Efficient Hardware Acceleration of Robust Volumetric Light Transport Simulation

DOI: 10.1111/cgf.14802

COMPUTER GRAPHICS *forum*
Volume 42 (2023), number 6, e14802

Efficient Hardware Acceleration of Robust Volumetric Light Transport Simulation

Nol Moonen and Andrei C. Jalba*

Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
nolmoonen@outlook.com, A.C.Jalba@tue.nl

Abstract

Efficiently simulating the full range of light effects in arbitrary input scenes that contain participating media is a difficult task. Unified points, beams and paths (UPBP) is an algorithm capable of capturing a wide range of media effects, by combining bidirectional path tracing (BPT) and photon density estimation (PDE) with multiple importance sampling (MIS). A computationally expensive task of UPBP is the MIS weight computation, performed each time a light path is formed. We derive an efficient algorithm to compute the MIS weights for UPBP, which improves over previous work by eliminating the need to iterate over the path vertices. We achieve this by maintaining recursive quantities as subpaths are generated, from which the subpath weights can be computed. In this way, the full path weight can be computed by only using the data cached at the two vertices at the ends of the subpaths. Furthermore, a costly part of PDE is the search for nearby photon points and beams. Previous work has shown that a spatial data structure for photon mapping can be implemented using the hardware-accelerated bounding volume hierarchy of NVIDIA's RTX GPUs. We show that the same technique can be applied to different types of volumetric PDE and compare the performance of these data structures with the state of the art. Finally, using our new algorithm and data structures we fully implement UPBP on the GPU which we, to the best of our knowledge, are the first to do so.

Keywords: global illumination, novel applications of the GPU, ray tracing, rendering

CCS Concepts: • Computing methodologies → Ray tracing; Reflectance modelling; Graphics processors

1. Introduction

Physically accurate light simulation is complex, as light behaviour depends on materials and media it interacts with. As a consequence, scenes that vary in geometric configuration and object parameters produce different light paths. While many different path sampling techniques have been developed over the years, no single technique is able to best capture all possible light paths, as they typically perform well on specific path subsets and poorly on others. The versatile bidirectional path tracing [LW93, VG94] (BPT) technique performs well on direct and smooth indirect lighting. Photon density estimation (PDE) techniques [JNSJ11] complement this nicely by handling well concentrated indirect lighting and caustics.

Multiple importance sampling [VG95] (MIS) combines multiple sampling techniques. After the individual techniques have been evaluated, the total contribution is a weighted sum of contributions. MIS can make the combined algorithm robust, meaning that it performs well regardless of the light effects in the scene. However, evaluation is costly due to having to evaluate multiple strategies.

Unified points, beams, and paths [KGH*14] (UPBP) uses MIS to combine PDE and BPT, which are complementary in terms of strengths and weaknesses. UPBP starts by sampling light paths originating from light sources. Photon maps are created, which are data structures that contain all points and segments of the light paths. Next, paths are sampled starting at the camera. At every step, the data structures built in the previous step are used to find nearby light data. Using the light and camera subpaths, the BPT and PDE estimators are evaluated. The contributions obtained by evaluating the estimators are combined using MIS, which basically means computing a weight based on the used estimator. Due to the large number of estimators present in UPBP, computing the MIS weights is computationally intensive. In Section 4, we derive an efficient algorithm for computing the MIS weights in UPBP. By reformulating

* Corresponding author

the weight so that it can be incrementally computed as subpaths are sampled, redundant computations are avoided. Here, we build upon previous work, specifically by adding support for participating media and additional volumetric estimators.

Recent advances in GPU hardware have led to the inclusion of hardware units to accelerate ray casting. Not only can this hardware be used to accelerate the generation of light paths, but it can also be creatively used for other purposes. In this work, we leverage these GPU capabilities to accelerate the UPBP method. To this end, in Section 5 we present data structures for PDE that make use of ray tracing frameworks. The selection of a performant data structure is crucial for improving the PDE estimator's speed and, in turn, the performance of UPBP.

In Section 2, we review highly-related work. We cover necessary background in Section 3. In Section 6, we evaluate our full GPU implementation and show that it significantly improves over a CPU-based implementation. Furthermore, we compare our individual data structures to state-of-the-art and find that in most cases, considerable speedups can be achieved. We conclude and discuss avenues for future work in Section 7.

2. Related Work

In this section, we briefly review highly-related works on efficient GPU light transport algorithms. While many different algorithms exist, we focus only on those that make up UPBP: BPT and (volumetric) PDE.

Virtually all global illumination algorithms rely on ray casting as a fundamental operation to locate the closest intersection of a ray with the scene. Work that uses the GPU for this operation mostly focuses on construction and traversal algorithms for acceleration structures. The most popular data structures are bounding volume hierarchies, on which Meister et al. recently conducted a survey [MOB*21]. Another main challenge in implementing ray casting on the GPU is dealing with thread divergence, stemming from random walks terminating at different lengths. As the GPU executes groups of threads at the same time, diverging threads remain idle. Aila and Laine [AL09] use a set of persistent threads that take a new path out of a queue when the current one is done. Novak et al. [NHD10] improve on this by removing the need for a queue and van Antwerpen [vA11a] refines it by grouping similar threads. Meister et al. [MBGB20] investigate reordering rays before traversing acceleration structures. For ray casting, full GPU solutions are available such as OptiX [PBD*10] that deal with the above concerns. In our approach, we have used OptiX as its newer versions utilizes NVIDIA's RTX ray tracing hardware.

As MIS is a core component of many light transport algorithms, the way MIS weights are computed has seen a few developments over the years. For BPT, Veach [Vea97] originally proposes a way of computing the MIS weight that only requires a single iteration over the sampled path vertices. For UPBP [KGH*14], this method is adapted for its reference implementation by Vévoda [Vév14]. However, iterating over all vertices in a path incurs a large cost, especially on GPUs where memory accesses should be minimized. In their GPU implementation of BPT, van Antwerpen [vA11b] formulates the weight differently such that it can be accumulated and stored at the subpath endpoints. This allows efficient computation by eliminating iteration over path vertices. Guo et al. [GHZ18] apply the same idea for volumetric BPT in layered BSDFs. For vertex connection and merging (VCM), a technique combining BPT and photon mapping, Georgiev et al. [GKDS12, Geo12] develop a similar scheme. We extend this scheme to support UPBP.

The selection of suitable data structure is crucial for the efficient evaluation of PDE on the GPU. For photon mapping and VCM, Davidovič et al. [DKHS14] compare the state-of-the-art and find hash grids to perform the best. On the CPU, for beam-based volumetric PDE, typically a BVH is used [JZJ08, JNSJ11]. Jarosz et al. [JNT*11] implement their progressive photon beams using a BVH and a GPU-based ray tracing framework. Progressive photon beams differs from UPBP as it terminates the camera subpath after the first diffuse scattering event. Evangelou et al. [EPVV21] use ray tracing hardware to implement a data structure for non-volumetric photon mapping. We extend this data structure to support several volumetric estimators.

3. Background

In this section, we briefly introduce the path integral formulation of [Vea97] and its extension to participating media [PKK00], which formulates the light simulation problem as an integral over light paths. We discuss how to unidirectionally sample light transport paths and how PDE and BPT combine two of these paths. Finally, we review how UPBP estimates the path integral by combining BPT and PDE.

3.1. Path integral formulation

The light transport problem consists of computing the amount of radiance that arrives at the camera from all light sources in the scene. The path integral formulation expresses the pixel intensity I as the integral $I = \int_{\mathcal{P}} f(\bar{\mathbf{x}})\,\mathrm{d}\mu(\bar{\mathbf{x}})$, where $\mathcal{P}$ is the space consisting of all possible light transport paths, $\mathrm{d}\mu$ is a product measure on $\mathcal{P}$ corresponding to area integration for surface vertices and volume integration for medium vertices. A k-length path $\bar{\mathbf{x}} = \mathbf{x}_0 \dots \mathbf{x}_k$ has $k \geq 1$ segments and $k+1$ vertices, and the integral sums up the contributions of all path lengths. Paths follow the direction of light and start with $\mathbf{x}_0$ on a light source, $\mathbf{x}_1 \dots \mathbf{x}_{k-1}$ are scattering events on surfaces or in media, and $\mathbf{x}_k$ is on the camera. The contribution of this path is described by $f(\bar{\mathbf{x}})$, the measurement contribution function, which is defined as

$$f(\bar{\mathbf{x}}) = L_\mathrm{e}(\mathbf{x}_0)T(\bar{\mathbf{x}})W_\mathrm{e}(\mathbf{x}_k)\,,$$

and illustrated in Figure 1. Here, $L_\mathrm{e}(\mathbf{x}_0)$ is the emitted radiance, $W_\mathrm{e}(\mathbf{x}_k)$ is the response function, and $T(\bar{\mathbf{x}})$ is the path throughput

$$T(\bar{\mathbf{x}}) = \left[\prod_{i=0}^{k-1} G(\mathbf{x}_i, \mathbf{x}_{i+1})T_\mathrm{r}(\mathbf{x}_i, \mathbf{x}_{i+1})\right]\left[\prod_{i=1}^{k-1} \rho(\mathbf{x}_i)\right],$$

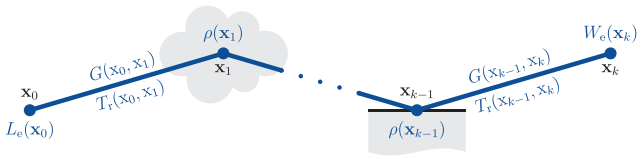


Figure 1: *The measurement contribution function. A light path of $k+1$ vertices and k segments is shown, with the first and last vertices being at the light source and the eye, respectively. The first scattering event occurs in a medium and the last on a surface.*

where G is the geometry term for segments, T_τ is the volumetric transmittance term for segments, ρ is the scattering function for inner vertices. G is defined as

$$G(\mathbf{x}, \mathbf{y}) = V(\mathbf{x}, \mathbf{y})(D(\mathbf{x} \to \mathbf{y})D(\mathbf{y} \to \mathbf{x}))/\|\mathbf{x} - \mathbf{y}\|^2 ,$$

where the visibility function $V(\mathbf{x}, \mathbf{y})$ is 1 if the path from $\mathbf{x}$ to $\mathbf{y}$ is not obstructed by geometry and 0 otherwise. $D(\mathbf{x} \to \mathbf{y})$ is a factor accounting for attenuation on surfaces and is defined as 1 if $\mathbf{x}$ is in media and $n_\mathbf{x} \cdot \omega_\mathbf{xy}$ otherwise, where $\omega_\mathbf{xy}$ is the normalized direction from $\mathbf{x}$ to $\mathbf{y}$ and $n_\mathbf{x}$ is the surface normal at $\mathbf{x}$. The last component of the path throughput is the scattering function at vertices, which is defined as

$$\rho(\mathbf{x}_i) = \begin{cases} \rho_s(\mathbf{x}_{i-1} \to \mathbf{x}_i \to \mathbf{x}_{i+1}) & \text{if } \mathbf{x}_i \text{ on a surface} , \\ \rho_p(\mathbf{x}_{i-1} \to \mathbf{x}_i \to \mathbf{x}_{i+1})\sigma_s(\mathbf{x}_i) & \text{if } \mathbf{x}_i \text{ in a medium} , \end{cases} \quad (1)$$

where ρ_s is the bidirectional scattering distribution function, ρ_p is the phase function, and σ_s denotes the scattering coefficient.

