Stochastic Lightcuts for Sampling Many Lights

Cem Yuksel

Abstract—We introduce stochastic lightcuts by combining the lighting approximation of lightcuts with stochastic sampling for efficiently rendering scenes with a large number of light sources. Our stochastic lightcuts method entirely eliminates the sampling correlation of lightcuts and replaces it with noise. To minimize this noise, we present a robust hierarchical sampling strategy, combining the benefits of importance sampling, adaptive sampling, and stratified sampling. Our approach also provides temporally stable results and lifts any restrictions on the light types that can be approximated with lightcuts. We present examples of using stochastic lightcuts with path tracing and indirect illumination with virtual lights, achieving more than an order of magnitude faster render times than lightcuts by effectively approximating direct illumination using a small number of light samples, in addition to providing temporal stability. Our comparisons to other stochastic sampling techniques demonstrate that we provide superior sampling quality that matches and improves the excellent convergence rates of the lightcuts approach.

Index Terms—Many Lights, Light Sampling, Stochastic Sampling, Lightcuts.

1 Introduction

The problem of rendering with a large number of light sources (a.k.a. the many-lights problem) has received considerable attention in computer graphics. While methods that provide scalable lighting solutions are often considered in the context of global illumination computation with many virtual light sources, the many-lights problem has growing applicability in computer graphics, as we continue to render more complex scenes with more actual light sources in them. Since various real environments are actually illuminated by a large number of light sources, such as shopping centers, supermarkets, theaters, and office spaces, just to name a few, providing a scalable lighting solution for handling many lights is an important problem in computer graphics.

Lightcuts [1] is one of the first methods introduced for efficiently handling many lights, and it is still a preferred method for various applications due to its performance and flexibility. On the other hand, lightcuts, like most other scalable lighting solutions, is temporally unstable, which leads to flickering and hinders its use in practice. The underlying cause of this instability for lightcuts is due to the sampling correlation it inherits in the lighting approximation.

We introduce *stochastic lightcuts* that incorporates stochastic sampling into the illumination estimation framework of lightcuts. As a result, we remove the sampling correlation from the lighting estimation. To minimize the stochastic sampling noise, we also introduce a robust hierarchical sampling strategy that combines the benefits of importance sampling, adaptive sampling (provided by lightcuts), and stratified sampling (using a light tree). Our method only modifies the light sampling order of lightcuts, so it does not hinder the flexibility, the applicability, or the impressive convergence rate of the lightcuts solution, and it can be easily incorporated into an existing implementation of lightcuts. On the contrary, our stochastic sampling solution introduces extra flexibility and allows using complex light sources, such as area lights, which can be difficult to handle using lightcuts. Furthermore, stochastic lightcuts allows placing a user-defined small upper bound on the number of light evaluations per shading computation, which can significantly improve the rendering performance (Figure 1). We show that our approach can be easily coupled with path tracing in addition to global illumination computation with virtual light sources, which are not necessarily point lights. With these properties, the stochastic lightcuts approach offers the most efficient scalable lighting solution and without the stability problems of existing alternatives.

This paper extends our prior work [2] by providing a more detailed discussions and an explanation of potential bias issues in light sampling that can be easily avoided. All our results are regenerated using an unbiased implementation of stochastic lightcuts.

2 Background

Rendering with many light sources has been an important problem in computer graphics, though it is often investigated in the context of global illumination computation with virtual light sources [3]. Earlier method use ordering the lights based on their potential contributions for minimizing shadow queries [4], importance sampling based on light intensities [5], light clustering using octrees [6], or precomputed visibility culling for selecting lights [7]. The idea of using many virtual point light sources (VPLs) for computing global illumination [8] attracted more attention to the many-lights problem and its more recent versions reinforced the importance of the many-lights problem [9], [10], [11], [12], [13], [14], [15], [16]. Yet, the many-lights problem is more general than virtual light evaluations and a large number of user-specified light sources appear in various production scenes. Since brute-force computation of direct lighting from many lights can easily be the bottleneck of rendering, some production renderers already provide functionalities for

• *C. Yuksel was with the School of Computing, University of Utah, Salt Lake City, UT, 84112, USA. E-mail: cem@cemyuksel.com*

Author's preprint. Manuscript received 17 Dec. 2019; revised 17 May 2020; accepted 24 May 2020.
Digital Object Identifier no. 10.1109/TVCG.2020.3001271

(a) Lightcuts
Max. light samples: 10
Render time: 4 minutes

(b) Lightcuts
Max. light samples: 1000
Render time: 92 minutes

(c) Stochastic Lightcuts (Ours)
Max. light samples: 10
Render time: 7 minutes

Fig. 1: *Comparison of lightcuts and our stochastic lightcuts for direct illumination estimation from* 1644 *light sources, rendered using path tracing with* 5 *bounces and* 64 *samples per pixel.* **(a)** *Lightcuts with up to* 10 *light samples produces a substantial amount of error and correlation artifacts.* **(b)** *Using up to* 1000 *light samples reduces the error, but still leads to visible flickering and takes more than* 20× *render time.* **(c)** *Our stochastic lightcuts method can produce a fast, temporally-stable, and low-noise lighting estimation with up to* 10 *samples.*

accelerating direct illumination computation from a large number of light sources in a scene [17], [18].

Lightcuts. A general and scalable solution to the many-lights problem is provided by lightcuts [1]. It starts with building a binary tree for representing the illumination from all lights in the scene. The scene lights are placed at the leaf nodes of this tree. Each internal node is used for approximating the illumination from all light sources within its subtree. This is typically implemented as picking one of the lights within the subtree as a representative light for the internal node, along with a scaling factor for its intensity to account for the illumination from the other lights of the subtree. The light tree is constructed once, prior to rendering, using a bottom-up approach by iteratively clustering similar node pairs (using a similarity metric) to form the internal nodes of the tree. When two nodes are paired, the representative light for their parent node is chosen randomly as one of the representative lights of the child nodes, using the cumulative intensities of the nodes as weights. During rendering, the light tree is evaluated at each shading point, starting from the root node and its representative light. After evaluating the representative light of a node, a conservative error bound is assigned to the node using its bounding box, indicating the maximum possible intensity contribution due to the illumination that can come from the subtree, assuming full visibility (i.e. no shadows). If this error bound is below a user-specified percentage (typically 2%) of the approximated total shading value, the lighting evaluation of the node is accepted. Otherwise, its child nodes are evaluated. Due to the construction of the light tree, one of the child nodes shares the same representative light as the parent node. Therefore, light from that child node can be quickly evaluated without the need for recomputing the shadows of the shared representative light. The extensions of the lightcuts method include a multi-dimensional version for handling volume scattering, depth of field, and motion blur [19]; a progressive GPU-friendly variant [20]; bidirectional lightcuts for improving the weighting scheme to support a wider range of materials [21]; and an out-of-core GPU implementation for rendering large scenes [22]. Recently, lightcuts was used for learning the light selection probability distributions for Monte Carlo sampling of direct illumination through Bayesian regression at render time [23].

