta fL_i(x_j, \omega^o_j) + \beta L_{\omega^i \omega^o}(x_j, \omega^o_j)] \cdot \frac{w(\omega^i_j | x_j)}{p^i(\omega^i_j | x_j)}$$

(25)

As we will show in the results section, such a control variate can be surprisingly effective. In practice, we have also observed that it is sufficient and sometimes beneficial to restrict its application to the first few vertices along a path. Moreover, while for a given function $h$ optimal variance reduction would require optimizing for $\beta$, we have obtained excellent results even with fixed $\beta$ in the range [0.5, 1].

4.3 Importance sampling with control variates

As shown by He and Owen [2014] combining importance sampling and control variates can theoretically lead to even lower variance estimators. For several importance sampling techniques $p_1, \ldots, p_m$, He and Owen suggest using the following estimator:

$$E[g] \approx \frac{[g(x) - \beta^T h(x) + I_h]}{p(x)}$$

(26)
where $p_o$ is a weighted average of the probabilities $p_o = \sum x, \alpha_p h = \alpha_1(p_1(x), \cdots, p_m(x))$ and $\alpha$ and $\beta$ are multi-dimensional parameters in $\mathbb{R}^n$.

In our case, since at each vertex we use both the BSDF defensive strategy, $p_o$, and the approximation-based importance sampling strategy $p_1 = FL_i$, we have chosen to employ a simpler combined control variate of the form:

$$E[g] \approx g(x) - \beta [p_1(x) - p_o(x)]$$

(27)

where $\alpha = (\alpha_1, \alpha_2)$ is the ratio of samples allocated to each of the two strategies, and $\beta$ is again a simple scalar control. The choice of the scalar function $p_1 - p_o$ as a control variate follows the approach described by Li et al [2013], and it is equally efficient as the original estimator [26], which is singular in $\beta$. Moreover, it has the advantage of having a zero integral.

5 Biased estimation

In the previous section we have covered many alternative unbiased estimators of the rendering equation that can be built on top of our online approximation of the underlying light field. In this section we will cover yet another biased estimator that can further reduce the overall error at the cost of some bias. The basic idea is to take the control variate estimator (25) with $\beta = 1$, and reparameterize it as:

$$L_{x_j}(x_j, \omega_j) \approx \gamma \left( L_{x_j}(x_j, \omega_j) f_{x_j}(\omega_j, \omega_j') - fL_{x_j}(x_j, \omega_j') \right) + \gamma \left( L_{x_j}(x_j, \omega_j) f_{x_j}(\omega_j, \omega_j') - fL_{x_j}(x_j, \omega_j') \right)$$

(28)

We look at this estimator as a predictor-corrector model, where the $L_{x_j}(x_j, \omega_j)$ term plays the role of the predictor, and the difference:

$$\gamma \left( L_{x_j}(x_j, \omega_j) f_{x_j}(\omega_j, \omega_j') - fL_{x_j}(x_j, \omega_j') \right)$$

plays the role of the corrector. By setting $\gamma < 1$ we simply bias the solution towards the predictor. Notice that, again, we can use such an estimator at any (or even every) vertex along a path. By setting $\gamma = 0$ and using it only at the very first diffuse vertex, we can reproduce the effect of the path space filtering algorithm by Keller et al [2014], except it is extended to handle all-frequency lighting and use the more efficient progressive spatio-temporal filtering approximation.

6 Results and discussion

In order to test the various algorithms we have used a reproduction of Eric Veach’s door ajar scene, notoriously designed to be hard for light transport simulation. We have also slightly modified the scene to make use of more modern rendering features, such as layered material models that are not cheap to evaluate, and gets orders of magnitude lower variance than the raw path sampling estimator. The biased estimator improves the results even further. Notice that applying it at the second bounce helps to reduce its bias and to hide some of its associated lower frequency noise.

Table 1 shows a comparison of all our estimators from sections (3) and (4) for various sample counts, as well as pure path tracing with and without next-event estimation. In particular, we compare the PSTF-based pure path providing / importance sampling estimator (PSTF-IS), the control variate estimator (PSTF-CV), the combined importance sampling + control variate estimator (PSTF-IS-CV) and finally the biased estimator with $\gamma = 0$ applied at the second vertex along a path (PSTF-B). All the PSTF-based estimators include next-event estimation. For the 1spp images, we used a warm-up phase of 32 frames to obtain nearly converged approximations. Notice how at 1spp the control variate based estimators provide for widely improved convergence even compared to the path guiding estimator (despite being far cheaper to evaluate), and gets orders of magnitude lower variance than the raw path sampling estimator. The biased estimator improves the results even further. Notice that applying it at the second bounce helps to reduce its bias and to hide some of its associated lower frequency noise.

