CHAPTER 18

Importance Sampling of Many Lights on the GPU

Pierre Moreau[1,2] and Petrik Clarberg[1]
[1]NVIDIA
[2]Lund University

ABSTRACT

The introduction of standardized APIs for ray tracing, together with hardware acceleration, opens up possibilities for physically based lighting in real-time rendering. Light importance sampling is one of the fundamental operations in light transport simulations, applicable to both direct and indirect illumination. This chapter describes a bounding volume hierarchy data structure and associated sampling methods to accelerate importance sampling of local light sources. The work is based on recently published methods for light sampling in production rendering, but it is evaluated in a real-time implementation using Microsoft DirectX Raytracing.

18.1 INTRODUCTION

A realistic scene may contain hundreds of thousands of light sources. The accurate simulation of the light and shadows that they cast is one of the most important factors for realism in computer graphics. Traditional real-time applications with rasterized shadow maps have been practically limited to use a handful of carefully selected dynamic lights. Ray tracing allows more flexibility, as we can trace shadow rays to different sampled lights at each pixel.

Mathematically speaking, the best way to select those samples is to pick lights with a probability in proportion to each light's contribution. However, the contribution varies spatially and depends on the local surface properties and visibility. Hence, it is challenging to find a single global probability density function (PDF) that works well everywhere.

The solution that we explore in this chapter is to use a hierarchical acceleration structure built over the light sources to guide the sampling [11, 22]. Each node in the data structure represents a cluster of lights. The idea is to traverse the tree from top to bottom, at each level estimating how much each cluster

E. Haines, T. Akenine-Möller (eds.), *Ray Tracing Gems*, https://doi.org/10.1007/978-1-4842-4427-2_18

contributes, and to choose which path through the tree to take based on random decisions at each level. Figure 18-1 illustrates these concepts. This means that lights are chosen approximately proportional to their contributions, but without having to explicitly compute and store the PDF at each shading point. The performance of the technique ultimately depends on how accurately we manage to estimate the contributions. In practice, the pertinence of a light or a cluster of lights, depends on its:

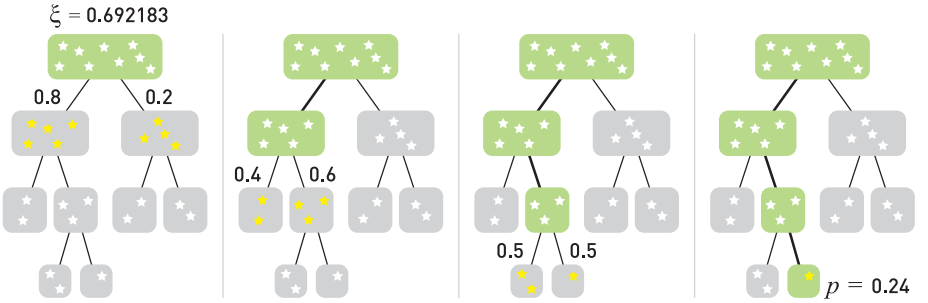


Figure 18-1. *All the light sources in the scene are organized in a hierarchy. Given a shading point X, we start at the root and proceed down the hierarchy. At each level, the importance of each immediate child with respect to X is estimated by a probability. Then, a uniform random number ξ decides the path through the tree, and at the leaf we find which light to sample. In the end, more important lights have a higher probability of being sampled.*

- *Flux:* The more powerful a light is, the more it will contribute.
- *Distance to the shading point:* The further away a light lies, the smaller the solid angle it subtends, resulting in less energy reaching the shading point.
- *Orientation:* A light source may not emit in all directions, nor do so uniformly.
- *Visibility:* Fully occluded light sources do not contribute.
- *BRDF at the shading point:* Lights located in the direction of the BRDF's main peaks will have a larger fraction of their energy reflected.

A key advantage of light importance sampling is that it is independent of the number and type of lights, and hence scenes can have many more lights than we can afford to trace shadow rays to and large textured area lights can be seamlessly supported. Since the probability distributions are computed at runtime, scenes can be fully dynamic and have complex lighting setups. With recent advances in denoising, this holds promise to reduce rendering time, while allowing more artistic freedom and more realistic results.

In the following, we discuss light importance sampling in more detail and present a real-time implementation that uses a bounding volume hierarchy (BVH) over the lights. The method is implemented using the Microsoft DirectX Raytracing (DXR) API, and source code is available.

18.2 REVIEW OF PREVIOUS ALGORITHMS

With the transition to path tracing in production rendering [21, 31], the visibility sampling is solved by tracing shadow rays toward sampled points on the light sources. When a shadow ray does not hit anything on its way from a shading point to the light, the point is deemed to be lit. By averaging over many such samples over the surfaces of the lights, a good approximation of the lighting is achieved. The approximation converges to ground truth as more samples are taken. However, with more than a handful of light sources, exhaustive sampling is not a viable strategy, not even in production rendering.

To handle the complexity of dynamic lighting with many lights, most techniques generally rely on building some form of spatial acceleration structure over the lights, which is then used to accelerate rendering by either culling, approximating, or importance-sampling the lights.

18.2.1 REAL-TIME LIGHT CULLING

Game engines have transitioned to use mostly physically based materials and light sources specified in physical units [19, 23]. However, for performance reasons and due to the limitations of the rasterization pipeline, only a few point-like light sources can be rendered in real time with shadow maps. The cost per light is high and the performance scales linearly with the number of lights. For area lights, the unshadowed contribution can be computed using linearly transformed cosines [17], but the problem of evaluating visibility remains.

To reduce the number of lights that need to be considered, it is common to artificially limit the influence region of individual lights, for example, by using an approximate quadratic falloff that goes to zero at some distance. By careful placement and tweaking of the light parameters, the number of lights that affect any given point can be limited.

Tiled shading [2, 28] works by binning such lights into screen-space tiles, where the depth bounds of the tiles effectively reduce the number of lights that need to be processed when shading each tile. Modern variants improve culling rates by splitting frusta in depth (2.5D culling) [15], by clustering shading points or lights [29, 30], or by using per-tile light trees [27].

A drawback of these culling methods is that the acceleration structure is in screen space. Another drawback is that the required clamped light ranges can introduce noticeable darkening. This is particularly noticeable in cases where many dim lights add up to a significant contribution, such as Christmas tree lights or indoor office illumination. To address this, Tokuyoshi and Harada [40] propose using stochastic light ranges to randomly reject unimportant lights rather than assigning fixed ranges. They also show a proof-of-concept of the technique applied to path tracing using a bounding sphere hierarchy over the light sources.

18.2.2 MANY-LIGHT ALGORITHMS

Virtual point lights (VPLs) [20] have long been used to approximate global illumination. The idea is to trace photons from the light sources and deposit VPLs at path vertices, which are then used to approximate the indirect illumination. VPL methods are conceptually similar to importance sampling methods for many lights. The lights are clustered into nodes in a tree, and during traversal estimated contributions are computed. The main difference is that, for importance sampling, the estimations are used to compute light selection probabilities rather than directly to approximate the lighting.

For example, *lightcuts* [44, 45] accelerate the rendering with millions of VPLs by traversing the tree per shading point and computing error bounds on the estimated contributions. The algorithm chooses to use a cluster of VPLs directly as a light source, avoiding subdivision to finer clusters or individual VPLs, when the error is sufficiently small. We refer to the survey by Dachsbacher et al. [12] for a good overview of these and other many-light techniques. See also the overview of global illumination algorithms by Christensen and Jarosz [8].

18.2.3 LIGHT IMPORTANCE SAMPLING

In early work on accelerating ray tracing with many lights, the lights are sorted according to contribution and only the ones above a threshold are shadow tested [46]. The contribution of the remaining lights is then added based on a statistical estimate of their visibility.

Shirley et al. [37] describe importance sampling for various types of light sources. They classify lights as bright or dim by comparing their estimated contributions to a user-defined threshold. To sample from multiple lights, they use an octree that is hierarchically subdivided until the number of bright lights is sufficiently small. The contribution of an octree cell is estimated by evaluating the contribution at a large number of points on the cell's boundary. Zimmerman and Shirley [47] use a uniform spatial subdivision instead and include an estimated visibility in the cells.