Matrix Row-Column Sampling. A prominent alternative approach to lightcuts is the matrix row-column sampling method [24] that provides an approximate evaluation of the entire lighting matrix for the scene. The rows and columns of this matrix correspond to the shading points and light sources, respectively. Instead of computing this entire matrix, as a brute-force lighting method would, a reduced matrix is computed, and the rest of the original matrix is approximated. This approach allows easily handling a wider range of light types than lightcuts and it can be accelerated using shadow maps. However, it is less adaptive than lightcuts, and it requires determining all shading points before lighting computation, which makes is less flexible. Extensions of this approach include an additional temporal dimension for reducing flickering [25], separation of local and global illumination components for handling glossy materials [26], introduction of cuts for adaptively evaluating a fraction of the lights [27], and a reduced matrix formulation that approximates the lighting matrix [28].

Lighting Grid Hierarchy. Recently, the lighting grid hierarchy (LGH) method [29] was introduced for rendering explosions with self-illumination by generating a large number of VPLs. LGH generates multiple representations of the entire illumination in the scene, each with a different resolution. During evaluation, lights at different distances are approximated by combining the approximations of different

resolutions. Its main advantage comes from the fact that it allows precomputing shadow maps for the entire hierarchy, which can lead to orders of magnitude faster computation for explosion rendering, as compared to lightcuts. However, LGH typically uses significantly more light samples than lightcuts, its precomputation benefits diminish when the shadow computation is less expensive than volume tracing, and it is limited in terms of the light types it can represent. LGH was also extended to provide a fast global illumination solution for real-time rendering [30].

Adaptive Tree Splitting. A recent importance sampling approach for handling the direct illumination of many lights is adaptive tree splitting [31], which is particularly effective for handling spot lights with limited field of view. This approach builds a bounding volume hierarchy for lights, considering their spatial distributions along with their directions. Similar to our solution, a hierarchical important sampling scheme is used for picking light samples. Adaptive tree splitting does not suffer from sampling correlation like lightcuts, but does not provide the adaptivity of lightcuts and its convergence rate is hindered by its importance formulation. This method was recently extended to achieve real-time rendering with a two-level hierarchy for fast updates in dynamic scenes [32].

A common limitation of all these scalable many-lights solutions is that they require a relatively large number of samples for producing stable/low-noise results. Therefore, traditional importance sampling [5] is still commonplace in practice. Our stochastic lightcuts method, in comparison, can produce temporally stable and relatively low noise results with fewer samples. Therefore, it is highly suitable for rendering algorithms that already rely on multi-sampling, such as path tracing, and significantly improve their performance by providing a low-cost estimation of lighting.

3 Stochastic Lightcuts

We introduce stochastic sampling into the lighting evaluation of lightcuts to eliminate its sampling correlation (Section 3.1), replacing the temporal instabilities of lightcuts with noise. For reducing this noise, we introduce a robust hierarchical importance sampling method (Section 3.2). Stochastic lightcuts can use the same light tree as lightcuts with only minor modifications to the information stored in each node and it can be easily implemented on top of an existing lightcuts implementation (Section 3.3). Therefore, we only describe the differences in the lighting approximation introduced by stochastic light evaluations, as compared to lightcuts.

3.1 Stochastic Sampling with Lightcuts

The sampling correlation of the lightcuts approach is related to the fact that the same light tree is used for rendering the entire image. This is unavoidable in general, because the light tree construction and storage can be expensive. The light tree, containing a representative light per node, forms a spatially varying ordering of light sources. The lighting estimation always begins with the representative light of the root node. Therefore, this light source is always included in the lighting estimation of the entire scene. Similarly, the representative lights at the higher levels of the hierarchy are more likely to be used for the lighting estimation. This predetermined order prior to rendering introduces sampling correlation. By using a conservative error bound, the error within a subtree can be limited to a user-defined percentage of the evaluated pixel color (usually set to 2%). However, this does not bound the total error of the entire lighting estimation. Therefore, errors due to sampling correlation can be substantial and often lead to temporal instability.

This important limitation of lightcuts was also recognized in prior work. The solution that is proposed as a part of the multidimensional lightcuts method [19] was simply storing a list of representative lights per node (such as 32) and randomly selecting one at render time, using the intensities of these pre-selected lights for importance sampling. This approach reduces the correlation, but does not eliminate it. It also inflates the light tree storage. More importantly, due to the performance advantages of sharing representative lights between a node an its parent, this preselection is not performed independently for each node and the sampling correlation is not completely eliminated.

We eliminate the correlation by simply ignoring the representative lights during lighting estimation. Instead, we randomly pick a light source within a given subtree. Any importance sampling scheme can be used here for determining the probability p_i of picking light source i within a subtree. Let p_s be the cumulative probability of all lights within the subtree s. The estimated illumination of the light subtree can be computed by simply scaling the illumination of the light with p_s/p_i. When the error of a node is above the user-defined threshold and we need to evaluate the child nodes, we only need to perform the lighting computation for one of the child nodes, since the other one must contain the light source that was randomly selected for evaluating the parent node. Therefore, just like the original lightcuts method, the light evaluation (including shadow computation) for the parent node is not wasted as we move deeper into the light tree.

This simple modification provides three important benefits:

1. It replaces the predefined order of lights with a randomized order and thereby completely eliminates the sampling correlation.
2. It allows using any type of light source. Since we do not rely on representative lights and we use the actual lights for computation, we impose no restrictions on the light type. The resulting illumination contribution of each light is merely scaled by the corresponding probabilities, as explained above.
3. We can limit the number of lights evaluated during lighting approximation without risking excessive correlation artifacts.

When using representative lights, we must rely on the error bound and traverse as deep into the light tree as necessary, and terminating "cut" selection prematurely (when a maximum number of light samples are computed) can have catastrophic results. This is because the predefined order of representative lights at the higher levels of the light tree can be pathological for estimating the lighting at some points

in the scene, and lead to excessive amounts of correlation artifacts. For example, if only a single light evaluation is permitted, the same representative light would be used for the entire scene. Using randomly selected lights, however, we would simply introduce noise by limiting the number of evaluated lights. This can be a significant advantage for some rendering algorithms, such as path tracing, that would ultimately reduce the noise using multi-sampling. Therefore, using path tracing, a faster and noisy lighting estimation would typically be preferable over a slower and more accurate one, and the performance improvements of using fewer light evaluations per lighting estimation can be substantial with minimal or no loss in the final image quality.

3.2 Hierarchical Importance Sampling

The stochastic sampling scheme mentioned above solves the sampling correlation problem of lightcuts. Regardless of how the probabilities are defined for randomly selecting the sampled light sources (assuming non-zero probabilities for lights with non-zero illumination), the solution is consistent and converges to the expected result with multi-sampling. However, if the probabilities are not good representations of the illumination contributions of each light source, the convergence rate can be slow, leading to excessive noise in lighting estimation.

Unfortunately, the optimal probabilities that would maximize the convergence are not constant for all points in the scene and they depend on the spatial relationship between the lights and the point where lighting is evaluated. Precomputing the probabilities for all points in the scene would be prohibitively expensive, so they must be computed at render time. To address this problem, we describe a robust hierarchical importance sampling scheme that only computes a fraction of the light probabilities using the light tree during the lighting evaluation at a given point.

Instead of directly selecting a random light within a subtree, we traverse the subtree step-by-step until we reach a leaf node. Starting with the root node of the subtree, at each step we randomly pick one of the child nodes using importance sampling. The probabilities of picking either child node are assigned based on the expected cumulative illumination contribution of each child node.