Figure 3 compares the performance by graphing over time the root mean square error (RMSE) of all our unbiased estimators. The comparison, obtained averaging over a variety of scenes, includes the k-d tree based importance-sampling estimator (KD-IS), the k-d tree based control variate estimator (KD-CV), the k-d tree based coupled importance-sampling plus control variate estimator (KD-IS-CV), the GMM based importance-sampling estimator...
with 4 components (GMM-4), the GMM based importance-sampling estimator with 8 components (GMM-8), and plain path-tracing with next-event estimation (PT). As our work is strictly addressing real-time rendering scenarios, we explicitly opted not to compare against previous path-guiding methods that were designed solely for offline rendering and had much larger overheads and parallelization bottlenecks. Notice how the control variate k-d tree based estimator at real-time settings (60ms) is roughly 13x faster than a pure path-tracing estimator (i.e. the path-tracing estimator has the same variance after 0.8s), and between 10x and 13x times faster than the GMM based path-guiding estimators. Path-guiding based approaches start becoming more efficient than the pure control variate estimator after 0.8-2 seconds depending on the implementation, whereas the combined importance-sampling plus control variate estimator starts providing an advantage after 0.2 seconds, although the advantage brought in by the control variate is asymptotically lost.

Figure 4 shows an equal-time comparison on a different scene where the biased estimator provides much less noisy results than simple path tracing, despite the much simpler light transport configuration. Figure 5 shows another equal-time comparison on a complex visibility scene where our biased estimator is compared to plain path-space filtering.

We also want to highlight that while other works on path guiding have overlooked this aspect, in order to make such comparisons meaningful it is absolutely critical to include next-event estimation, since this seemingly basic technique can reduce variance thousand-folds in the presence of complex lighting, and is by itself way more effective than path guiding alone.

Finally, we speculate that pure path guiding / importance sampling may not be very cost-effective when compared to a path-tracer with next-event estimation due to the fact that imperfect importance sampling techniques can lead to higher variance samples, that can only be partially mitigated by multiple importance sampling. While Owen and Zhou [Owen and Zhou 2000] have discussed safer strategies to minimize the additional variance due to locally suboptimal importance sampling techniques, these involve costly convex optimization of the multiple importance sampling weights. Moreover, it is important to notice that in many cases path guiding only helps with a tiny portion of a path: for example, in the door ajur scene it helps going through the door. However, both inside the first room and once in the back room, where multiple bounces among the walls diffuse out the overall lighting, path guiding cannot provide much help - though it still adds its associated overheads. Interestingly, Vorba et al. 2015a have recently introduced a simpler online MIS optimization technique performing stochastic gradient descent on KL divergence that seems to minimize the negative impact of suboptimal importance sampling decisions by reducing their probability. While we did not have a chance to test this yet, even this simple algorithm is bound to increase memory traffic and incur additional synchronization overhead, as it requires the use of spin-locks that are potentially expensive in a massively parallel scenario.

6.1 Future work

Besides applying the same techniques to the handling of participating media, a natural extension of this work would be to combine it with adjoint-driven Russian roulette and splitting [Vorba and Krivánek 2016], which might help further reduce variance at a low additional cost. Another venue would be to couple it with progressive photon mapping [Hachisuka et al. 2008; Hachisuka and Jensen 2009] or vertex merging techniques [Hachisuka et al. 2012]. Yet another potential area is coupling it with the latest results on improved multiple importance sampling [Ivo et al. 2019; Karlík et al. 2019; Sbert and Elvira 2019]. Finally, it would be interesting to automate the choice of the constant γ in our biased estimator to minimize total error, seen as the sum of bias and variance.