3.2. Path integral estimation

Closed-form solutions for the path integral only exist for very restricted cases and in practice, the path integral is approximated using an estimator. Monte Carlo integration is the 'de facto' method for estimating this integral. It has the form $\langle I \rangle = (1/m) \sum_{i=1}^{m} f(\overline{\mathbf{x}}_i)/p(\overline{\mathbf{x}}_i)$, which averages the contributions of m independent randomly-sampled paths $\overline{\mathbf{x}}_i$ that are sampled with probability density function (PDF) $p(\overline{\mathbf{x}}_i)$. Estimation methods differ in the way paths are generated and in their corresponding path PDFs.

Bidirectional light transport algorithms construct a path $\overline{\mathbf{x}}$ by combining the endpoint of a subpath from a light source with the endpoint of a subpath from the camera. For the remainder of this paper, let $\overline{\mathbf{y}} = \mathbf{y}_0 \dots \mathbf{y}_{s-1}$ denote a light subpath of s vertices, where vertex $\mathbf{y}_0$ is a point on a light source. Let $\overline{\mathbf{z}} = \mathbf{z}_0 \dots \mathbf{z}_{t-1}$ denote an eye subpath of t vertices, where $\mathbf{z}_0$ is a vertex on the camera.

3.3. Path probability density function

Unidirectional sampling extends the path segment by segment. The PDF $p(\overline{\mathbf{y}})$ of such a path $\overline{\mathbf{y}}$ is the joint distribution of its vertices via a chain of conditional vertex PDFs. We will now state the forward path PDFs, sampled in the same direction as the random walk, and the reverse path PDFs, sampled in the opposite direction of the random walk. The reverse PDFs are needed to assign a weight when combining multiple sampling strategies, as will become apparent later. The notation below also applies to $\overline{\mathbf{z}}$.

Forward vertex PDFs.

$$p_i^{\rightarrow}(\overline{\mathbf{y}}) = \begin{cases} p(\mathbf{y}_0) & \text{if } i = 0 , \\ p_{\omega,i}^{\rightarrow}(\overline{\mathbf{y}}) g_i^{\rightarrow}(\overline{\mathbf{y}}) p_{\tau,i}^{\rightarrow}(\overline{\mathbf{y}}) & \text{otherwise} , \end{cases} \quad (2)$$

$$p_{\omega,i}^{\rightarrow}(\overline{\mathbf{y}}) = \begin{cases} p_\omega(\mathbf{y}_0 \to \mathbf{y}_1) & \text{if } i = 1 , \\ p_\omega(\mathbf{y}_{i-2} \to \mathbf{y}_{i-1} \to \mathbf{y}_i) & \text{if } i > 1 , \end{cases} \quad (3)$$

$$g_i^{\rightarrow}(\overline{\mathbf{y}}) = G(\mathbf{y}_{i-1}, \mathbf{y}_i) , \quad (4)$$

$$p_{\tau,i}^{\rightarrow}(\overline{\mathbf{y}}) = p_\tau(\mathbf{y}_{i-1} \to \mathbf{y}_i) . \quad (5)$$

In the equations above, $p(.)$ denotes an unconditional PDF expressed w.r.t. the Euclidean volume on $\mathbb{R}^3$, p_ω denotes a directional PDF w.r.t. the solid angle measure and p_τ denotes a distance PDF w.r.t. the Euclidean length on $\mathbb{R}^1$. Geometry factor G converts the solid angle $\times$ length product measure to a volume measure. The PDF of a subpath $\overline{\mathbf{y}}$ with s vertices is $p_s(\overline{\mathbf{y}}) = \prod_{i=0}^{s-1} p_i^{\rightarrow}(\overline{\mathbf{y}})$.

Reverse vertex PDFs.

$$p_i^{\leftarrow}(\overline{\mathbf{y}}) = \begin{cases} p(\mathbf{y}_k) & \text{if } i = k , \\ p_{\omega,i}^{\leftarrow}(\overline{\mathbf{y}}) g_i^{\leftarrow}(\overline{\mathbf{y}}) p_{\tau,i}^{\leftarrow}(\overline{\mathbf{y}}) & \text{otherwise} , \end{cases} \quad (6)$$

$$p_{\omega,i}^{\leftarrow}(\overline{\mathbf{y}}) = \begin{cases} p_\omega(\mathbf{y}_{k-1} \leftarrow \mathbf{y}_k) & \text{if } i = k-1 , \\ p_\omega(\mathbf{y}_i \leftarrow \mathbf{y}_{i+1} \leftarrow \mathbf{y}_{i+2}) & \text{if } i < k-1 , \end{cases} \quad (7)$$

$$g_i^{\leftarrow}(\overline{\mathbf{y}}) = G(\mathbf{y}_i, \mathbf{y}_{i+1}) , \quad (8)$$

$$p_{\tau,i}^{\leftarrow}(\overline{\mathbf{y}}) = p_\tau(\mathbf{y}_i \leftarrow \mathbf{y}_{i+1}) . \quad (9)$$

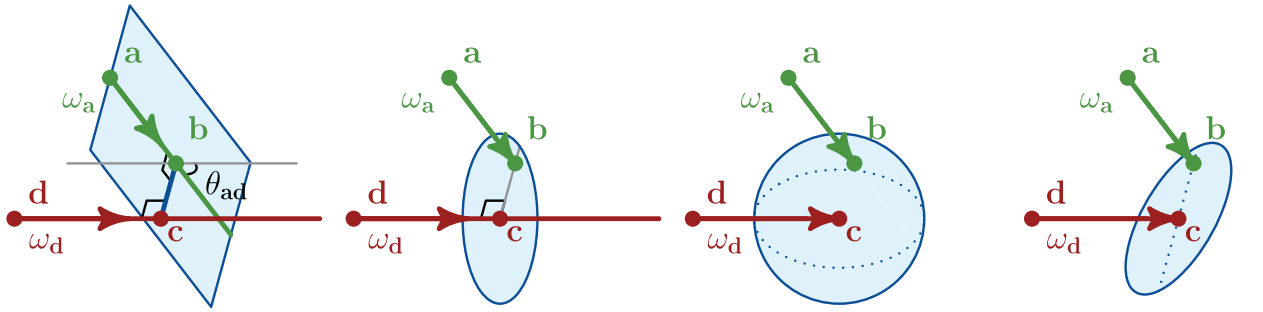
The arrow above the symbols distinguishes between a forward and a reverse PDF.

3.4. Bidirectional path tracing

BPT is a light transport algorithm that combines a light subpath with a camera subpath by connecting their endpoints with an additional segment. As a result, a k-length path can be constructed in $k+2$ different ways, yielding $k+2$ different sampling techniques. Let $\langle I \rangle_{\text{BPT},s}$ denote the BPT estimator and $p_{\text{BPT},s}$ its PDF for evaluation on a light subpath with s vertices and a camera subpath with $t = k+1-s$ vertices, $0 \le s \le k+1$. The estimator and its PDF are given by

$$\langle I \rangle_{\text{BPT},s} = L_e(\mathbf{y}_0) \frac{T(\overline{\mathbf{y}})}{p(\overline{\mathbf{y}})} \rho(\mathbf{y}_{s-1}) T_\tau(\mathbf{y}_{s-1}, \mathbf{z}_{t-1}) G(\mathbf{y}_{s-1}, \mathbf{z}_{t-1}) \rho(\mathbf{z}_{t-1}) \frac{T(\overline{\mathbf{z}})}{p(\overline{\mathbf{z}})} W_e(\mathbf{z}_0) , \quad (10)$$

$$p_{\text{BPT},s} = p(\overline{\mathbf{y}}) p(\overline{\mathbf{z}}) . \quad (11)$$



(a) Beam-Beam 1D (b) Point-Beam 2D (c) Point-Point 3D (d) Point-Point 2D

Figure 2: *The photon density estimators used in unified points, beams, and paths. (a-c) are applied for volumes, (d) for surfaces. The shared input of these estimators are the photon beam* $(\mathbf{a}, \omega_\mathbf{a})$ *and the camera beam* $(\mathbf{d}, \omega_\mathbf{d})$*. The* P-B2D*,* P-P3D*, and* P-P2D *estimators sample a photon point* **b** *and the* P-P3D *and* P-P2D *sample a camera point* **c**.