Note that how these probabilities are computed has utmost importance. Using a poor strategy can result in a worse convergence behavior than the simplest importance sampling scheme that randomly selects a light purely based on its intensity [5].

3.2.1 Computing Child Node Probabilities

Let p_1 and p_2 represent the probabilities of selecting either one of the child nodes for estimating the lighting at a scene point $\mathbf{x}$. Following the error estimation mechanism of lightcuts, a naive choice for determining the importance weights w_1 and w_2 for picking the child nodes would be using the maximum possible illumination that can come from each child node. The corresponding probabilities are defined as $p_1 = w_1/(w_1 + w_2)$ and $p_2 = w_2/(w_1 + w_2)$. These weights can be calculated for each node j using a conservative estimate for the maximum geometry term $G_j(\mathbf{x})$, computed using the node's bounding box and its light direction bounds [1], [31]; another conservative estimate for bounding the BRDF (Bidirectional Reflectance Distribution Function) of the material $M_j(\mathbf{x}, \omega)$ [1], where ω is the view direction; the total intensity of the lights within the node $\mathbf{I}_j$; and the minimum distance to the node's bounding box $d_j^{\min}(\mathbf{x})$. Thus, we can write

$$w_j = \frac{F_j(\mathbf{x}, \omega)\,\|\mathbf{I}_j\|}{\left(d_j^{\min}(\mathbf{x})\right)^2}\,, \tag{1}$$

where $F_j(\mathbf{x}, \omega) = G_j(\mathbf{x})M_j(\mathbf{x}, \omega)$ is the *reflectance bound*, combining the geometry and material terms. Note that our notation slightly differs from the notation of the original lightcuts method [1], as the inverse-square attenuation factor $1/(d_j^{\min}(\mathbf{x}))^2$ is not hidden in $G_j(\mathbf{x})$. Also note that in this notation we do not consider directional light sources, which would not have the inverse-square attenuation factor, so they need to be treated differently.

While the formulation in Equation 1 might appear reasonable at first glance, it would lead to unacceptable weights for importance sampling. This is because when $\mathbf{x}$ is within the bounding box of a child node, $d_j^{\min}(\mathbf{x})$ becomes zero and the weight becomes infinity. When $\mathbf{x}$ is outside of the bounding box of the other child node, the probability of selecting this other child node becomes zero, regardless of how much illumination it represents. Therefore, this sampling scheme would not necessarily converge to the correct result.

A typical solution to this problem would be replacing the closest distance $d_j^{\min}(\mathbf{x})$ with the distance to a point within the bounding box of a node. Indeed, this is the approach used by adaptive tree splitting [31] by computing the distance to the centroid of the bounding box. However, regardless of how this point is chosen, the singularity of inverse-square distance to a point, where the weight would go to infinity, would not be entirely avoided. Also, picking a particular point within the bounding box may lead to a weight that can be a poor estimator of the incoming illumination from within the entire bounding box, resulting in high noise and low convergence. In effect, this solution reduces the singularity in Equation 1 from an entire bounding box to a point, but does not always improve the convergence of importance sampling, as compared to merely considering the total light intensities [5].

On the other hand, we cannot completely ignore the inverse-square attenuation term, as it can be a significant factor in bounding the illumination contribution of distant lights. Therefore, simply considering the total light intensity $\mathbf{I}_j$ or its modulation with the reflectance bound $F_j(\mathbf{x}, \omega)$ are not ideal solutions either.

Our solution is simply combining the weights computed using the minimum distance term $d_j^{\min}(\mathbf{x})$ and weights computed without a distance term. At the higher levels of the hierarchy, where the node bounding boxes are large and the minimum distance term can be a poor indicator of the expected illumination coming from a node, we simply ignore the distance term. For lower levels of the hierarchy, where the node bounding boxes are sufficiently far from $\mathbf{x}$, we include the distance term. More precisely, we decide whether to include the distance term by comparing the minimum distance $d_j^{\min}(\mathbf{x})$ to the size of the bounding box

ℓ_j (i.e. the length of its diagonal). The distance term is included only if $d_j^{\min}(\mathbf{x}) > \alpha\ell_j$ for both child nodes, where α is a user-defined scaling coefficient. We use $\alpha = 1$ for all examples in this paper. Let j and k indicate two child nodes of an internal node. We compute the weights using

$$w_j = F_j(\mathbf{x},\omega)\,\|\mathbf{I}_j\|\;\Lambda_{j,k}(\mathbf{x})\;, \tag{2}$$

where the attenuation term $\Lambda_{j,k}(\mathbf{x})$ is

$$\Lambda_{j,k}(\mathbf{x}) = \begin{cases} \frac{1}{\left(d_j^{\min}(\mathbf{x})\right)^2} & \text{if } d_j^{\min}(\mathbf{x}) > \alpha\ell_j \text{ and } d_k^{\min}(\mathbf{x}) > \alpha\ell_k \\ 1 & \text{otherwise.} \end{cases} \tag{3}$$

This simple solution completely avoids the singularity. The distance to the node bounding box is only included when it is expected to improve the importance estimator, that is when the node is sufficiently far, which is determined by the size of a node's bounding box, so it is independent of the numerical scale of the scene.

For incorporating directional lights with no attenuation factor, we slightly modify the weight computation, such that

$$w_j = F_j(\mathbf{x},\omega)\left(\left(\|\mathbf{I}_j\| - \|\mathbf{I}_j^D\|\right)\;\Lambda_{j,k}(\mathbf{x}) + \|\mathbf{I}_j^D\|\right)\;, \tag{4}$$

where $\|\mathbf{I}_j^D\|$ is the total intensity of directional lights within the node. Thus, the attenuation term does not impact the importance of directional lights.

3.2.2 Dead Branches

We use the term *dead branch* to refer to the subtree under a node that can contribute no illumination purely due to the reflectance bounds of all lights within the subtree (not considering visibility/shadows). We call all nodes within a dead branch *dead nodes*.

When the reflectance bound $F_j(\mathbf{x},\omega)$ for the root node of a dead branch is zero, the node is assigned zero probability and the dead branch is automatically avoided during our hierarchical importance sampling traversal. However, this behavior is not guaranteed. This is because a typical computation of the reflectance bound would be based on a conservative estimate, using the bounding box of the light node [1], [31]. This avoids the potentially-enormous cost of considering each individual light within a subtree. On the other hand, the conservative bound $F_j(\mathbf{x},\omega)$ of an internal node can be non-zero even when it is zero for *all* individual lights within the subtree. Thus, a dead branch can be assigned a non-zero probability.

Including dead branches in lighting estimation does not break the lightcuts algorithm, especially when it is permitted to converge using as many light samples as necessary to satisfy the error threshold, but it introduces unnecessary computation cost. More importantly, when the maximum light sample count is limited, selecting a dead branch can effectively *waste* a light sample and negatively impact the quality of the lighting estimation. This is particularly important for fast lighting estimation with a small number of samples using algorithms that rely on multi-sampling, such as path tracing. Therefore, dead branches should be avoided when detected.