For real-time ray tracing with many lights, Schmittler et al. [36] restrict the influence region of lights and use a *k*-d tree to quickly locate the lights that affect each point. Bikker takes a similar approach in the Arauna ray tracer [5, 6], but it uses a BVH with spherical nodes to more tightly bound the light volumes. Shading is done Whitted-style by evaluating all contributing lights. These methods suffer from bias as the light contributions are cut off, but that may potentially be alleviated with stochastic light ranges as mentioned earlier [40].

In the Brigade real-time path tracer, Bikker [6] uses *resampled importance sampling* [39]. A first set of lights is selected based on a location-invariant probability density function, and then this set is resampled by more accurately estimating the contributions using the BRDF and distances to pick one important light. In this approach, there is no hierarchical data structure.

The Iray rendering system [22] uses a hierarchical light importance sampling scheme. Iray works with triangles exclusively and assigns a single flux (power) value per triangle. A BVH is built over the triangular lights and traversed probabilistically, at each node computing the estimated contribution of each subtree. The system encodes directional information at each node by dividing the unit sphere into a small number of regions and storing one representative flux value per region. Estimated flux from BVH nodes is computed based on the distance to the center of the node.

Conty Estevez and Kulla [11] take a similar approach for cinematic rendering. They use a 4-wide BVH that also includes analytic light shapes, and the lights are clustered in world space including orientation by using bounding cones. In the traversal, they probabilistically select which branch to traverse based on a single uniform random number. The number is rescaled to the unit range at each step, which preserves stratification properties (the same technique is used in hierarchical sample warping [9]). To reduce the problem of poor estimations for large nodes, they use a metric for adaptively splitting such nodes during traversal. Our real-time implementation is based on their technique, with some simplifications.

18.3 FOUNDATIONS

In this section, we will first review the foundations of physically based lighting and importance sampling, before diving into the technical details of our real-time implementation.

18.3.1 LIGHTING INTEGRALS

The radiance L_o leaving a point X on a surface in viewing direction $\mathbf{v}$ is the sum of emitted radiance L_e and reflected radiance L_r, under the geometric optics approximation described by [18]:

$$L_o(X,\mathbf{v}) = L_e(X,\mathbf{v}) + L_r(X,\mathbf{v}), \tag{1}$$

$$\text{where} \quad L_r(X,\mathbf{v}) = \int_\Omega f(X,\mathbf{v},\mathbf{l})\, L_i(X,\mathbf{l})(\mathbf{n}\cdot\mathbf{l})\, d\omega \tag{2}$$

and where f is the BRDF and L_i is the incident radiance arriving from a direction $\mathbf{l}$. In the following, we will drop the X from the notation when we speak about a specific point. Also, let the notation $L(X \leftarrow Y)$ denote the radiance emitted from a point Y in the direction toward a point X.

In this chapter, we are primarily interested in the case where L_i comes from a potentially large set of local light sources placed within the scene. The algorithm can, however, be combined with other sampling strategies for handling distant light sources, such as the sun and sky.

The integral over the hemisphere can be rewritten as an integral over all the surfaces of the light sources. The relationship between solid angle and surface area is illustrated in Figure 18-2. In fact, a small patch dA at a point Y on a light source covers a solid angle

$$d\omega = \frac{|\mathbf{n}_Y \cdot -\mathbf{l}|}{\|X-Y\|^2} dA, \tag{3}$$

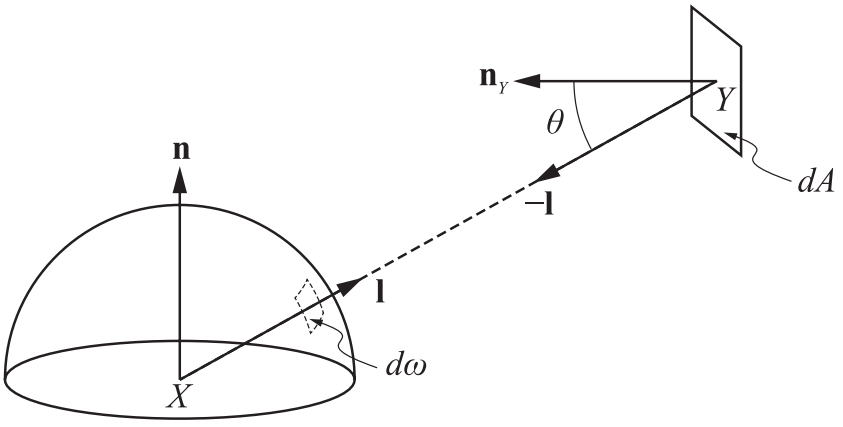


Figure 18-2. *The differential solid angle $d\omega$ of a surface patch dA at a point Y on a light source is a function of its distance $\|X - Y\|$ and the angle $\cos\theta = |\mathbf{n}_Y \cdot -\mathbf{l}|$ at which it is viewed.*

i.e., there is an inverse square falloff by distance and a dot product between the light's normal $\mathbf{n}_Y$ and the emitted light direction $-\mathbf{l}$. Note that in our implementation, light sources may be single-sided or double-sided emitters. For single-sided lights, we set the emitted radiance $L(X \leftarrow Y) = 0$ if $(\mathbf{n}_Y \cdot -\mathbf{l}) \leq 0$.

We also need to know the visibility between our shading point X and the point Y on the light source, formally expressed as

$$v\left(X \leftrightarrow Y\right) = \begin{cases} 1 & \text{if } X \text{ and } Y \text{ are mutually visible,} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

In practice, we evaluate v by tracing shadow rays from X in direction $\mathbf{l}$, with the ray's maximum distance $t_{\max} = \|X - Y\|$. Note that to avoid self-intersections due to numerical issues, the ray origin needs to be offset and the ray shortened slightly using epsilons. See Chapter 6 for details.

Now, assuming that there are m light sources in the scene, the reflected radiance in Equation 2 can be written as

$$L_r\left(X, \mathbf{v}\right) = \sum_{i=1}^{m} L_{r,i}\left(X, \mathbf{v}\right), \quad \text{where} \tag{5}$$

$$L_{r,i}\left(X, \mathbf{v}\right) = \int_{\Omega} f\left(X, \mathbf{v}, \mathbf{l}\right) L_i\left(X \leftarrow Y\right) v\left(X \leftrightarrow Y\right) \max\left(\mathbf{n} \cdot \mathbf{l}, 0\right) \frac{|\mathbf{n}_Y \cdot -\mathbf{l}|}{\left\|X - Y\right\|^2} dA_i. \tag{6}$$

That is, L_r is the sum of the reflected light from each individual light $i = \{1, \ldots, m\}$. Note that we clamp $\mathbf{n} \cdot \mathbf{l}$ because light from points backfacing to the shading point cannot contribute. The complexity is linear in the number of lights m, which may become expensive when m is large. This leads us to the next topic.

18.3.2 IMPORTANCE SAMPLING

As discussed in Section 18.2, there are two fundamentally different ways to reduce the cost of Equation 5. One method is to limit the influence regions of lights, and thereby reduce m. The other method is to sample a small subset of lights $n \ll m$. This can be done in such a way that the result is *consistent*, i.e., it converges to the ground truth as n grows.