7 Appendix

Recall that a basis function $b_k \in \mathcal{H}$ is, like radiance, a function $b_k : \mathcal{R} \rightarrow \mathcal{R}$ on the ray space manifold $\mathcal{R} = \mathbb{M} \times S^2$, a product of the set of scene surfaces $\mathbb{M}$ and the sphere of directions $S^2$. The natural measure on ray space is the throughput measure:

$$d\mu(x, \omega) = dA(x) \times d\sigma^+(\omega) \quad (29)$$

In order to compute the throughput measure probability $P_T(x, \omega)$ of sampling a path landing on a point $x$ from direction $-\omega$, we will start by considering the area probability $P_A(x, y)$ of sampling a path whose last two path vertices are, respectively, first $y$ and finally $x$. This is the product of two factors: the probability of sampling a path landing on $y$, times the area probability $p(x|y)$ of sampling $x$ given $y$. The first factor can be obtained by integrating over all paths of all possible lengths $l$, so that we have:

$$P_A(x, y) = p(x|y) \cdot \sum_{l=0}^{\infty} \int p(x_0 \cdots x_l) p(y|x_l) dA(x_0) \cdots dA(x_l) \quad (30)$$

Now, if we consider the fact that the point $y$ can be deterministically obtained tracing a ray from $x$ in direction $\omega$, i.e. $y = h(x, \omega)$, we can convert between the area measure probability $P_A(x, y)$ and the throughput measure $P_T(x, \omega)$ with the formula:

$$P_T(x, \omega) = P_A(x, y) \cdot \frac{1}{G(x, y)} \quad (31)$$
| Method       | 1 spp RMSE | 32 spp RMSE | 60s RMSE |
|--------------|------------|-------------|----------|
| PT w/o NEE   | 0.625053   | 0.625053    | 0.564618 |
| PT w/ NEE    | 0.618978   | 0.495649    | 0.155520 |
| PSTF-IS      | 0.608228   | 0.424356    | 0.110337 |
| PSTF-CV      | 0.564476   | 0.367088    | 0.126974 |
| PSTF-IS-CV   | 0.536593   | 0.374096    | 0.128888 |
| PSTF-B       | 0.508294   | 0.215587    | 0.072084 |

Table 1: From top to bottom: 1. path tracing without NEE, 2. path tracing with NEE, 3. PSTF-IS, 4. PSTF-CV, 5. PSTF-IS-CV, 6. PSTF-B From left to right: 1 spp for the first column, 32 spp for the second column, and a same-time comparison for all the rows of the third column, after 60 seconds.
Figure 5: An equal-time comparison of PSF (left) and the biased PSTF estimator (right) applied at the first bounce, at 32spp, with the same pixel-sized spatial hashing. The progressive nature of PSTF allows for much quicker convergence.

ALGORITHM 1: basic PSTF path-tracing skeleton, tracking the field $L_0$

```plaintext
sample_value = 0;
sample_weight = 1;
(x_0, \omega_0, p_0^T) = sample_camera();
// set the MIS weight to 1 (no other technique here)
w_0 = 1;
f_0 = 1;
for j = 1 in \infty:
x_j = intersect (x_{j-1}, \omega_{j-1}^i);
\omega_{j}^i = \omega_{j-1}^i;
// increment the touched finite-elements’ counters
increment_counts (x_j, \omega_{j}^i);
// accumulate the field \hat{L} at the previous vertex (x_{j-1}, \omega_{j-1}^o)
// using information from the current
if j > 0:
    update_value = \hat{L}^{old}(x_j, \omega_j^i) \cdot f_{j-1} \cdot w_{j-1}/p_{j-1}^T;
    update_approx (x_{j-1}, \omega_{j-1}^o);
    // accumulate the local emission at this vertex
    sample_value += sample_weight \cdot L_0(x_j, \omega_j^i);
// and accumulate it to \hat{L}^{new} at this vertex
update_value = \hat{L}^{new}(x_j, \omega_j^i);
update_approx (x_j, \omega_j^i);
// perform next event estimation
new_value = next_event_estimation (x_j, \omega_j^i);
// add its contribution to the output
sample_value += sample_weight \cdot new_value;
// sample a scattering event
(\omega_{j+1}, f_{j+1}, p_{j+1}^T, w_{j+1}) = scattering_event (x_j, \omega_j^i);
// update the sample weight for this path
sample_weight = sample_weight \cdot f_{j} \cdot w_{j}/p_{j}^T;
return sample_value;
```

