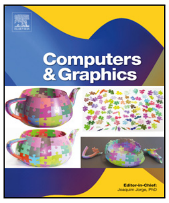
Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag



Technical Section

Path guiding for wavefront path tracing: A memory efficient approach for GPU path tracers[☆]

Bora Yalçıner [*], Ahmet Oğuz Akyüz

Middle East Technical University, Computer Engineering Department, Ankara, Turkey

ARTICLE INFO

Keywords:
Graphics processors
Monte Carlo rendering
Path tracing
Path guiding

ABSTRACT

We propose a path-guiding algorithm to be incorporated into the wavefront style of path tracers (WFPTs). As WFPTs are primarily implemented on graphics processing units (GPUs), the proposed method aims to leverage the capabilities of the GPUs and reduce the hierarchical data structure and memory usage typically required for such techniques. To achieve this, our algorithm only stores the radiant exitance on a single global sparse voxel octree (SVO) data structure. Probability density functions required to guide the rays are generated *on-the-fly* using this data structure. The proposed approach reduces the scene-related persistent memory requirements compared to other path-guiding techniques while producing similar or better results depending on scene characteristics. To our knowledge, our algorithm is the first one that incorporates path guiding into a WFPT.

1. Introduction

Path tracing family of techniques became one of the standard methods for generating photo-realistic imagery [1]. The primary motivation to use these methods is their implementation simplicity and generated image quality. Furthermore, recent advances in graphics hardware enable interactive implementations of such techniques. These techniques tackle the complex recursive light-transport integral by applying numeric Monte Carlo integration, a process known as sampling.

Many sampling schemes are proposed throughout the literature that either sample sub-sections of the integral (e.g., next-event estimation) or reflectance portion of the integral [2]. Such sampling schemes are comparatively simpler because their data is readily available in the initial scene definition. Other parts of the integrand mostly depend on the layout of the elements described in the scene. Extracting a probability field of light distribution over the scene is a critical component of a robust photo-realistic image estimator. This problem is tackled by a family of algorithms which are collectively known as *path guiding* algorithms [3–5].

Most path-guiding methods utilize a hierarchical discretization of the light field or a combination of analytically defined functions that fit this light field. The generation of this probability field relies on the light transport simulation itself; thus, path-guiding methods progressively learn this field from path tracing either during runtime or in a preprocessing step. This progressive nature of path guiding necessitates the usage of highly adaptive data structures, which inherently do not suit the GPU architecture well. Adaptive discretization, which relies on adaptive memory management, is not a GPU-friendly operation. Another problem is that such a probability field has a large memory requirement due to its being high-dimensional.

To this end, we propose a wavefront path guiding algorithm that is GPU-friendly and designed to fully utilize the GPU's capabilities. Our main contributions are thus (1) on-the-fly generation of the radiant exitance field, which resides on an SVO data structure; (2) hardware-accelerated approximate cone tracing for an efficient query of the radiant exitance, (3) GPU-friendly parallel product path guiding scheme that utilizes warp-level intrinsics, and (4) a heuristic that judiciously combines generated samples for improved final image quality.

2. Previous work

The rendering equation is defined by the following integral [1]:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{\Omega} f_s(x,\omega_i,\omega_o) L_i(x,\omega_i)\cos\theta_i d\omega_i, \tag{1}$$

where the outgoing radiance $L_o(x,\omega_o)$ may contribute to another location x_k, thus, becoming $L_i(x_k,-\omega_i)$. As such, the above equation can be recursively expanded, resulting in a series of chained integrals with a theoretically infinite recursion depth. Because of that, it is not analytically integrable and is usually evaluated by using the following Monte Carlo estimator, where the emitted radiance $L_e(.)$ is typically omitted:

$$\hat{L}_o(x,\omega_o) = \frac{1}{N}\sum_{i=1}^{N} \frac{f_s(x,\omega_i,\omega_o)L_i(x,\omega_i)\cos\theta_i}{p(\omega_i|x,\omega_o)}. \tag{2}$$

☆ This article was recommended for publication by Prof. J. Jorge.
* Corresponding author.
E-mail addresses: yalciner.bora@metu.edu.tr (B. Yalçıner), akyuz@ceng.metu.edu.tr (A.O. Akyüz).

https://doi.org/10.1016/j.cag.2024.103945
Received 22 November 2023; Received in revised form 14 May 2024; Accepted 14 May 2024
Available online 17 May 2024

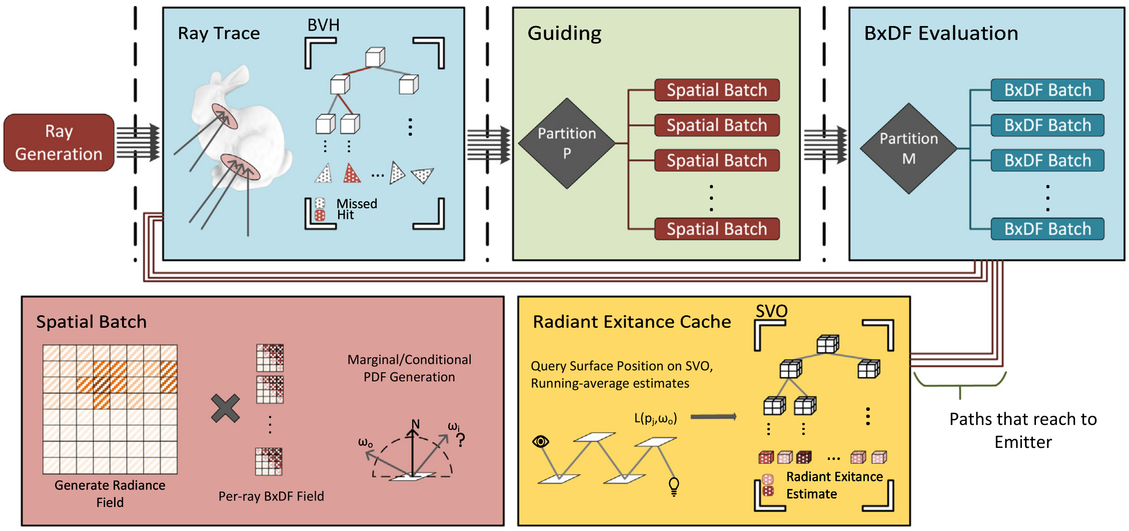


Fig. 1. The top-down view of the entire path-guiding algorithm. Blue rectangles of the image show the wavefront path tracing operations. Other colored parts are the additional steps required for guiding the rays. $Partition_p$ and $Partition_m$ sections represent partitioning the rays by position and material, respectively. Device code is executed for each spatial batch. Each batch generates an incoming radiance field, which is incorporated into the sampling scheme. Paths that reach an emitter contribute to an approximation of the radiant exitance, which is cached on an SVO.

The variance of such an estimator, visually noticed as noise, is directly related to the similarity between PDF p and the integrand. As finding a single optimal PDF is usually not practical, multiple PDFs are typically combined using multiple importance sampling (MIS) [6].

The BSDF portion of the integrand, $f_s(.)$, is traditionally used for importance sampling due to its fully or partially analytic nature. Additionally, parameters required for evaluating it do not recursively depend on other surfaces; thus, $f_s(.)$ can be directly sampled in an efficient manner. However, fitting a density function for the incoming radiance portion of the integrand, $L_i(.)$, is more cumbersome. Finding a plausible density function for the incoming radiance field falls into the domain of path-guiding algorithms.

2.1. Path guiding

Scene radiance field is a high-dimensional data requiring three dimensions for spatial information and two for spherical direction information. Since the directional distribution of the incoming radiance field is not available from the outset, most path-guiding methods either pre-generate it [4,7–9] or progressively learn it using already computed path chains [5,10].

Path guiding was pioneered by the works of Lafortune et al. [11] and Jensen [3]. These approaches utilize histogram-based techniques, which suffer from memory scalability issues when dealing with high-resolution data. Schüßler et al. proposed a 9D Gaussian mixture model (GMM) incorporating the incident and the previous point's information on the path chain [12]. Combinations of analytically defined functions for approximating the radiance field are proposed as well [4,13]. Product sampling, which is the sampling of not only the incoming radiance but its product with BSDF, is also incorporated into these approaches [14].

Ruppert et al. utilize a kd-tree for spatial subdivision and a series of von Mises–Fisher distributions (vMF) for the directional portion of the field [15]. This method tackles the variance seams that occur when a spatial portion of the data structure is at a low-resolution state by adjusting the vMFs to the sampler path's point of view. Müller et al. proposed the practical path guiding method [5]. This algorithm utilizes sparse data structures in the global radiance field's spatial and directional portions. Product extension is also proposed for this method [16].

Learning-based methods are also proposed for path guiding. Among these, neural network techniques are either pre-trained via generic scenes [8,9,17] or trained on a per-scene basis [7]. Reinforcement learning-based techniques are also used by resembling this problem into Q-learning [18–20]. However, these approaches store the directional portion of the radiance field densely, which is not scalable in terms of memory. For the spatial portion, point samples [19] or dense arrays [20] are utilized. The Bayesian regression model is also applied to efficiently sample light sources for improved next-event estimation sampling [10].