18.3.2.1 MONTE CARLO METHOD

Let Z be a discrete random variable with values $z \in \{1, \ldots, m\}$. The probability that Z is equal to some value z is described by the discrete PDF $p(z) = P(Z = z)$, where $\sum p(z) = 1$. For example, if all values are equally probable, then $p(z) = \frac{1}{m}$. If we have a function $g(Z)$ of a random variable, its expected value is

$$\mathbb{E}\left[g(Z)\right] = \sum_{z \in Z} g(z)p(z), \tag{7}$$

i.e., each possible outcome is weighted by how probable it is. Now, if we take n random samples $\{z_1, \ldots, z_n\}$ from Z, we get the n-sample *Monte Carlo estimate* $\tilde{g}_n(z)$ of $\mathbb{E}\left[g(Z)\right]$ as follows:

$$\tilde{g}_n(z) = \frac{1}{n}\sum_{j=1}^{n} g(z_j). \tag{8}$$

In other words, the expectation can be estimated by taking the mean of random samples of the function. We can also speak of the corresponding Monte Carlo *estimator* $\tilde{g}_n(Z)$, which is the mean of the function of the n independent and identically distributed random variables $\{Z_1, \ldots, Z_n\}$. It is easy to show that $\mathbb{E}\left[\tilde{g}_n(Z)\right] = \mathbb{E}\left[g(Z)\right]$, i.e., the estimator gives us the correct value.

Since we are taking random samples, the estimator $\tilde{g}_n(Z)$ will have some variance. As discussed in Chapter 15, the variance decreases linearly with n:

$$\mathrm{Var}\left[\tilde{g}_n(Z)\right] = \frac{1}{n}\mathrm{Var}\left[g(Z)\right]. \tag{9}$$

These properties show that the Monte Carlo estimator is consistent. In the limit, when we have infinitely many samples, the variance is zero and it has converged to the correct expected value.

To make this useful, note that almost any problem can be recast as an expectation. We thus have a consistent way of estimating the solution based on random samples of the function.

18.3.2.2 LIGHT SELECTION IMPORTANCE SAMPLING

In our case, we are interested in evaluating the sum of light reflected from all the light sources (Equation 5). This sum can be expressed as an expectation (cf., Equation 7) as follows:

$$L_r(X,\mathbf{v}) = \sum_{i=1}^{m} L_{r,i}(X,\mathbf{v}) = \sum_{i=1}^{m} \frac{L_{r,i}(X,\mathbf{v})}{P(Z=i)} P(Z=i) = \mathbb{E}\left[\frac{L_{r,Z}(X,\mathbf{v})}{p(Z)}\right]. \tag{10}$$

Following Equation 8, the Monte Carlo estimate $\tilde{L}_r$ of the reflected light from all light sources is therefore

$$\tilde{L}_r(X,\mathbf{v}) = \frac{1}{n}\sum_{j=1}^{n} \frac{L_{r,z_j}(X,\mathbf{v})}{p(z_j)}, \tag{11}$$

that is, we sum the contribution from a randomly selected set of lights $\{z_1, \ldots, z_n\}$, divided by the probability of selecting each light. This estimator is always consistent, independent of how few samples n we take. However, the more samples we take, the smaller the variance of the estimator will be.

Note that nothing discussed so far makes any assumptions on the distribution of the random variable Z. The only requirement is that $p(z) > 0$ for all lights where $L_{r,z} > 0$, otherwise we would risk ignoring the contribution from some lights. It can be shown that the variance is minimized when $p(z) \propto L_{r,z}(X,\mathbf{v})$ [32, 38]. We will not go into the details here, but when the probability density function is exactly proportional to the function that we are sampling, the summation in the Monte Carlo estimator reduces to a sum of constant terms. In that case the estimator has zero variance.

In practice, this is not achievable because $L_{r,z}$ is unknown for a given shading point, but we should aim for selecting lights with a probability as close as possible to their relative contribution to the shading point. In Section 18.4, we will look at how $p(z)$ is computed.

18.3.2.3 LIGHT SOURCE SAMPLING

To estimate the reflected radiance using Equation 11, we also need to evaluate the integral $L_{r,z_j}(X,\mathbf{v})$ for the randomly selected set of lights. The expression in Equation 6 is an integral over the surface of the light that involves both BRDF and visibility terms. In graphics, this is not practical to evaluate analytically. Therefore, we again resort to Monte Carlo integration.

The surface of the light source is sampled uniformly with s samples $\{Y_1, \ldots, Y_s\}$. For triangle mesh lights, each light is a triangle, which means that we pick points uniformly on the triangle using standard techniques [32] (see Chapter 16). The probability density function for the samples on a triangle i is $p(Y) = \frac{1}{A_i}$, where A_i is the area of the triangle. The integral over the light is then evaluated using the Monte Carlo estimate

$$\tilde{L}_{r,i}(X, \mathbf{v}) = \frac{A_i}{s} \sum_{k=1}^{s} f(X, \mathbf{v}, \mathbf{l}_k) L_i(X \leftarrow Y_k) v(X \leftrightarrow Y_k) \max(\mathbf{n} \cdot \mathbf{l}_k, 0) \frac{\left| \mathbf{n}_{Y_k} \cdot -\mathbf{l}_k \right|}{\left\| X - Y_k \right\|^2}. \tag{12}$$

In the current implementation, $s = 1$ as we trace a single shadow ray for each of the n sampled light sources, and $\mathbf{n}_{Y_k} = \mathbf{n}_i$ since we use the geometric normal of the light source when evaluating its emitted radiance. Smooth normals and normal mapping are disabled by default for performance reasons, because they often have negligible impact on the light distribution.

18.3.3 RAY TRACING OF LIGHTS

In real-time applications, a common rendering optimization is to separate the geometric representation from the actual light-emitting shape. For example, a light bulb can be represented by a point light or small analytic sphere light, while the visible light bulb is drawn as a more complex triangle mesh.

In this case, it is important that the emissive property of the light geometry matches the intensity of the actual emitter. Otherwise, there will be a perceptual difference between how bright a light appears in direct view and how much light it casts into the scene. Note that a light source is often specified in photometric units in terms of its luminous flux (lumen), while the emissive intensity of an area light is given in luminance (cd/m^2). Accurate conversion from flux to luminance therefore needs to take the surface area of the light's geometry into account. Before rendering, these photometric units are finally converted to the radiometric quantities that we use (flux and radiance).

Another consideration is that when tracing shadow rays toward an emitter, we do not want to inadvertently hit the mesh representing the light source and count the emitter as occluded. The geometric representation must therefore be invisible to shadow rays, but visible for other rays. The Microsoft DirectX Raytracing API allows control of this behavior via the `InstanceMask` attribute on the acceleration structure and by the `InstanceInclusionMask` parameter to `TraceRay`.

For *multiple importance sampling* (MIS) [41], which is an important variance reduction technique, we must be able to evaluate light sampling probabilities given samples generated by other sampling strategies. For example, if we draw a sample over the hemisphere using BRDF importance sampling that hits a light source after traversal, we compute its probability had the sample been generated with light importance sampling. Based on this probability together with the BRDF sampling probability, a new weight for the sample can be computed using, for example, the power heuristic [41] to reduce the overall variance.

A practical consideration for MIS is that if the emitters are represented by analytic shapes, we cannot use hardware-accelerated triangle tests to search for the light source in a given direction. An alternative is to use custom intersection shaders to compute the intersections between rays and emitter shapes. This has not yet been implemented in our sample code. Instead, we always use the mesh itself as the light emitter, i.e., each emissive triangle is treated as a light source.

18.4 ALGORITHM

In the following, we describe the main steps of our implementation of light importance sampling. The description is organized by the frequency at which operations occur. We start with the preprocessing step that can happen at asset-creation time, which is followed by the construction and updating of the light data structure that runs once per frame. Then, the sampling is described, which is executed once per light sample.

18.4.1 LIGHT PREPROCESSING

For mesh lights, we precompute a single flux value Φ_i per triangle i as a preprocess, similar to Iray [22]. The flux is the total radiant power emitted by the triangle. For diffuse emitters, the flux is

$$\Phi_i = \iint_{\Omega} L_i(X)\left(\mathbf{n}_i \cdot \omega\right) d\omega \, dA_i, \tag{13}$$

where $L_i(X)$ is the emitted radiance at position X on the light's surface. For non-textured emitters, the flux is thus simply $\Phi_i = \pi A_i L_i$, where L_i is the constant radiance of the material and A_i is the triangle's area. The factor π comes from the integral of the cosine term over the hemisphere. To handle textured emitters, which in our experience are far more common than untextured ones, we evaluate Equation 13 as a preprocess at load time.