A dead branch is automatically detected and avoided when $F_j(\mathbf{x},\omega) = 0$ at its root node. Otherwise, we detect a dead branch further down its subtree, where the importance weights w_1 and w_2 of the two child nodes of a node are both zero. Note that testing whether $w_1 + w_2$ is zero is sufficient for detecting a dead branch, because all possible subtrees within a dead branch include a node with $w_1 + w_2 = 0$, since all leaf nodes of a dead branch contain lights i with $F_i(\mathbf{x},\omega) = 0$. However, the node that satisfies this condition does not have to be the root node of the dead branch.

Once a dead branch is detected, we can simply terminate the hierarchical traversal by returning any light within the subtree. The selection of a light within a dead branch is inconsequential, since none of them should have any effective illumination. Therefore, we can simply return the representative light of an internal node and skip the evaluation of the light source to save computation.

Yet, simply terminating the traversal and returning an arbitrary light within the subtree of a dead branch would still waste a light sample. Therefore, the solution we presented in our prior work [2] is backtracking the hierarchical traversal and making sure that we do not select a light within a dead branch. This can be done by moving back up the light tree, skipping the detected dead node by updating its probability to zero, and continuing the traversal using a different path down the light tree. Note that in some cases the entire light tree can be a dead branch (i.e. a *dead tree*) if the point $\mathbf{x}$, where the illumination is computed, cannot be illuminated by any light in the scene (based on the reflectance bound alone, not considering visibility).

While this solution is effective in avoiding dead branches, it does not form an unbiased estimator. For example, consider an internal node of the light tree with two child nodes, one of which is a dead branch. Let p be the probability of picking the dead branch when evaluating the parent node during hierarchical importance sampling, and $\mathbf{I}$ be the expected illumination under the other child node. Since one of the child nodes is a dead branch, the expected illumination of the parent node is also $\mathbf{I}$. However, the backtracking approach detects the dead branch with only p probability. If it is detected, the expected illumination of the parent node can be computed from the other child node as $\mathbf{I}$; otherwise, it is taken as $\mathbf{I}/(1-p)$. Indeed, backtracking, when a dead branch is detected, provides a more accurate lighting estimation, but it is detected with only p probability. Thus, the expected illumination of the parent node is overestimated as $p\mathbf{I} + (1-p)\mathbf{I}/(1-p)$. Note that simply returning an arbitrary light from the dead branch instead would produce no light with p probability and $\mathbf{I}/(1-p)$ with $1-p$ probability, resulting in a correct estimation of $\mathbf{I}$. Therefore, even though dead branches do waste light samples, we cannot simply avoid them by stochastically altering the probabilities without introducing bias to the lighting estimation.

For unbiased lighting estimation, when a dead branch is detected, we simply return a *null light*. There is no need to evaluate this light (or compute shadows for it), since we know that it cannot contribute any illumination. Yet, we must still count it as a light sample to avoid introducing bias.

3.3 Implementation

The lightcuts algorithm determines a "cut" through the light tree by evaluating it starting from its root node. Each

(a) Lightcuts
Max. light samples: 10
Render time: 27 seconds

(b) Lightcuts
Max. light samples: 1000
Render time: 14 minutes

(c) Stochastic Lightcuts (Ours)
Max. light samples: 10
Render time: 58 seconds

Fig. 2: *Comparison of our stochastic lightcuts method to lightcuts for direct illumination computation from* 14 *light fixtures (on the ceiling), each containing* 100 *light sources, rendered using* 64 *samples per pixel.* **(a)** *Lightcuts with up to* 10 *light samples produces prominent stripes on the table, on the floor, and in the background, due to the shadows of the light fixture details.* **(b)** *When up to* 1000 *light samples are permitted with lightcuts, the stripes become less noticeable in a still frame, but such artifacts still persist and lead to substantial flickering in animations, and it also takes almost* $30\times$ *longer time to render.* **(c)** *Our stochastic lightcuts method achieves a low-noise solution with no visible artifacts using only* 10 *light samples per lighting estimation.*

Algorithm 1: Pseudocode of selecting a light within a subtree using hierarchical importance sampling.

```
1  function SelectLight(j)
       // j is the root node of a light subtree
2      p ← 1 // initialize the probability of picking the light
3      r ← a random value in [0, 1)
4      while j is not leaf do
5          w1 ← the weight of the first child node of j
6          w2 ← the weight of the second child node of j
7          if w1 + w2 > 0 then
8              p1 ← w1/(w1 + w2)
9              if r < p1 then
10                 p ← p · p1 // update the probability
11                 r ← r/p1 // rescale the random value
12                 j ← the first child node of j
13             else
14                 p ← p · (1 − p1) // update the probability
15                 r ← (r − p1)/(1 − p1) // rescale
16                 j ← the second child node of j
17             end
18         else
               // j is a dead node.
19             return (null light, p) // no light sample found
20         end
21     end
22     i ← the light sample of the leaf node j
23     return (i, p)
```

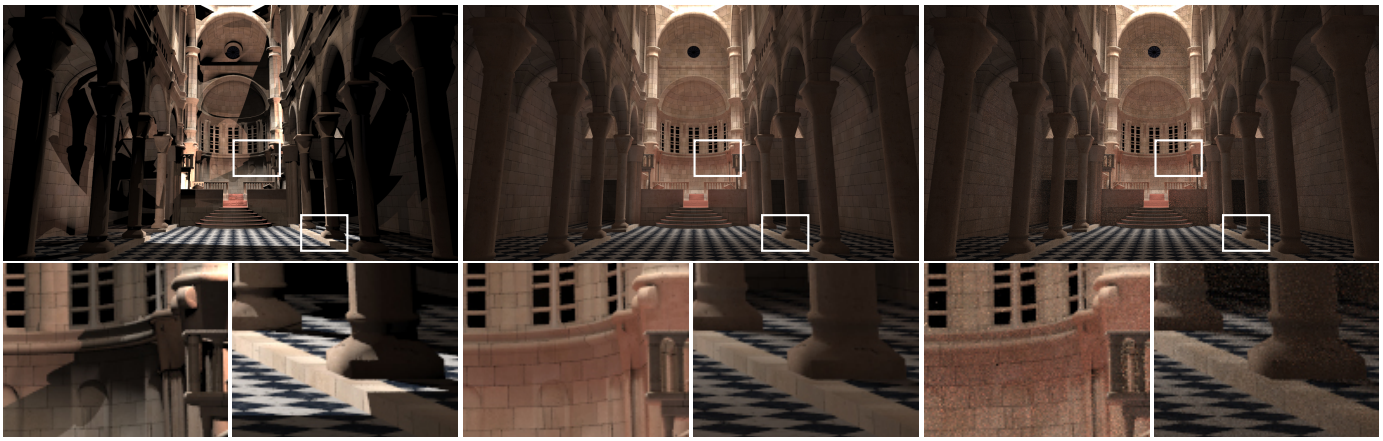
evaluated light tree node is added to a light sample list (typically implemented using a heap) to be returned. The nodes in the light sample list with an error bound greater than the error threshold are replaced by their child nodes (in the order of the largest error bound). When replacing a node with its child nodes, only one of the child nodes requires a full evaluation, while the other one (containing the light sample of the parent node) uses the parent node's data and only updates its error bound and cluster intensity.

