Real-Time Rendering with Lighting Grid Hierarchy

Daqi Lin
University of Utah

Cem Yuksel
University of Utah

Figure 1: *An example frame rendered using our real-time global illumination solution with one million virtual point lights, computed by our method, using* $\alpha = 2$ *and* 4×4 *interleaved sampling. The render time is 24 ms on an NVIDIA RTX 2080 GPU at* 1280×720 *resolution.*

ABSTRACT

We present an extension of the lighting grid hierarchy method for real-time rendering with many lights on the GPU. We describe efficient methods for parallel construction of the lighting grid hierarchy and using it with deferred rending. We also present a method for estimating shadows from many lights with a small number of shadow samples using the ray tracing API on the GPU. We show how our approach can be used for real-time global illumination computation with virtual point lights.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**; **Ray tracing**.

KEYWORDS

Many lights, global illumination, ray tracing, importance sampling

ACM Reference Format:
Daqi Lin and Cem Yuksel. 2019. Real-Time Rendering with Lighting Grid Hierarchy. In *Symposium on Interactive 3D Graphics and Games (I3D '19), May 21–23, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, Article 8, 10 pages. https://doi.org/10.1145/3321361

I3D '19, May 21–23, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6310-5/19/05…$15.00
https://doi.org/10.1145/3321361

1 INTRODUCTION

Rendering with a large number of light sources (i.e. the many-lights problem) has been an important challenge in computer graphics. While there exists elegant offline rendering methods that provide sub-linear performance in the number of light sources [Hašan et al. 2007; Walter et al. 2005; Yuksel and Yuksel 2017], it still remains an open problem for real-time rendering.

In this paper we provide an extension on the recently-introduced lighting grid hierarchy method [Yuksel and Yuksel 2017], which was originally developed for rendering explosions by representing their illumination using many point lights, and we make it suitable for general-purpose real-time rendering on the GPU. Given a large number of light sources, we construct a lighting grid hierarchy on the GPU and use it for efficiently approximating the total lighting contribution from all lights in a deferred renderer. We achieve this by rendering the lights within the lighting grid hierarchy as range-limited light volumes and using a small number of shadow samples for approximating the shadows from all lights via a new importance sampling algorithm. The computation of the chosen shadow samples is performed using the recently-introduced ray tracing API on the GPU and a screen-space filter is used for eliminating the high-frequency noise of shadow sampling. We show how our method can be used for computing global illumination with a large number of virtual point lights at real-time frame rates (Figure 1). The technical contributions in this paper include:

- An efficient method for lighting grid hierarchy construction on the GPU,
- An importance sampling algorithm for estimating shadow contributions of all lights with a fixed memory footprint,

- A hybrid ray tracing-rasterization approach for rendering high-quality diffuse-dominant global illumination in complex scenes using many virtual lights, including dynamic lighting and dynamic geometry at real-time frame rates.

2 BACKGROUND

In this section we briefly overview the related work in computer graphics regarding rendering with many-lights and real-time global illumination computation. We also provide a summary of the lighting grid hierarchy method [Yuksel and Yuksel 2017].

2.1 Prior Work

The many-lights problem received considerable attention in computer graphics [Dachsbacher et al. 2014], starting with ordering lights based on their contributions [Ward 1994], stochastic sampling [Shirley et al. 1996], light clustering using octrees [Paquette et al. 1998], and precomputed visibility culling [Fernandez et al. 2002]. The lightcuts method [Walter et al. 2005] provides a highly efficient scalable solution to the many lights problem by forming a binary light tree from the light sources. Its extensions address high-dimensional integrations [Walter et al. 2006], progressive GPU implementation [Davidovič et al. 2012], bidirectional sampling [Walter et al. 2012], and out-of-core GPU implementation for large scenes [Wang et al. 2013]. An alternative solution to the many-lights problem forms a lighting matrix and approximates its solution [Hašan et al. 2007]. The extensions of this approach include a method for handling glossy surfaces [Davidovic et al. 2010], reducing flickering by processing animated sequences [2008], and using cuts [Ou and Pellacini 2011] or a reduced matrix [Huo et al. 2015] for accelerating the computation. Recently, the lighting grid hierarchy method [Yuksel and Yuksel 2017] was introduced for rendering explosions by representing their illumination using many virtual point lights. We extend this method in this paper by providing a GPU-friendly variant that is suitable for real-time rendering with many lights, so we discuss this method in more detail below (Sec. 2.2).

Most interactive global illumination methods aim to provide a fast estimation of the rendering equation [Kajiya 1986] with geometry approximations using voxels [Crassin et al. 2011; Kaplanyan and Dachsbacher 2010], surfels [Christensen 2008; Ritschel et al. 2009a], or spheres [Ren et al. 2006; Sloan et al. 2007]; lighting approximations using photons [Hachisuka et al. 2008; Jensen 1996], virtual point lights [Keller 1997; Segovia et al. 2006a], or spherical functions [Green et al. 2007; Ramamoorthi and Hanrahan 2001; Sloan et al. 2002]; screen space techniques [Mara et al. 2016; McGuire and Mara 2014; Moreau et al. 2016; Nichols et al. 2009; Ritschel et al. 2009b]; caching [Jendersie et al. 2016; Vardis et al. 2014]; or reconstruction from sparse samples [Krivanek et al. 2005; McGuire et al. 2017; Silvennoinen and Lehtinen 2017].

The method we describe in this paper uses virtual point lights (VPLs) [Keller 1997]. Advantages of using VPLs include easy implementation, stable appearance, and scalability. Due to the low-frequency nature of diffuse reflection, VPLs are particularly effective in rendering diffuse indirect reflection, which, in many cases, is the most important part of global illumination. However, the singularity of point lights cause practical problems. Simply clamping the inverse square attenuation leads to energy loss. Energy compensation methods that use path tracing [Kollig and Keller 2006], screen space sampling [Novák et al. 2011] or a mixture with photon mapping [Sriwasansak et al. 2018] have been developed to solve this issue, but they are computationally expensive, particularly for real-time rendering.

An important obstacle for using VPLs in real-time rendering has been the challenge of efficiently handling many light sources. Clustered shading [Olsson et al. 2012] is the first method that presented real-time rendering performance with one million point lights; however, it assumes local illumination. Stochastic light culling [Tokuyoshi and Harada 2016] achieves interactive rates by fitting VPLs into the tiled shading framework [Olsson and Assarsson 2011], but introduces banding artifacts that are difficult to filter. Forward light cuts [Laurent et al. 2016] can compute the illumination of many VPLs using a multi-scale radiance cache, but shadows are not accounted. More recently, Estevez and Kulla [2018] introduced an efficient method for importance sampling many lights during path tracing by stochastically traversing a bounding volume hierarchy of light clusters, and this method is recently extended to real-time rendering [Moreau and Clarberg 2019].

Computing shadows from many VPLs has been another related challenge. There are solutions for real-time shadow computation from hundreds of lights [Olsson et al. 2015, 2014], but scenes using VPLs for indirect illumination usually require thousands of VPLs or more. Traditionally, shadows are computed for each of the VPLs, but this can be too expensive for real-time rendering, unless combined with a subsampling technique. Harada et al. [2013] proposed a method for efficiently casting shadow rays to lights within each render tile, but it does not solve the problem for virtual point lights with potentially global influence radius. Imperfect shadow maps [Ritschel et al. 2008] use a point cloud representation of the scene geometry to significantly reduce the shadow mapping cost at the expense of shadow quality.