To integrate the radiance, we rasterize all emissive triangles in *texture space*. The triangles are scaled and rotated so that each pixel represents exactly one texel at

the largest mip level. The integral is then computed by loading the radiance for the corresponding texel in the pixel shader and by accumulating its value atomically. We also count the number of texels and divide by that number at the end.

The only side effect of the pixel shader is atomic additions to a buffer of per-triangle values. Due to the current lack of floating-point atomics in DirectX 12, we use an NVIDIA extension via NVAPI [26] to do floating-point atomic addition.

Since the pixel shader has no render target bound (i.e., it is a `void` pixel shader), we can make the viewport arbitrarily large within the API limits, without worrying about memory consumption. The vertex shader loads the UV texture coordinates from memory and places the triangle at an appropriate coordinate in texture space so that it is always within the viewport. For example, if texture wrapping is enabled, the triangle is rasterized at pixel coordinates

$$(x, y) = (u - \lfloor u \rfloor, v - \lfloor v \rfloor) \cdot (w, h), \tag{14}$$

where w, h are the dimensions of the largest mip level of the emissive texture. With this transform, the triangle is always in view, independent of the magnitude of its (pre-wrapped) UV coordinates.

We currently rasterize the triangle using one sample per pixel, and hence only accumulate texels whose centers are covered. Tiny triangles that do not cover any texels are assigned a default nonzero flux to ensure convergence. Multisampling, or conservative rasterization with analytic coverage computations in the pixel shader, can be used to improve accuracy of the computed flux values.

All triangles with $\Phi_i = 0$ are excluded from further processing. Culling of zero flux triangles is an important practical optimization. In several example scenes, the majority of the emissive triangles lie in black regions of the emissive textures. This is not surprising, as often the emissiveness is painted into larger textures, rather than splitting the mesh into emissive and non-emissive meshes with separate materials.

18.4.2 ACCELERATION STRUCTURE

We are using a similar acceleration structure as Conty Estevez and Kulla [11], that is, a bounding volume hierarchy [10, 33] built from top to bottom using binning [43]. Our implementation uses a binary BVH, meaning that each node has two children. In some cases, a wider branching factor may be beneficial.

We will briefly introduce how binning works, before presenting different existing heuristics used during the building process, as well as minor variants thereof.

18.4.2.1 BUILDING THE BVH

When building a binary BVH from top to bottom, the quality and speed at which the tree is built depends on how the triangles are split between the left and right children at each node. Analyzing all the potential split locations will yield the best results, but this will also be slow and is not suitable for real-time applications.

The approach taken by Wald [43] consists of uniformly partitioning the space at each node into bins and then running the split analysis on those bins only. This implies that the more bins one has, the higher the quality of the generated tree will be, but the tree will also be more costly to build.

18.4.2.2 LIGHT ORIENTATION CONE

To help take into account the orientation of the different light sources, Conty Estevez and Kulla [11] store a light orientation cone in each node. This cone is made of an axis and two angles, θ_o and θ_e: the former bounds the normals of all emitters found within the node, whereas the latter bounds the set of directions in which light gets emitted (around each normal).

For example, a single-sided emissive triangle would have $\theta_o = 0$ (there is only one normal) and $\theta_e = \frac{\pi}{2}$ (it emits light over the whole hemisphere). Alternatively, an emissive sphere would have $\theta_o = \pi$ (it has normals pointing in all directions) and $\theta_e = \frac{\pi}{2}$, as around each normal, light is still only emitted over the whole hemisphere; θ_e will often be $\frac{\pi}{2}$, except for lights with a directional emission profile or for spotlights, where it will be equal to the spotlight's cone angle.

When computing the cone for a parent node, its θ_o will be computed such that it encompasses all the normals found in its children, whereas θ_e is simply computed as the maximum of each child's θ_e.

18.4.2.3 DEFINING THE SPLIT PLANE

As mentioned earlier, an axis-aligned split plane has to be computed to split the set of lights into two subsets, one for each child. This is usually achieved by computing a cost metric for each possible split and picking the one with the lowest cost. In the context of a binned BVH, we tested the *surface area heuristic* (SAH) (introduced by Goldsmith and Salmon [14] and formalized by MacDonald and Booth [24]) and the *surface area orientation heuristic* (SAOH) [11], as well as different variants of those two methods.

For all the variants presented below, the binning performed while building the BVH can be done either on the largest axis only (of a node's axis-aligned bounding box (AABB)) or on all three axes and the split with the lowest cost is selected. Only considering the largest axis will result in lower build time but also lower tree quality, especially for the variants taking the light orientations into account. More details on those trade-offs can be found in Section 18.5.

SAH The SAH focuses on the surface area of the AABB of the resulting children as well as on the number of lights that they contain. If we define the left child as $L = \cup_{j=0}^{i} \text{bin}_j$ and the right child as $R = \cup_{j=i+1}^{k} \text{bin}_j$, where k is the number of bins and $i \in [0, k-1]$, the cost for the split creating L and R as children is

$$cost(L,R) = \frac{n(L)a(L) + n(R)a(R)}{n(L \cup R)a(L \cup R)}, \tag{15}$$

where $n(C)$ and $a(C)$ return the number of lights and the surface area of a potential child node C, respectively.

SAOH The SAOH is based on the SAH and includes two additional weights: one based on the bounding cone around the directions in which the lights emit light, and another based on the flux emitted by the resulting clusters. The cost metric is

$$\text{cost}(L,R,s) = k_r(s)\frac{\Phi(L)a(L)M_\Omega(L) + \Phi(R)a(R)M_\Omega(R)}{a(L \cup R)M_\Omega(L \cup R)}, \tag{16}$$

where s is the axis on which the split is occurring, $k_r(s) = \text{length}_{\text{max}}/\text{length}_s$ is used to prevent thin boxes, and M_Ω is an orientation measure [11].

VH The *volume heuristic* (VH) is based on the SAH and replaces the surface area measure $a(C)$ in Equation 15 by the volume $v(C)$ of a node C's AABB.

VOH The *volume orientation heuristic* (VOH) similarly replaces the surface area measure in the SAOH (Equation 16) by the volume measure.

18.4.3 LIGHT IMPORTANCE SAMPLING

We now look at how the lights are actually sampled based on the acceleration structure described in the previous section. First, the light BVH is probabilistically traversed in order to select a single light source, and then a light sample is generated on the surface of that light (if it is an area light). See Figure 18-1.

18.4.3.1 PROBABILISTIC BVH TRAVERSAL

When traversing the acceleration data structure, we want to select the node that will lead us to the lights that contribute the most to the current shading point, with a probability for each light that is proportional to its contribution. As mentioned in Section 18.4.2, the contribution depends on many parameters. We will use either approximations or the exact value for each parameter, and we will try different combinations to optimize quality versus performance.

Distance This parameter is computed as the distance between the shading point and the center of the AABB of the node being considered. This favors nodes that are close to the shading point (and by extension lights that are close), if the node has a small AABB. However, in the first levels of the BVH, the nodes have large AABBs that contain most of the scene, giving a poor approximation of the actual distance between the shading point and some of the lights contained within that node.

Light Flux The flux of a node is computed as the sum of the flux emitted by all light sources contained within that node. This is actually precomputed when building the BVH for performance reasons; if some light sources have changing flux values over time, the precomputation will not be an issue because the BVH will have to be rebuilt anyway since the flux is also used for guiding the building step.

Light Orientation The selection so far does not take into consideration the orientation of the light source, which could give as much weight to a light source that is shining directly upon the shading point as to another light source that is backfacing. To that end, Conty Estevez and Kulla [11] introduced an additional term to a node's importance function that conservatively estimates the angle between the light normal and direction from the node's AABB center to the shading point.

Light Visibility To avoid considering lights that are located below the horizon of a shading point, we use the clamped $\mathbf{n} \cdot \mathbf{l}$ term in the importance function of each node. Note that Conty Estevez and Kulla [11] use this clamped term, multiplied by the surface's albedo, as an approximation to the diffuse BRDF, which will achieve the same effect of discarding lights that are beneath the horizon of the shading point.