The implementation of the stochastic lightcuts algorithm involves only two relatively minor changes. First, instead of simply using the representative light of a light tree node, we use hierarchical importance sampling to select a light within the subtree of the node. If the node is determined to be a dead node, we simply skip adding the node to the light sample list. If the root node is a dead node, we return an empty light sample list. Second, each light sample returned must also be accompanied by the probability of selecting the light sample within the subtree of the selected node. Therefore, the light selection probabilities must be updated when replacing a node in the light sample list with its child nodes.

Algorithm 1 shows the pseudocode of the light selection algorithm using our hierarchical importance sampling approach. This algorithm returns the light sample and the probability of selecting it within the given subtree. Notice that Algorithm 1 must traverse the given light subtree down to a leaf node or a detected dead node. Therefore, it includes additional computation cost, as opposed to simply returning a pre-selected representative light used by the original lightcuts algorithm.

4 Results

We evaluate our stochastic lightcuts method by first comparing it to lightcuts and then comparing its sampling quality to other related stochastic sampling techniques. The results are generated with a custom renderer using Intel's Embree ray tracing kernels [33]. The error threshold for lightcuts and stochastic lightcuts is set to 2%. All timing results



(a) Lightcuts
Max. light samples: 10
Render time: 1 minute

(b) Lightcuts
Max. light samples: 1000
Render time: 2 hours

(c) Stochastic Lightcuts (Ours)
Max. light samples: 10
Render time: 1.8 minutes

Fig. 3: *Comparison of our stochastic lightcuts method to lightcuts in a scene illuminated by one million virtual spherical lights, generated with up to* 8 *bounces from* 120 *light sources near the top of the dome, rendered using* 64 *samples per pixel.* **(a)** *Lightcuts with up to* 10 *light samples produces severe visual artifact in the form of sharp shadow lines.* **(b)** *Using up to* 1000 *light samples with lightcuts makes these artifacts less prominent, but the result still flickers in animation and the render time becomes about* 80× *longer.* **(c)** *Our stochastic lightcuts method produces a temporally-stable solution with only* 10 *light samples.*

are measured on a computer with dual Intel Xeon CPUs running at 2.4 GHz (16 total cores). Some render times we report in this paper are different than our prior work [2], since we are using a more optimized implementation of lightcuts and a newer version of Embree.

4.1 Comparison to Lightcuts

We compare the results of lightcuts and stochastic lightcuts using three different scenes. Figure 1 shows path tracing in a scene with 1644 light sources. Note that modern production scenes can include significantly more lights. Figure 2 shows direct illumination from light fixtures that not only illuminate the scene but also cast shadows. Finally, Figure 3 shows a scene illuminated by one million virtual spherical lights (VSLs) [34] generated from 120 original lights. All lights in each scene are placed in a single light tree. We use simple materials and an optimized ray tracing implementation on the CPU, so a significant portion of the render times are spent in lighting estimation.

In these three figures, we show two different settings for lightcuts. The first one limits the maximum number of light samples to 10 (Figures 1a, 2a, and 3a). This causes the lightcuts algorithm to produce a fast estimate of lighting, but this relatively small limit on light samples does not allow lightcuts to converge using its error threshold parameter. Therefore, the resulting images contain large blocks of dark areas (Figure 1a), randomized shadow patterns (Figure 2a), and inconsistent illumination throughout the scenes with sharp illumination variations (Figure 3a). This is because lightcuts with a small limit on the light sample count cannot traverse deep enough into the light tree and the resulting lighting estimation is produced using the few representative lights near the root of the light tree. The dark regions with sharp shadow boundaries appear where these representative lights are shadowed. Since lightcuts generates the light tree stochastically (to avoid bias in light tree construction), a different tree is built for each frame, resulting in extreme amounts of temporal flickering. Forcing lightcuts to use the same light tree for each frame on an animation would eliminate the flickering when the illumination in the scene is static, but visual artifacts of the poor lighting estimation would remain.

The second setting we show for lightcuts allows the algorithm to converge using its error threshold (Figures 1b, 2b, and 3b). In this case, we limit the maximum light samples to 1000, which is close to brute-force lighting evaluation in Figure 1, as it contains only 1644 lights. However, lightcuts does not come close to this maximum limit and returns an estimate when all light subtrees under the chosen "cut" satisfy the error bound. In fact, using an error threshold of 2%, lightcuts typically returns an estimation using only a few hundred light sources, regardless of the total number of lights in the scene. This allows lightcuts to provide an efficient lighting estimation with fast convergence rates. On the other hand, always using hundreds of lights in lighting estimation limits the performance gain, as compared to brute-force rendering (as in Figures 1 and 2). More importantly, the resulting solution still contains sampling correlation that leads to visible flickering in animations. This flickering is substantial even in the scene in Figure 3, where the majority of the lighting is indirect illumination with gradual variations over the image. This temporal instability can be reduced by choosing a smaller error threshold, but this would also inflate the render time. Indeed, letting lightcuts to converge using its error threshold (Figures 1b, 2b, and 3b) leads to render times that are already 20× to 120× slower than using a relatively small limit on the maximum number of light samples in our test (Figures 1a, 2a, and 3a).

Our stochastic lightcuts solution, in comparison, provides a fast lighting estimation with low noise, using a maximum of 10 light samples per lighting estimation (Figures 1c, 2c, and 3c). Since there is no sampling correlation with stochastic lightcuts, the results do not contain visual artifacts

and it is temporally stable, using a different light tree at each frame. The stochastic lightcuts algorithm takes considerably more time than lightcuts with the same number of samples, due to the cost of hierarchical traversal of the light tree down to leaf nodes. Also, sampling correlation, while detrimental to quality, improves performance with better cache utilization and leads to shorter render times using lightcuts with the same number of light samples. Nonetheless, using only 10 light samples, we can provide a solution that is similar to the lightcuts solution with up to 1000 light samples, and without its temporal instability. The render times are also about an order of magnitude shorter. It should be noted that multi-sampling used in these scenes help reduce the resulting noise with stochastic lightcuts, but does not help the lightcuts solutions at all.

4.2 Stochastic Sampling Comparisons

We evaluate the sampling quality of our stochastic lightcuts method by comparing it to traditional importance sampling [5], adaptive tree splitting [31], and multidimensional lightcuts that stores up to 32 representative lights per light tree node [19]. For our tests, we use a deceptively simple scene shown in Figure 4. This scene includes 10,000 virtual lights generated from a single point light source with no bounce. A horizontally-placed divider occludes the original light source and forms darker regions in the bottom half of the image, where only a small portion of the virtual lights provide non-zero illumination. The walls in the top half of the image are illuminated by almost all lights in the scene. What makes this scene interesting for testing sampling is this duality in the overall illumination, the substantial variation in intensities between the virtual lights and the original light source, and the fact that the illumination from the lights are not limited to a local region. Our results with different stochastic sampling techniques are shown in Figure 5 and the root mean square errors are provided in Table 1 as averages of 10+ renders (the standard deviations in RMSE of different renders are shown in parentheses). We use only a single sample per pixel in the test in Figure 5 to clearly show the quality of light sampling. Using multiple samples (i.e. multiple lighting estimations) per pixel reduces the noise for all methods but multidimensional lightcuts, which receives a relatively minor improvement from multi-sampling due to sampling correlation.

In this scene, traditional importance sampling (Figure 5a) provides a good sampling quality, particularly for the top half of the image. The bottom half of the image, however, receives more noise because of the high probability of sampling the original light, which is shadowed, and the approximately equal probability of sampling any virtual light.