3.5. Photon density estimation

PDE combines a camera subpath with a light subpath by finding nearby endpoints of light subpaths and merging the two endpoints together. This results in one less vertex in the complete path, than the sum of vertices of the subpaths. We use Křivánek et al.'s [KGH*14] formulation of the radiance estimators in the path integral framework. Let $\mathbf{a} := \mathbf{y}_{s-2}$, $\mathbf{b} := \mathbf{y}_{s-1}$, $\mathbf{c} := \mathbf{z}_{t-1}$, $\mathbf{d} := \mathbf{z}_{t-2}$. The light subpath is sampled up to **a**, and the ray $(\mathbf{a}, \omega_\mathbf{a})$ defines a *photon beam*. Sampling a distance along the ray would create a *photon point* at **b**. The same holds on the camera side: the camera subpath is sampled up to **d** and the ray $(\mathbf{d}, \omega_\mathbf{d})$ defines a *camera beam*, sampling a distance along the camera beam creates a *camera point* **c**. This setup is shown in Figure 2. We define the contribution of the light subpath up to $\mathbf{y}_i$ as $C_\mathrm{l}(\mathbf{y}_0 \dots \mathbf{y}_i)$ and the contribution of the camera subpath up to $\mathbf{z}_j$ as $C_\mathrm{c}(\mathbf{z}_0 \dots \mathbf{z}_j)$,

$$C_\mathrm{l}(\mathbf{y}_0 \dots \mathbf{y}_i) = L_\mathrm{e}(\mathbf{y}_0)(T(\mathbf{y}_0 \dots \mathbf{y}_i)\rho(\mathbf{y}_i))/(p(\mathbf{y}_0 \dots \mathbf{y}_i)p^{\rightarrow}_{\omega,i+1}(\bar{\mathbf{y}}))\,,$$
$$C_\mathrm{c}(\mathbf{z}_0 \dots \mathbf{z}_j) = W_\mathrm{e}(\mathbf{z}_0)(T(\mathbf{z}_0 \dots \mathbf{z}_j)\rho(\mathbf{z}_j))/(p(\mathbf{z}_0 \dots \mathbf{z}_j)p^{\rightarrow}_{\omega,j+1}(\bar{\mathbf{z}}))\,.$$

The scattering function ρ at query location **c** is evaluated using the direction of the light subpath, which may or may not actually pass through this location. This is because **b** and **c** are interpreted to be the same point, but do not actually share a position. To describe this behaviour, ρ (see Equation (1)) is amended as

$$\rho(\mathbf{b}, \mathbf{c}) = \begin{cases} \rho_\mathrm{s}(\mathbf{a} \to \mathbf{b}, \mathbf{c} \to \mathbf{d}) & \text{if } \mathbf{c} \text{ is on a surface} \\ \rho_\mathrm{p}(\mathbf{a} \to \mathbf{b}, \mathbf{c} \to \mathbf{d})\sigma_\mathrm{s}(\mathbf{b}) & \text{if } \mathbf{c} \text{ is in a medium} \end{cases}. \quad (12)$$

Depending on whether or not a photon point and/or a camera point are sampled, as well as the dimension of the kernel employed for the merge, a large number of estimators exists. The estimators are assigned an acronym using the following naming scheme: the light data type (photon **P**oint or photon **B**eam), the camera data type (camera **P**oint or camera **B**eam), and the dimension of the blur employed (1D, 2D, or 3D). Křivánek et al. find the minimum-blur estimators to introduce the least amount of bias, which is why UPBP includes one minimum-blur estimator for volumetric PDE from each category. Beam-Point is not included, which they state is too costly to evaluate. In addition, a surface estimator in the form of P-P2D is included. Furthermore, Křivánek et al. distinguish between 'long' and 'short' beams. We assume the use of long beams for each of the estimators. For each of these techniques, we state the estimator function and their PDF. The PDFs of the B-B1D, P-B2D, P-P3D and P-P2D estimators are derived by Vévoda [Vév14], who use the name SURF instead of P-P2D. In these equations, K_d denotes a normalized d-dimensional kernel. All estimators are prefixed by $C_\mathrm{l}(\mathbf{y}_0 \dots \mathbf{a})$ and postfixed by $C_\mathrm{c}(\mathbf{z}_0 \dots \mathbf{d})$, all PDFs are prefixed by $p(\mathbf{y}_0 \dots \mathbf{b})$ and postfixed by $p(\mathbf{z}_0 \dots \mathbf{c})$. These terms are excluded in the equations below in favour of readability. The estimator PDFs for B-B1D and P-B2D are slightly simplified using Equation (2). Let $\langle I \rangle_{v,s}$ denote the estimator and $p_{v,s}$ the PDF for a PDE technique v that is evaluated on a light subpath with s vertices and a camera subpath with $t = k + 2 - s$ vertices, $2 \le s \le k$. For each technique, Figure 2 shows its geometric configuration, and Equations (13)–(20) show their estimators and PDF, that is,

$$\langle I \rangle_{\text{B-B1D},s} = T_\mathrm{r}(\mathbf{a}, \mathbf{b})\frac{\rho(\mathbf{b}, \mathbf{c})K_1(\mathbf{b}, \mathbf{c})}{\sin\theta_\mathbf{ad}}T_\mathrm{r}(\mathbf{c}, \mathbf{d})\,, \quad (13)$$

$$p_{\text{B-B1D},s} = \frac{p^{\rightarrow}_{\omega,s-1}(\bar{\mathbf{y}})G(\mathbf{a}, \mathbf{b})}{p(\mathbf{b})}\frac{\sin\theta_\mathbf{ad}}{K_1(\mathbf{b}, \mathbf{c})}\frac{G(\mathbf{c}, \mathbf{d})p^{\rightarrow}_{\omega,t-1}(\bar{\mathbf{z}})}{p(\mathbf{c})}\,,$$
$$= \frac{\sin\theta_\mathbf{ad}}{p^{\rightarrow}_{\tau,s-1}(\bar{\mathbf{y}})K_1(\mathbf{b}, \mathbf{c})p^{\rightarrow}_{\tau,t-1}(\bar{\mathbf{z}})}\,, \quad (14)$$

$$\langle I \rangle_{\text{P-B2D},s} = \frac{T_\mathrm{r}(\mathbf{a}, \mathbf{b})}{p^{\rightarrow}_{\tau,s-1}(\bar{\mathbf{y}})}\rho(\mathbf{b}, \mathbf{c})K_2(\mathbf{b}, \mathbf{c})T_\mathrm{r}(\mathbf{c}, \mathbf{d})\,, \quad (15)$$

$$p_{\text{P-B2D},s} = K_2^{-1}(\mathbf{b}, \mathbf{c})\frac{G(\mathbf{b}, \mathbf{c})p^{\rightarrow}_{\omega,t-1}(\bar{\mathbf{z}})}{p(\mathbf{c})} = \frac{K_2^{-1}(\mathbf{b}, \mathbf{c})}{p^{\rightarrow}_{\tau,t-1}(\bar{\mathbf{z}})}\,, \quad (16)$$

$$\langle I \rangle_{\text{P-P3D},s} = \frac{T_\mathrm{r}(\mathbf{a}, \mathbf{b})}{p^{\rightarrow}_{\tau,s-1}(\bar{\mathbf{y}})}\rho(\mathbf{b}, \mathbf{c})K_3(\mathbf{b}, \mathbf{c})\frac{T_\mathrm{r}(\mathbf{c}, \mathbf{d})}{p^{\rightarrow}_{\tau,t-1}(\bar{\mathbf{z}})}\,, \quad (17)$$

$$p_{\text{P-P3D},s} = K_3^{-1}(\mathbf{b}, \mathbf{c})\,, \quad (18)$$

$$\langle I\rangle_{\text{P-P2D},s} = \frac{T_r(\mathbf{a},\mathbf{b})}{p^{\rightarrow}_{\tau,s-1}(\overline{\mathbf{y}})}\rho(\mathbf{b},\mathbf{c})K_2(\mathbf{b},\mathbf{c})\frac{T_r(\mathbf{c},\mathbf{d})}{p^{\rightarrow}_{\tau,t-1}(\overline{\mathbf{z}})}\,, \tag{19}$$

$$p_{\text{P-P2D},s} = K_2^{-1}(\mathbf{b},\mathbf{c})\,. \tag{20}$$

3.6. Unified points, beams, and paths

To capitalize on the strengths and to mitigate the weaknesses of the individual estimators, *unified point, beams, and paths* [KGH*14] (UPBP) combines the BPT and PDE estimators using MIS. With MIS, the contributions of multiple estimators are summed according to the number of samples and a weighting function. For UPBP, this results in the estimator

$$\begin{aligned}
\langle I\rangle_{\text{UPBP}} = \frac{1}{n_{\text{BPT}}} &\sum_{i=1}^{n_{\text{BPT}}}\sum_{s=0}^{k+1} w_{\text{BPT},s,t}(\overline{\mathbf{x}}_i)\quad \langle I\rangle_{\text{BPT},s}(\overline{\mathbf{x}}_i)\\
+\frac{1}{n_{\text{B-B1D}}} &\sum_{i=1}^{n_{\text{B-B1D}}}\sum_{s=2}^{k} w_{\text{B-B1D},s,t}(\overline{\mathbf{x}}_i)\quad \langle I\rangle_{\text{B-B1D},s}(\overline{\mathbf{x}}_i)\\
+\frac{1}{n_{\text{P-B2D}}} &\sum_{i=1}^{n_{\text{P-B2D}}}\sum_{s=2}^{k} w_{\text{P-B2D},s,t}(\overline{\mathbf{x}}_i)\quad \langle I\rangle_{\text{P-B2D},s}(\overline{\mathbf{x}}_i)\\
+\frac{1}{n_{\text{P-P3D}}} &\sum_{i=1}^{n_{\text{P-P3D}}}\sum_{s=2}^{k} w_{\text{P-P3D},s,t}(\overline{\mathbf{x}}_i)\quad \langle I\rangle_{\text{P-P3D},s}(\overline{\mathbf{x}}_i)\\
+\frac{1}{n_{\text{P-P2D}}} &\sum_{i=1}^{n_{\text{P-P2D}}}\sum_{s=2}^{k} w_{\text{P-P2D},s,t}(\overline{\mathbf{x}}_i)\quad \langle I\rangle_{\text{P-P2D},s}(\overline{\mathbf{x}}_i)\,.
\end{aligned} \tag{21}$$