Node Importance Using the different parameters just defined, the importance function given a shading point X and a child node C is defined as

$$\text{importance}(X, C) = \frac{\Phi(C)\,|\cos\theta_i'|}{\|X - C\|^2} \times \begin{cases} \cos\theta' & \text{if } \theta' < \theta_e, \\ 0 & \text{otherwise,} \end{cases} \tag{17}$$

where $\|X - C\|$ is the distance between shading point X and the center of the AABB of C, $\theta_i' = \max\left(0, \theta_i - \theta_u\right)$, and $\theta' = \max\left(0, \theta - \theta_o - \theta_u\right)$. The angles θ_e and θ_o come from the light orientation cone of node C. The angle θ is measured between the light orientation cone's axis and the vector from the center of C to X. Finally, θ_i is the incident angle and θ_u the uncertainty angle; these can all be found in Figure 18-3.

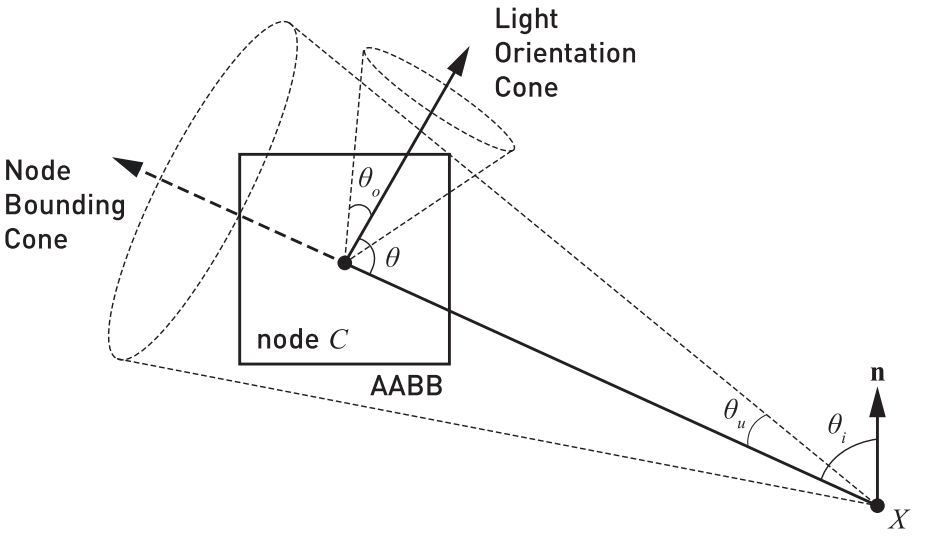


Figure 18-3. *Description of the geometry used for computing the importance of a child node C as seen from a shading point X. In Figure 18-1, the importance is computed twice at each step in the traversal, once for each child. The angle θ_u and the axis from X to the center of the AABB represent the smallest bounding cone containing the whole node and are used to compute conservative lower bounds on θ_i and θ.*

18.4.3.2 RANDOM NUMBER USAGE

A single uniform random number is used to decide whether to take the left or the right branch. The number is then rescaled and used for the next level. This technique preserves stratification (cf., hierarchical sample warping [9]) while also avoiding the cost of generating new random numbers at every level of the hierarchy. The rescaling of a random number ξ to find a new random number ξ' is done as follows:

$$\xi' = \begin{cases} \dfrac{\xi}{p(L)} & \text{if } \xi < p(L), \\ \dfrac{\xi - p(L)}{p(R)} & \text{otherwise,} \end{cases} \tag{18}$$

where $p(C)$ is the probability of selecting node C, computed as the importance of that node divided by the total importance:

$$p(L) = \frac{\text{importance}(L)}{\text{importance}(L) + \text{importance}(R)}. \tag{19}$$

Care must be taken to ensure enough random bits are available due to the limits of floating-point precision. For scenarios with huge numbers of lights, two or more random numbers may be alternated or higher precision used.

18.4.3.3 SAMPLING THE LEAF NODE

At the end of the traversal, a leaf node containing a certain number of light sources has been selected. To decide which triangle to sample, we can either uniformly pick one of the triangles stored in the leaf node or use an importance method similar to the one used for computing the node's importance during the traversal. For importance sampling, we consider the closest distance to the triangle and the largest $\mathbf{n} \cdot \mathbf{l}$ bound of the triangle; including the triangle's flux and its orientation to the shading point could further improve the results. Currently, up to 10 triangles are stored per leaf node.

18.4.3.4 SAMPLING THE LIGHT SOURCE

After a light source has been selected through the tree traversal, a light sample needs to be generated on that light source. We use the sampling techniques presented by Shirley et al. [37] for generating the light samples uniformly over the surfaces of different types of lights.

18.5 RESULTS

We demonstrate the algorithm for multiple scenes with various numbers of lights, where we measure the rate at which the error decreases, the time taken for building the BVH, and the rendering time.

The rendering is accomplished by first rasterizing the scene in a G-buffer using DirectX 12, followed by light sampling in a full-screen ray tracing pass using a single shadow ray per pixel, and finally temporally accumulating the frames if no movements occurred. All numbers are measured on an NVIDIA GeForce RTX 2080 Ti and an Intel Xeon E5-1650 at 3.60 GHz, with the scenes being rendered at a resolution of 1920 × 1080 pixels. For all the results shown in this chapter, the indirect lighting is never evaluated and we instead use the algorithm to improve the computation of direct lighting.

We use the following scenes, as depicted in Figure 18-4, in our testing:

- *Sun Temple:* This scene features 606,376 triangles, out of which 67,374 are textured emissive; however, after the texture pre-integration described in Section 18.4.1, only 1,095 emissive triangles are left. The whole scene is lit by textured fire pits; the part of the scene shown in Figure 18-4 is only lit by two fire pits located far on the right, as well as two other small ones located behind the camera. The scene is entirely diffuse.

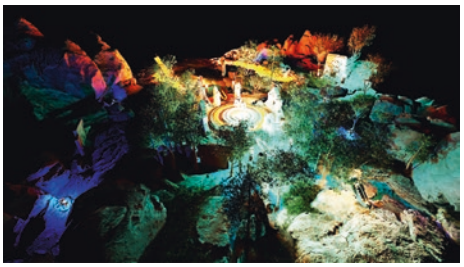
Sun Temple

Bistro (view 1)

Bistro (view 2)

Bistro (view 3)

Paragon Battlegrounds: Dawn (PBG-D)



Paragon Battlegrounds: Ruins (PBG-R)

Figure 18-4. *Views of all the different scenes that were used for testing.*

- *Bistro:* The Bistro scene has been modified to make the meshes of many of the different light sources actually emissive. In total, there are 20,638 textured emissive triangles, out of 2,829,226 total triangles. Overall, the light sources mainly consist of small light bulbs, with the addition of a few dozen small spotlights and a few emissive shop signs. The scene is mostly diffuse, with the exception of the bistro's windows and the Vespa.
- *Paragon Battlegrounds:* This scene is made of three different parts, of which we only use two: Dawn (PBG-D) and Ruins (PBG-R). Both consist of a mix of large emissive area lights located in the ground, as well as small ones such as runes engraved in rocks or small lights on the turrets; most of the materials are specular, with the exception of the trees. PBG-D features 90,535 textured emissive triangles, of which 53,210 are left after the texture integration; the whole scene is made of 2,467,759 triangles (emissive ones included). In comparison, PBG-R features 389,708 textured emissive triangles, of which 199,830 are left after the texture integration; the whole scene is made of 5,672,788 triangles (emissive ones included).

Note that although all these scenes are currently static, dynamic scenes are supported in our method by rebuilding the light acceleration structure per frame. Similar to how DXR allows refitting of the acceleration structure, without changing its topology, we could choose to update only the nodes in a pre-built tree if lights have not moved significantly between frames.