Adaptive tree splitting (Figure 5b) improves the sampling quality for the darker bottom region of the image, as compared to traditional importance sampling. On the other hand, the singularity in its importance formulation leads to poor importance estimates for the top half of the image. As a result, the strong original light source is not sampled as often and the importance estimates considerably deviate from the illumination contributions of the selected lights, leading to even more noise than purely considering light intensities, like traditional importance sampling.

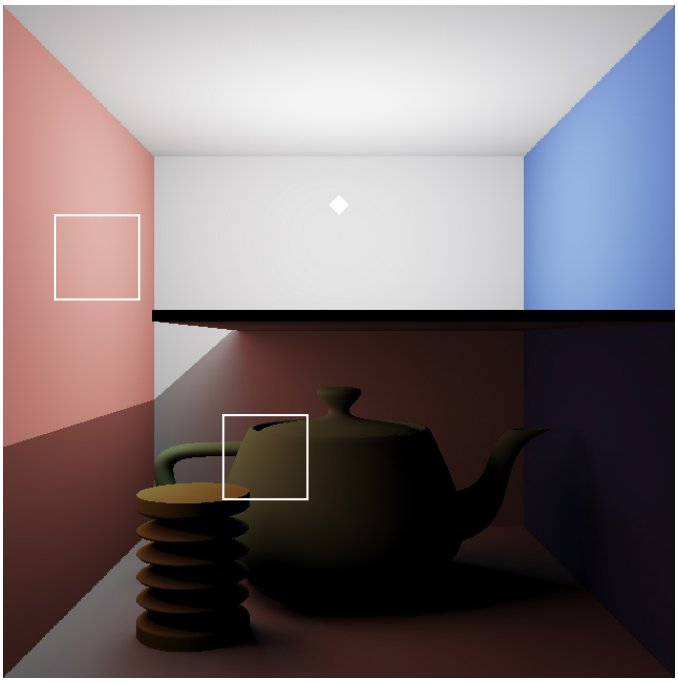
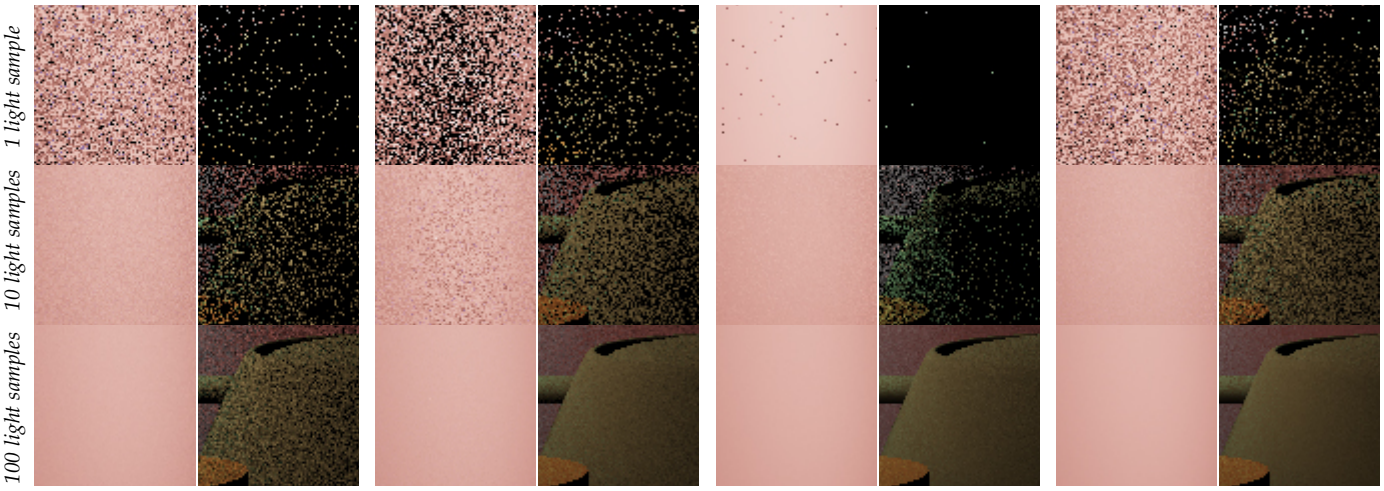


Fig. 4: *The test scene used for comparing different stochastic sampling algorithms in Figure 5. The scene contains* $10,000$ *virtual lights generated from a single point light with no bounces.*

The multidimensional lightcuts method (Figure 5c) augments lightcuts by storing 32 representative lights per light tree node. One of these representative lights is randomly selected (using proportional probabilities to light intensities) during the lighting estimation of the node. Thus, it introduces some flavor of stochastic sampling in lightcuts, but the benefits are highly limited. It preserves the excellent convergence rates of lightcuts, such that the representative lights, chosen during the light tree construction, are likely to produce low error. On the other hand, sampling correlation is not entirely avoided, so the results display a similar temporal instability to lightcuts. In this test scene, it provides excellent results for the top half, but the bottom half is either completely dark (with 1 light sample) or extremely noisy and unstable (with 10 light samples). Using 100 light samples virtually eliminates the noise, but the temporal instability persists. While the results provide relatively low root mean square error (see Table 1), sampling correlation leads to excessive degradation in visual quality, especially with relatively few light samples.

Our stochastic lightcuts method (Figure 5d) provides the best sampling quality. Using a single light sample, it slightly improves the sampling quality of traditional importance sampling in the top half of the image, and it achieves a more significant improvement with more light samples. This improvement is due to the importance estimation of stochastic lightcuts, which is locally-adaptive and provides improved accuracy in importance estimation, considering the light distances from the point where lighting is evaluated. In the bottom half, it leads to a superior sampling quality than all other methods. Like multidimensional lightcuts, it preserves the excellent convergence rate of lightcuts when using a relatively large number of light samples, but it does not suffer from temporal instabilities.



(a) Importance Sampling **(b)** Adaptive Tree Splitting **(c)** Multidimensional Lightcuts **(d) Stochastic Lightcuts (Ours)**

Fig. 5: *Comparison of the sampling quality of our stochastic lightcuts method to other related sampling techniques using the test scene in Figure 4, rendered with a single sample per pixel using* 1, 10, *and* 100 *light samples for lighting estimation.* **(a)** *Traditional importance sampling produces a relatively low-noise image in well-lit areas (left column), but the noise increases in darker areas (right column).* **(b)** *Adaptive tree splitting provides improved sampling quality for darker regions (right column), but introduces more noise in well-lit areas (left column), as compared to traditional importance sampling.* **(c)** *Multidimensional lightcuts produces low noise in well-lit areas (left column), but sampling quality degrades substantially in darker regions (right column) and the results are temporally unstable.* **(d)** *Our stochastic lightcuts method produces the best sampling quality in all cases.*

TABLE 1: *Root mean square error (RMSE) for Figure 5. The standard deviations in RMSE of 10+ renders are in parentheses.*

	1 light sample	10 light samples	100 light samples
Importance Sampling	70.86 (0.13)	22.41 (0.03)	10.11 (0.01)
Adaptive Tree Splitting	91.99 (0.09)	20.24 (0.03)	8.31 (0.01)
Multidimensional Lightcuts	34.37 (0.14)	17.31 (2.39)	5.35 (0.48)
Stochastic Lightcuts (Ours)	64.86 (2.17)	17.13 (1.45)	5.10 (0.39)