2.2 Lighting Grid Hierarchy

The lighting grid hierarchy method [Yuksel and Yuksel 2017] provides an effective solution to the many-lights problem, though it was originally introduced for rendering explosions with self-illumination by representing the volumetric illumination data as many virtual point lights. As opposed to alternative solutions to the many-lights problem, lighting grid hierarchy provides a temporally stable computation. It also allows efficiently precomputing and storing shadows for all lights, which leads to orders of magnitude faster computation with volumetric shadows needed for rendering explosions. In this paper we extend this approach by providing an efficient parallel construction method, presenting a technique for efficiently computing the lighting from the hierarchy on the GPU, and introducing an importance sampling algorithm that avoids shadow precomputation, all of which are crucial for achieving real-time frame rates.

The lighting grid hierarchy method represents the entire illumination from all lights at multiple resolutions. Each level of the hierarchy corresponds to a different resolution representation that approximates the original lights using fewer light sources. A level is

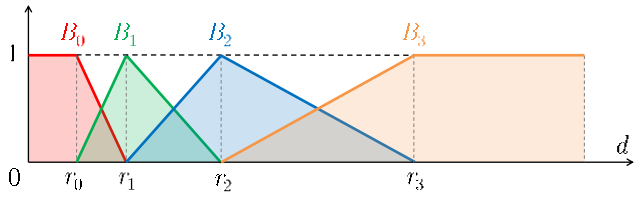


Figure 2: *The blending functions B_0, B_1, B_2, and B_3 of lighting grid hierarchy with $\ell_{\max} = 3$, forming a partition of unity for any distance d from the point where lighting is computed.*

constructed by placing a volumetric grid that encapsulates the original lights. The vertices of the grid approximate the lights around them, such that the contribution of each original light is distributed to the eight neighboring grid vertices using trilinear weights. A *grid light* is generated from each grid vertex with non-zero illumination and placed at the illumination center of the original lights it represents. The highest resolution grid forms level 1 with the set of light sources $\mathbb{S}_1$. Higher levels ℓ of the hierarchy use grid cells with twice the size in all dimensions as compared to the level $\ell - 1$ right below them. The highest (coarsest) level $\ell_{\max}$ typically has a single cell with 8 vertices, forming $\mathbb{S}_{\ell_{\max}}$. Therefore, the number of levels constructed depends on the resolution of level 1. The original lights are kept at level zero, forming $\mathbb{S}_0$.

For providing an efficient solution to the many-lights problem, a lighting grid hierarchy approximates the light coming from different distances using different levels of the hierarchy, providing different resolution representation of the original lighting. This is accomplished using *blending functions* that form a partition of unity for any distance from the point where lighting is computed (Figure 2). These blending functions determine the influence regions of the lights at each grid level and the incoming illumination from a light is modulated by the corresponding blending function value. Let h_ℓ be the grid size of level ℓ. The non-zero regions of the blending functions are determined by distances $r_\ell = \alpha h_\ell$, where α is a user-defined parameter that determines the accuracy of the lighting approximation. Larger α values lead to blending functions with larger non-zero regions and result in using more grid lights for estimating lighting with higher accuracy.

3 RENDERING WITH MANY LIGHTS

Our rendering algorithm uses the lighting grid hierarchy method [Yuksel and Yuksel 2017] to efficiently evaluate the illumination from a large number of point lights. In our experiments we use this algorithm for computing indirect illumination with many virtual point lights (VPLs) [Keller 1997], though our lighting computation is independent from how the point lights are generated.

We begin with constructing a lighting grid hierarchy from the given point lights on the GPU (Sec. 3.1). We use this lighting grid hierarchy within a deferred renderer for efficiently estimating the illumination from all lights (Sec. 3.2). While computing the lighting, we stochastically pick a fixed number of shadow samples to be computed via ray tracing on the GPU (Sec. 3.3). Finally, we filter the computed shadows to eliminate the high-frequency sampling noise and use the result as shadow ratio estimators for computing the final lighting approximation.

3.1 Lighting Grid Hierarchy Construction

The problem of constructing a lighting grid hierarchy is similar to the particle-to-grid transfer operations used in hybrid Lagrangian-Eulerian simulation systems [Gao* et al. 2018; Wu et al. 2018]. Each level of the hierarchy can be constructed using either *scatter* [Gao* et al. 2018] or *gather* [Wu et al. 2018] operations. The scatter approach loops over each light and adds its illumination to the corresponding grid vertices. Since the parallel scatter loop involves atomic operations, it can be highly inefficient for higher (coarser) levels of the hierarchy, as the small number of target grid vertices at these levels lead to frequent thread contentions in atomic operations. The gather approach, on the other hand, loops over each grid vertex and finds the corresponding lights that contribute to the vertex. This eliminates the need for atomic operations, but requires search operations for finding the corresponding lights. This search can be accelerated by a pre-ordering step [Wu et al. 2018], which can also be expensive to compute.

To provide an efficient parallel construction algorithm, we split the construction process into two steps. In the first step we scatter the contributions of each input light to the first grid level $\mathbb{S}_1$ with the highest resolution. Since this level involves a relatively small percentage of thread contentions, the related atomic operations can be performed efficiently. In the second step we build the rest of the levels using a gather approach. To avoid an expensive pre-ordering step, we build these levels using the grid lights of the first level $\mathbb{S}_1$, which are already ordered by construction. This approach, as opposed to generating all levels directly from the input lights $\mathbb{S}_0$, leads to some smoothing in the final lighting approximation from the lighting grid hierarchy, but provides a highly efficient mechanism for the parallel construction process. Since VPLs are placed only on surfaces, a significant portion of the grid vertices in the volume may contain no illumination, especially for the lower (finer) levels of the hierarchy. Therefore, the construction process is completed by a stream compaction pass that is applied for each level to remove the large percentage of unused grid vertices.

Since we use a bottom-up construction of the hierarchy by building $\mathbb{S}_1$, we must first determine the size of the grid cells h_1. We begin with computing the bounding box of all input lights and set the grid size of the top level $\mathbb{S}_{\ell_{\max}}$, which only contains a single cell (i.e. 8 grid lights), as the longest edge of this bounding box. The grid size for the first level $\mathbb{S}_1$ is computed using $h_1 = h_{\ell_{\max}} / 2^{\ell_{\max}-1}$. In our implementation the number of lighting grid hierarchy levels $\ell_{\max}$ is controlled by a user-specified parameter.