ALGORITHM 2: basic PSTF path-tracing skeleton, tracking the field $L_{\omega\nu}$

```plaintext
sample_value = 0;
sample_weight = 1;
(x_0, \omega_0, p_0^T) = sample_camera();
// set the MIS weight to 1 (no other technique here)
w_0 = 1;
f_0 = 1;
for j = 1 in \infty:
x_j = intersect (x_{j-1}, \omega_{j-1}^i);
\omega_{j}^i = \omega_{j-1}^i;
// increment the touched finite-elements’ counters
increment_counts (x_j, \omega_{j}^i);
// accumulate the field \hat{L} at the previous vertex (x_{j-1}, \omega_{j-1}^o)
// using information from the current
if j > 0:
    update_value = \hat{L}^{old}(x_j, \omega_j^i) \cdot f_{j-1} \cdot w_{j-1}/p_{j-1}^T;
    update_approx (x_{j-1}, \omega_{j-1}^o);
    // accumulate the local emission at this vertex
    sample_value += sample_weight \cdot L_0(x_j, \omega_j^i);
// and accumulate it to \hat{L}^{new} at this vertex
update_value = \hat{L}^{new}(x_j, \omega_j^i);
update_approx (x_{j-1}, \omega_{j-1}^o);
// perform next event estimation
new_value = next_event_estimation (x_j, \omega_j^i);
// add its contribution to the output
sample_value += sample_weight \cdot new_value;
// and (optionally) use it to update \hat{L}^{\omega\nu}_{\omega\nu}
// at this vertex (x_j, \omega_j^i) (recalling this
// technique represents light transported
// once, i.e. $T_{L_{\omega\nu}}$)
update_approx (x_j, \omega_j^i);
// sample a scattering event
(\omega_{j+1}, f_{j+1}, p_{j+1}^T, w_{j+1}) = scattering_event (x_j, \omega_j^i);
// update the sample weight for this path
sample_weight = sample_weight \cdot f_{j} \cdot w_{j}/p_{j}^T;
return sample_value;
```
where $G(x, y)$ is the usual geometric throughput:

$$G(x, y) = \frac{\omega \cdot n_x |\omega \cdot n_y|}{|x - y|^2} \quad (32)$$

Finally, we can convert between the area probability $p(x|y)$ and the projected solid angle probability $p_S^y(-\omega)$ of sampling the direction $-\omega$ at vertex $y$ using the identity:

$$p(x|y) = p_S^y(-\omega)G(x, y) \quad (33)$$

and combining the expressions we get:

$$P_T(x, \omega) = p_S^y(-\omega) \cdot \sum_{l=0}^{\infty} \int p(x_0 \cdot \ldots \cdot x_l)p(y|x_l)dA(x_0) \ldots dA(x_l) \quad (34)$$

with the position $y = h(x, \omega)$.

Acknowledgements We thank Cem Cebenoyan and Aaron Lefohn for supporting this work and Jacob Munkberg and Pascal Gautron for carefully reviewing it and providing valuable feedbacks. We would also like to acknowledge insightful discussions and related work by Alexander Keller, Nickolaus Binder, Sascha Fricke and Hannes Hergerth. The staircase scene is a modified version of a scene from the Rendering Resources archive [Bitterli 2016]. The door ajar scene is a modified version of the reconstruction performed by Miika Aittala, Samuli Laine, and Jaakko Lehtinen of the original scene by Eric Veach.

References

Bekaert, P., Slusallek, P., Cool, R., Havran, V., and Seidel, H.-P. 2003. A custom designed density estimation method for light transport. Research Report MPI-I-2003-4-004, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany; September.

Binder, N., Fricke, S., and Keller, A. 2018. Fast path space filtering by jittered spatial hashing. In ACM SIGGRAPH 2018 Talks, ACM, New York, NY, USA, SIGGRAPH ’18, 71:1–71:2.

Bitterli, B., Jakob, W., Novák, J., and Jarosz, W. 2017. Reversible jump metropolis light transport using inverse mappings. ACM Trans. Graph. 36, 1 (Oct.), 1:1–1:12.

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

Chaos Group. 2008. Light Cache GI. https://docs.chaosgroup.com/display/VRAYSKETCHUP/Light+Cache+GI [Online].

Clarberg, P., Jarosz, W., Akenine-Möller, T., and Jensen, H. W. 2005. Wavelet importance sampling: Efficiently evaluating products of complex functions. In ACM SIGGRAPH 2005 Papers, ACM, New York, NY, USA, SIGGRAPH ’05, 1166–1175.

Dahm, K., and Keller, A. 2017. Learning light transport the reinforced way. In ACM SIGGRAPH 2017 Talks, ACM, New York, NY, USA, SIGGRAPH ’17, 73:1–73:2.

Hachisuka, T., and Jensen, H. W. 2009. Stochastic progressive photon mapping. ACM Trans. Graph. 28, 5, 141.