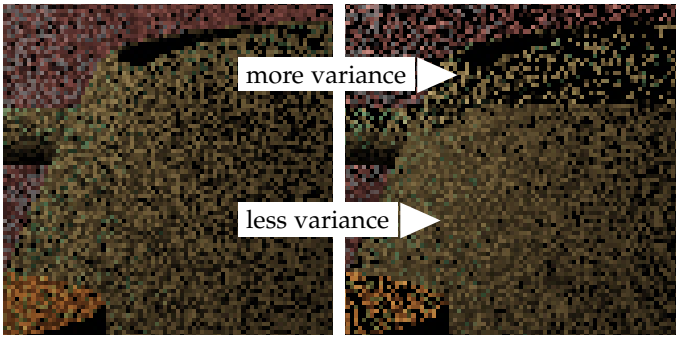


Fig. 6: *The same scene in Figure 4 rendered using stochastic lightcuts with two different light trees generated from the same light sources, showing that different trees can result in different distribution of variance over the image. Both images use 10 light samples.*

5 Discussion and Future Work

While algorithmically similar to lightcuts, our stochastic lightcuts solution is conceptually closer to traditional importance sampling [5]. The light tree is mainly used for efficiently computing importance sampling weights for lights. Therefore, stochastic lightcuts is safe to use with an arbitrarily small number of maximum light samples (such as one), though adding more light samples reduces the noise faster than traditional importance sampling. When the error threshold is set to zero, it merely provides a different and more efficient way of importance sampling, exceeding the excellent convergence rates of lightcuts. Using a non-zero error threshold, it provides an effective mechanism for adaptive light sampling.

Note that using "cuts" for sampling the light tree does not introduce bias to light sampling. The cuts effectively split the light tree into multiple subtrees, each of which is sampled stochastically. In that respect, cuts merely define the stratification for sampling. Using an error threshold for the cut selection does not introduce bias either, since the error threshold is only used for stratification but not for sample selection. When a large threshold is used, the cut selection produces fewer light subtrees (and therefore fewer light samples), but does not bias the light selection process.

Since the light tree is mainly used for computing the importance weights, the construction of the light tree impacts the quality of the sampling scheme. Our experiments with stochastic lightcuts revealed that using the same set of light sources, different light trees lead to different convergence rates at different parts of the image. Indeed, a poorly constructed light tree means poor importance sampling weights. In all our tests, we use the light tree construction algorithm of the original lightcuts method [1], which consistently provides superior sampling quality for stochastic lightcuts, as compared to prior methods [5], [19], [31]. However, we have also noticed a more significant variation in sampling noise with our method, depending on

the construction of the light tree. This can be seen in Table 1, which includes the standard deviation of root mean square error (RMSE) measured for 10+ rendered images with each method. Notice that our method (and multidimensional lightcuts) has significantly larger standard deviation, meaning RMSE varies depending on the quality of the tree.

In fact, the quality of the tree can also vary locally. Figure 6 shows two different light trees used for rendering the same scene. Notice that, as compared to the image on the left, the image on the right has more variance at some parts and less variance at others, showing that the variance distribution over the image depends on the light tree. This suggests that the results of stochastic lightcuts can be further improved by a specialized light tree construction algorithm, which would be an interesting direction for future work. Note that unlike the original lightcuts method that must use a stochastic light tree construction algorithm to avoid introducing bias during its deterministic lighting estimation, stochastic lightcuts can use a deterministic light tree construction algorithm, as it would not introduce bias during rendering with its stochastic lighting estimation.

An important advantage of stochastic lightcuts is that it allows achieving a fast lighting estimation using a very small number of light samples. This is unlike any prior scalable lighting solution [1], [19], [24], [29], [31], all of which require a large number of samples in practice. For example, adaptive tree splitting [31] often needs a user-specified *split threshold* to force it to skip the top levels of the light tree, thereby requiring more light samples. This is because the importance estimation of adaptive tree splitting is less reliable closer to the top of the tree where the bounding boxes are large. In that respect, stochastic lightcuts offers the most efficient solution to the many lights problem, especially when combined with multi-sampling or noise reduction techniques, which are commonplace with path tracing.

While stochastic lightcuts provides a good approximation with a few light samples, adding more light samples quickly reduces the error. Therefore, instead of using n light samples for lighting estimation and an average of m evaluations (which would mean $n \times m$ total light samples), it is more favorable to use $n \times m$ light samples within a single lighting estimation. This allows stochastic lightcuts to pick a deeper cut and thereby reduce the sampling error accordingly.

The weight function for importance sampling we present in Equations 2 and 3 is effectively equivalent to traditional importance sampling for the upper levels of the light tree, where the bounding boxes are large. That is why, our formulation does not perform worse than traditional importance sampling [5]. However, our importance formulation is not optimal and it is possible to use other weight functions with stochastic lightcuts that could outperform our weight function. For example, in the scene in Figure 1 our weight function leads to about 34% lower RMSE than the weight function of traditional importance sampling (using $w_j = \|\mathbf{I}_j\|$) and about 50% lower RMSE than random sampling (using $p_1 = \frac{1}{2}$). Yet, future research may develop more effective weight functions for stochastic lightcuts.

Another interesting future direction would be coupling stochastic lightcuts with Bayesian online regression [23], which has been shown to reduce the sampling noise by relying on lightcuts to provide the initial estimation and learning the lighting pattern in the scene. Replacing lightcuts with stochastic lightcuts in this context might introduce superior performance in terms of both computation time and sampling quality. It would also be interesting to evaluate how much of an improvement one could expect achieve on top of stochastic lightcuts by introducing a on-the-fly learning method. Furthermore, a similar learning approach can be used for guiding the light tree construction algorithm for either updating the light tree at render time or improving its construction for the next frame.

6 Conclusion

We have presented the stochastic lightcuts method, which introduces stochastic sampling concepts into the lighting estimation of lightcuts. We have also presented a robust hierarchical importance sampling approach that uses the existing light tree for improving the estimation, especially when used with a small number of light samples. Our approach eliminates the sampling correlation problems of lightcuts that lead to temporal instabilities and allows incorporating different light types, making our solution suitable for a much wider range of applications. Furthermore, because we can effectively estimate the illumination using a small number of light evaluations, we can achieve more than an order of magnitude faster lighting estimation than lightcuts. When more light evaluations are needed, our approach provides (and improves) the excellent convergence rates of lightcuts. Our tests also show that the stochastic lightcuts method can robustly handle substantial variations in light properties, such as light intensity. Furthermore, it is extremely easy to implement on top of an existing lightcuts implementation. Therefore, we expect our method to be quickly adopted by production renderers, particularly because robustly and efficiently handling a large number of light sources has been an important practical problem.

Acknowledgments

We would like to thank Pete Shirley for helpful suggestions, Daqi Lin for identifying the bias problem with the original dead branch avoidance algorithm, and Ian Mallett for his help in theoretical evaluation of this bias.

References

[1] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg, "Lightcuts: A scalable approach to illumination," *ACM Transactions on Graphics (Proceedings of SIGGRAPH '05)*, vol. 24, no. 3, pp. 1098–1107, 2005.