3.2 Lighting Computation

If the lighting grid is densely populated, such that each grid vertex contains a light with non-zero intensity, lights around a shaded point can be directly gathered from the grid. However, the stream compaction pass we use for eliminating grid lights with zero intensities prevents trivially finding the lights around a shaded points from their grid locations. Therefore, we use the lighting grid hierarchy within a deferred renderer for estimating the illumination from all lights with a light rasterization step. After generating G-buffers for the scene geometry, we rasterize the lights in the lighting grid hierarchy as (coarse) spheres (approximated using cubes in practice) [Dachsbacher and Stamminger 2006]. Since the blending function

values for the lights in the lighting grid hierarchy are zero beyond the distance $2r_\ell$ from the light sources, for each light we draw a sphere with $2r_\ell$ radius. The exception is the 8 lights in the top level of the hierarchy, which use blending functions that do not go to zero with increasing distance, so these lights can be handled separately by drawing a screen-size quad. This process produces fragments for each pixel that the lights can potentially illuminate with a non-zero blending function value. Yet, the blending functions can still evaluate to zero for some of these fragments, since each grid light at the higher levels of the hierarchy has a minimum illumination radius $r_\ell/2$ within which the blending function is zero (see Figure 2). Therefore, we compute the blending function for each fragment and discard the fragment if it is zero.

We perform the lighting computation for each fragment with a non-zero blending function value and accumulate the result without considering shadows. Shadows are computed separately, as explained below (Sec. 3.3).

3.3 Shadow Sampling

The lighting grid hierarchy allows estimating the illumination using a small subset of all lights. Yet, in practice lighting computation of each pixel still involves hundreds of lights with non-zero blending function values (especially with relatively large α parameters) and computing shadows for each one of these lights can be prohibitively expensive for real-time rendering. Even though the lighting grid hierarchy method allows precomputing shadows (such as shadow maps) with a reasonable memory usage, this precomputation can easily become the bottleneck of the entire rendering process. Therefore, we instead stochastically estimate shadows using a fixed number of samples, which are computed via ray tracing on the GPU. We pick these shadow samples during the lighting computation (Sec. 3.2). The shadows computed from these samples are then used as shadow ratio estimators [Heitz et al. 2018].

We must pick the shadow samples randomly, independent of the order in which the lights are processed, to avoid introducing bias in shadow sampling. Furthermore, this random sampling should be performed independently for each pixel to avoid correlation in sampling. This process requires considering the set of all lights that have non-zero illumination contributions for each pixel. Moreover, since the illumination contributions of each light in the lighting grid hierarchy can vary drastically, using an importance sampling scheme is crucial for reducing the variance in sampling.

We pick a small number of shadow samples for each pixel during the lighting computation, while rendering the lights as spheres (Sec. 3.2). These shadow samples are evaluated after the lighting computation via tracing shadow rays on the GPU from the pixel positions towards the selected shadow sample locations. The lighting grid hierarchy we construct contains the variance of the illumination center for each light. We use these variance values for randomly picking shadow sample positions to produce area shadows, as opposed to using the illumination centers directly. Each one of the shadow samples of a pixel is selected independently. Therefore, it is possible to have multiple shadow samples of a pixel belonging to the same light source, though it is improbable in practice, considering that each pixel is illuminated by hundreds of lights. Nonetheless, even if two shadow samples of a pixel use the same light, they are likely to send shadow rays towards slightly different directions.

Let f_i be the probability density of picking the light source i for shadow sampling, such that the probability of picking the light source is $p_i = f_i / \sum_j^n f_j$, where n is the total number of lights in the hierarchy. These f_i values are determined per pixel based on the illumination contribution of each light for importance sampling, such that f_i is non-zero if and only if the light has non-zero illumination contribution (disregarding potential shadowing). During lighting computation, we store a running total for the cumulative probability density $\hat{f}_i = \sum_j^i f_j$ for each pixel. For each fragment with a non-zero f_i value, we decide whether to use it as a shadow sample stochastically with probability $\tilde{p}_i = f_i / \hat{f}_i$, using the accumulated probability density $\hat{f}_i$ while rendering the light source i. This stochastic decision is performed separately for each shadow sample of the pixel. If the light is selected as a shadow sample, it overwrites the previously selected sample. Note that the first light of a pixel with $\tilde{p}_1 = f_1 / \hat{f}_1 = 1$ is always selected as a shadow sample, though succeeding lights can overwrite the shadow sample. At the end of the lighting computation, this process provides k shadow samples each with the desired probability p_i (see Appendix A for a proof), where k is the number of shadow samples per pixel, controlled as a user-defined parameter.

After the lighting computation, during which k shadow samples are picked, we trace shadow rays on the GPU to determine a binary shadow value for each sample. The average of the k samples provide the shadow value for the pixel, which can be used as a shadow ratio estimator [Heitz et al. 2018].

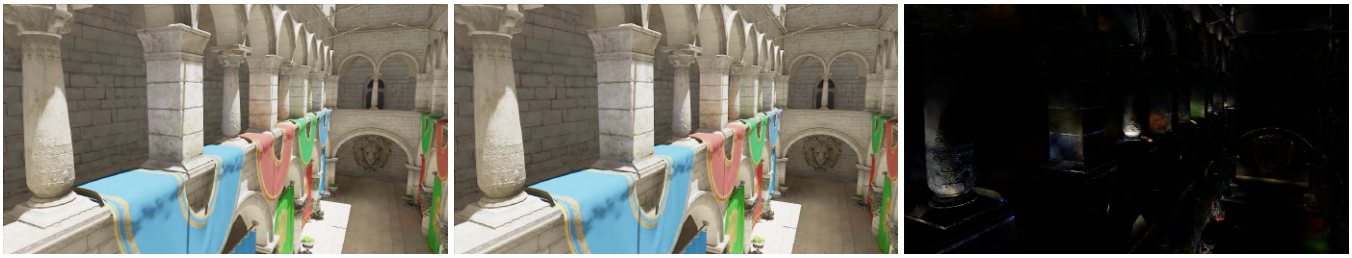
Stochastic shadow sampling, as explained above, leads to a substantial amount of noise when using a small number of shadow samples. For eliminating the high-frequency noise in shadow sampling a screen-space bilateral filter can be applied to the computed shadow values before using them as shadow ratio estimators [Heitz et al. 2018]. In our tests we use a wavelet-based filter [Dammertz et al. 2010], which we found to be more effective for filtering the shadow noise of VPLs used for computing diffuse-dominant global illumination.

4 IMPLEMENTATION AND RESULTS

We evaluate our method by computing indirect illumination with VPLs [Keller 1997]. The VPLs are generated by tracing light rays up to three diffuse bounces. When the lighting condition changes, we regenerate all VPLs and construct a new lighting grid hierarchy. We use the DirectX ray tracing API for both generating VPLs and computing shadow rays. All performance results are generated using an NVIDIA RTX 2080 graphics card at 1280×720 resolution.

4.1 Additional Optimizations

The process of picking the shadow samples (Sec. 3.3) involves atomic operations for updating the running total for the cumulative probability density $\hat{f}_i$ and overwriting the shadow samples. However, in practice the impact of thread contentions during the lighting computation on the final result can be negligible. An exception is rendering to very small viewports. In our tests we found that disabling thread locks produces virtually identical results with 10–20% improvement in render times. Therefore, the results in



(a) *Sample shadows from* $\mathbb{S}_1$ *Render time: 32.5 ms*

(b) *Sample shadows from* $\mathbb{S}_2$ *Render time: 28.5 ms*

(c) *Difference* ×8

Figure 3: *Comparison of shadow sampling with and without using the lowest (finest) level of the hierarchy* $\mathbb{S}_1$*, using 100K VPLs, 4 shadow samples, and* $\alpha = 1$*.*

this paper are generated without thread locks, unless otherwise indicated.