In the above estimator, it becomes clear how many different techniques UPBP actually evaluates. It estimates the pixel value found by tracing a single camera subpath through the pixel, being merged with and connected to light subpaths using technique v. Here, n_v denotes the number of light subpaths used to evaluate technique v. $w_{v,s,t}$ is the weight for technique v evaluated on a light subpath of s vertices and a camera subpath of t vertices. In Section 3.4, t is defined as $k+1-s$ for BPT and in Section 3.5, t is defined as $k+2-s$ for PDE. Note that the PDE techniques cannot be evaluated on a light source or on the camera lens, hence the subpaths need at least two vertices. A provably good and frequently used choice for the weighting function is the balance heuristic

$$\hat{w}_v(\overline{\mathbf{x}}) = n_v p_v(\overline{\mathbf{x}}) / \sum_{l=1}^{m} n_l p_l(\overline{\mathbf{x}})\,.$$

The balance heuristic assigns a weight $\hat{w}_v$ to each technique v proportional to the PDFs of all m techniques, which minimizes variance. For UPBP, this results in Equation (22), where $p_{v,s}$ denotes the PDF of a path created by technique v from a subpath with s vertices on the light subpath, that is,

$$\begin{aligned}
w_{v,s,t}(\overline{\mathbf{x}}) &= \frac{n_v p_{v,s}(\overline{\mathbf{x}})}{g_v(\overline{\mathbf{x}})}\\
g_v(\overline{\mathbf{x}}) &= \sum_{j=0}^{k+1} n_{\text{BPT}} p_{\text{BPT},j}(\overline{\mathbf{x}})\\
&\quad + \sum_{j=2}^{k} n_{\text{B-B1D}} p_{\text{B-B1D},j}(\overline{\mathbf{x}}) + n_{\text{P-B2D}} p_{\text{P-B2D},j}(\overline{\mathbf{x}})\\
&\quad + \sum_{j=2}^{k} n_{\text{P-P3D}} p_{\text{P-P3D},j}(\overline{\mathbf{x}}) + n_{\text{P-P2D}} p_{\text{P-P2D},j}(\overline{\mathbf{x}})\,.
\end{aligned} \tag{22}$$

A single iteration of UPBP is performed in two stages. In the first stage, light subpaths $\overline{\mathbf{y}}$ are generated and the vertices on surfaces and the vertices and beams in media are stored. Data structures are created for the accelerated lookup of the vertices and beams. In Section 5, we describe a data structure that leverages ray tracing hardware. In the second stage, one camera subpath $\overline{\mathbf{z}}$ is generated for each pixel. At every camera beam and vertex generated, all relevant PDE estimators are evaluated by querying the data structures for nearby beams and points. At every camera vertex, the BPT estimator is evaluated with all vertices of a randomly chosen light subpath that was stored in the first phase. Each contribution is multiplied by the MIS weight and added to the pixel value. The light and camera subpaths are stochastically terminated up to a user-defined maximum path length. Next section presents an efficient algorithm for the computation of the MIS weight.

4. Recursive Algorithm

At the heart of UPBP lies the path weight computation, performed each time a light path $\overline{\mathbf{x}}$ is formed by evaluating technique v on a light subpath with s vertices and a camera subpath with t vertices. A weight $w_{v,s,t}(\overline{\mathbf{x}})$ is computed for this path, with which the radiance along the light path is multiplied before accumulating. This weight depends on the path PDF, which is the product of the PDFs of the path vertices. As subpaths are extended, the calculated probability density of the vertices earlier in the path remains constant. Hence, the part of this product that represents vertices earlier in the path remains constant. If we can reformulate the above weight to split out these constant terms, it can be computed more efficiently by caching a part of it. We will explain below how a set of recursive quantities can be maintained as both subpaths are generated, which represent the subpath weights. When the subpaths are combined to form a light path, the full weight can be obtained by only taking into account the quantities stored at the light and camera vertex. This allows for its progressive computation, similar to how the estimator contributions in Equations (10), (13), (15), (17), and (19) typically are computed. In this section, we omit $\overline{\mathbf{x}}$ when it is clear what path is used.

4.1. Subpath weights

Before we derive a recursive formulation of the path weight for the subpaths, we first write Equation (22) such that we separate out one

term for the light subpath and one term for the camera subpath. We rewrite the weight as

$$\begin{aligned} w_{v,s,t} &= \frac{1}{h_{v,s}} \\ h_{v,s} &= \sum_{j=0}^{k+1} \frac{n_{\text{BPT}} p_{\text{BPT},j}}{n_v p_{v,s}} \\ &\quad + \sum_{j=2}^{k} \frac{n_{\text{B-B1D}} p_{\text{B-B1D},j} + n_{\text{P-B2D}} p_{\text{P-B2D},j}}{n_v p_{v,s}} \\ &\quad + \sum_{j=2}^{k} \frac{n_{\text{P-P3D}} p_{\text{P-P3D},j} + n_{\text{P-P2D}} p_{\text{P-P2D},j}}{n_v p_{v,s}} . \end{aligned} \tag{23}$$

We now split the weight into three parts: one sum that depends on only light subpath vertices $w_{v,s}^{\text{light}}$, one quantity that depends on the "local" information of the connection or merge $w_{v,s}^{\text{local}}$, and one sum that depends only on camera subpath vertices $w_{v,s}^{\text{camera}}$, *i.e.*

$$w_{v,s,t} = 1/(w_{v,s}^{\text{light}} + w_{v,s}^{\text{local}} + w_{v,s}^{\text{camera}}) . \tag{24}$$

The two partial sums and local quantity have different shapes depending on technique v for which we compute the weight. We can derive a common formula for the PDE techniques, as their PDFs share a common form. For $v \in \{\text{B-B1D}, \text{P-B2D}, \text{P-P3D}, \text{P-P2D}\}$, we write

$$p_{v,s} = p_s(\overline{\mathbf{y}}) e_{v,s} p_t(\overline{\mathbf{z}}) , \tag{25}$$

where $e_{v,s}$ is the *estimator specific* term of technique v in Equations (14), (16), (18), and (20). Note that we cannot write the BPT PDF in the same way, as its subpaths have different lengths than the PDE subpaths. We will now define $w_{v,s}^{\text{light}}$, $w_{v,s}^{\text{local}}$, and $w_{v,s}^{\text{camera}}$ for all estimators. Since we have a common form for the PDE PDFs, we can also group the derivation of these quantities for PDE.

Bidirectional path tracing. For $v = \text{BPT}$ we have

$$\begin{aligned} w_{v,s}^{\text{light}} &= \sum_{j=0}^{s-1} \frac{p_{\text{BPT},j}}{p_{\text{BPT},s}} \\ &\quad + \sum_{j=2}^{s} \frac{n_{\text{B-B1D}} p_{\text{B-B1D},j} + n_{\text{P-B2D}} p_{\text{P-B2D},j}}{n_{\text{BPT}} p_{\text{BPT},s}} , \\ &\quad + \sum_{j=2}^{s} \frac{n_{\text{P-P3D}} p_{\text{P-P3D},j} + n_{\text{P-P2D}} p_{\text{P-P2D},j}}{n_{\text{BPT}} p_{\text{BPT},s}} \end{aligned} \tag{26}$$

$$w_{v,s}^{\text{local}} = \sum_{j=s}^{s} \frac{p_{\text{BPT},j}}{p_{\text{BPT},j}} = 1 , \tag{27}$$

$$\begin{aligned} w_{v,s}^{\text{camera}} &= \sum_{j=s+1}^{k+1} \frac{p_{\text{BPT},j}}{p_{\text{BPT},s}} \\ &\quad + \sum_{j=s+1}^{k} \frac{n_{\text{B-B1D}} p_{\text{B-B1D},j} + n_{\text{P-B2D}} p_{\text{P-B2D},j}}{n_{\text{BPT}} p_{\text{BPT},s}} . \\ &\quad + \sum_{j=s+1}^{k} \frac{n_{\text{P-P3D}} p_{\text{P-P3D},j} + n_{\text{P-P2D}} p_{\text{P-P2D},j}}{n_{\text{BPT}} p_{\text{BPT},s}} \end{aligned} \tag{28}$$

An important property of the light and camera sums is that they are symmetric. If we take the perspective of the camera subpath and reverse the numbering of the vertices, the weight of the camera subpath becomes the same sum as the light subpath weight above (but for t instead of s). We use this symmetric property to only derive the recursive quantities for both sums by making a single derivation.

Photon density estimation. For $v \in P$ we have

$$\begin{aligned} w_{v,s}^{\text{light}} &= \frac{n_{\text{BPT}}}{n_v} \sum_{j=0}^{s-1} \frac{p_{\text{BPT},j}}{p_{v,s}} \\ &\quad + \sum_{j=2}^{s-1} \frac{n_{\text{B-B1D}} p_{\text{B-B1D},j} + n_{\text{P-B2D}} p_{\text{P-B2D},j}}{n_v p_{v,s}} , \\ &\quad + \sum_{j=2}^{s-1} \frac{n_{\text{P-P3D}} p_{\text{P-P3D},j} + n_{\text{P-P2D}} p_{\text{P-P2D},j}}{n_v p_{v,s}} \end{aligned} \tag{29}$$

$$w_{v,s}^{\text{local}} = \sum_{j=s}^{s} \sum_{b \in P} \frac{n_b p_{b,s}}{n_v p_{v,s}} = 1 + \sum_{b \in P \setminus \{v\}} \frac{n_b e_{b,s}}{n_v e_{v,s}} , \tag{30}$$

$$\begin{aligned} w_{v,s}^{\text{camera}} &= \frac{n_{\text{BPT}}}{n_v} \sum_{j=s}^{k+1} \frac{p_{\text{BPT},j}}{p_{v,s}} \\ &\quad + \sum_{j=s+1}^{k} \frac{n_{\text{B-B1D}} p_{\text{B-B1D},j} + n_{\text{P-B2D}} p_{\text{P-B2D},j}}{n_v p_{v,s}} . \\ &\quad + \sum_{j=s+1}^{k} \frac{n_{\text{P-P3D}} p_{\text{P-P3D},j} + n_{\text{P-P2D}} p_{\text{P-P2D},j}}{n_v p_{v,s}} \end{aligned} \tag{31}$$

The same symmetry property holds also for the PDE sums. Note that the local quantity does not reduce to 1 as for BPT, but instead we have to take all PDE techniques in P into account. Here, $\setminus$ denotes set difference.