Hachisuka, T., Ogaki, S., and Jensen, H. W. 2008. Progressive photon mapping. ACM Trans. Graph. 27, 5, 130.

Hachisuka, T., Pantaleoni, J., and Jensen, H. W. 2012. A path space extension for robust light transport simulation. ACM Trans. Graph. 31, 6 (Nov.), 191:1–191:10.

Hanika, J., Droske, M., and Fasching, L. 2015. Manifold next event estimation. Comput. Graph. Forum 34, 4 (July), 87–97.

Hanika, J., Kaplanyan, A., and Dachschafer, C. 2015. Improved half vector space light transport. Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering) 34, 4 (June), 65–74.

He, H., and Owen, A. B. 2014. Optimal mixture weights in multiple importance. Research report, November.

Herholz, S., Elek, O., Vorba, J., Lensch, H., and Krivánek, J. 2016. Product importance sampling for light transport path guiding. Comput. Graph. Forum 35, 4 (July), 67–77.

Ivo, K., Vévoda, P., Grötzmann, P., Skřivan, T., Slusallek, P., and Krivánek, J. 2019. Optimal multiple importance sampling. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2019) 38, 4 (July), 37:1–37:14.

Jakob, W., and Marschner, S. 2012. Manifold exploration: A markov chain monte carlo technique for rendering scenes with difficult specular transport. ACM Trans. Graph. 31, 4 (July), 58:1–58:13.

Kajiy, J. T. 1986. The rendering equation. In Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, NY, USA, SIGGRAPH ’86, 143–150.

Kaplanyan, A. S., Hanika, J., and Dachschafer, C. 2014. The natural-constraint representation of the path space for efficient light transport simulation. ACM Transactions on Graphics (Proc. SIGGRAPH) 33, 4.

Karlík, O., Šik, M., Vévoda, P., Skřivan, T., and Krivánek, J. 2019. Mis compensation: Optimizing sampling techniques in multiple importance sampling. ACM Trans. Graph. (SIGGRAPH Asia 2019) 38, 6.

Kelemen, C., Szirmay-Kalos, L., Antal, G., and Csonka, F. 2002. A simple and robust mutation strategy for the Metropolis light transport algorithm. In Computer Graphics Forum, 531–540.

Keller, A., Dahm, K., and Binder, N. 2014. Path space filtering. In ACM SIGGRAPH 2014 Talks, ACM, New York, NY, USA, SIGGRAPH ’14, 68:1–68:1.

Lafortune, E., and Willems, Y. 2016. A 5d tree to reduce the variance of monte carlo ray tracing. Rendering Techniques 1995 (Proceedings of the Sixth Eurographics Workshop on Rendering), 11–20.

Li, W., Tan, Z., and Chen, R. 2013. Two-stage importance sampling with mixture proposals. Journal of the American Statistical Association 108, 504, 1350–1365.
Müller, T., Gross, M., and Novák, J. 2017. Practical path guiding for efficient light-transport simulation. *Comput. Graph. Forum* 36, 4 (July), 91–100.

Müller, T., Mcwilliams, B., Rousselle, F., Gross, M., and Novák, J. 2019. Neural importance sampling. *ACM Trans. Graph.* 38, 5 (Oct.), 145:1–145:19.

Owen, A. B., and Zhou, Y. 2000. Safe and effective importance sampling.

Pantaleoni, J. 2017. Charted metropolis light transport. *ACM Trans. Graph.* 36, 4 (July), 75:1–75:14.

Pantaleoni, J. 2019. Importance Sampling of Many Lights with Reinforcement Lightcuts Learning. *arXiv:1911.10217* (Dec.).

Sbert, M., and Elvira, V. 2019. Generalizing the Balance Heuristic Estimator in Multiple Importance Sampling. *arXiv:1903.11908* (Mar.).

Veach, E., and Guibas, L. J. 1997. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH ’97, 65–76.

Veach, E. 1997. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University.

Vorba, J., and Křivánek, J. 2016. Adjoint-driven russian roulette and splitting in light transport simulation. *ACM Trans. Graph.* 35, 4 (July), 42:1–42:11.

Vorba, J., Karlík, O., Šik, M., Ritschel, T., and Křivánek, J. 2014. On-line learning of parametric mixture models for light transport simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)* 33, 4 (aug).

Vorba, J., Hanika, J., Herholz, S., Müller, T., Křivánek, J., and Keller, A. 2019. Path tracing in production. In *ACM SIGGRAPH Courses*, ACM, New York, NY, USA, 18:1–18:77.