Note that each level of the lighting grid hierarchy encodes the entire illumination of all input lights with a different resolution. Therefore, it is possible to skip using the actual input lights altogether and begin the lighting computation using the first level of the hierarchy $\mathbb{S}_1$ instead. This can significantly reduce the overdraw caused by rendering spheres for each light source and accelerate the lighting computation, but it also introduces some smoothing to the estimated illumination. Yet, in the case of using VPLs for computing indirect illumination, this additional smoothing can even be preferable in practice. Therefore, all results in this paper are generated using the lighting grid hierarchy starting from $\mathbb{S}_1$.

In addition, due to the large number of grid lights in $\mathbb{S}_1$ and their relatively small illumination radii, skipping $\mathbb{S}_1$ grid lights for shadow sampling can significantly reduce the memory bandwidth and computation time, without obvious impact on the render quality. In our tests we observed an additional 10–15% speedup by skipping $\mathbb{S}_1$ for shadow sampling with almost identical render results, as can be seen in Figure 3. Therefore, the results in this paper do not use $\mathbb{S}_1$ for shadow sampling, unless otherwise specified.

4.2 Lighting Grid Hierarchy Construction

Rendering begins with constructing a lighting grid hierarchy, which is reconstructed every time the illumination changes and a new set of VPLs are generated. Table 1 compares the computation time of parallel lighting grid hierarchy construction using scatter operations on the input VPLs and our gather approach using $\mathbb{S}_1$ for computing the higher levels of the hierarchy. The computation time of each step is listed in the table. The first step computes the collective bounding box of the lights and the final step merges the grid lights of all levels into a single buffer to avoid multiple draw calls during light rasterization. Notice that our gather method is more than an order of magnitude faster than the scatter approach. The first two (compute bounds and compute $\mathbb{S}_1$) and the last (merge levels) operations are identical in both cases. The difference in performance comes from the thread contentions of the scatter operations for computing the higher levels of the hierarchy. In comparison, we can efficiently construct a lighting grid hierarchy by generating higher levels from $\mathbb{S}_1$.

A qualitative comparison of lighting grid hierarchy construction methods is provided in Figure 4. Notice that the two methods

Table 1: *Breakdown of lighting grid hierarchy construction time.*

	Scatter VPLs	Gather from $\mathbb{S}_1$
Compute bounds	1.7 ms	1.7 ms
Compute $\mathbb{S}_1$	2.3 ms	2.3 ms
Compute $\mathbb{S}_2$	2.3 ms	1.0 ms
Compute $\mathbb{S}_3$	4.7 ms	1.0 ms
Compute $\mathbb{S}_4$	23 ms	2.0 ms
Compute $\mathbb{S}_5$	107 ms	1.0 ms
Compute $\mathbb{S}_6$	405 ms	1.1 ms
Compute $\mathbb{S}_7$	1,563 ms	1.5 ms
Merge levels	0.5 ms	0.5 ms
Total	**2,110 ms**	**12.1 ms**

The timings are generated by averaging 256 frames using 100K VPLs with the scene in Figure 5.

for parallel lighting grid hierarchy construction produces similar results. Yet, an extra level of smoothing and light leakage can be observed when using our gather operations from $\mathbb{S}_1$ (Figure 4b).

As one would expect, the lighting grid hierarchy construction time depends on the number of VPLs. The total construction times for different number of VPLs can be seen in Table 2, showing that the construction time using our gather approach grows sublinearly with the increasing number of VPLs.

(a) *Scatter VPLs* **(b)** *Gather from* $\mathbb{S}_1$

Figure 4: *Lighting grid hierarchy construction methods using (a) scatter operations on the input VPLs and (b) our gather operations using the first level* $\mathbb{S}_1$ *of the hierarchy, producing similar results.*

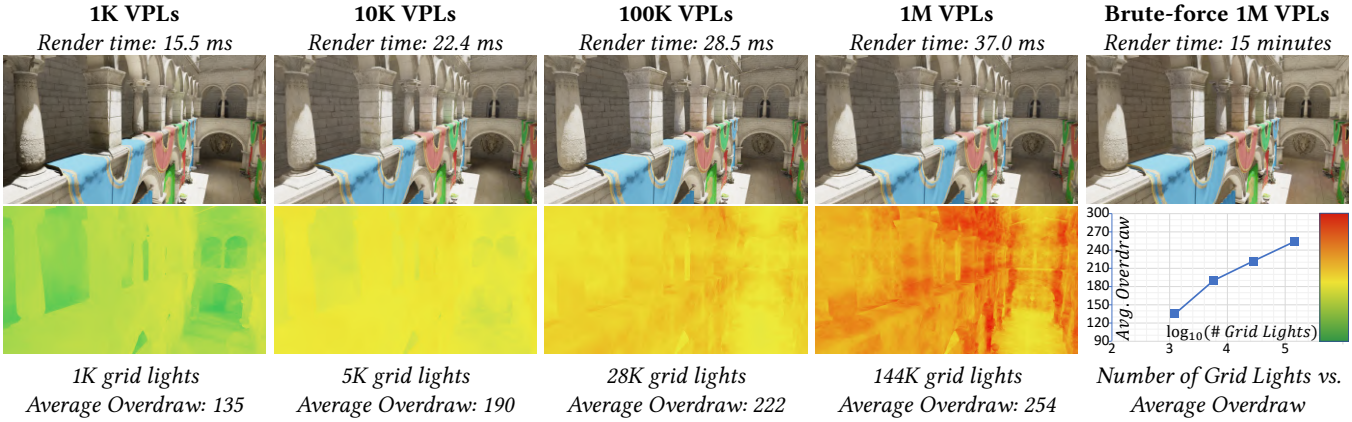


Figure 5: *Heatmaps showing the number of overdraw per pixel during the lighting computation for lighting grid hierarchies generated from different numbers of VPLs and a log plot of number of grid lights used in lighting computation vs. average overdraw per pixel, showing the logarithmic growth in average overdraw as compared to the increasing number of lights. The first row shows the corresponding render results using our method with 4 shadow samples per pixel and the brute-force reference generated by computing shadows of each VPL.*

4.3 Rendering

Figure 1 shows an example image rendered using our method for computing global illumination with VPLs. As expected, our method can produce high-quality global illumination, since we can efficiently compute lighting from a large number of VPLs. The performance and the quality of our results depend on the number of VPLs used, the number of shadow samples per pixel, and the parameter α of the lighting grid hierarchy method that determines the number of light samples used for estimating the illumination.

Obviously, using more VPLs leads to a better approximation of global illumination and it also increases the render time. Since we do not directly use the VPLs in the lighting computation, the performance of our results depend on the number of grid lights in the higher levels of the hierarchy. In our tests we set the number of levels for the lighting grid hierarchy according to the number of VPLs and we pick the largest number of levels, such that the number of lights in $\mathbb{S}_1$ is less than half of the number of VPLs. Thus, in our test the number of lights in the hierarchy is roughly proportional to the number of VPLs. However, as shown in Table 2, the render time does not linearly scale with the number of VPLs and we achieve a sublinear growth in render time with increasing VPL count.