There are real-time and GPU-focused path-guiding approaches as well. Derevyannykh proposed a screen space parametric mixture model for path guiding [21]. Due to the screen space nature of the method, only the first bounce is guided. Dittebrandt et al. propose a real-time path-guiding scheme that utilizes compressed quad-trees and a visibility cache for light sampling [22].

Despite several techniques being available for both CPU- and GPU-based path guiding, to our knowledge, none of these algorithms are tailored toward a WFPT style path tracer, which has unique design requirements, as discussed next.

2.2. Wavefront path tracing (WFPT)

WFPT is the state-of-the-art design of the graphics device path tracing algorithm [23]. Although API-backed, host-style execution methods exist in the device [24], efficient warp execution mandates some form of partitioning internally.

A straightforward method for parallelizing the path tracing algorithm on a host (i.e., CPU) system is to assign each recursive random walk sequence to a single logical core. However, such parallelization is ill-suited for GPUs because each warp executes in a lock-step fashion. Due to the random nature of the walks, neighboring threads may execute different evaluation routines (e.g., different material and shading computations), which forces the warps to serialize the execution.

The wavefront method segregates the ray casting and material (BSDF) evaluation/sampling routines, allowing recursion to be evaluated in a lock-step fashion. Specifically, for each recursion depth, threads perform the ray-casting operation to determine the incident location. Once the evaluating locations are found, rays are partitioned with respect to the material evaluation parameters. By partitioning the rays in this way, the computational routines are common among the threads, which leads to improved performance and efficiency in execution.

The main issue with this approach is the high memory requirement. This approach requires storing walk states after each operation, which becomes increasingly burdensome as the number of parallel walks increases. In practice, graphics devices often require thousands of walks to be executed to saturate the device, leading to a significant memory burden.

Our primary motivation to describe this proposed method comes from this central issue. The accompanying path-guiding methods for path tracing on graphics devices should not further hinder the available memory or, at the very least, minimize its impact.

3. Proposed method

3.1. Wavefront path guiding

Wavefront path guiding (WFPG) introduces additional steps to the WFPT algorithm. Before partitioning with respect to the BSDF, rays are partitioned by position. Then, the partitioned rays *collaboratively* generate an estimated radiance field, which is utilized for path guiding. After guiding is conducted, rays continue the WFPT steps as usual. The main overview of this algorithm is given in Algorithm 1 and visualized in Fig. 1. In the following, we describe each part of the algorithm in detail.

Algorithm 1 Wavefront path guiding.

```
Input–Output
$R_1 = \{r_1, r_2 \dots\}$                                  ▷ Set of initial rays
$B = \{(p_1, R_{p,1}), (p_2, R_{p,2}) \dots\}$              ▷ Position bins
$M = \{(m_1, R_{m,1}), (m_2, R_{m,2}) \dots\}$              ▷ Material bins
Start
Initially Generate rays from the camera and populate $R_1$
for $i = 1$ to MaxDepth do
    $B_i = \{(p_1, R_{p_1}), (p_2, R_{p_2}) \dots\} \leftarrow$ Partition-S($R_i$)
    $N_i = \{(n_1, R_{n_1}), (n_2, R_{n_2}) \dots\} \leftarrow$ Partition-M($R_i$)
    for all $(p_j, R_{p_j}) \in B_i$ do
        $R^j_{i+1} \leftarrow$ GuideRays($(p_j, R_{p_j})$)
    end for
    for all $(n_j, R_{n_j}) \in N_i$ do
        for all $r_k \in R_{n_j}$ do
            EvaluateBSDF($n_j, r_k$)
        end for
    end for
    $R_{i+1} = \{R^1_{i+1}, R^2_{i+1} \dots\}$               ▷ Next set of rays
end for
for all paths that reach an emitter do
    UpdateExitance(SVO)
end for
```

3.2. Radiant exitance caching using sparse voxel octree

Unlike other methods that store an incoming radiance field over a spatially discretized volume, we approximate radiant exitance. The main reason for this approach is to reduce memory usage, as radiant exitance is a directionless quantity. This quantity is extracted during Monte Carlo integration and cached in a sparse voxel octree (SVO) [25, 26].

To explain the caching scheme, we resort to the recursively expanded version of the rendering equation, in which path chains are explicitly written [27]:

$$L(p_1 \to p_0) = \sum_{k=1}^{\infty} P(\bar{p}_k), \tag{3}$$

where $\bar{p}_k = p_0, p_1, \dots, p_k$ represents all points along the path of a ray, with p_0 on the image plane. The radiance that reaches p_0 from such a path is then:

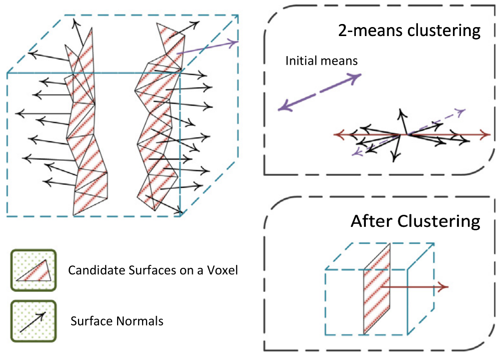


Fig. 2. The approximate normals of the voxel are calculated by conducting a k-means clustering with $k = 2$. This prevents opposite normals from canceling each other out.

$$P(\bar{p}_k) = \underbrace{\int \cdots \int_{\Omega}}_{k-1} L_e(p_k \to p_{k-1}) \times \left(\prod_{j=1}^{k-1} f_s(p_{j+1} \to p_j \to p_{j-1}) G(p_{j+1} \to p_j) \right) \times dA(p_2) \cdots dA(p_k). \tag{4}$$

where f_s is the BSDF and G is the geometry term. The radiance from the path vertex p_k toward p_{k-1} can be extracted from the total throughput as follows:

$$L(p_k \to p_{k-1}) = \frac{T(\bar{p}_n)}{T(\bar{p}_k)} L_e(p_n \to p_{n-1}). \tag{5}$$

Thus, for every path, the position p_k and the throughput $T(\bar{p}_k)$ are stored for every depth on the path. When an emitter is found, its radiance is backpropagated at every depth, and its local radiance estimate is found. The position p_k is used to query the SVO to find the leaf voxel, and finally, these local radiance estimates are accumulated to approximate the radiant exitance for that leaf. This operation corresponds to UpdateExitance(SVO) routine in Algorithm 1.

The SVO is generated using Crassin et al.'s approach [28]. The scene is conservatively voxelized in 3D space, and voxels are generated. After the voxelization step, the tree hierarchy is generated using the method described by Karras et al. using Morton code sorting [29].

Since the SVO has a limited voxel resolution, scenes with thin objects would not be adequately represented due to the SVO's volumetric subdivision. To alleviate this, we approximate the surface orientation via normals. Thus, each node of the SVO stores two surface normals, and the radiant exitance corresponding to each normal direction is separately stored. The radiant exitance that is stored on the leaf nodes of the SVO is propagated toward the inner nodes of the tree structure. In our experiments, we found a simple bottom-up averaging scheme to be sufficient. This information is required to query incoming radiance using cone tracing.

To estimate normals, the surface fragment normals are obtained during the voxelization process and are subjected to a simple k-means clustering procedure with $k = 2$. This process is demonstrated in Fig. 2. The initial means for the algorithm is a random vector selected from the surface elements inside the voxel and its opposite. Through iterative refinement, these vectors are updated to represent the two dominant directions of the surface elements better. At the end, the first cluster's representative vector $\vec{N}$ and its opposite $-\vec{N}$ are selected as the representative directions for that voxel. The motivation behind this approach is to allow a voxel to become an omnidirectional source. If the clustering results were directly used, certain fragments whose normals point away from the dominant directions would not make a contribution. The clustering approach also prevents normals from canceling each other out, which could occur if a simple average was used.

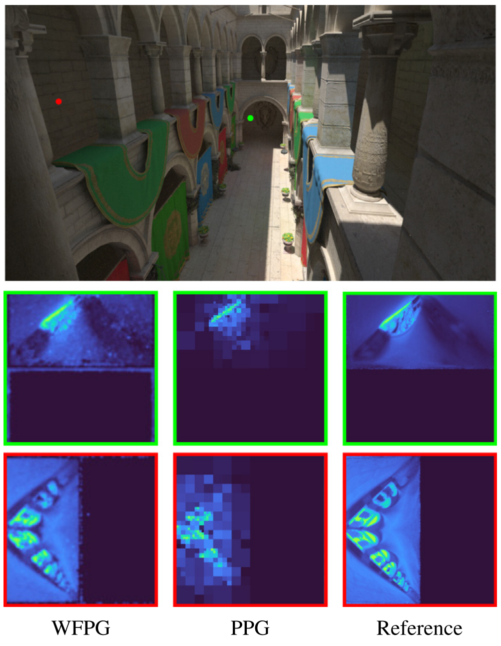


Fig. 3. Normalized radiance fields of our method (WFPG) and Müller et al.'s method (PPG). In this instance, our method produces a radiance field with a resolution of 128^2. Both methods are trained using an equal number of samples (2048 per pixel). The reference radiance field is generated via path tracing and has a resolution of 256^2 (2^{16} samples per pixel).