4.2. Recursive formulation

Now that we have a term that represents the subpath weight, we want to formulate it such that it can be computed in a forward manner, rather than to have to solve the sum by iterating over the vertices. To this end, we write the weight as

$$w_{v,s,t} = 1/(\overline{w}_{v,s-1}(\overline{\mathbf{y}}) + w_{v,s}^{\text{local}} + \overline{w}_{v,t-1}(\overline{\mathbf{z}})) , \tag{32}$$

with $\overline{w}_{v,s-1}(\overline{\mathbf{y}})$ and $\overline{w}_{v,t-1}(\overline{\mathbf{z}})$ recursive formulations of the partial light and camera subpath weights $w_{v,s}^{\text{light}}$ and $w_{v,s}^{\text{camera}}$, respectively.

We derive these formulas only for the light subpath, as the derivations also apply to the camera subpath due to symmetry of Equations (26) and (28) as well as of Equations (29) and (31). The recursive formulation requires us to address how to evaluate $p_{v,i}$ for $i < s$, having unidirectionally sampled a path $\overline{\mathbf{y}}$. We do this by interpreting the path from s to i as having been sampled from the camera, by maintaining reverse probabilities as we sample $\overline{\mathbf{y}}$. For the PDE techniques, there is an additional complication as the kernel in the estimator-specific factors depends on the mutual position of the merged light and camera vertex. These do not actually exist, as the subpath of s vertices is unidirectionally sampled and only a light vertex exists at vertex $\mathbf{y}_i$. To deal with this, we assume a constant kernel that is not dependent on these positions, which is the same assumption Vévoda[Vév14] makes.

For convenience, we define $f_i(\overline{\mathbf{y}})$ to be the sum of all $e_{v,i}(\overline{\mathbf{y}})$ times their respective n_v terms for $v \in P$. Note that $f_i(\overline{\mathbf{z}})$ is not equal to $f_i(\overline{\mathbf{y}})$, as the estimator-specific term for P-B2D is not symmetric with respect to the sample direction.

4.2.1. *Bidirectional path tracing*

Filling in the path PDF definitions in Equation (26), we can cancel out large parts of the PDFs. This results in the equation below for $v = \text{BPT}$, the derivation of which is provided in the supplementary material

$$w_{v,s}^{\text{light}} = \sum_{j=0}^{s-1} \prod_{i=j}^{s-1} \frac{p_i^{\leftarrow}(\overline{\mathbf{y}})}{p_i^{\rightarrow}(\overline{\mathbf{y}})} + \frac{1}{n_{\text{BPT}}} \sum_{j=2}^{s} f_j(\overline{\mathbf{y}}) p_{j-1}^{\leftarrow}(\overline{\mathbf{y}}) \prod_{i=j}^{s-1} \frac{p_i^{\leftarrow}(\overline{\mathbf{y}})}{p_i^{\rightarrow}(\overline{\mathbf{y}})} .$$

We write the above equation as the following recursive quantity

$$q_{\text{BPT},0} = \frac{p_0^{\leftarrow}}{p_0^{\rightarrow}} , \quad q_{\text{BPT},i} = p_i^{\leftarrow} \left(\frac{f_{i+1}}{n_{\text{BPT}}} + \frac{1}{p_i^{\rightarrow}} + \frac{1}{p_i^{\rightarrow}} q_{\text{BPT},i-1} \right),$$

with which we write the light subpath weight for BPT as

$$\overline{w}_{\text{BPT},s-1} := q_{\text{BPT},s-1} . \tag{33}$$

If instead of the light subpath notation $\overline{\mathbf{y}}$ we use camera subpath notation $\overline{\mathbf{z}}$, the above formulas apply without modification to Equation (28), since the light and camera sums are symmetric. This now allows us to substitute $\overline{w}_{\text{BPT},s-1}(\overline{\mathbf{y}})$ and $\overline{w}_{\text{BPT},t-1}(\overline{\mathbf{z}})$ into Equation (32).

4.2.2. *Photon density estimation*

Filling in the path PDF definitions in Equation (29), we again can cancel out large parts of the PDFs. This results in the equation below for $v \in P$, the derivation of which is also given in the supplementary material, that is,

$$w_{v,s}^{\text{light}} = \frac{1}{n_v e_{v,s}} \left(\frac{n_{\text{BPT}}}{p_{s-1}^{\rightarrow}} \sum_{j=0}^{s-1} \prod_{i=j}^{s-2} \frac{p_i^{\leftarrow}(\overline{\mathbf{y}})}{p_i^{\rightarrow}(\overline{\mathbf{y}})} + \sum_{j=2}^{s-1} f_j(\overline{\mathbf{y}}) \prod_{i=j}^{s-1} \frac{p_{i-1}^{\leftarrow}(\overline{\mathbf{y}})}{p_i^{\rightarrow}(\overline{\mathbf{y}})} \right).$$

We write the term within brackets in the above equation as the following recursive quantity

$$q_{\text{PDE},0} = \frac{n_{\text{BPT}}}{p_0^{\rightarrow}} , \quad q_{\text{PDE},1} = \frac{n_{\text{BPT}}}{p_1^{\rightarrow}} \left(\frac{p_0^{\leftarrow}}{p_0^{\rightarrow}} + 1 \right),$$
$$q_{\text{PDE},i} = n_{\text{BPT}} \frac{1}{p_i^{\rightarrow}} + \frac{p_{i-1}^{\leftarrow}}{p_i^{\rightarrow}} (f_i + q_{\text{PDE},i-1}) ,$$

with which we write the light subpath weight for $v \in P$ as

$$\overline{w}_{v,s-1} := q_{\text{PDE},s-1} / n_v e_{v,s} . \tag{34}$$

If instead of the light subpath notation $\overline{\mathbf{y}}$ we use camera subpath notation $\overline{\mathbf{z}}$, the above formulas apply without modification to Equation (31), since the light and camera sums are symmetric. This now allows us to substitute $\overline{w}_{v,s-1}(\overline{\mathbf{y}})$ and $\overline{w}_{v,t-1}(\overline{\mathbf{z}})$ for $v \in P$ into Equation (32).

4.3. Forward evaluation

Now that we have derived two recursive quantities, q_{BPT} and q_{PDE}, with which we can construct the weight $\overline{w}_v$ for each estimator v, we would like to store these quantities at every light and camera vertex, as subpaths are traced. This allows us to construct the path weight only from information stored at these two vertices. However, we cannot evaluate $q_{\text{BPT},i}$ and $q_{\text{PDE},i}$ as soon as we sample vertex i. Both recursive quantities depend on reverse probabilities that are not known when sampling subpath vertex i: $p_i^{\leftarrow}$ depends on the next two vertices via $p_\omega(\mathbf{y}_i \leftarrow \mathbf{y}_{i+1} \leftarrow \mathbf{y}_{i+2})$; similarly, $p_{i-1}^{\leftarrow}$ depends on the next vertex. To deal with this problem, we separate three terms that can be stored at vertex i. From these three quantities, the actual weights can be constructed in a constant amount of operations. Note that f_i can be stored at i, as $\sin\theta_{\mathbf{y}_{i-2}\mathbf{y}_i}$ is known after sampling the BSDF at $i-1$ and $p_\tau^{\leftarrow}(\mathbf{y}_{i-1}, \mathbf{y}_i)$ is known when the next vertex is found. We will now repeat q_{BPT} and q_{PDE} and separate these quantities, that is,

$$\begin{aligned} q_{\text{BPT},i} &= p_i^{\leftarrow} \left(\frac{f_{i+1}}{n_{\text{BPT}}} + \frac{1}{p_i^{\rightarrow}} + \frac{1}{p_i^{\rightarrow}} q_{\text{BPT},i-1} \right) \\ &\overset{1}{=} p_i^{\leftarrow} \Bigg(\frac{f_{i+1}}{n_{\text{BPT}}} + \underbrace{\frac{1}{p_i^{\rightarrow}}}_{d_i^{\text{shared}}} + \underbrace{p_{\tau,i-1}^{\leftarrow} p_{\omega,i-1}^{\leftarrow} \frac{g_{i-1}^{\leftarrow}}{p_{i-1}^{\leftarrow} p_i^{\rightarrow}} q_{\text{BPT},i-1}}_{d_i^{\text{BPT}}} \Bigg), \end{aligned} \tag{35}$$

$$\begin{aligned} q_{\text{PDE},i} &= n_{\text{BPT}} \frac{1}{p_i^{\rightarrow}} + \frac{p_{i-1}^{\leftarrow}}{p_i^{\rightarrow}} (f_i + q_{\text{PDE},i-1}) \\ &\overset{1}{=} n_{\text{BPT}} \underbrace{\frac{1}{p_i^{\rightarrow}}}_{d_i^{\text{shared}}} + \underbrace{p_{\tau,i-1}^{\leftarrow} p_{\omega,i-1}^{\leftarrow} \frac{g_{i-1}^{\leftarrow}}{p_i^{\rightarrow}} \left(f_i + q_{\text{PDE},i-1} \right)}_{d_i^{\text{PDE}}}, \end{aligned} \tag{36}$$

where in steps (1) we make use of Equation (6). Note that d_i^{shared} appears in both $q_{\text{BPT},i}$ and $q_{\text{PDE},i}$. These formulas apply to both the light subpaths $\overline{\mathbf{y}}$ and camera subpaths $\overline{\mathbf{z}}$.