[2] C. Yuksel, "Stochastic lightcuts," in *High-Performance Graphics (HPG 2019)*. The Eurographics Association, 2019.

[3] C. Dachsbacher, J. Křivánek, M. Hašan, A. Arbree, B. Walter, and J. Novák, "Scalable realistic rendering with many-light methods," *Computer Graphics Forum*, vol. 33, no. 1, pp. 88–104, 2014.

[4] G. Ward, "Adaptive shadow testing for ray tracing," in *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, 1994, pp. 11–20.

[5] P. Shirley, C. Wang, and K. Zimmerman, "Monte carlo techniques for direct lighting calculations," *ACM Transactions on Graphics*, vol. 15, no. 1, pp. 1–36, 1996.

[6] E. Paquette, P. Poulin, and G. Drettakis, "A light hierarchy for fast rendering of scenes with many lights," *Computer Graphics Forum*, vol. 17, no. 3, pp. 63–74, 1998.

[7] S. Fernandez, K. Bala, and D. P. Greenberg, "Local illumination environments for direct lighting acceleration," in *Proceedings of the 13th Eurographics Workshop on Rendering*, 2002, pp. 7–14.
[8] A. Keller, "Instant radiosity," in *Proceedings of ACM SIGGRAPH '97*, 1997, pp. 49–56.
[9] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek, "Interactive global illumination using fast ray tracing," in *Proceedings of Eurographics Workshop Rendering*, 2002, pp. 15–24.
[10] I. Wald, C. Benthin, and P. Slusallek, "Interactive global illumination in complex and highly occluded environments," in *Proceedings of Eurographics Workshop on Rendering*, 2003, pp. 74–81.
[11] B. Segovia, J. C. Iehl, R. Mitanchey, and B. Péroche, "Bidirectional instant radiosity," in *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, ser. EGSR '06, 2006, pp. 389–397.
[12] S. Laine, H. Saransaari, J. Kontkanen, J. Lehtinen, and T. Aila, "Incremental instant radiosity for real-time indirect illumination," in *Proceedings of Eurographics Workshop Rendering*, 2007, pp. 277–286.
[13] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz, "Imperfect shadow maps for efficient computation of indirect illumination," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia '08)*, vol. 27, no. 5, pp. 129:1–129:8, 2008.
[14] J. Novák, D. Nowrouzezahrai, C. Dachsbacher, and W. Jarosz, "Virtual ray lights for rendering scenes with participating media," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 60:1–60:11, 2012.
[15] J. Novák, D. Nowrouzezahrai, C. Dachsbacher, and W. Jarosz, "Progressive virtual beam lights," *Computer Graphics Forum*, vol. 31, no. 4, pp. 1407–1413, 2012.
[16] Z. Dong, T. Grosch, T. Ritschel, J. Kautz, and H.-P. Seidel, "Real-time indirect illumination with clustered visibility," in *Vision, Modeling, and Visualizaion Workshop 2009*, 2009, pp. 187–196.
[17] A. C. Estevez and C. Kulla, "Importance sampling of many lights with adaptive tree splitting," in *ACM SIGGRAPH 2017 Talks*, ser. SIGGRAPH '17, 2017, pp. 33:1–33:2.
[18] C. Nichols, "Understanding adaptive lights," http://www.chaosgroup.com/blog/understanding-adaptive-lights, 2016.
[19] B. Walter, A. Arbree, K. Bala, and D. P. Greenberg, "Multidimensional lightcuts," *ACM Transactions on Graphics (Proceedings of SIGGRAPH '06)*, vol. 25, no. 3, pp. 1081–1088, 2006.
[20] T. Davidovič, I. Georgiev, and P. Slusallek, "Progressive lightcuts for gpu," in *ACM SIGGRAPH '12 Talks*, 2012, p. 1.
[21] B. Walter, P. Khungurn, and K. Bala, "Bidirectional lightcuts," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 59:1–59:11, 2012.
[22] R. Wang, Y. Huo, Y. Yuan, K. Zhou, W. Hua, and H. Bao, "Gpu-based out-of-core many-lights rendering," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 210:1–210:10, 2013.
[23] P. Vévoda, I. Kondapaneni, and J. Křivánek, "Bayesian online regression for adaptive direct illumination sampling," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 125:1–125:12, Jul. 2018.
[24] M. Hašan, F. Pellacini, and K. Bala, "Matrix row-column sampling for the many-light problem," *ACM Transactions on Graphics (Proceedings of SIGGRAPH '07)*, vol. 26, no. 3, 2007.
[25] M. Hašan, E. Velázquez-Armendariz, F. Pellacini, and K. Bala, "Tensor clustering for rendering many-light animations," in *Proceedings of Eurographics Workshop on Rendering*, 2008, pp. 1105–1114.
[26] T. Davidovič, J. Křivánek, M. Hašan, P. Slusallek, and K. Bala, "Combining global and local virtual lights for detailed glossy illumination," *ACM Trans. Graph. (Proceedings of SIGGRAPH '10)*, vol. 29, no. 6, pp. 143:1–143:8, 2010.
[27] J. Ou and F. Pellacini, "Lightslice: Matrix slice sampling for the many-lights problem," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia '11)*, vol. 30, no. 6, pp. 179:1–179:8, 2011.
[28] Y. Huo, R. Wang, S. Jin, X. Liu, and H. Bao, "A matrix sampling-and-recovery approach for many-lights rendering," *ACM Transactions on Graphics*, vol. 34, no. 6, pp. 210:1–210:12, 2015.
[29] C. Yuksel and C. Yuksel, "Lighting grid hierarchy for self-illuminating explosions," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)*, vol. 36, no. 4, pp. 110:1–110:10, 2017.
[30] D. Lin and C. Yuksel, "Real-time rendering with lighting grid hierarchy," *Proc. ACM Comput. Graph. Interact. Tech. (Proceedings of I3D 2019)*, vol. 2, no. 1, pp. 8:1–8:17, 2019, to appear.
[31] A. C. Estevez and C. Kulla, "Importance sampling of many lights with adaptive tree splitting," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 1, no. 2, pp. 25:1–25:17, Aug. 2018.
[32] P. Moreau, M. Pharr, and P. Clarberg, "Dynamic many-light sampling for real-time ray tracing," in *High-Performance Graphics - Short Papers*, M. Steinberger and T. Foley, Eds. The Eurographics Association, 2019.
[33] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst, "Embree: A kernel framework for efficient cpu ray tracing," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 143:1–143:8, Jul. 2014.
[34] M. Hašan, J. Křivánek, B. Walter, and K. Bala, "Virtual spherical lights for many-light rendering of glossy scenes," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia '09)*, vol. 28, no. 5, pp. 143:1–143:6, 2009.

Cem Yuksel is an associate professor in the School of Computing at the University of Utah. Previously, he was a postdoctoral fellow at Cornell University, after receiving his PhD in Computer Science from Texas A&M University in 2010. His research interests are in computer graphics and related fields, including physically-based simulations, realistic image synthesis, rendering techniques, global illumination, sampling, GPU algorithms, graphics hardware, modeling complex geometries, knitted structures, and hair modeling, animation, and rendering.