3.3. On-the-fly generation of local radiance field

To generate the incoming radiance over a surface, we utilize a modified cone tracing approach to reduce aliasing artifacts that would be caused by sampling the environment using infinitesimally thin rays [30, 31].

Given a location on the scene, p_k, the omnidirectional incoming radiance field $L(p_k, \omega_i)$ is stratified into equal solid angle patches, ω. The SVO is queried for each patch by tracing a cone toward that direction to find the incident hit position, which can be found using different approaches. The volumetric estimation proposed by Crassin et al. [31] is efficient but is prone to light leaks. Empty space skipping cone tracing [25] can also be used, but we found it to be slower compared to a third alternative. In this alternative, we use the underlying device's hardware-accelerated ray tracing capabilities to find the intersection point. After the hit point is found, the radiant exitance stored in the SVO is queried using the cone aperture, hit position, and distance.

As the cone with an aperture of ω travels into the scene, the area A of the disk at the base of the cone increases. This (projected) area can be computed by using the following formula:

$$A = r^2\omega, \tag{6}$$

where r is the distance between the apex and the cone base. At this distance, the area of the disk would be equal to πR^2, with R being the disk's radius. As we already know the leaf voxel that contains the intersection point, we traverse up the SVO to find the voxel whose area is closest to the disk's area (the square of its side length approximates the voxel's cross-section area). The radiant exitance in this node is then sampled by multiplying the corresponding voxel normal with the cone's principal direction.

To show the effectiveness of the proposed approach, Fig. 3 compares our method's generated radiance field and that of the practical path-guiding technique [5] together with the ground-truth reference as seen from two different viewpoints. It can be seen from the figures that our scheme better approximates the actual incoming radiance field.

Algorithm 2 PARTITION-S Routine. Partition the paths that have hit p_i to series of bins b_j using an SVO with the depth d.

```
Input
$R = \{r_1, r_2 \dots\}$                                   ▷ Rays that are going to be partitioned
$SVO = \{(n_1)^1, (n_1, \dots)^2 \dots (n_1, \dots)^d\}$            ▷ $(n_1)^1$ is the root
Output
$B = \{(p_1, R_{p_1}), (p_2, R_{p_2}) \dots\}$                ▷ Pair of positions and ray sets
Buffer
$I = \{b_1, b_2 \dots\}$                                    ▷ Bin id for each ray
Start
Clear $B$, $I$
for all $r_i \in R$ do
    $p_i \leftarrow$ RAYPOSITION($r_i$)
    $n_i^d \leftarrow$ DESCENDLEAF($p_i$)                       ▷ Find the leaf node
    ATOMICADD($n_i^d$, 1)
    $b_i \leftarrow$ NODEID($n_i^d$)
end for
for all $l \in$ SVO (in bottom-up fashion, up to $l_{min}$) do
    for all $n^l \in (n \dots)^l$ in SVO level $l$ do
        $C = \{c_1, c_2, \dots c_8\}$                       ▷ Node children's path count
        $T \leftarrow c_1 + \dots + c_8$
        if $T \geq c_{ray}$ or $l = l_{min}$ then
            MARKNODE($n_i^l$)                        ▷ This node has sufficient rays
        end if
    end for
end for
for all $b_i \in I$ do                                   ▷ Find the node for each bin
    $n_i^d \leftarrow$ TONODE($b_i$)
    $n_i^l \leftarrow$ ASCENDANDFINDMARKED($n_i^d$)
    $b_i \leftarrow$ NODEID($n_i^l$)
end for
$B \leftarrow$ PARTITION($I$, $R$)
```

3.4. Positional binning using SVO

Executing the aforementioned radiance field generation scheme would be too costly if evaluated at every point. To amortize this cost, we employ a binning scheme that generates a single radiance field for nearby points. The primary assumption of this approach is that similar regions of the scene would receive similar radiance.

Our positional binning scheme is described in Algorithm 2. Since we already have the scene's SVO hierarchy, we utilize that for the partitioning scheme. Each path atomically increments a value on the leaves of the SVO. Then, these values are accumulated for each level of the SVO in a bottom-up fashion. Two user-defined parameters, referred to as l_{min} and c_{ray}, are employed to control the partitioning process. The l_{min} parameter sets the minimum limit for the tree level up to which binning can be performed. The c_{ray} parameter, on the other hand, determines the threshold for the number of rays considered sufficient for each bin. A sample output for this process is shown in Fig. 4.

Once binning is complete, the radiance field is generated for each bin in a GPU-oriented manner. That is, a single device block is utilized for each partition, and the threads on that block *simultaneously* generate the radiance field. The resolution of this radiance field is another parameter of our method. In our experiments, we used a maximum resolution of 128 × 128 (corresponds to 64 KiB of memory) due to shared memory limitations. This local radiance field is stored in the shared memory available for each block — in other words, no persistent GPU memory is used.

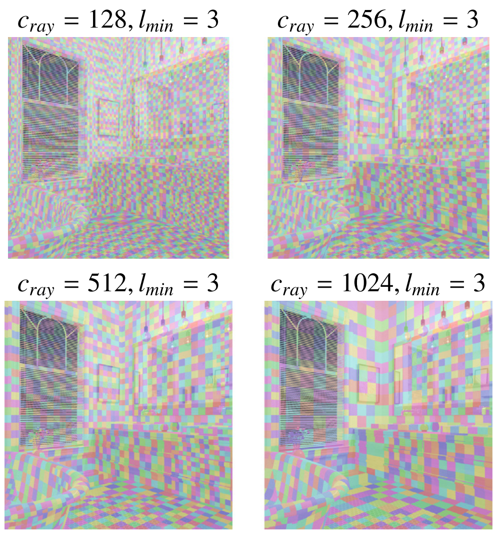

$c_{ray} = 128, l_{min} = 3$
$c_{ray} = 256, l_{min} = 3$
$c_{ray} = 512, l_{min} = 3$
$c_{ray} = 1024, l_{min} = 3$

Fig. 4. False color representation for the binning process for the initial rays coming from the camera. For each colored region, a single local radiance field is generated. Note that the region size increases with the ray count threshold parameter, c_{ray}.

Algorithm 3 GuideRays routine. Given a bin with partitioned rays, generate incident radiance field, generate PDF and CDF, and sample either using path guiding or BSDF via MIS.

```
Input
$(p_j, R_{p_j})$                              ▷ Partitioned position and rays
Output
$R^j{}_{i+1}$                                 ▷ Guided rays
Buffer
$L(p_i, \omega_i)$                            ▷ Incoming Radiance Field on shared memory
$PDF(\omega_i), CDF(\omega_i)$                ▷ PDF and CDF on shared memory
Start
$p_o \leftarrow$ SelectOrigin($R_{p_j}$)
for all $\omega_i \in \Omega$ do
    $L(p_o, \omega_i) \leftarrow$ ConeTrace($SVO, p_o, \omega_i$)
end for
$CDF(\omega_i), PDF(\omega_i) \leftarrow$ GeneratePDF-CDF($L$)
for all $r_k \in R_{p_j}$ do
    $M \leftarrow$ AcquireMaterial($r_k$)
    $r_{k_{next}} \leftarrow$ MIS($PDF(w_i), CDF(w_i), M$)
end for
$R^j{}_{i+1} = \{r_{1_{next}} \dots\}$
```

3.5. Path guiding

The local radiance field generated after the binning process can now be used for path guiding. To this end, we first compute the probability and cumulative density functions (PDF) and (CDF) for sampling according to these distributions. For the sampling process, the radiance field is treated as a piecewise constant 2D function. Traditional inverse sampling methods can be used in parallel [32]. Each row of the 2D field is assigned to a single warp, which applies an inclusive scan (prefix sum) using warp-level intrinsics to generate the CDF for each row. The marginal portion of the CDF is calculated similarly. For non-product path guiding (see below), the results of this approach can be used directly. The pseudocode for this phase of our approach is shared in Algorithm 3.

3.6. Product path guiding

In product path guiding, we multiply the BSDF at each point with its corresponding radiance field. The main problem that needs to be solved is the efficient computation of this product. For this purpose, we utilize the approach of Estevez et al. [33], which was initially proposed for environment mapping. This method utilizes a two-layer hierarchical system. The lower level is the original radiance field. The higher level is the subsampled representation of this field into the resolution of the BSDF field. Based on the constraints of the underlying GPU, we found using an 8×8 resolution appropriate for the higher level. The process then involves element-wise multiplication of the BSDF and low-resolution radiance fields. This approach uses a different parallelization scheme compared to the previous one. In this scheme, each warp (group of threads) handles a single point instead of each thread. Using warp-level intrinsics, each warp collaboratively generates a multiplied upper layer and samples from it.