Table 2 also shows that using a single shadow sample per pixel is significantly cheaper than 4. It is important to note that most of the extra cost of having additional shadow samples is related to the process of picking shadow samples during lighting computation. The actual shadow computation via tracing shadow rays is much faster in comparison.

The sublinear growth in render time with increasing VPL count can also be observed by investigating the number of overdraw (i.e. the number of fragments generated per pixel) during the lighting computation. Note that the lighting computation is the bottleneck of our system and the actual computation is proportional to the number of fragments generated. Figure 5 shows heatmaps indicating the number of overdraw per pixel for different number of VPLs.

Table 2: *Computation & render times with different number of VPLs.*

Number of VPLs	1K	10K	100K	1M
VPL Generation	0.1 ms	0.2 ms	0.4 ms	2.5 ms
Hierarchy Construction	7.0 ms	9.2 ms	12.1 ms	32.0 ms
Render time (no shadow)	5.6 ms	8.1 ms	11.2 ms	16.0 ms
Render time (1 shadow/pixel)	10.1 ms	14.3 ms	18.4 ms	24.3 ms
Render time (4 shadows/pixel)	15.5 ms	22.4 ms	28.5 ms	37.0 ms

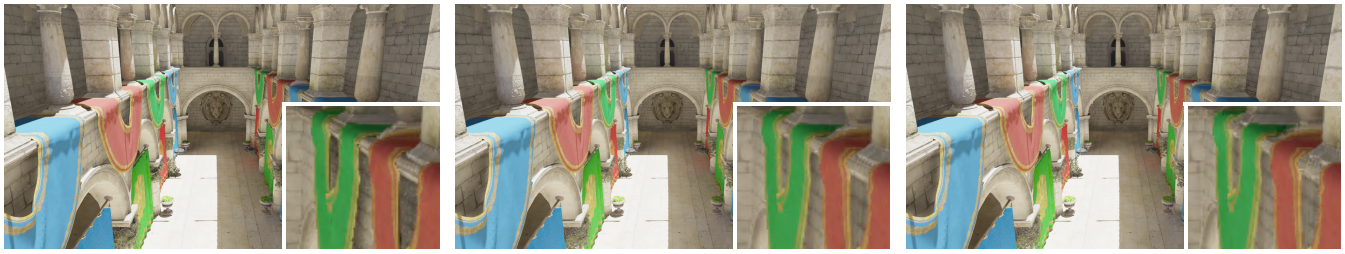
The timings are generated using the camera angle in Figure 5.

Notice that the number of lights in the hierarchy grows proportional to the number of VPLs, but overdraw grows approximately logarithmically with increasing number of lights.

One way to reduce the number of overdraw is using interleaved sampling [Keller and Heidrich 2001; Segovia et al. 2006b; Wald et al. 2002], which splits the frame buffer into a small number of tiles. The lights at each level of the hierarchy are distributed evenly to each tile and each light is rasterized onto only one of the tiles. Since the tiles have much lower resolution than the combined frame buffer, lights rendered onto a tile produces fewer fragments than rendering onto the entire frame. The final image of the combined frame buffer is constructed during the final compositing pass. By reducing overdraw in lighting computation, interleaved sampling significantly improves the total render time, but also leads to additional smoothing in the final illumination estimation.

Figure 6 shows a comparison of images generated with and without interleaved sampling. Since indirect illumination is mostly smooth, the differences caused by the additional smoothing of interleaved sampling are not easy to notice. Yet, there exists some additional smoothing in indirect shadows, especially visible behind the draping cloth shown in the insets.

Table 3 provides the break-down of computation time for all rendering operations with and without interleaved sampling. Notice that without interleaved sampling lighting computation takes most of the rendering time (82.1% in this example). Using interleaved sampling with 2×2 and 4×4 significantly reduces the lighting



(a) *No interleaved sampling*
Render time: 30.2 ms

(b) 2×2 *interleaved sampling*
Render time: 15.2 ms

(c) 4×4 *interleaved sampling*
Render time: 10.1 ms

Figure 6: *Comparison of our method with and without interleaved sampling using 100K VPLs, 4 shadow samples, and* $\alpha = 1$.

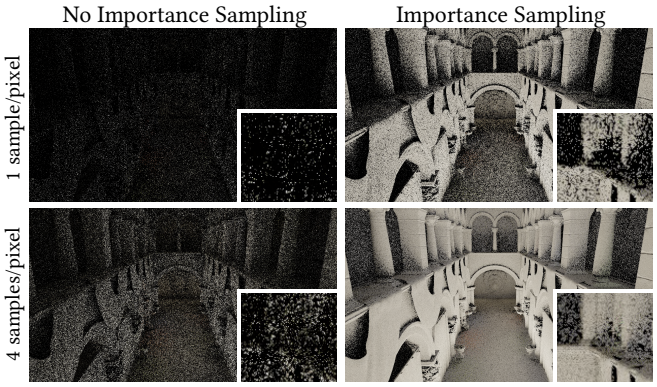


Figure 7: *Shadowed indirect illumination without shadow filtering, generated with and without importance sampling using 1 and 4 shadow samples per pixel.*

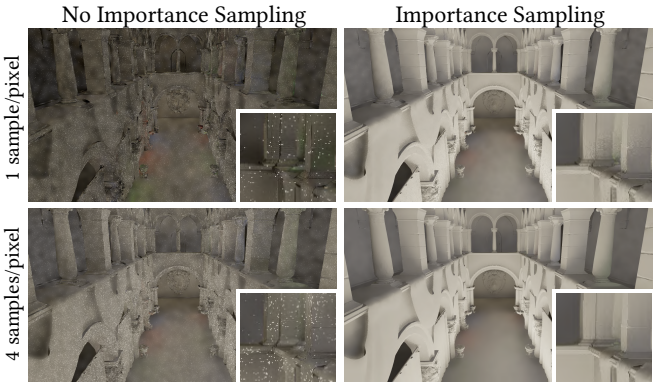


Figure 8: *Shadowed indirect illumination with shadow filtering, generated with and without importance sampling using 1 and 4 shadow samples per pixel.*

Table 3: *Computation time of rendering operations.*

Interleaved sampling	not used		2×2		4×4	
G-buffer generation	1.1 ms	3.5%	1.1 ms	7.0%	1.0 ms	5.9%
Lighting computation	24.8 ms	82.1%	9.0 ms	59.4%	3.8 ms	25.5%
Shadow ray tracing	2.3 ms	7.7%	2.2 ms	14.5%	2.2 ms	28.9%
Shadow filtering	1.9 ms	6.3%	2.8 ms	18.4%	2.9 ms	37.4%
Final compositing	0.1 ms	0.4%	0.1 ms	0.8%	0.1 ms	2.3%
Total render time	**30.2 ms**		**15.2 ms**		**10.1 ms**	

The timings are generated with 100K VPLs, 4 shadow samples per pixel, and $\alpha = 1$*, using the camera angle in Figure 6. Thread locks are used in lighting computation for only* 4×4 *interleaved sampling to prevent race conditions, which can be prominent in this case. The total render times do not include 0.4 ms used for generating VPLs and 12.1 ms used for constructing the lighting grid hierarchy.*

computation time. For dynamic scenes where lighting condition or scene geometry constantly changes, the cost of VPL generation and lighting grid hierarchy also need to be taken account into the total render cost.