As the subpath is traced, we update and store quantities d_i^{shared}, d_i^{BPT}, and d_i^{PDE} at each vertex i. For practical reasons, the first ver-

tices ($\mathbf{y}_0$ and $\mathbf{z}_0$) are not stored, see Section 4.4. We obtain the initialization of these quantities by writing out $q_{\text{BPT},1}$ and $q_{\text{PDE},1}$ and factoring out the terms, that is,

$$d_1^{\text{shared}} = 1/p_1^{\rightarrow}\,, \tag{37}$$

$$d_1^{\text{BPT}} = g_0^{\leftarrow}/(p_0^{\rightarrow} p_1^{\rightarrow})\,, \tag{38}$$

$$d_1^{\text{PDE}} = (g_0^{\leftarrow} n_{\text{BPT}})/(p_0^{\rightarrow} p_1^{\rightarrow})\,. \tag{39}$$

We find the recursive formulation of the d-quantities by recursively expanding $q_{\text{BPT},i-1}$ and $q_{\text{PDE},i-1}$ in Equations (35) and (36) to obtain

$$d_i^{\text{shared}} = 1/p_i^{\rightarrow}\,, \tag{40}$$

$$d_i^{\text{BPT}} = \frac{g_{i-1}^{\leftarrow}}{p_i^{\rightarrow}}\left(\frac{f_i}{n_{\text{BPT}}} + d_{i-1}^{\text{shared}} + p_{\tau,i-2}^{\leftarrow} p_{\omega,i-2}^{\leftarrow} d_{i-1}^{\text{BPT}}\right), \tag{41}$$

$$d_i^{\text{PDE}} = \frac{g_{i-1}^{\leftarrow}}{p_i^{\rightarrow}}\left(f_i + n_{\text{BPT}} d_{i-1}^{\text{shared}} + p_{\tau,i-2}^{\leftarrow} p_{\omega,i-2}^{\leftarrow} d_{i-1}^{\text{PDE}}\right). \tag{42}$$

We now know how to initialize the d-quantities and update them as the subpaths are traced. From these quantities, q_{BPT} and q_{PDE} are constructed. Recall that the weight of a full path constructed by technique v from a light subpath $\overline{\mathbf{y}}$ with s vertices and a camera subpath $\overline{\mathbf{z}}$ with t vertices is given by Equation (32). For each technique, we will now explain how the full path weight is constructed from the quantities stored at the connected or merged vertices.

4.3.1. *Bidirectional path tracing*

The subpath weight for BPT is given by Equation (33). The quantity q_{BPT} is constructed from recursive quantities given in Equation (35). Combining this for a light subpath $\overline{\mathbf{y}}$ with s vertices and a camera subpath $\overline{\mathbf{z}}$ with t vertices gives (for $v = \text{BPT}$)

$$\overline{w}_{v,s-1}(\overline{\mathbf{y}}) = p_{s-1}^{\leftarrow}\left(\frac{f_s}{n_{\text{BPT}}} + d_{s-1}^{\text{shared}} + p_{\tau,s-2}^{\leftarrow} p_{\omega,s-2}^{\leftarrow} d_{s-1}^{\text{BPT}}\right), \tag{43}$$

$$\overline{w}_{v,t-1}(\overline{\mathbf{z}}) = p_{t-1}^{\leftarrow}\left(\frac{f_t}{n_{\text{BPT}}} + d_{t-1}^{\text{shared}} + p_{\tau,t-2}^{\leftarrow} p_{\omega,t-2}^{\leftarrow} d_{t-1}^{\text{BPT}}\right). \tag{44}$$

Recall that $w_{v,s}^{\text{local}} = 1$. Note that the above equations only hold for the general case, in which $s > 1$ and $t > 1$. This is because we do not store the recursive quantities at the first vertices. In the next section we discuss how to handle the cases when $s \leq 1$ or $t \leq 1$.

4.3.2. *Photon density estimation*

The subpath weight for $v \in P$ is given by Equation (34). The quantity q_{PDE} is constructed from the recursive quantities as given in Equation (36). Combining this for a light subpath $\overline{\mathbf{y}}$ with s vertices and a camera subpath $\overline{\mathbf{z}}$ with t vertices yields

$$\overline{w}_{v,s-1}(\overline{\mathbf{y}}) = \frac{1}{n_v e_{v,s}}\left(n_{\text{BPT}} d_{s-1}^{\text{shared}} + p_{\tau,s-2}^{\leftarrow} p_{\omega,s-2}^{\leftarrow} d_{s-1}^{\text{PDE}}\right), \tag{45}$$

$$\overline{w}_{v,t-1}(\overline{\mathbf{z}}) = \frac{1}{n_v e_{v,s}}\left(n_{\text{BPT}} d_{t-1}^{\text{shared}} + p_{\tau,t-2}^{\leftarrow} p_{\omega,t-2}^{\leftarrow} d_{t-1}^{\text{PDE}}\right). \tag{46}$$

Note that both these equations contain $e_{v,s}$, which is the estimator-specific term for technique v on a light subpath with s and a camera subpath with t vertices; $w_{v,s}^{\text{local}}$ is given by Equation (30). Note that PDE techniques are only evaluated when $s > 1$ and $t > 1$, so no special care is required when this is not the case.

With these equations out of the way, we now focus on implementation. In the first phase of UPBP, as light subpaths are generated, quantities d^{shared}, d^{BPT}, and d^{PDE} are maintained and stored in memory for each vertex. In the second phase, camera subpaths are generated and d^{shared}, d^{BPT}, and d^{PDE} are computed for the camera side. When a technique is evaluated using the current camera subpath and a light subpath, the d-quantities for the light subpath and camera subpath are used to fill Equations (43) and (44), or Equations (45) and (46), depending on the technique used. Finally, these two quantities are plugged in Equation (24) using either Equation (27) or Equation (30).

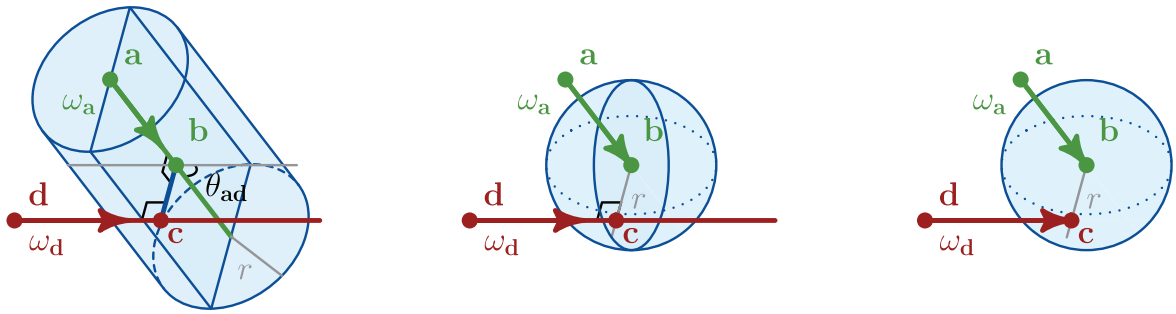
4.4. Special cases

In our derivation we have made a couple of assumptions. Specifically, we disregarded that $p_{\text{BPT},s,0} = 0$ due to the probability of hitting a pinhole camera lens being zero. Furthermore, $p_{\text{BPT},s,1}$ actually needs multiplication by the number of light paths, as every light vertex is connected to the camera. Moreover, in practice the probability $p_0^{\rightarrow}$ is different depending on whether a vertex is sampled on the light source or camera lens, to connect to a subpath or form the start of a new subpath. Finally, in the above scheme we disregarded BPT techniques when $s \leq 1$ or $t \leq 1$, since the subpath weights in these cases are never stored. In the supplementary material we lift all these assumptions and we adapt the scheme to take these facts into account.

As not all estimators may be evaluated on all combinations of vertices, the following must be ensured by an implementation: the PDF of the BPT estimator should be zero when $\mathbf{y}_{s-1}$ or $\mathbf{z}_{t-1}$ is on a specular surface. The PDF of the B-B1D, P-B2D, and P-P3D estimators should be zero when $\mathbf{y}_{s-1}$ or $\mathbf{z}_{t-1}$ are not in media. Finally, the PDF of the P-P2D estimator should be zero when $\mathbf{y}_{s-1}$ or $\mathbf{z}_{t-1}$ are in media or on a specular surface.

5. Photon Query Using Ray Tracing

Having presented an efficient method to compute the MIS weights for UPBP, we will focus on how to efficiently implement photon queries for points and beams. We do this by formulating the query as a ray-tracing problem, such that hardware-accelerated ray tracing frameworks can be used to solve it. We have selected OptiX [NVIb] for this purpose as, in recent versions, it makes use of the RTX GPU's hardware-accelerated BVH. We assume the reader to be familiar with OptiX and RTX, and refer to the OptiX programming guide [NVIa] and Turing whitepaper [Bur18] for further details. For



(a) The B-B1D estimator uses a photon beam (**a**, ω_a). Storing a cylinder based on the photon beam allows a camera beam (**d**, ω_d) query with an arbitrary direction to call an intersection program upon intersection with the AABB of the cylinder. The intersection program then either rejects or computes points **b** and **c**.

(b) The P-B2D estimator estimator uses a photon point **b**. Storing a sphere based on this point allows a camera beam (**d**, ω_d) query with an arbitrary direction to call an intersection program upon intersection with the AABB of the sphere. The intersection program computes point **c**.

(c) The P-P3D and SURF estimators use a photon point **b**. Storing a sphere centered on this point allows a camera point **c** query to register the point upon intersection with the AABB of the sphere.

Figure 3: *For each estimator, we show the 3D volume stored by the BVH in order to use ray tracing hardware to accelerate the photon query. In each figure, radius r denotes the width of the kernel.*

each PDE estimator in UPBP, we will present a bounding box program that produces an axis-aligned bounding box (AABB) and an intersection program that rejects or confirms ray-AABB intersections. Using these two components, we can evaluate the estimator when needed.

In the construction of these data structures, the width of the estimator kernel is important. It denotes the maximum distance between points or beams taken into account by the estimator. Note that in this context, a beam has no thickness and is equivalent to a ray. Furthermore, the kernel width is often called the radius of the estimator; in our work, we set the radii as in Vévoda [Vév14].

Beam-Beam, one-dimensional

The B-B1D estimator considers a one-dimensional blur between a photon beam and a camera beam. This setup is shown in Figure 2a. A one-dimensional blur is performed on the segment that is found by the intersection of the camera beam with the plane perpendicular to both photon beam and camera beam, which is the rectangular plane in the figure. We cannot store the plane directly in the BVH, as its orientation for the estimator depends on the mutual positions of the photon and camera beams. Therefore, we must consider an arbitrary orientation of this plane, as the stored representative must function for any arbitrary camera ray. Since it must be perpendicular to the photon beam, this results in a cylinder centred on the photon beam with a radius equal to the width of the 1D kernel, see Figure 3a.