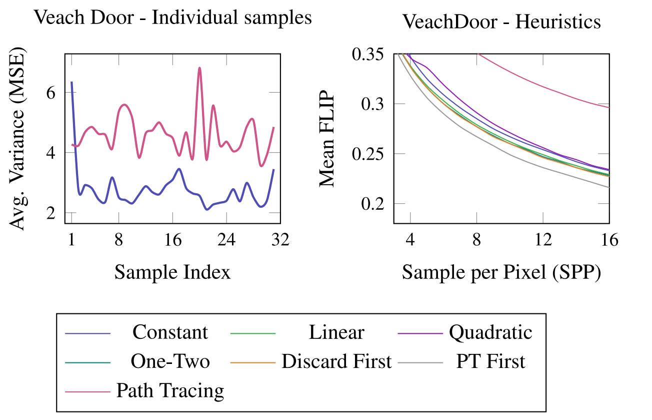


Fig. 5. Single sample variance of the proposed and traditional path-tracking methods. Each sample on the graph is considered in isolation. This graph exposes our method's learning scheme. The general trend of the light distribution is immediately learned in a couple of samples. In this example, the benefits stabilize after about the 20th sample. The first sample of path guiding has higher error than pure path tracing because we sample an omnidirectional field, whereas path tracing samples a hemispherical one.

The result is then used for the first stage of sampling, which can be done as explained in the previous section. We then find the corresponding block in the lower level (i.e., higher resolution) radiance field and perform a second stage of sampling according to the distribution in this block. For example, if the higher and lower levels are 8×8 and 128×128 respectively, the second sampling samples from a 16×16 field.

3.7. Sample combination heuristic

As our approach progressively learns about the radiant exitance distribution, initial samples may not benefit sufficiently from path guiding. With each primary ray sample, the distribution will be better learned, and the benefits will improve. However, after a certain number of samples, the field may saturate and only undergo incremental changes. This section describes several heuristics that experimentally combine different sampling schemes given the described behavior. Given a set of N full image samples and weights $S = \{(S_1, W_1), (S_2, W_2), \dots, (S_N, W_N)\}$, the resulting radiance-field of the generated image I can be computed with the given heuristics function $h(i)$ as follows:

$$I = \frac{\sum_{i=1}^{N} W_i S_i h(i)}{\sum_{i=1}^{n} W_i h(i)}. \tag{7}$$

Several heuristic functions are shown below:

$$h(i) = \begin{cases} i & i < 5 \\ 5 & \text{otherwise} \end{cases} \quad \text{(Linear)}$$

$$h(i) = \begin{cases} i^2 & i < 5 \\ 25 & \text{otherwise} \end{cases} \quad \text{(Quadratic)}$$

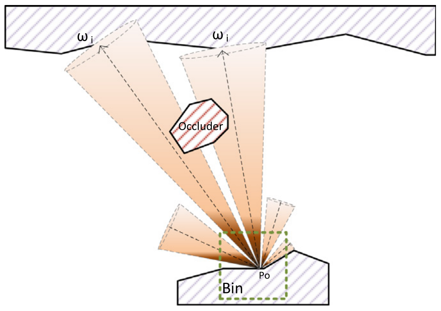


Fig. 6. Aliasing illustration, assuming radiance field is generated over the volume represented by the green dashed square. The radiance field is generated from a point p_o. The contribution of a small occluder (shaded red) could not be captured due to the low-resolution radiance field. Cone rays miss the occluder, and the radiant exitance of the surface behind is queried.

$$h(i) = \begin{cases} 1 & i = 1 \\ 2 & \text{otherwise} \end{cases} \qquad \text{(One-Two)}$$

$$h(i) = \begin{cases} 0 & i = 1 \\ 1 & \text{otherwise} \end{cases} \qquad \text{(Discard First)}$$

In addition to these heuristics, we experimented with two more, namely "Constant" and "PT First". In the former, each sample has constant weight, and in the latter, the first sample directly comes from the first path-tracing sample without path guiding being applied. The remaining samples are generated with path guiding and are equally weighted.

The results for different combinations are shown in Fig. 5. Here, the left graph shows the mean squared error for each sample in isolation. The pink curve corresponds to pure path tracing and the purple curve to our approach. It can seen that for the first sample, our approach has higher error as we sample an omnidirectional field, whereas path tracing samples a hemispherical one. In our case, the following samples produce lower variance than path tracing as the light distribution is learned. The benefits stabilize after a certain point. On the right-hand side of the same figure, we show the results of different sample combination heuristics with respect to the HDR-FLIP metric [34]. It can be observed that among the proposed strategies, the best combination strategy is "PT First", which we use for the results produced in this paper.

4. Implementation

We have implemented our algorithm using CUDA. For hardware-accelerated ray tracing, we use the OptiX Framework [35]. Our source code is publicly available in [36]. In the following, we discuss several important implementation issues.

OptiX & Shared Memory: Since OptiX does not expose inline ray-tracing capabilities, we could not utilize the shared memory and the device's hardware-accelerated ray-tracing capabilities in a single kernel execution. Therefore, we use a small persistent buffer to segregate the OptiX ray-tracing pipeline launches with the sampling kernel launches. The allocation amount depends on the number of processors on the device. In our experiments, 8 to 16 MiB of memory was enough to saturate mid to high-end GPU. This segregation is unnecessary for other APIs, such as DXR and Vulkan, since hardware-accelerated ray tracing inside a compute shader and access to the shared memory can be done simultaneously.

Captured radiance field's origin and aliasing: Selecting the reference point for the radiance field generation is not simple. Directly selecting the center point of the collaborating rays would create self-occlusions, or directly using the center of the partitioned region would make variance seams toward the edges of the partitioned area. Instead, we randomly select a candidate hit location and use it as a radiance field origin at each iteration.

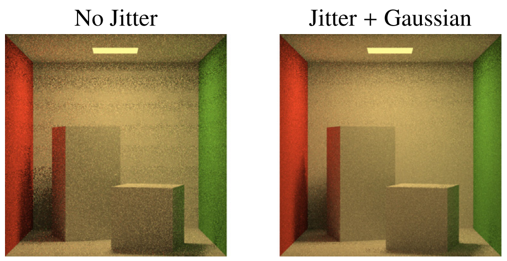


Fig. 7. Demonstration of jittering and filtering. NEE is turned off for demonstration purposes. Variance seams are visible without jittering. Positional and directional jitter and Gaussian blur minimize variance seams.

Like in the real-time rendering paradigm, aliasing is an issue, even with the cone tracing approach (Fig. 6). High-frequency illumination or occlusion could not be captured due to the relatively low-resolution radiance field. To reduce this issue, we jitter the sampling directions to capture these high-frequency regions, at least on some iterations.

Both directional and spatial jittering minimize variance seams. However, the radiance field we generate corresponds precisely to the spatial location being rendered; any other rays trailing a slightly different part of the scene would need a slightly modified field to compensate for the difference in perspective. To make the generated radiance field plausible for all the rays in the bin, we filter the radiance field using Gaussian blur. The effects of these antialiasing approaches are illustrated in Fig. 7.

PDF Domains: For an estimator to be unbiased, the sampling domain should encapsulate the sampled radiance field, meaning that its PDF should be non-zero where the radiance field is non-zero. Generated radiance fields are common for groups of rays, and they are generated from a singular location on the spatial domain; sampling for some rays would not satisfy this unbiasedness constraint. Blurring the generated radiance field alleviates this issue somewhat, but there can still be zero values on the radiance field due to self-occlusion. Because of that, we set a constant non-zero epsilon ($\epsilon = 10^{-2}$) value as the initial value of the radiance field.

Multiple Importance Sampling (MIS): Although we explain our algorithm as if the radiance field guiding scheme is the only sampler for the given path, in practice, traditional next-event estimation (NEE) and BSDF sampling schemes are combined using MIS. MIS requires the combined PDFs to be present for calculation. To acquire the BSDF sampler's PDF, BSDF data needs to be accessed, which may raise an issue of branching discrepancies between threads within a warp. However, due to the spatial binning scheme, nearby regions usually have the same BSDF, and branch divergence is minimal. This is not an issue with the product path guiding scheme since the entire warp is responsible for a single ray.

Radiance field projection on to 2D Cartesian space. To represent the radiance field on a 2D Cartesian grid, we utilize concentric octahedral mapping [37]. This method is known to give better results on lower resolutions than other classical projection techniques, such as spherical projection.

5. Results & validation

In this section, we evaluate our method under various test scenarios. For clarity, our method is named WFPG throughout this section. In all tests, next-event estimation is enabled. All of the GPU measurements are done using a 3070Ti Mobile GPU. Lastly, we generate radiance fields in a decaying manner, meaning that with each path depth, we decrease the size of the generated radiance field by two in each dimension. This is motivated by the fact that earlier bounces make a greater contribution to the final image due to higher throughput.