Importance sampling for shadow computation is a crucial component of our method. Figures 7 and 8 compare the results of indirect shadows before and after filtering, computed with and without importance sampling. Notice that shadows are extremely noisy without importance sampling and the filtered shadows still contain a substantial amount of lower-frequency noise even with 4 shadow samples per pixel. In comparison, we achieve superior results with a single shadow sample per pixel using importance sampling. Our shadows are further improved using more samples.

The lighting grid hierarchy method with smaller α values produces a smoother lighting approximation by considering fewer lights for each lighting computation. A comparison with two different α values is shown in Figure 9. Since indirect illumination is mostly smooth, the differences between the two alpha values are difficult to notice. On the other hand, doubling the alpha value triples the render time for this example.

(a) $\alpha = 1$ *(Render time: 31 ms)*

(b) $\alpha = 2$ *(Render time: 93 ms)*

Figure 9: *While the accuracy of the lighting approximation improves with increasing* α *parameter of lighting grid hierarchy, the quality improvement in indirect illumination can be difficult to see.*

(a) *VXGI*
Render time: 21 ms

(b) 4×4 *interleaved sampling*
Render time: 15 ms

(c) *No interleaved sampling*
Render time: 93 ms

(d) *Path Tracing Reference*
Render time: 2 hours

Figure 10: *Comparison of global illumination computation using (a) voxel cone tracing generated with the highest quality settings in NVIDIA VXGI 2.0, (b) our method with* 4×4 *interleaved sampling and* $\alpha = 1.5$*, (c) our method with no interleaved sampling and* $\alpha = 2$*, and (d) path tracing reference. The given render times for VXGI and our method do not include construction time. 100K VPLs are used for generating our results.*

Global illumination computation with a large number of VPLs produces high-quality results and our approach, especially when combined with interleaved sampling, provides the necessary performance for achieving real-time frame rates. We compare the results of our method to the NVIDIA VXGI implementation of voxel cone tracing [Crassin et al. 2011] and path tracing [Kajiya 1986] in Figure 10. Notice that using our method we can produce a reasonably close solution to the path tracing reference at interactive frame rates (Figure 10c). Using interleaved sampling with our method (Figure 10b) introduces additional smoothing in lighting approximation, but provides a highly efficient solution to real-time global illumination computation. Notice that the indirect shadows (e.g. behind the draping cloth and on the columns) are properly reproduced using our method, but with some extra smoothing as compared to the path tracing reference. In comparison to VXGI (Figure 10a), we achieve closer results to the path tracing reference with relatively less smoothing introduced to the indirect illumination estimation.

Although further reducing the render time, ignoring shadows for indirect lighting can produce unrealistic results, which is more pronounced in brightly lit scenes. Figure 11 shows different scenes rendered using our algorithm and compares our method with 1 and 4 shadows samples per pixel to direct illumination only and indirect illumination without shadows. Notice that without indirect shadows the resulting indirect illumination is overestimated and it becomes relatively flat.

While the lighting grid hierarchy method provides a temporally stable solution to rendering with many lights, it does not eliminate any underlying flickering of VPLs. In dynamic scenes, where the direct illumination (and/or scene geometry) changes, we regenerate all VPLs independently at each frame. Since VPL positions are distributed randomly in the scene, the resulting VPL distribution is not temporally stable, even if the scene is static. Therefore, if all VPLs are regenerated at every frame, the resulting indirect illumination estimation may include substantial amount of flickering, regardless of which method is used for computing the illumination from the VPLs. One solution is incorporating methods that re-use VPLs to minimize flickering [Hedman et al. 2016]. Alternatively, applying a temporal anti-aliasing (TAA) filter only to the indirect illumination can substantially reduce the flickering, but can also introduce ghosting artifacts. The solution we prefer is applying TAA only to the indirect shadows, which reduces flickering without noticeable ghosting artifacts.

5 DISCUSSION AND CONCLUSION

We have introduced an extension of the lighting grid hierarchy method that makes it suitable for real-time rendering with many lights. We have also shown how our method can be used for efficiently computing high-quality global illumination at real-time frame rates. Our method can handle dynamic scenes, including dynamic lighting.

One important issue with all efficient solutions to the many-lights problem is that light leakage is unavoidable. This is certainly the case with the lighting grid hierarchy method and our parallel construction approach that uses $\mathbb{S}_1$ for generating the higher levels of the hierarchy leads to additional light leakage.

Moreover, unless a relatively large α value is used, lighting grid hierarchy can introduce some smoothing to the illumination estimation. Our shadow computation with a small number of shadow samples introduces an additional level of smoothing due to shadow filtering. When combined with interleaved sampling, the smoothing of our lighting estimation is further amplified.

Nonetheless, our method provides an effective solution to the many-lights problem for real-time rendering.

Note that, regardless of whether VPLs are completely or partially regenerated at every frame, the lighting grid hierarchy must be reconstructed when there is any change to the VPL data. This construction introduces computation cost beyond rendering for dynamic scenes. An interesting future direction would be exploring dynamic updates to previously constructed lighting grid hierarchy that could reduce the initialization cost for dynamic scenes.

ACKNOWLEDGMENTS

We thank Morgan McGuire [2017] for providing the scene files we used in our tests and the anonymous reviewers for their helpful feedback.

REFERENCES

Per Christensen. 2008. Point-based approximate color bleeding. *Pixar Technical Notes* 2, 5 (2008), 6.

Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. 2011. Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 1921–1930.

Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák. 2014. Scalable Realistic Rendering with Many-Light Methods. *Computer Graphics Forum* 33, 1 (2014), 88–104.

Carsten Dachsbacher and Marc Stamminger. 2006. Splatting indirect illumination. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. ACM, 93–100.