We use different abbreviations for some of the methods used in this section. Methods starting with "BVH_" will traverse the BVH hierarchy in order to select a triangle. The suffix after "BVH_" refers to which information is being used during the traversal: "D" for the distance between the viewpoint and a node's center, "F" for the flux contained in a node, "B" for the $\mathbf{n} \cdot \mathbf{l}$ bound, and finally "O" for the node orientation cone. The method Uniform uses MIS [41] to combine samples obtained by sampling the BRDF with samples obtained by randomly selecting an emissive triangle among all emissive triangles present in the scene with a uniform probability.

When MIS [41] is employed, we use the power heuristic with an exponent of 2. The sample budget is shared equally between sampling the BRDF and sampling the light source.

18.5.1 PERFORMANCE

18.5.1.1 ACCELERATION STRUCTURE CONSTRUCTION

Building the BVH using the SAH, with 16 bins on only the largest axis, takes about 2.3 ms on Sun Temple, 26 ms on Bistro, and 280 ms on Paragon Battlegrounds. Note that the current implementation of the BVH builder is CPU-based, is single-threaded, and does not make use of vector operations.

Binning along all three axes at each step is roughly 2× slower due to having three times more split candidates, but the resulting tree may not perform better at runtime. The timings presented here use the default setting of 16 bins per axis. Decreasing that number makes the build faster, e.g., 4 bins is roughly 2× faster, but again quality suffers. For the remaining measurements, we have used the highest-quality settings, as we expect that the tree build will not be an issue once the code is ported to the GPU and used for game-like scenes with tens of thousands of lights.

The build time with SAOH is about 3× longer than with SAH. The difference is mainly due to the extra lighting cone computations. We iterate once over all lights to compute the cone direction and a second time to compute the angular bounds. Using an approximate method or computing bounds bottom-up could speed this up.

Using the volume instead of the surface area did not result in any performance change for building.

18.5.1.2 RENDER TIME PER FRAME

We measured the rendering times with trees built using different heuristics and with all the sampling options turned on. See Table 18-1. Similarly to the build performance, using the volume-based metrics instead of surface area did not significantly impact the rendering time (usually within 0.2 ms of the surface area–based metric). Binning along all three axes or only the largest axis also has no significant impact on the rendering time (within 0.5 ms of each other).

Table 18-1. *Rendering times in milliseconds per frame with four shadow rays per pixel, measured over 1,000 frames and using the SAH and SAOH heuristics with different build parameters. The BVH_DFBO method was used with MIS, 16 bins were used for the binning, and at most one triangle was stored per leaf node.*

	SAH		SAOH	
	Largest Axis	All Axes	Largest Axis	All Axes
Sun Temple	16.9 ± 0.27	17.5 ± 0.10	17.3 ± 0.47	16.2 ± 0.30
Bistro (view 1)	30.3 ± 0.18	30.3 ± 0.61	31.8 ± 0.26	30.4 ± 0.20
Bistro (view 2)	38.8 ± 0.43	36.9 ± 0.30	39.6 ± 0.31	38.3 ± 1.12
Bistro (view 3)	31.2 ± 0.60	32.3 ± 0.19	33.0 ± 0.17	32.7 ± 0.20
PBG-D	23.6 ± 0.22	23.6 ± 0.19	23.7 ± 0.59	23.3 ± 0.20
PBG-R	40.5 ± 0.14	39.8 ± 0.15	41.9 ± 0.57	41.0 ± 0.16

When testing different maximum amounts of triangles per leaf node (1, 2, 4, 8, and 10), the rendering times were found to be within 5 % of each other with 1 and 10 being the fastest. Results for two of the scenes can be found in Figure 18-5, with similar behavior observed in the other scenes. The computation of the importance of each triangle adds a noticeable overhead. Conversely, storing more triangles per leaf node will result in shallower trees and therefore quicker traversal. It should be noted that the physical size of the leaf nodes was not changed (i.e., it was always set to accept up to 10 triangle IDs), only the amount that the BVH builder was allowed to put in a leaf node. Also for these tests, leaf nodes were created as soon as possible rather than relying on a leaf node creation cost.

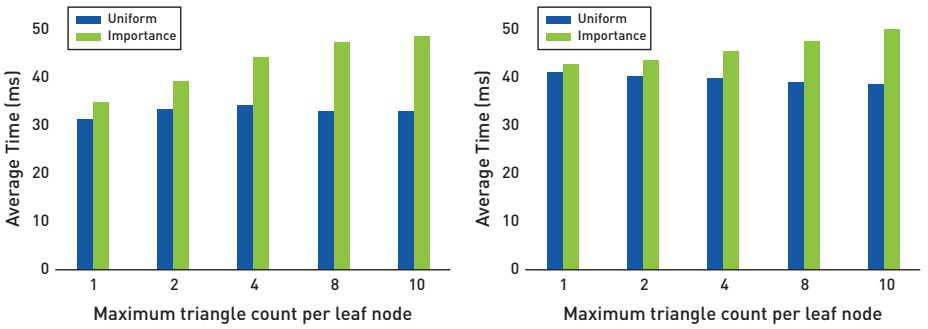


Figure 18-5. *Rendering times in milliseconds per frame for various maximum numbers of triangles per leaf node for Bistro (view 1) (left) and PBG-R (right), with and without importance sampling for triangle selection within the leaves. In all cases the BVH was built with 16 bins along all three axes using SAOH, and BVH_DFBO was used for the traversal.*

The use of SAOH over SAH results in similar rendering times overall, but the use of a BVH over the lights as well as which terms are considered for each node's importance do have an important impact, with BVH_DFBO being between 2× and 3× slower than Uniform. This is shown in Figure 18-6. This boils down to the additional bandwidth required for fetching the nodes from the BVH as well as the additional instructions for computing the $\mathbf{n} \cdot \mathbf{l}$ bound and the weight based on the orientation cone. This extra cost could be reduced by compressing the BVH nodes (using 16-bit floats instead of 32-bit floats, for example); the current nodes are 64 bytes for the internal nodes and 96 bytes for the external ones.

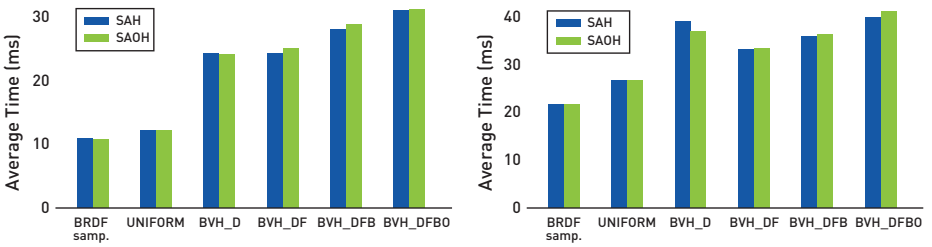


Figure 18-6. *Comparisons in Bistro (view 1) (left) and PBG-R (right) of rendering times in milliseconds per frame using the different traversal methods, compared to sampling the BRDF to get the light sample direction. All methods use 4 samples per pixel, and BVH-based methods use 16 bins along all three axes.*

18.5.2 IMAGE QUALITY

18.5.2.1 BUILD OPTIONS

Overall, the volume variants perform worse than their surface-area equivalents, and methods using 16 bins perform better than their counterparts only using 4 bins. As for how many axes should be considered for defining the best split, considering all three axes leads to lower mean squared error results in most cases compared to only using the largest axis, but not always. Finally, SAOH variants are usually better than or at least on par with their SAH equivalents. This can be highly dependent on how they formed their nodes at the top of the BVH: as those nodes contain most of the lights in the scene, they represent a poor spatial and directional approximation of the emissive surfaces that they contain.

This can be seen in Figure 18-7 in the area around the pharmacy shop sign (pointed at by the red arrow), for example at point *A* (pointed at by the white arrow). When using SAH, point *A* is closer to the green node than the magenta one, resulting in a higher chance of choosing the green node and therefore missing the green light emitted by the cross sign even though that green light is important, as can be seen in Figure 18-4 for Bistro (view 3). Conversely, with SAOH the point *A* has a high

chance of selecting the node containing the green light, improving the convergence in that region. However, it is possible to find regions where SAH will give better results than SAOH for similar reasons.