Table 1
Single sample per pixel timings (ms) of the wavefront path guiding stages. Each Depth$_n$ is the total computation time of our algorithm with (bottom) and without (top) product path guiding for the nth bounce of a ray path. The miscellaneous portion includes partitioning and material evaluation routines. For product path guiding, BSDF resolution is 8 × 8. The bottom row shows the time of pure path tracing.

	1920 × 1080		1280 × 1280	
	SPONZA	VEACHDOOR	BATHROOM	CORNELLBOX
WFPG	$l_{min} = 5$, $c_{ray} = 512$, SVO $= 256^3$			
Depth$_1$ (128 × 128)	48.29	35.76	33.78	74.46
	199.67	180.66	143.37	134.88
Depth$_2$ (64 × 64)	25.36	21.23	22.54	31.24
	109.46	122.96	99.94	82.72
Depth$_3$ (32 × 32)	16.68	16.84	14.20	12.39
	76.67	107.43	79.90	56.16
Depth$_4$ (16 × 16)	10.96	17.25	11.10	7.15
	58.03	105.77	73.90	47.56
Update Exitance	4.21	4.07	6.5	4.37
Miscellaneous	75.21	70.80	102.76	48.33
Total	180.71	165.95	190.88	177.84
	523.25	591.69	506.37	374.02
SVO Generation	45.45	17.81	35.54	44.39
PT	74.00	80.8	92.73	54.42

Table 2
WFPG statistics for selected scenes. Parameters for our method are the same as in Table 1. Statistics for only the first three path depths are provided.

		Depth$_1$ 128 × 128	Depth$_2$ 64 × 64	Depth$_3$ 32 × 32
SPONZA	(1)	2005/1034	2713/570	2604/444
	(2)	262.8	88.9	21.3
VEACHDOOR	(1)	2218/935	2637/707	2859/602
	(2)	290.7	86.41	23.42
BATHROOM	(1)	1867/829	3158/444	3022/418
	(2)	244.7	103.5	24.8
CORNELLBOX	(1)	4861/337	7151/180	7048/136
	(2)	637.1	234.3	57.7

(1) Bin count/avg. ray per bin
(2) Generated PDF and CDF memory (MiB)

5.1. Profiling

In Table 1, we share the overall timing calculations for our algorithm. The WFPG portion shows the time spent at each depth of our algorithm. The lightly shaded rows (top) are for pure path guiding, and the darkly shaded ones (bottom) are for product path guiding. The time for SVO generation, which is done only once, is shown at the bottom. The PT row shows the time of pure path tracing without using our method. It can be seen that the overall cost of our path-guiding algorithm is approximately two times of path tracing. The cost of product path guiding is higher due to per-ray product field calculation, multiplication, and layered sampling.

Table 2 shows several statistics of the WFPG on different scenes. These are the bin counts, average rays per bin, and the combined PDF and CDF memory sizes at each bounce. Dense methods such as Dahm and Keller [19] and Kim et al. [20] would be required to hold these dense structures in persistent memory, whereas in our case, this memory is transient in the sense that it is used as shared memory for each bounce and released for the next one.

5.2. Equal sample/time comparison

In the top row of Fig. 8, we compare our algorithm's path-guiding and product path-guiding versions with each other and against path tracing under an equal sample scenario for five test scenes. Two error metrics are used. For the top two rows, HDR-FLIP [34], which is a perceptual metric, is used, and for the bottom two rows, the Mean Square Error (MSE), which is a numeric metric, is used. It is worth noting that we applied the MSE metric on tone-mapped [38] frames as otherwise critical but minor errors in dark pixels would be subjugated by relatively less important but higher magnitude errors in lighter pixels.

In these results, the "Cornell Box" scene is a reproduction of the original scene, while the "Cornell Enclosed" is the same scene except that a glass enclosure surrounds the light source. This effectively disables NEE, as no shadow ray can directly reach the light source. The comparison of these two scenes shows that our method performs noticeably better under indirect illumination. This is expected as when NEE connects a scene point to the light source, it dominates the shading of the point, minimizing the impact of path guiding. As for the other three scenes, both path guiding and product path guiding variants of our method perform better than path tracing.

As our method requires extra computations that increase the run time, we evaluate its performance under an equal time setting. This is shown at the bottom row of Fig. 8. Except for the "Cornell Box", "Sponza", and "Bathroom" scenes for which direct lighting is dominant, our method outperforms the pure path tracing approach.

An interesting behavior can be seen in the "Bathroom" scene. Here, our path-guiding methods outperform regular path tracing under the equal sample scenario; however, in the equal-time setting, path tracing eventually outperforms both proposed methods. One explanation for this is that the dominant light sources (the filaments of the light bulbs) are very small; therefore, even if the rays are guided toward the voxels that contain the light sources, they may not reach the filaments. This also depends on the resolution of the voxels and the radiance field. Secondly, because the mirror is perfectly specular, path guiding using the radiant exitance information produces sub-optimal results. These combined constraints and the additional overhead of our guiding schemes result in this slight underperformance of our approach in the long run for this scene.

Table 3
Memory requirements of different methods. WFPG method parameters are $l_{min} = 5$, $c_{ray} = 512$ and SVO $= 128^3$. All other methods use their default parameters. The iterative training SPP value of Ruppert et al.'s method is 4. For "Sponza" and "Veach Door" scenes, the resolution is 1920 × 1080. For the Bathroom scene, the resolution is 1280 × 1280.

	Depth	Ours	Scene-related Memory (MiB)		
			Training	Müller et al.	Ruppert et al.
SPONZA	4	3.96	16 t	19.0	3.2
			32 t	26.0	5.5
			64 t	37.0	20.1
VEACHDOOR	6	1.26	32 t	40.3	7.4
			64 t	56.3	14.8
			128 t	77.0	30.0
BATHROOM	10	2.00	128 t	83.4	15.4
			256 t	121.32	31.1
			512 t	167.03	63.9

5.3. Memory utilization comparison

Table 3 shows the scene-related memory requirements of our method and other state-of-the-art path-guiding methods. In our case, the scene-related memory is the SVO memory. For Müller et al. [5], it is the memory consumed by sd-trees. Finally, for Ruppert et al. [15], it is the total memory of vMF coefficients and the kd-tree. As can be seen from the table, the scene-related memory cost of other methods increases together with the training time due to the refinement of the data structures. On the other hand, our method's memory requirement is not only smaller but also does not increase over time.