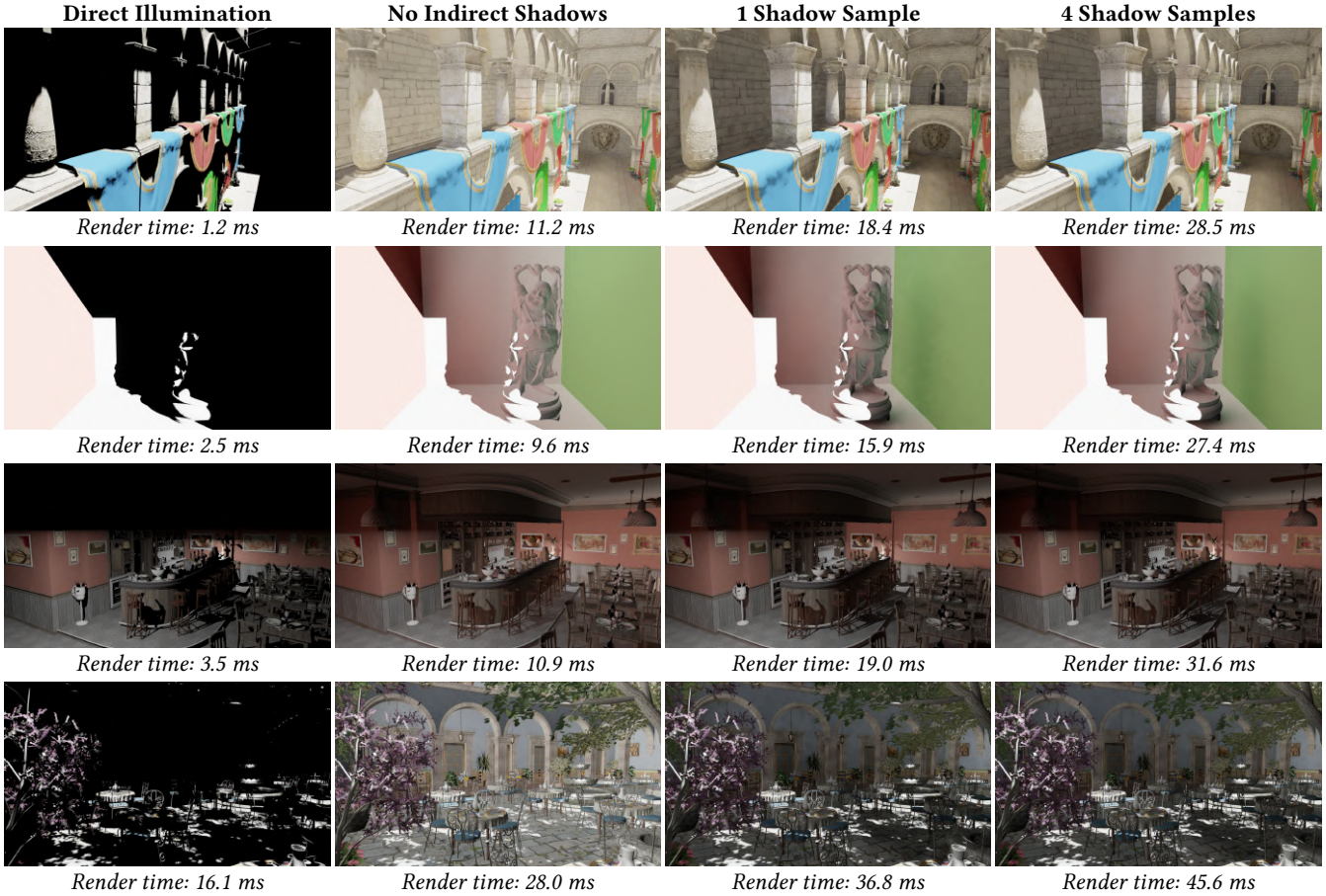


Figure 11: *Comparison of rendering time using different scenes and different settings. All scenes are rendered using lighting grid hierarchy generated from 100k VPLs, with no interleaved sampling and $\alpha = 1$. The reported render times do not include VPL generation and lighting grid hierarchy construction. Models downloaded from Morgan McGuire's Computer Graphics Archive [McGuire 2017].*

Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik Lensch. 2010. Edge-avoiding À-Trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics*. Eurographics Association, 67–75.

Tomáš Davidovič, Iliyan Georgiev, and Philipp Slusallek. 2012. Progressive lightcuts for GPU. In *ACM SIGGRAPH 2012 Talks*. ACM, 1.

T Davidovic, J Krivnek, M Hasan, P Slusallek, and K Bala. 2010. Combining global and local lights for high-rank illumination effects. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 29, 5 (2010).

Alejandro Conty Estevez and Christopher Kulla. 2018. Importance sampling of many lights with adaptive tree splitting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 2 (2018), 25.

Sebastian Fernandez, Kavita Bala, and Donald P Greenberg. 2002. Local Illumination Environments for Direct Lighting Acceleration. *Rendering Techniques* 2002 (2002), 13th.

Ming Gao*, Xinlei Wang*, Kui Wu*, Andre Pradhana, Eftychios Sifakis, Cem Yuksel, and Chenfanfu Jiang. 2018. GPU Optimization of Material Point Methods. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2018)*, Article 254 (2018), 12 pages. (*Joint First Authors).

Paul Green, Jan Kautz, and Frédo Durand. 2007. Efficient reflectance and visibility approximations for environment map rendering. In *Computer Graphics Forum*, Vol. 26. Wiley Online Library, 495–502.

Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. 2008. Progressive photon mapping. In *ACM Transactions on Graphics (TOG)*, Vol. 27. ACM, 130.

Takahiro Harada, Jay McKee, and Jason C Yang. 2013. Forward+: A step toward film-style shading in real time. *GPU Pro* 4 (2013), 115–134.

Miloš Hašan, Fabio Pellacini, and Kavita Bala. 2007. Matrix row-column sampling for the many-light problem. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 26.

Miloš Hašan, Edgar Velázquez-Armendariz, Fabio Pellacini, and Kavita Bala. 2008. Tensor Clustering for Rendering Many-light Animations. In *Proceedings of Eurographics Workshop on Rendering*. 1105–1114.

Peter Hedman, Tero Karras, and Jaakko Lehtinen. 2016. Sequential monte carlo instant radiosity. In *Proceedings of the 20th ACM SIGGRAPH symposium on interactive 3D graphics and games*. ACM, 121–128.

Eric Heitz, Stephen Hill, and Morgan McGuire. 2018. Combining analytic direct illumination and stochastic shadows. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2.

Yuchi Huo, Rui Wang, Shihao Jin, Xinguo Liu, and Hujun Bao. 2015. A matrix sampling-and-recovery approach for many-lights rendering. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 210.

Johannes Jendersie, David Kuri, and Thorsten Grosch. 2016. Precomputed illuminance composition for real-time global illumination. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 129–137.

Henrik Wann Jensen. 1996. Global illumination using photon maps. In *Rendering TechniquesâĂŹ 96*. Springer, 21–30.

James T Kajiya. 1986. The rendering equation. In *ACM Siggraph Computer Graphics*, Vol. 20. ACM, 143–150.

Anton Kaplanyan and Carsten Dachsbacher. 2010. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM, 99–107.

Alexander Keller. 1997. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 49–56.

Alexander Keller and Wolfgang Heidrich. 2001. Interleaved sampling. In *Rendering Techniques 2001*. Springer, 269–276.

Thomas Kollig and Alexander Keller. 2006. Illumination in the presence of weak singularities. In *Monte Carlo and Quasi-Monte Carlo Methods 2004*. Springer, 245–257.

Jaroslav Krivanek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. 2005. Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics* 11, 5 (2005), 550–561.

Gilles Laurent, Cyril Delalandre, Grégoire de La Rivière, and Tamy Boubekeur. 2016. Forward Light Cuts: A Scalable Approach to Real-Time Global Illumination. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 79–88.

Michael Mara, Morgan McGuire, Derek Nowrouzezahrai, and David P Luebke. 2016. Deep g-buffers for stable global illumination approximation.. In *High Performance Graphics*. 87–98.