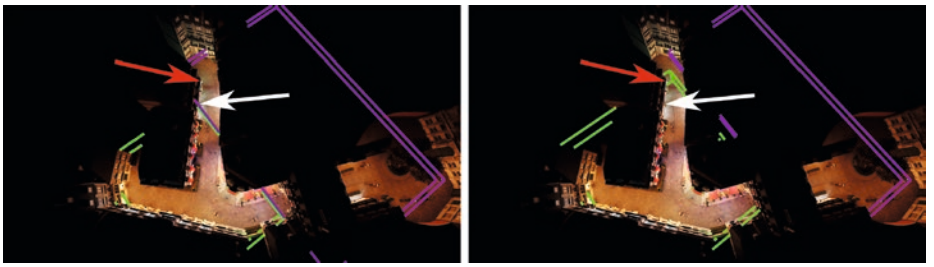


Figure 18-7. *Visualization of the second level of the BVH when built using SAH (left) and SAOH (right); the AABB of the left child is colored in green whereas the one of the right child is in magenta. In both cases, 16 bins were used and all three axes were considered.*

18.5.2.2 TRIANGLE AMOUNT PER LEAF NODE

As more triangles are stored in leaf nodes, the quality will degrade when using a uniform selection of the triangles because it will do a poorer job than the tree traversal. Using importance selection reduces the quality degradation compared to uniform selection, but it still performs worse than using only the tree. The results for Bistro (view 3) can be seen on the right in Figure 18-8.

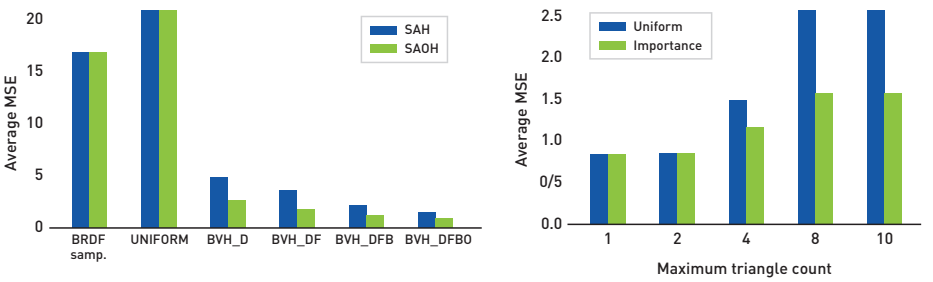


Figure 18-8. *Comparisons in Bistro (view 3) of mean squared error (MSE) results for the different traversal methods, compared to sampling the BRDF to get the light sample direction (left) and various maximum amounts of triangles for BVH_DFBO (right). All methods use 4 samples per pixel, and BVH-based methods use 16 bins along all three axes.*

18.5.2.3 SAMPLING METHODS

In Figure 18-9 we can see the resulting images when using and combining different sampling strategies for the Bistro (view 2) scene.

Figure 18-9. *Visual results at 4 samples per pixel (SPP) (left) and 16 SPP (right), using the different sampling strategies defined in Section 18.4.3. All BVH-based methods use a BVH built with SAOH evaluated for 16 bins along all axes. The BVH techniques use MIS: half their samples sample the BRDF and half traverse the light acceleration structure.*

As expected, using light sampling greatly outperforms the BRDF sampling approach, by being able to find some valid light paths at each pixel. Using the BVH with the distance as an importance value allows picking up of the contributions from nearby light sources, as can be seen for the two white light sources placed on each side of the door of the bistro, the different lights placed on its facade, or its windows.

When also considering the flux of the light sources during the traversal, we can observe a shift from a mostly blueish color (from the hanging small blue light sources closest to the ground) to a more yellowish tone coming from the different street lights, which might be located farther away but are more powerful.

Using the $\mathbf{n} \cdot \mathbf{l}$ bounds does not make much of a difference in this scene, except for the reflections on the Vespa (mostly visible on the 16 SPP images), but the effects can be way more pronounced in other scenes. Figure 18-10 shows an example from Sun Temple. There, using the bounds on $\mathbf{n} \cdot \mathbf{l}$ results in the back of the column on the right receiving more light and being distinguishable from the shadow it casts on the nearby wall, as well as the architectural details of the ceiling of the enclave in which the statue is located becoming visible.

Figure 18-10. *Visual results when not using the n · l bounds (left) compared to using it (right). Both images use 8 SPP (4 BRDF samples and 4 light samples) and a BVH binned along all three axes with 16 bins using SAH, and both take into account the distance and flux of the light.*

Even without SAOH, the orientation cone still has a small impact on the final image; for example, the facades in Figure 18-9 (at the end of the street and in the right-hand corner of the image) are less noisy compared to not using the orientation cones.

The use of an acceleration structure significantly improves the quality of the rendering, as seen in Figure 18-8, with between 4× and 6× improved average MSE score over the Uniform method even when only considering the distance to a node for that node's importance function. Incorporating the flux, the $\mathbf{n} \cdot \mathbf{l}$ bound and the orientation cone give a further 2× improvement.

18.6 CONCLUSION

We have presented a hierarchical data structure and sampling methods to accelerate light importance sampling in real-time ray tracing, similar to what is used in offline rendering [11, 22]. We have explored sampling performance on the GPU, taking advantage of hardware-accelerated ray tracing. We have also presented results using different build heuristics. We hope this work will inspire future work in game engines and research to incorporate better sampling strategies.

While the focus of this chapter has been on the sampling problem, it should be noted that any sample-based method usually needs to be paired with some form of denoising filter to remove the residual noise, and we refer the reader to recent real-time methods based on advanced bilateral kernels [25, 34, 35] as a suitable place to start. Deep learning–based methods [3, 7, 42] also show great promise. For an overview of traditional techniques, refer to the survey by Zwicker et al. [48].

For the sampling, there are a number of worthwhile avenues for improvement. In the current implementation, we bound $\mathbf{n} \cdot \mathbf{l}$ to cull lights under the horizon. It would be helpful to also incorporate BRDF and visibility information to refine the sampling probabilities during tree traversal. On the practical side, we want to move the BVH building code to the GPU for performance reasons. That will also be important for supporting lights on dynamic or skinned geometry.

ACKNOWLEDGEMENTS

Thanks to Nicholas Hull and Kate Anderson for creating the test scenes. The Sun Temple [13] and Paragon Battlegrounds scenes are based on assets kindly donated by Epic Games. The Bistro scene is based on assets kindly donated by Amazon Lumberyard [1]. Thanks to Benty et al. [4] for creating the Falcor rendering research framework, and to He et al. [16] and Jonathan Small for the Slang shader compiler that Falcor uses. We would also like to thank Pierre Moreau's advisor Michael Doggett at Lund University. Lastly, thanks to Aaron Lefohn and NVIDIA Research for supporting this work.

REFERENCES

[1] Amazon Lumberyard. Amazon Lumberyard Bistro, Open Research Content Archive (ORCA). http://developer.nvidia.com/orca/amazon-lumberyard-bistro, July 2017.

[2] Andersson, J. Parallel Graphics in Frostbite—Current & Future. Beyond Programmable Shading, SIGGRAPH Courses, 2009.

[3] Bako, S., Vogels, T., McWilliams, B., Meyer, M., Novák, J., Harvill, A., Sen, P., DeRose, T., and Rousselle, F. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Transactions on Graphics 36*, 4 (2017), 97:1–97:14.

[4] Benty, N., Yao, K.-H., Foley, T., Kaplanyan, A. S., Lavelle, C., Wyman, C., and Vijay, A. The Falcor Rendering Framework. https://github.com/NVIDIAGameWorks/Falcor, July 2017.

[5] Bikker, J. Real-Time Ray Tracing Through the Eyes of a Game Developer. In *IEEE Symposium on Interactive Ray Tracing* (2007), pp. 1–10.