Storing an AABB of this cylinder can result in a bounding box with much empty space. To remedy this, we use the technique described by Wald et al. [WMZ*20]. They describe how an instance transform of a unit cylinder can be used to perform an initial rejection test, which can be performed in hardware. This effectively implements an oriented bounding box, which encloses the beam much more tightly. Only when the oriented bounding box is intersected, the costly software intersection test is performed.

The query ray of this data structure is equal to the camera beam, with which the hardware checks for an intersection with the AABB. If an intersection is found, an intersection program checks whether the ray intersects the perpendicular plane. This additional test is required as, for example, the camera beam may be parallel to the photon beam, intersecting the cylinder but not the plane. When the intersection program finds that the camera beam intersects the plane, based on the mutual orientations of the photon and camera beams, points **b** and **c** are found and the estimator is evaluated.

Point-Beam, two-dimensional

The P-B2D estimator considers a two-dimensional blur between a photon point and a camera beam, see Figure 2b. The blur is performed on the disk in which the photon point lies, perpendicular to the camera ray. Like the Beam-Beam estimator, we cannot store this disk directly as its orientation depends on the camera ray. Instead, we consider an arbitrary rotation of this disk, which results in a sphere with a radius equal to the kernel width. This is shown in Figure 3b; we store the AABB of the resulting sphere in the BVH.

The ray with which the data structure is queried, is equal to the camera beam. If the ray hits the AABB, the intersection program checks whether the ray intersects the disk that is centred at the photon point and is perpendicular to the ray. This test is required as the ray origin or endpoint may lie in the sphere, in which case the sphere is hit but the disk may not be. When the intersection program finds the camera beam to be intersecting the disk, point **c** is found.

Point-Point, two- and three-dimensional

For P-P3D and P-P2D, a three- or two-dimensional blur is performed between a photon point and a camera point, see Figure 2c and Figure 2d. The blur is performed either on the 3D positions of the points or their 2D positions, by interpreting them to lie in the

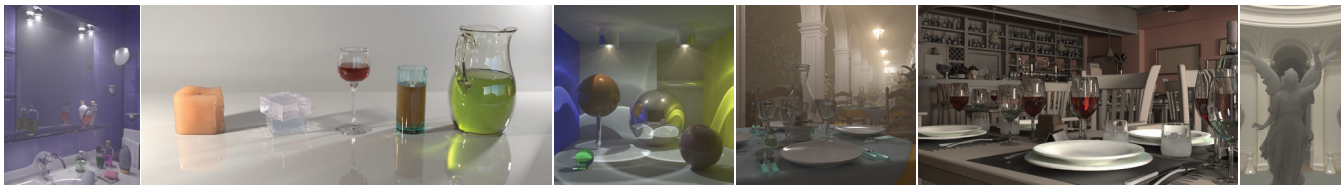


Figure 4: *Benchmark scenes.* Left-to-right*: Bathroom, Still Life, Mirror Balls, San Miguel, Bistro, and Sun Temple; results obtained with our GPU renderer.*

same plane. We store the AABB of a sphere centred on the photon point with a radius equal to the kernel width, shown in Figure 3c. If a camera point is inside the sphere, we know that the distance between the points is smaller or equal to the radius of the sphere.

One problem with the point-point estimator is that the query is a point, whereas the ray tracing framework requires a ray. To remedy this, the trace call is made with a ray with a very small extent. The origin of the ray is equal to the point and an arbitrary direction is chosen. The two-dimensional case is exactly how Evangelou et al. [EPVV21] use the RTX BVH as a photon map data structure.

6. Results

We separately evaluate a full GPU implementation of UPBP and the individual GPU data structures. We compare our GPU implementations with SmallUPBP [Vév15], which is the CPU-based implementation of UPBP used by Křivánek et al. [KGH*14] and described in detail in [Vév14]. All experiments are performed on a system with an AMD Ryzen 5 2600 CPU with six cores and 3.4 GHz, 16 GB of RAM, and an NVIDIA RTX 2070 GPU.

In all experiments, we used six different scenes that contain both sparse and dense media, see Figure 4. Bathroom, Still Life, and Mirror Balls appear in [KGH*14] and are run with the same parameters. A full list of scene parameters is provided in the supplementary materials. This list includes the number of threads used, as this for the CPU implementation is limited by the available system RAM.

6.1. GPU implementation

We have implemented UPBP to completely run on the GPU using our new MIS weight algorithm and accelerated data structures. Furthermore, this implementation also makes use of the RTX hardware for the generation of light and camera paths. We make the source code publicly available at https://github.com/nolmoonen/gpuupbp. Figure 5 shows the performance compared to a CPU implementation running single- and multi-threaded. The running times of the CPU implementation are given in Table 1; the CPU implementation uses the default data structures as explained in the next section. As can be seen from Figure 5, our GPU implementation achieves a speedup of 21 to 69 times over the single-threaded SmallUPBP and 4 to 15 times over the multi-threaded version. Most of the GPU running time is taken up by the camera kernel, which generates the camera subpaths, queries the data structures, and evaluates the estimators. A small fraction is taken by the construction of the data structures and the remainder by the light kernel, which generates the

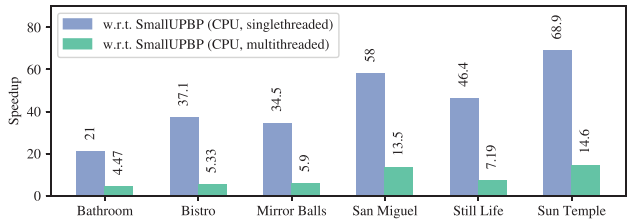


Figure 5: *Speedup of our GPU implementation over the single- and multi-threaded CPU versions. Values are per-iteration averages, measured over 120 iterations.*

Table 1: *Baseline CPU running times in seconds for both ST – single-threaded and MT – multi-threaded versions of SmallUPBP.*

	Bathroom	Bistro	Mirror Balls	San Miguel	Still Life	Sun Temple
ST	134.01	127.22	58.33	115.06	50.72	419.78
MT	28.46	18.28	9.97	26.84	7.86	89.00

light subpaths. The number of light and camera subpaths is equal in our experiments. Therefore, and since our new algorithm allows for computing the weight for a pair of paths in constant time, most time spent in the GPU implementation is due to data structure queries.

6.2. Data structures

To properly compare data structures on the CPU and GPU, we separated building and querying from the remainder of the algorithm, such that we could only profile those specific parts. We compared the data structure for each estimator of the previous section with its CPU counterpart and a reference BVH implementation, and a linear BVH (LBVH) [Kar12], see the recent survey by Meister et al. [MOB*21]. A list of the number of elements used in each estimator is provided in supplementary material.

6.2.1. *Beam-Beam, one-dimensional*

Table 2 gives the performance of the data structure of the Beam-Beam estimator. SmallUPBP uses a grid data structure where each beam is referenced in every cell it intersects. While the LBVH has a lower construction time than our data structure, in most cases our data structure spends less time performing the queries. Thus, overall, our implementation outperforms the LBVH.

Table 2: *Running times (ms) and speedup over the CPU data structure, for the* B-B1D *data structures; 20 iterations.*

Scene	Bathroom Build	Query	Total ×	Bistro Build	Query	Total ×	Mirror Balls Build	Query	Total ×
Grid (CPU)	628	14650	-	34	9055	-	220	11653	-
Ours (GPU)	79	388	**32.8**	86	102	**48.7**	59	307	**32.5**
LBVH (GPU)	23	3061	5.0	22	240	34.8	17	1877	6.3

Scene	San Miguel Build	Query	Total ×	Still Life Build	Query	Total ×	Sun Temple Build	Query	Total ×
Grid (CPU)	64	30547	-	10	624	-	63	79166	-
Ours (GPU)	54	75	**239.4**	27	18	14.6	82	343	**186.7**
LBVH (GPU)	17	336	86.8	8	17	**25.9**	28	2845	27.6

Table 3: *Running times (ms) and speedup over the CPU data structure, for the* P-B2D *data structures; 20 iterations.*

Scene	Bathroom Build	Query	Total ×	Bistro Build	Query	Total ×	Mirror Balls Build	Query	Total ×
Embree (CPU)	213	9702	-	11	6262	-	106	6770	-
Ours (GPU)	163	37	**49.9**	56	73	**49.3**	94	50	**48.1**
LBVH (GPU)	33	283	31.4	21	146	37.7	22	174	35.3

Scene	San Miguel Build	Query	Total ×	Still Life Build	Query	Total ×	Sun Temple Build	Query	Total ×
Embree (CPU)	158	5682	-	31	9712	-	226	37177	-
Ours (GPU)	113	45	**37.2**	34	74	**90.9**	177	286	**81.0**
LBVH (GPU)	24	6397	0.9	11	284	33.2	39	9004	4.1

Table 4: *Running times (ms) and speedup over the CPU data structure, for the* P-P3D *data structures; 20 iterations.*

Scene	Bathroom Build	Query	Total ×	Bistro Build	Query	Total ×	Mirror Balls Build	Query	Total ×
Hash grid (CPU)	50	578	-	5	2266	-	25	240	-
Ours (GPU)	27	2	22.9	15	80	24.1	15	1	16.9
Hash grid (GPU)	9	9	**36.7**	6	15	**110.8**	6	4	**29.7**
LBVH (GPU)	15	17	20.5	7	48	42.0	10	7	16.6

Scene	San Miguel Build	Query	Total ×	Still Life Build	Query	Total ×	Sun Temple Build	Query	Total ×
Hash grid (CPU)	36	491	-	7	400	-	51	2747	-
Ours (GPU)	20	2	**24.5**	8	3	39.5	105	52	**18.0**
Hash grid (GPU)	8	625	0.8	6	4	**45.8**	16	1216	2.3
LBVH (GPU)	12	897	0.6	7	17	17.4	22	1326	2.1

6.2.2. *Point-Beam, two-dimensional*

Table 3 shows the performance of the data structures for the Point-Beam estimators. SmallUPBP uses a CPU-based ray tracing framework, Embree, to accelerate the search. Similar to the Beam-Beam estimator, our data structure needs more time building, but less time querying than the LBVH, so that again, our data structure is faster overall.