Morgan McGuire. 2017. Computer Graphics Archive. (July 2017). https://casual-effects.com/data

Morgan McGuire and Michael Mara. 2014. Efficient GPU screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)* 3, 4 (2014), 73–85.

Morgan McGuire, Mike Mara, Derek Nowrouzezahrai, and David Luebke. 2017. Real-time global illumination using precomputed light field probes. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2.

Pierre Moreau and Petrik Clarberg. 2019. *Importance Sampling of Many Lights on the GPU*. Apress, Berkeley, CA, 255–283.

Pierre Moreau, Erik Sintorn, Viktor Kämpe, Ulf Assarsson, and Michael C Doggett. 2016. Photon splatting using a view-sample cluster hierarchy. In *High Performance Graphics*. 75–85.

Greg Nichols, Jeremy Shopf, and Chris Wyman. 2009. Hierarchical image-space radiosity for interactive global illumination. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 1141–1149.

Jan Novák, Thomas Engelhardt, and Carsten Dachsbacher. 2011. Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *Symposium on Interactive 3D Graphics and Games*. ACM, 119–124.

Ola Olsson and Ulf Assarsson. 2011. Tiled shading. *Journal of Graphics, GPU, and Game Tools* 15, 4 (2011), 235–251.

Ola Olsson, Markus Billeter, and Ulf Assarsson. 2012. Clustered deferred and forward shading. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*. Eurographics Association, 87–96.

Ola Olsson, Markus Billeter, Erik Sintorn, Viktor Kämpe, and Ulf Assarsson. 2015. More efficient virtual shadow maps for many lights. *IEEE Transactions on Visualization and Computer Graphics* 21, 6 (2015), 701–713.

Ola Olsson, Erik Sintorn, Viktor Kämpe, Markus Billeter, and Ulf Assarsson. 2014. Efficient virtual shadow maps for many lights. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 87–96.

Jiawei Ou and Fabio Pellacini. 2011. LightSlice: matrix slice sampling for the many-lights problem. *ACM Trans. Graph.* 30, 6 (2011), 179–1.

Eric Paquette, Pierre Poulin, and George Drettakis. 1998. A Light Hierarchy for Fast Rendering of Scenes with Many Lights. *Computer Graphics Forum* 17, 3 (1998), 63–74.

Ravi Ramamoorthi and Pat Hanrahan. 2001. An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 497–500.

Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. 2006. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 977–986.

Tobias Ritschel, Thomas Engelhardt, Thorsten Grosch, H-P Seidel, Jan Kautz, and Carsten Dachsbacher. 2009a. Micro-rendering for scalable, parallel final gathering. In *ACM Transactions on Graphics (TOG)*, Vol. 28. ACM, 132.

Tobias Ritschel, Thorsten Grosch, Min H Kim, H-P Seidel, Carsten Dachsbacher, and Jan Kautz. 2008. Imperfect shadow maps for efficient computation of indirect illumination. In *ACM Transactions on Graphics (TOG)*, Vol. 27. ACM, 129.

Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. 2009b. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM, 75–82.

Benjamin Segovia, Jean Claude Iehl, Richard Mitanchey, and Bernard Péroche. 2006a. Bidirectional Instant Radiosity.. In *Rendering Techniques*. 389–397.

Benjamin Segovia, Jean Claude Iehl, Richard Mitanchey, and Bernard Péroche. 2006b. Non-interleaved deferred shading of interleaved sample patterns. In *Graphics Hardware*. 53–60.

Peter Shirley, Changyaw Wang, and Kurt Zimmerman. 1996. Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics (TOG)* 15, 1 (1996), 1–36.

Ari Silvennoinen and Jaakko Lehtinen. 2017. Real-time global illumination by precomputed local reconstruction from sparse radiance probes. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 230.

Peter-Pike Sloan, Naga K Govindaraju, Derek Nowrouzezahrai, and John Snyder. 2007. Image-based proxy accumulation for real-time soft global illumination. In *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on*. IEEE, 97–105.

Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, Vol. 21. ACM, 527–536.

Jamorn Sriwasansak, Adrien Gruson, and Toshiya Hachisuka. 2018. Efficient Energy-Compensated VPLs using Photon Splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1 (2018), 16.

Yusuke Tokuyoshi and Takahiro Harada. 2016. Stochastic light culling. *Journal of Computer Graphics Techniques Vol* 5, 1 (2016).

K Vardis, G Papaioannou, and A Gkaravelis. 2014. Real-Time Radiance Caching using Chrominance Compression. *Journal of Computer Graphics Techniques* 3, 4 (2014).

Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. 2002. Interactive global illumination. (2002).

Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. 2006. Multidimensional Lightcuts. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '06)* 25, 3 (2006), 1081–1088.

Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P Greenberg. 2005. Lightcuts: a scalable approach to illumination. *ACM Transactions on graphics (TOG)* 24, 3 (2005), 1098–1107.

Bruce Walter, Pramook Khungurn, and Kavita Bala. 2012. Bidirectional Lightcuts. *ACM Transactions on Graphics* 31, 4, Article 59 (2012), 11 pages.

Rui Wang, Yuchi Huo, Yazhen Yuan, Kun Zhou, Wei Hua, and Hujun Bao. 2013. GPU-based Out-of-core Many-lights Rendering. *ACM Transactions on Graphics* 32, 6, Article 210 (2013), 10 pages.

Gregory J Ward. 1994. Adaptive shadow testing for ray tracing. In *Photorealistic Rendering in Computer Graphics*. Springer, 11–20.

Kui Wu, Nghia Truong, Cem Yuksel, and Rama Hoetzlein. 2018. Fast Fluid Simulations with Sparse Volumes on the GPU. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2018)* 37, 2 (2018), 157–167.

Can Yuksel and Cem Yuksel. 2017. Lighting grid hierarchy for self-illuminating explosions. *ACM TOG (Proc. SIGGRAPH)* 36, 4 (2017), 110.

A SHADOW SAMPLING PROBABILITIES

Given a stream of lights $x_1, x_2, x_3, \ldots, x_n$ with discrete probability densities $f_1, f_2, f_3, \ldots, f_n$, the objective is to pick light i as a shadow sample with probability $p_i = f_i / \sum_j^n f_j$. The algorithm in Sec. 3.3 keeps a running total for the cumulative probability density $\hat{f}_i = \sum_j^i f_j$ and for each light i decides whether to pick it as a shadow sample with probability $\tilde{p}_i = f_i / \hat{f}_i$. We can prove that this produces the desired probabilities p_i by induction. When $n = 1$, $\tilde{p}_i = p_i = 1$, so the base case is satisfied. For induction, suppose that a light k, where $k < n$, is selected as a shadow sample with probability $f_k / \hat{f}_{n-1}$. The algorithm replaces k with next light n with probability $\tilde{p}_n = f_n / \hat{f}_n$. If light n is selected, its probability $\tilde{p}_n = p_n$ provides the desired probability. Otherwise, light k is preserved as the shadow sample with probability $\hat{f}_{n-1} / \hat{f}_n$; thus, the total probability of picking light k becomes $(f_k / \hat{f}_{n-1})(\hat{f}_{n-1} / \hat{f}_n)$, which leads to the desired probability $p_k = f_k / \hat{f}_n$.