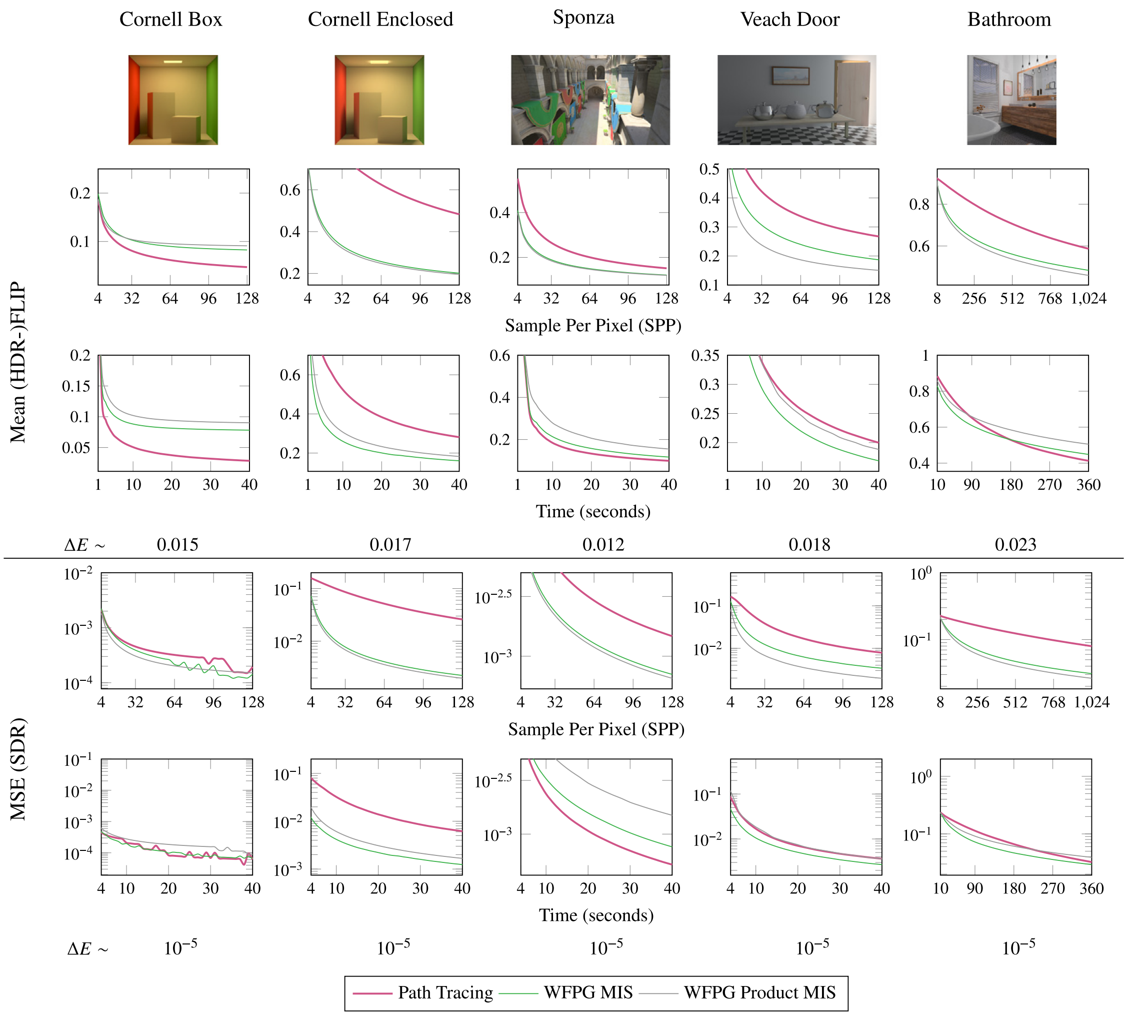


Fig. 8. Equal sample (top row) and equal time (bottom row) comparisons between the proposed method and path tracing. The results of the proposed method are shown with product path guiding both activated and deactivated. WFPG Parameters for "Sponza Lion", "Veach Door", and "Bathroom" scenes are the same as in Table 1. Due to scene simplicity, Cornell Box parameters are $l_{min} = 4$, $c_{ray} = 512$, and SVO $= 256^3$. Overall, parameters are selected to generate around 2000 – 3000 bins per depth iteration consistently. ΔE represents the mean difference of our methods from the ground-truth for each scene rendered with 150 000 samples to demonstrate unbiasedness.

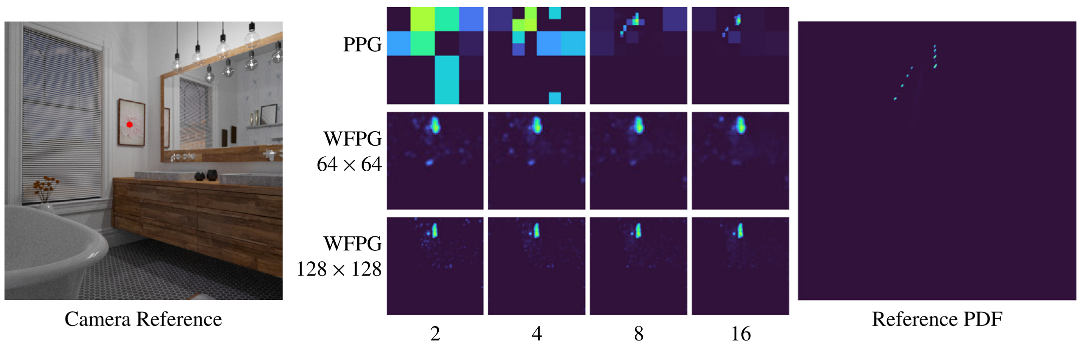
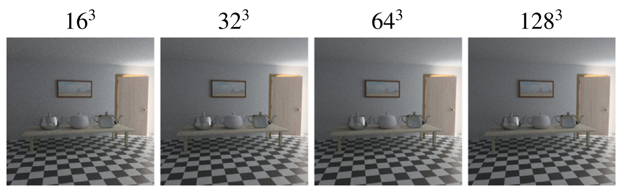


Fig. 9. Convergence of Müller et al.'s method and our method. The reference PDF is generated using path tracing over the red region. For our method, 64 × 64 and 128 × 128 radiance fields are generated. Müller et al.'s method uses default parameters.

In addition to the scene-related memory cost, our method also requires path memory. In complexity notation, it can be represented as $\Theta(p \times d)$ where p is the path count, and d is the maximum traversal depth of the paths. Both of the other approaches are CPU-based path-guiding algorithms. However, we can argue that their potential wavefront-style implementation on the GPU would also require the same amount of path memory as ours.

Given that the SVO memory is the only path guiding related data structure that we hold in persistent memory, we analyze the effect of different SVO resolutions on render quality. This is shown in Fig. 10 for resolutions from 16^3 to 128^3. As can be seen from this figure, the render quality increases up to a point but remains intact afterward. The memory usage of the SVO also increases proportionally. In our analysis, we found the 128^3 SVO resolution to work well for all of our test scenes.



	16^3	32^3	64^3	128^3
(1)	0.179	0.154	0.144	0.144
(2)	$3.19 \cdot 10^{-3}$	$2.23 \cdot 10^{-3}$	$1.91 \cdot 10^{-3}$	$1.92 \cdot 10^{-3}$
(3)	0.02	0.07	0.31	1.26

Fig. 10. Performance change with respect to SVO resolution. The row values are as follows: (1) Mean FLIP. (2) Mean Square Error. (3) SVO Memory MiB.

5.4. *Radiance field validation*

We compare the reference and generated radiance fields over specific regions in different scenes to validate the generated radiance field. Reference radiance fields are generated via path tracing. Two such comparisons can be seen in Figs. 3 and 9. The latter figure exposes the advantage of our method compared to Müller et al.'s method. Our method does not require adapting its directional data structure. The directional data is dense and more or less immediately captures the radiance field; further refinement reduces residual noise.

5.5. *Product path guiding*

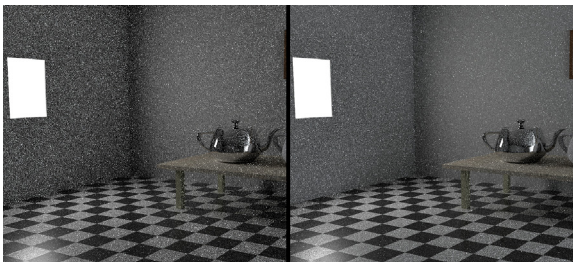
In our experiments, low-resolution (8 × 8) product path guiding mostly eliminates sidedness problems that occur when rays are partitioned around a thin reflective surface with dramatic radiance differences between their sides. Due to omnidirectional generation, rays may probabilistically select the other side of the thin object. The "Veach Door" scene is an excellent example, as the illumination in this scene comes from a bright light source in the back room. As such, regular path guiding may steer more rays toward this direction. With product path guiding, however, if this direction cannot illuminate a surface due to its normal facing away from it, rays will not be guided toward it. In Fig. 8, the scene that most benefits from product path guiding is "Veach Door" due to this characteristic of the scene. Despite this, in equal time comparison, the regular path guiding appears to be still better due to the extra computations involved in product path guiding.

However, if we slightly modify this scene by placing the room in an omnidirectional environment map where the majority of the illumination comes from a window on the wall, product path guiding may outperform regular path guiding even under an equal-time setting. This is illustrated in Fig. 11. It can be seen that despite fewer rays being traced for product path guiding, each ray "counts" more, yielding a final image with reduced noise. Product path guiding can be helpful in these scenarios where thin walls separate an intense illumination between two regions.

5.6. *Comparison with the literature*

We conducted an equal sample comparison between Müller et al.'s [5], Ruppert et al.'s [15], and our methods. We refrain from conducting equal-time comparisons due to underlying architectural differences. Moreover, to prevent renderer-based differences from altering the results, each technique is compared against the reference image of that renderer.

Results can be seen in Table 4. Comparisons provide HDR-FLIP heat maps and mean HDR-FLIP values [34]. On the left of the FLIP heat maps is the reference image of the Mitsuba Renderer. The images' mean square error (MSE) is also given as a separate row. Both compared method parameters are run with their default parameters. As both methods require training and rendering samples, we set the sample count of our process to the sum of these values.



	(a)	(b)
Runtime (60s)	404 spp	126 spp
Mean FLIP	0.467	**0.310**
MSE	0.027	**0.009**

Fig. 11. An equal-time comparison between the non-product (a) and product (b) version of the proposed method showcasing a scenario where product path guiding outperforms regular path guiding in equal time.

We opted for a 256^3 resolution for the SVO, although Fig. 10 suggests that 128^3 resolution is adequate for capturing the radiant exitance field. This is true for the "VeachDoor" scene, but the smaller resolution may not be sufficient on larger scenes such as the "Sponza". Because the memory overhead of increasing the resolution to 256^3 is insignificant, we opted for this higher resolution for the comparisons.

For the "Bathroom" scene, Müller et al.'s and Ruppert et al.'s methods yield very similar error scores, which are both lower than our error score (lower is better in this case). This scene represents a worst-case scenario for our algorithm due to the ideal specular reflection of the mirror. As we represent the illumination using radiant exitance, we guide more rays toward the mirror, despite the fact that the light reflected off the mirror only illuminates the perfect reflection directions.

For the "Veach Door" scene, our error score for product path guiding lies in between the other two methods. In this scene, our product path-guiding version outperforms the regular path-guiding version due to the reason explained in the previous section. Finally, for the "Sponza" scene, WFPG product path guiding yields the lowest error score with a small margin.