[6] Bikker, J. *Ray Tracing in Real-Time Games*. PhD thesis, Delft University, 2012.

[7] Chaitanya, C. R. A., Kaplanyan, A. S., Schied, C., Salvi, M., Lefohn, A., Nowrouzezahrai, D., and Aila, T. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Transactions on Graphics 36*, 4 (2017), 98:1–98:12.

[8] Christensen, P. H., and Jarosz, W. The Path to Path-Traced Movies. *Foundations and Trends in Computer Graphics and Vision 10*, 2 (2016), 103–175.

[9] Clarberg, P., Jarosz, W., Akenine-Möller, T., and Jensen, H. W. Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions. *ACM Transactions on Graphics 24*, 3 (2005), 1166–1175.

[10] Clark, J. H. Hierarchical Geometric Models for Visibility Surface Algorithms. *Communications of the ACM 19*, 10 (1976), 547–554.

[11] Conty Estevez, A., and Kulla, C. Importance Sampling of Many Lights with Adaptive Tree Splitting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 2 (2018), 25:1–25:17.

[12] Dachsbacher, C., Křivánek, J., Hašan, M., Arbree, A., Walter, B., and Novák, J. Scalable Realistic Rendering with Many-Light Methods. *Computer Graphics Forum 33*, 1 (2014), 88–104.

[13] Epic Games. Unreal Engine Sun Temple, Open Research Content Archive (ORCA). http://developer.nvidia.com/orca/epic-games-sun-temple, October 2017.

[14] Goldsmith, J., and Salmon, J. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications 7*, 5 (1987), 14–20.

[15] Harada, T. A 2.5D Culling for Forward+. In *SIGGRAPH Asia 2012 Technical Briefs* (2012), pp. 18:1–18:4.

[16] He, Y., Fatahalian, K., and Foley, T. Slang: Language Mechanisms for Extensible Real-Time Shading Systems. *ACM Transactions on Graphics 37*, 4 (2018), 141:1–141:13.

[17] Heitz, E., Dupuy, J., Hill, S., and Neubelt, D. Real-Time Polygonal-Light Shading with Linearly Transformed Cosines. *ACM Transactions on Graphics 35*, 4 (2016), 41:1–41:8.

[18] Kajiya, J. T. The Rendering Equation. *Computer Graphics (SIGGRAPH)* (1986), 143–150.

[19] Karis, B. Real Shading in Unreal Engine 4. Physically Based Shading in Theory and Practice, SIGGRAPH Courses, August 2013.

[20] Keller, A. Instant Radiosity. In *Proceedings of SIGGRAPH* (1997), pp. 49–56.

[21] Keller, A., Fascione, L., Fajardo, M., Georgiev, I., Christensen, P., Hanika, J., Eisenacher, C., and Nichols, G. The Path Tracing Revolution in the Movie Industry. In *ACM SIGGRAPH Courses* (2015), pp. 24:1–24:7.

[22] Keller, A., Wächter, C., Raab, M., Seibert, D., van Antwerpen, D., Korndörfer, J., and Kettner, L. The Iray Light Transport Simulation and Rendering System. arXiv, https://arxiv.org/abs/1705.01263, 2017.

[23] Lagarde, S., and de Rousiers, C. Moving Frostbite to Physically Based Rendering 3.0. Physically Based Shading in Theory and Practice, SIGGRAPH Courses, 2014.

[24] MacDonald, J. D., and Booth, K. S. Heuristics for Ray Tracing Using Space Subdivision. *The Visual Computer 6*, 3 (1990), 153–166.

[25] Mara, M., McGuire, M., Bitterli, B., and Jarosz, W. An Efficient Denoising Algorithm for Global Illumination. In *Proceedings of High-Performance Graphics* (2017), pp. 3:1–3:7.

[26] NVIDIA. NVAPI, 2018. https://developer.nvidia.com/nvapi.

[27] O'Donnell, Y., and Chajdas, M. G. Tiled Light Trees. In *Symposium on Interactive 3D Graphics and Games* (2017), pp. 1:1–1:7.

[28] Olsson, O., and Assarsson, U. Tiled Shading. *Journal of Graphics, GPU, and Game Tools 15*, 4 (2011), 235–251.

[29] Olsson, O., Billeter, M., and Assarsson, U. Clustered Deferred and Forward Shading. In *Proceedings of High-Performance Graphics* (2012), pp. 87–96.

[30] Persson, E., and Olsson, O. Practical Clustered Deferred and Forward Shading. Advances in Real-Time Rendering in Games, SIGGRAPH Courses, 2013.

[31] Pharr, M. Guest Editor's Introduction: Special Issue on Production Rendering. *ACM Transactions on Graphics 37*, 3 (2018), 28:1–28:4.

[32] Pharr, M., Jakob, W., and Humphreys, G. *Physically Based Rendering: From Theory to Implementation*, third ed. Morgan Kaufmann, 2016.

[33] Rubin, S. M., and Whitted, T. A 3-Dimensional Representation for Fast Rendering of Complex Scenes. *Computer Graphics (SIGGRAPH) 14*, 3 (1980), 110–116.

[34] Schied, C., Kaplanyan, A., Wyman, C., Patney, A., Chaitanya, C. R. A., Burgess, J., Liu, S., Dachsbacher, C., Lefohn, A., and Salvi, M. Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination. In *Proceedings of High-Performance Graphics* (2017), pp. 2:1–2:12.

[35] Schied, C., Peters, C., and Dachsbacher, C. Gradient Estimation for Real-Time Adaptive Temporal Filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 2 (2018), 24:1–24:16.

[36] Schmittler, J., Pohl, D., Dahmen, T., Vogelgesang, C., and Slusallek, P. Realtime Ray Tracing for Current and Future Games. In *ACM SIGGRAPH Courses* (2005), pp. 23:1–23:5.

[37] Shirley, P., Wang, C., and Zimmerman, K. Monte Carlo Techniques for Direct Lighting Calculations. *ACM Transactions on Graphics 15*, 1 (1996), 1–36.

[38] Sobol, I. M. *A Primer for the Monte Carlo Method*. CRC Press, 1994.

[39] Talbot, J. F., Cline, D., and Egbert, P. Importance Resampling for Global Illumination. In *Rendering Techniques* (2005), pp. 139–146.

[40] Tokuyoshi, Y., and Harada, T. Stochastic Light Culling. *Journal of Computer Graphics Techniques 5*, 1 (2016), 35–60.

[41] Veach, E., and Guibas, L. J. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proceedings of SIGGRAPH* (1995), pp. 419–428.

[42] Vogels, T., Rousselle, F., McWilliams, B., Röthlin, G., Harvill, A., Adler, D., Meyer, M., and Novák, J. Denoising with Kernel Prediction and Asymmetric Loss Functions. *ACM Transactions on Graphics 37*, 4 (2018), 124:1–124:15.

[43] Wald, I. On Fast Construction of SAH-Based Bounding Volume Hierarchies. In *IEEE Symposium on Interactive Ray Tracing* (2007), pp. 33–40.

[44] Walter, B., Arbree, A., Bala, K., and Greenberg, D. P. Multidimensional Lightcuts. *ACM Transactions on Graphics 25*, 3 (2006), 1081–1088.

[45] Walter, B., Fernandez, S., Arbree, A., Bala, K., Donikian, M., and Greenberg, D. P. Lightcuts: A Scalable Approach to Illumination. *ACM Transactions on Graphics 24*, 3 (2005), 1098–1107.

[46] Ward, G. J. Adaptive Shadow Testing for Ray Tracing. In *Eurographics Workshop on Rendering* (1991), pp. 11–20.

[47] Zimmerman, K., and Shirley, P. A Two-Pass Solution to the Rendering Equation with a Source Visibility Preprocess. In *Rendering Techniques* (1995), pp. 284–295.

[48] Zwicker, M., Jarosz, W., Lehtinen, J., Moon, B., Ramamoorthi, R., Rousselle, F., Sen, P., Soler, C., and Yoon, S.-E. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum 34*, 2 (2015), 667–681.