6.2.3. *Point-Point, two- and three-dimensional*

Table 4 and Table 5 give the data structure performance for the Point-Point estimators. SmallUPBP uses a hash grid, which, for comparison, we also implemented on the GPU. Unlike the beam-based estimators, it is not immediately clear which data structure is best. For the volumetric P-P3D estimator, the hash grid seems to perform best on all scenes, with the exception of San Miguel and Sun

Table 5: *Running times (ms) and speedup over the CPU data structure, for the* P-P2D *data structures; 20 iterations.*

Scene	Bathroom Build	Query	Total ×	Bistro Build	Query	Total ×	Mirror Balls Build	Query	Total ×
Hash grid (CPU)	424	28646	-	331	10298	-	310	5165	-
Ours (GPU)	415	171	**49.7**	356	80	24.4	371	205	9.5
Hash grid (GPU)	52	1946	14.6	52	265	**33.7**	29	378	**13.5**
LBVH (GPU)	88	833	31.6	72	531	17.7	58	482	10.1

Scene	San Miguel Build	Query	Total ×	Still Life Build	Query	Total ×	Sun Temple Build	Query	Total ×
Hash grid (CPU)	145	3955	-	116	1713	-	169	4525	-
Ours (GPU)	273	78	11.7	107	44	12.2	291	30	14.7
Hash grid (GPU)	20	366	10.6	22	80	18.1	22	248	17.5
LBVH (GPU)	36	192	**18.0**	33	56	**20.7**	37	124	**29.5**

Temple. These two scenes share the property that the light sources are relatively small compared to the scene geometry and are concentrated in a small part of the scene. This causes a non-uniform distribution of light samples, which is a situation a hash grid performs notoriously poor on. For the surface-based P-P2D estimator, our data structure spends the least amount of time performing the queries, but most time constructing it. As a result, there is no clear winner.

7. Conclusions and Future Work

Our main contribution is an efficient method to evaluate the MIS weight for UPBP, which provides an algorithmic speedup by removing the need to iterate over the path vertices. Thanks to this, the MIS weight can now be evaluated in constant time instead of being proportional to the path length. The key idea is to split the path weight into three independent parts: one for the light subpath, one for the camera subpath, and a quantity depending on both subpath ends. The subpath weight is formulated as a recursive quantity that can be evaluated as the subpath is traced. Upon forming a full path by combining two subpaths, the weight is computed only from data cached at the light and camera vertex that are merged with PDE, or connected with BTP.

We also showed how the hardware-accelerated bounding volume hierarchy of NVIDIA's RTX graphics cards can be used to implement a data structure for PDE. By reformulating the problem, we implemented three different photon maps for volumetric PDE. We also compared the performance of these data structures to CPU-based reference implementations and hardware-accelerated state of the art. We have found that in many cases, the RTX-based photon maps offer a significant improvement over state-of-the-art approaches. To further analyse the performance of the proposed algorithm and hardware-accelerated photon maps, we implemented the full UPBP algorithm on the GPU using CUDA. [1] We evaluated our full GPU solution on a varied selection of scenes and compared running times with single- and multi-threaded CPU implementations. We found that our GPU implementation is able to achieve a speedup of up to 15 times compared to the best-performing prior work.

Work by Bitterli and Jarosz [BJ17] and Deng et al. [DJBJ19] further generalizes the theory of photon points and beams to photon planes and volumes. This results in an even larger set of volumetric photon density estimators, geared primarily towards simulating thin participating media. UPBP handles thin media mostly with the B-B1D estimator, which is also one of the most computationally-intensive parts of the algorithm. It would be interesting to see if these estimators can be incorporated into UPBP and if the RTX data structures could be applied to this extended set of estimators.

Several, more theoretical, future challenges stated by Křivánek et al. [KGH*14] still remain unsolved and are not addressed by our work. Specifically, finding a theoretical basis for radius reduction of the beam-based estimators and for selecting the number of light subpaths used for each technique are still open problems.

Acknowledgements

We thank the authors of SmallUPBP for making its source code, as well as the Bathroom, Still Life, and Mirror Balls scenes, publicly available [Vév15]. We thank Morgan McGuire for making the San Miguel scene available [McG17], NVIDIA's ORCA [NVI17] for publishing the Bistro and Sun Temple scenes, and the Stanford Computer Graphics Laboratory [Sta] for making the Lucy statue available.

References

[AL09] Aila T., Laine S.: Understanding the efficiency of ray traversal on GPUs. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, Association for Computing Machinery, pp. 145–149.

[BJ17] Bitterli B., Jarosz W.: Beyond points and beams: Higher-dimensional photon samples for volumetric light transport. *ACM Transactions on Graphics 36*, 4 (July 2017).

[1] Our GPU implementation is publicly available and can be found at https://github.com/nolmoonen/gpuupbp.

[Bur18] Burnes A.: NVIDIA Turing architecture deep dive whitepaper available now for download, Sept. 2018.

[DJBJ19] Deng X., Jiao S., Bitterli B., Jarosz W.: Photon surfaces for robust, unbiased volumetric density estimation. *ACM Transactions on Graphics 38*, 4 (July 2019).

[DKHS14] Davidovič T., Křivánek J., Hašan M., Slusallek P.: Progressive light transport simulation on the GPU: Survey and improvements. *ACM Transactions on Graphics 33*, 3 (June 2014).

[EPVV21] Evangelou I., Papaioannou G., Vardis K., Vasilakis A. A.: Fast radius search exploiting ray tracing frameworks. *Journal of Computer Graphics Techniques (JCGT) 10*, 1 (Feb 2021), 25–48.

[Geo12] Georgiev I.: Implementing Vertex Connection and Merging. Tech. rep., Saarland University, 2012.

[GHZ18] Guo Y., Hašan M., Zhao S.: Position-free monte carlo simulation for arbitrary layered bsdfs. *ACM Transactions on Graphics 37*, 6 (Dec 2018).

[GKDS12] Georgiev I., Křivánek J., Davidovič T., Slusallek P.: Light transport simulation with vertex connection and merging. *ACM Transactions on Graphics 31*, 6 (Nov 2012).

[JNSJ11] Jarosz W., Nowrouzezahrai D., Sadeghi I., Jensen H. W.: A comprehensive theory of volumetric radiance estimation using photon points and beams. *ACM Transactions on Graphics 30*, 1 (Feb 2011).

[JNT*11] Jarosz W., Nowrouzezahrai D., Thomas R., Sloan P.-P., Zwicker M.: Progressive photon beams. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (New York, NY, USA, 2011), SA '11, Association for Computing Machinery.

[JZJ08] Jarosz W., Zwicker M., Jensen H. W.: The beam radiance estimate for volumetric photon mapping. In *ACM SIGGRAPH 2008 Classes* (New York, NY, USA, 2008), SIGGRAPH '08, Association for Computing Machinery.

[Kar12] 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* (Goslar, DEU, 2012), EGGH-HPG'12, Eurographics Association, pp. 33–37.

[KGH*14] Křivánek J., Georgiev I., Hachisuka T., Vévoda P., Šik M., Nowrouzezahrai D., Jarosz W.: Unifying points, beams, and paths in volumetric light transport simulation. *ACM Transactions on Graphics 33*, 4 (July 2014).

[LW93] Lafortune E. P., Willems Y. D.: Bi-directional path tracing. In *Proceedings of Compugraphics '93* (Alvor, Portugal, 1993), pp. 145–153.

[MBGB20] Meister D., Boksansky J., Guthe M., Bittner J.: On ray reordering techniques for faster GPU ray tracing. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2020), I3D '20, Association for Computing Machinery.

[McG17] McGuire M.: Computer graphics archive, July 2017.

[MOB*21] Meister D., Ogaki S., Benthin C., Doyle M. J., Guthe M., Bittner J.: A survey on bounding volume hierarchies for ray tracing. *Computer Graphics Forum* (2021).

[NHD10] NovÃąk J., Havran V., Dachsbacher C.: Path regeneration for interactive path tracing. In *Eurographics 2010 - Short Papers* (2010), Lensch H. P. A., Seipel S. (Eds.), The Eurographics Association.

[NVIa] NVIDIA: NVIDIA OptiX – programming guide.

[NVIb] NVIDIA: NVIDIA OptiX ray tracing engine.

[NVI17] NVIDIA: Open research content archive (orca), July 2017.

[PBD*10] Parker S. G., 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 Transactions on Graphics 29*, 4 (July 2010).

[PKK00] Pauly M., Kollig T., Keller A.: Metropolis light transport for participating media. In *Rendering Techniques 2000* (Vienna, 2000), Péroche B., Rushmeier H., (Eds.), Springer Vienna, pp. 11–22.

[Sta] Stanford Computer Graphics Laboratory: The stanford 3d scanning repository.

[vA11a] van Antwerpen D.: Improving SIMD efficiency for parallel Monte Carlo light transport on the GPU. In *Proceedings of ACM SIGGRAPH Symposium on High Performance Graphics* (New York, NY, USA, 2011), HPG '11, Association for Computing Machinery, pp. 41–50.

[vA11b] van Antwerpen D.: Recursive MIS Computation for Streaming BDPT on the GPU. Tech. rep., Delft University of Technology, 2011.

[Vea97] Veach E.: *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, Stanford, CA, USA, 1997. AAI9837162.

[Vév14] Vévoda P.: *Robust Light Transport Simulation in Participating Media*. Master's thesis, Charles University in Prague, 2014.

[Vév15] Vévoda P.: SmallUPBP - A (not too) small physically based volumetric renderer, 2015.

[VG94] Veach E., Guibas L.: Bidirectional estimators for light transport. In *Eurographics Rendering Workshop 1994 Proceedings* (Darmstadt, Germany, 1994), pp. 147–162.

[VG95] Veach E., Guibas L. J.: Optimally combining sampling techniques for monte carlo rendering. In *SIGGRAPH 95 Proceedings* (1995), Addison-Wesley, pp. 419–428.

[WMZ*20] WALD I., MORRICAL N., ZELLMANN S., MA L., USHER W., HUANG T., PASCUCCI V.: Using hardware ray transforms to accelerate ray/primitive intersections for long, thin primitive types. *Proceedings of the ACM in Computer Graphics and Interactive Techniques. 3*, 2 (Aug 2020).

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Supporting Information