Given that the compared methods use different renderers (an earlier version of the Mitsuba renderer) and architectures (GPU vs. CPU), the high degree of similarity between the algorithms suggests that our approach demonstrates competitive performance despite having a smaller memory impact.

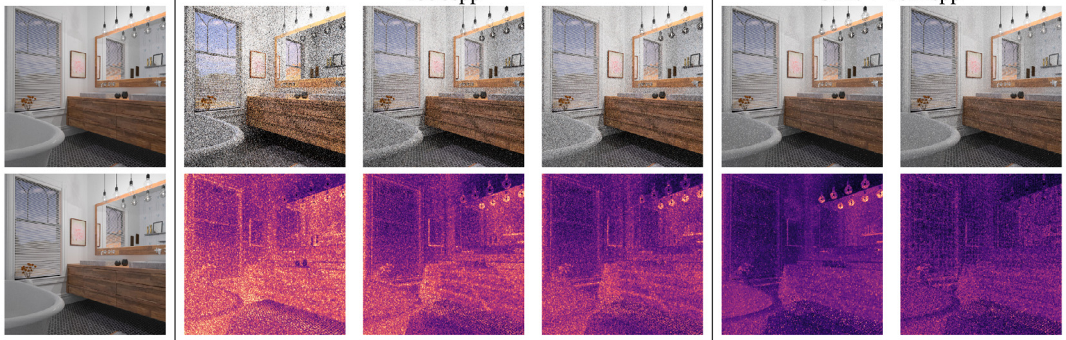
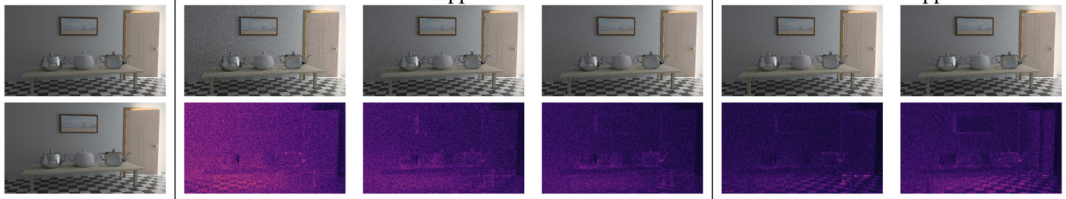
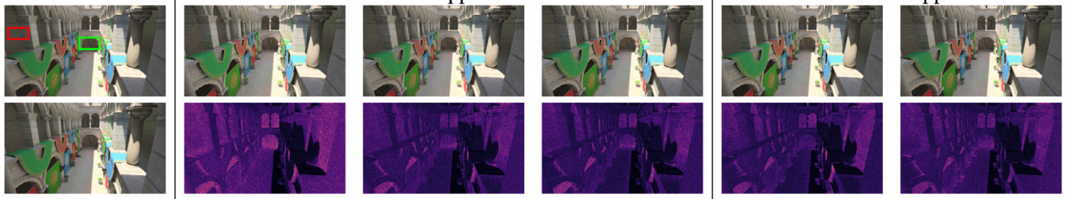
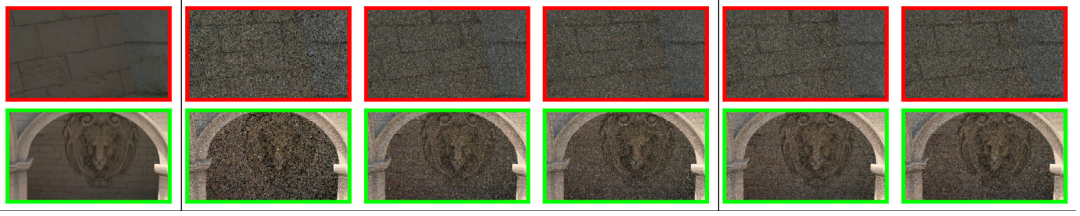
6. Limitations & future work

There are several limitations of the proposed method, some of which are shared by the other path-guiding methods as well. Here, we highlight the most important ones that can be addressed by future work.

Densely generated radiance fields: Generated radiance fields may not capture high-frequency features. These would require a higher resolution capture, which asymptotically requires $\mathcal{O}(n^2)$ amount of work. As a future work, asymmetric cones could be used to query the incident location. A minimal data structure could orchestrate this approach. "Compressed Directional Quadtree" (CDQ), proposed by Dittebrandt et al. can be a candidate for this [22].

Radiant exitance and highly specular objects: To minimize memory footprint, we deemed it necessary to hold only the radiant exitance in the SVO data structure. However, in scenarios such as the "Bathroom" scene, this approach proves insufficient, as discussed earlier. To address this issue, cones can be bounced from specular surfaces to

Table 4
The comparison between the two state-of-the-art and our path-guiding algorithms is shown. We used the default parameters of the literature methods except for the sample counts. We set our sample count (1536) to the sum of the training and rendering sample counts used for each method. Our parameters were $l_{min} = 5$, $c_{ray} = 512$, and the SVO resolution equal to 128^3. The maximum path depth of the scenes were 10 for "Bathroom", 4 for "Sponza", and 6 for "Veach Door".

	Reference	PT	WFPG	WFPG Product	Ruppert et al.	Müller et al.
			1536spp		512t + 1024spp	
Bathroom (Ours / Mitsuba)						
	FLIP Mean	0.533	0.446	0.363	0.272	**0.270**
	MSE	0.125	0.114	**0.092**	0.113	0.098
			96spp		32t + 64spp	
VeachDoor (Ours / Mitsuba)						
	FLIP Mean	0.295	0.207	0.162	**0.142**	0.177
	MSE	0.089	0.069	0.036	**0.015**	0.023
			48spp		16t + 32spp	
CrySponza (Ours / Mitsuba)						
	FLIP Mean	0.226	0.170	**0.165**	0.168	0.176
	MSE	0.001	0.001	0.001	0.001	0.001
						

continue to query the next hit location. Alternatively, the aforementioned CDQ can be used to segment the radiant exitance on regions with high specularity. Determination of these regions could be done at initialization time during SVO generation since it does not rely on light interaction. The refinement of the CDQ, however, would be performed during the runtime phase.

Volumetric subdivision of the scene: In the context of most path-guiding methods, spatial subdivision schemes are often volumetric. Similarly, in our case, the spatial binning scheme is also volumetric. This creates problems when an infinitely thin and two-sided surface occupies this volume. As discussed in this paper, product path guiding could mitigate this issue. Arguably, most scenes involve mostly reflective materials; a surface-based subdivision approach would be beneficial.

Selective path guiding: Since our method generates radiance fields on the fly, it would reduce computational cost to avoid these calculations in regions that would minimally benefit from path guiding, such as those that receive strong direct lighting. This optimization would dramatically improve the computation time of scenes containing mixed regions dominated by direct and indirect illumination.

7. Conclusion

In this paper, we proposed the first GPU-oriented wavefront style path guiding method that does not rely on dynamic memory management — an operation that does not suit the GPU architecture. The proposed method pre-generates an SVO data structure by voxelizing surfaces and refines the radiant exitance field during rendering. This structure is then utilized to generate PDFs to guide rays on the fly. This leads to a smaller memory requirement than the existing methods without hampering image quality. We also showed how to perform product sampling under this setting. By sharing our source code, we hope to stimulate future research for GPU path guiding, which could be vital for real-time path tracing.

Table A.5
Comparison between different GPU-based path tracer implementations and ours using the "Kitchen" scene in 1920 × 1080 resolution. As can be seen from the per-sample timings, wavefront path tracers have an inherent implementation overhead compared to MegaKernel-based ones. However, our timings are on par with well-known ray tracer architectures. The image shown is generated by our renderer.

Renderer - Framework	Type	Time per sample (ms)
NVIDIA-PT-SDK (DXR)	MegaKernel	40.2 (NEE On)
		26.0 (NEE Off)
PBRT-v4 (CUDA)	Wavefront	71.2
Mitsuba3 (CUDA)	MegaKernel	48.8
Ours (CUDA)	Wavefront	67.2

CRediT authorship contribution statement

Bora Yalçıner: Visualization, Validation, Software, Conceptualization. **Ahmet Oğuz Akyüz:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The reference images reported in this paper were partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

Appendix A. Validation

We share the results of an evaluation to compare the run-time performance of our ray tracing architecture with well-known architectures in the literature [39–42]. As seen in Table A.5, the run-time performance of our GPU-based WFPT implementation is similar to PBRT-v4's WFPT implementation. Both ours and PBRT-v4's WFPT results are somewhat slower than Megakernel-based architectures due to the implementation overhead of manually managing path states.

Appendix B. Occupancy analysis

We have conducted GPU resource usage of our product sampling version of radiance field generation kernels. Results can be seen in Table B.6. Shared memory utilization is at its limit for 128 × 128 field generation kernel (68.3 KiB). Each multiprocessor has 128 KiB of shared memory, of which 102 KiB is available for the user on a 3070Ti mobile GPU. Doubling the radiance field resolution will quadruple the memory requirement, which will not fit into the shared memory. To achieve maximum utilization of a multiprocessor, we select a block size of 512, which results in 33% occupancy. Achieving a higher occupancy would have required simplifying the kernels, but this is not feasible due to the complexity of the sampling routines.

Table B.6
GPU resource usage statistics of the radiance field generation kernels. All kernels are launched via 512 threads per block.

Field resolution	Shared Mem. (KiB)	Registers	Occupancy
128 × 128	68.3	128	33%
64 × 64	19.01	128	33%
32 × 32	6.72	128	33%

References

[1] Kajiya JT. The rendering equation. SIGGRAPH Comput Graph 1986;20(4):143–50.
[2] Heitz E, d'Eon E. Importance sampling microfacet-based BSDFs using the distribution of visible normals. In: Proceedings of the 25th eurographics symposium on rendering. EGSR '14, Goslar, DEU: Eurographics Association; 2014, p. 103–12.
[3] Jensen HW. Importance driven path tracing using the photon map. In: Rendering techniques. 1995.
[4] Vorba J, Karlík O, Šik M, Ritschel T, Křivánek J. On-line learning of parametric mixture models for light transport simulation. ACM Trans Graph 2014;33(4).
[5] Müller T, Gross M, Novák J. Practical path guiding for efficient light-transport simulation. Comput Graph Forum 2017;36(4):91–100.
[6] Veach E, Guibas LJ. Optimally combining sampling techniques for Monte Carlo rendering. In: Proceedings of the 22nd annual conference on computer graphics and interactive techniques. SIGGRAPH '95, New York, NY, USA: Association for Computing Machinery; 1995, p. 419–28.
[7] Müller T, Mcwilliams B, Rousselle F, Gross M, Novák J. Neural importance sampling. ACM Trans Graph 2019;38(5).
[8] Huo Y, Wang R, Zheng R, Xu H, Bao H, Yoon S-E. Adaptive incident radiance field sampling and reconstruction using deep reinforcement learning. ACM Trans Graph 2020;39(1).
[9] Bako S, Meyer M, DeRose T, Sen P. Offline deep importance sampling for Monte Carlo path tracing. Comput Graph Forum 2019.
[10] Vévoda P, Kondapaneni I, Křivánek J. Bayesian online regression for adaptive direct illumination sampling. ACM Trans Graph 2018;37(4).
[11] Lafortune EP, Willems YD. A 5D tree to reduce the variance of Monte Carlo ray tracing. In: Rendering techniques '95 (proceedings of the 6th eurographics workshop on rendering). 1995, p. 11–20.
[12] Schüßler V, Hanika J, Jung A, Dachsbacher C. Path guiding with vertex triplet distributions. Comput Graph Forum 2022;41(4):1–15.
[13] Dodik A, Papas M, Öztireli C, Müller T. Path guiding using spatio-directional mixture models. Comput Graph Forum 2022;41(1):172–89.
[14] Herholz S, Elek O, Vorba J, Lensch H, Křivánek J. Product importance sampling for light transport path guiding. Comput Graph Forum 2016;35(4):67–77.
[15] Ruppert L, Herholz S, Lensch HPA. Robust fitting of parallax-aware mixtures for path guiding. ACM Trans Graph 2020;39(4).
[16] Diolatzis S, Gruson A, Jakob W, Nowrouzezahrai D, Drettakis G. Practical product path guiding using linearly transformed cosines. In: Computer graphics forum (proceedings of eurographics symposium on rendering). Vol. 39, 2020, (4).
[17] Zhu S, Xu Z, Sun T, Kuznetsov A, Meyer M, Jensen HW, Su H, Ramamoorthi R. Photon-driven neural reconstruction for path guiding. ACM Trans Graph 2021;41(1).
[18] Sutton RS, Barto AG. Reinforcement learning: an introduction. Cambridge, MA, USA: A Bradford Book; 2018.
[19] Dahm K, Keller A. Learning light transport the reinforced way. In: ACM SIGGRAPH 2017 talks. SIGGRAPH '17, New York, NY, USA: Association for Computing Machinery; 2017.
[20] Kim J, Kim YM. Fast and lightweight path guiding algorithm on GPU. In: Lee S-H, Zollmann S, Okabe M, Wünsche B, editors. Pacific graphics short papers, posters, and work-in-progress papers. The Eurographics Association; 2021.
[21] Derevyannykh M. Real-time path-guiding based on parametric mixture models. In: Pelechano N, Vanderhaeghe D, editors. Eurographics 2022 - short papers. The Eurographics Association; 2022.
[22] Dittebrandt A, Hanika J, Dachsbacher C. Temporal sample reuse for next event estimation and path guiding for real-time path tracing. In: Dachsbacher C, Pharr M, editors. Eurographics symposium on rendering - DL-only track. The Eurographics Association; 2020.
[23] Laine S, Karras T, Aila T. Megakernels considered harmful: Wavefront path tracing on GPUs. In: Proceedings of the 5th high-performance graphics conference. HPG '13, New York, NY, USA: Association for Computing Machinery; 2013, p. 137–43.
[24] Zheng S, Zhou Z, Chen X, Yan D, Zhang C, Geng Y, Gu Y, Xu K. LuisaRender: A high-performance rendering framework with layered and unified interfaces on stream architectures. ACM Trans Graph 2022;41(6).
[25] Laine S, Karras T. Efficient sparse voxel octrees – analysis, extensions, and implementation. NVIDIA technical report NVR-2010-001, NVIDIA Corporation; 2010.

[26] Crassin C, Neyret F, Lefebvre S, Eisemann E. GigaVoxels: Ray-guided streaming for efficient and detailed voxel rendering. In: Proceedings of the 2009 symposium on interactive 3D graphics and games. I3D '09, New York, NY, USA: Association for Computing Machinery; 2009, p. 15–22.

[27] Pharr M, Jakob W, Humphreys G. Physically based rendering: from theory to implementation. 3rd ed.. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2016.

[28] Crassin C, Green S. Octree-based sparse voxelization using the GPU hardware rasterizer. Patrick Cozzi and Christophe Riccio: CRC Press; 2012, chap. 22.

[29] Karras T. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In: Proceedings of the fourth ACM SIGGRAPH / eurographics conference on high-performance graphics. EGGH-HPG'12, Goslar, DEU: Eurographics Association; 2012, p. 33–7.

[30] Amanatides J. Ray tracing with cones. ACM SIGGRAPH Comput Graph 1984;18(3):129–35.

[31] Crassin C, Neyret F, Sainz M, Green S, Eisemann E. Interactive indirect illumination using voxel cone tracing: A preview. In: Symposium on interactive 3D graphics and games. I3D '11, New York, NY, USA: Association for Computing Machinery; 2011, p. 207.

[32] Shirley P, Laine S, Hart D, Pharr M, Clarberg P, Haines E, Raab M, Cline D. Sampling transformations zoo. Berkeley, CA: A Press; 2019, p. 223–46, chap. Sampling.

[33] Conty Estevez A, Lecocq P. Fast product importance sampling of environment maps. In: ACM SIGGRAPH 2018 talks. SIGGRAPH '18, New York, NY, USA: Association for Computing Machinery; 2018.

[34] Andersson P, Nilsson J, Shirley P, Akenine-Möller T. Visualizing errors in rendered high dynamic range images. In: Theisel H, Wimmer M, editors. Eurographics 2021 - short papers. The Eurographics Association; 2021.

[35] Parker SG, Bigler J, Dietrich A, Friedrich H, Hoberock J, Luebke D, McAllister D, McGuire M, Morley K, Robison A, Stich M. Optix: A general purpose ray tracing engine. ACM Trans Graph 2010;29(4).

[36] Yalciner B. METU ray - GPU-based renderer/framework.. 2024, URL: https://github.com/yalcinerbora/meturay. [Accessed 20 February 2024].

[37] Clarberg P. Fast equal-area mapping of the (hemi)sphere using SIMD. J Graph Tools 2008;13(3):53–68.

[38] Reinhard E, Stark M, Shirley P, Ferwerda J. Photographic tone reproduction for digital images. In: Seminal graphics papers: pushing the boundaries, volume 2. 2023, p. 661–70.

[39] Bitterli B. Rendering resources. 2016, URL: https://benedikt-bitterli.me/resources/. [Accessed 20 February 2024].

[40] Pharr M, Jakob W, Humphreys G. Physically based rendering: from theory to implementation. 4th ed.. Cambridge, MA, USA: The MIT Press; 2023.

[41] Jakob W, Speierer S, Roussel N, Vicini D. Dr.Jit: A just-in-time compiler for differentiable rendering. Trans Graph (Proc SIGGRAPH) 2022;41(4). http://dx.doi.org/10.1145/3528223.3530099.

[42] NVIDIA. NVIDIA® RTX Path Tracing. 2023, URL: https://github.com/NVIDIAGameWorks/RTX-Path-Tracing. [Accessed 20 February 2024].