Stochastic Substitute Trees for Real-Time Global Illumination

Wolfgang Tatzgern
Graz University of Technology

Benedikt Mayr
Graz University of Technology

Bernhard Kerbl
TU Wien

Markus Steinberger
Graz University of Technology

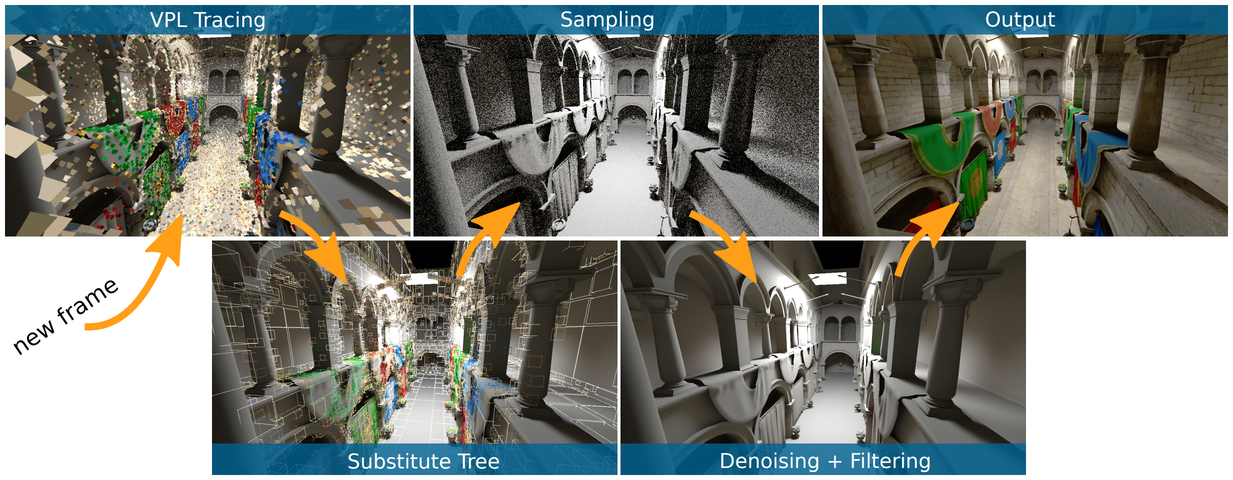


Figure 1: Global illumination with stochastic substitute trees: For every frame, we distribute VPLs (0.4 ms**), build a substitute tree over the VPLs (**0.9 ms**), stochastically sample the tree (**3.4 ms**), apply CNN denoising (**17.4 ms**) and temporal filtering (**0.9 ms**).**

ABSTRACT

With the introduction of hardware-supported ray tracing and deep learning for denoising, computer graphics has made a considerable step toward real-time global illumination. In this work, we present an alternative global illumination method: The stochastic substitute tree (SST), a hierarchical structure inspired by lightcuts with light probability distributions as inner nodes. Our approach distributes virtual point lights (VPLs) in every frame and efficiently constructs the SST over those lights by clustering according to Morton codes. Global illumination is approximated by sampling the SST and considers the BRDF at the hit location as well as the SST nodes' intensities for importance sampling directly from inner nodes of the tree. To remove the introduced Monte Carlo noise, we use a recurrent autoencoder. In combination with temporal filtering, we deliver real-time global illumination for complex scenes with challenging light distributions.

CCS CONCEPTS

• **Computing methodologies** → *Ray tracing*; *Graphics processors.*

KEYWORDS

Global illumination, many lights, ray tracing, real-time

ACM Reference Format:
Wolfgang Tatzgern, Benedikt Mayr, Bernhard Kerbl, and Markus Steinberger. 2020. Stochastic Substitute Trees for Real-Time Global Illumination. In *Symposium on Interactive 3D Graphics and Games (I3D '20), May 5–7, 2020, San Francisco, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3384382.3384521

1 INTRODUCTION

Real-time global illumination (GI) is one of the most sought-after features in computer graphics. Path tracing renownedly provides an elegant solution to simulating global light transport by recursively sampling along a large number of rays that traverse the scene. Today, almost all competitive path tracers leverage the massively parallel processing power of the graphics processing unit (GPU). Recent GPUs have been fitted with dedicated ray tracing modules that help mitigate the bottleneck of ray-based scene traversal. Furthermore, specialized inferencing units (*tensor cores*) have been added to evaluate pre-trained convolutional neural networks (CNN) to filter undersampled images. Although backed by hardware, ray tracing and inference remain prohibitively expensive. Hence, these novel GPU features are mostly used in combination with rasterization for bulk geometry rendering, contributing minor visual enhancements.

In contrast to path tracing, *instant radiosity* (IR) describes an alternative solution to GI on hardware rasterization [Keller 1997]. IR effectively reduces the problem of indirect illumination to trivial direct illumination by introducing a large number of *virtual point lights* (VPL) that are distributed throughout the scene. In addition to avoiding image-order complexity, such *many-light* rendering approaches are particularly effective when combined with the concept of *lightcuts*: instead of full surface shading with all lights in the scene, lightcuts select a subset of VPLs and amplify their intensity for shading [Walter et al. 2005]. However, generating lightcuts is computationally intensive and usually implies off-line processing.

In this paper, we describe a new approach for fast, high-quality global illumination on modern GPUs by extending the concept of stochastic lightcuts (SLC) [Yuksel 2019]. Our *Stochastic Substitute Trees* (SST) use custom substitute lights whose geometric properties are purpose-built to stochastically approximate a continuous light distribution. In the context of real-time GI, we show how SSTs provide an ideal complement to recently introduced GPU capabilities: in combination with ray tracing for visibility, deep learning for image reconstruction and adaptive temporal filtering, SSTs compare favorably to other state-of-the-art solutions in terms of performance and temporal stability. We make the following contributions:

- We introduce a novel method for generating substitutes for inner light tree nodes that model their light contribution as distributions. To the best of our knowledge, we are the first to propose using representatives whose geometric properties are disjoint from the initial VPLs.
- We show how SSTs can be generated in parallel on the GPU each frame for real-time animations and dynamic lighting.
- We demonstrate how the SST can be efficiently sampled for Lambertian and glossy materials.
- We demonstrate our approach in NVIDIA Falcor and evaluate it against state-of-the-art methods for global illumination on a variety of static and animated scenes.
- We discuss the combination with hardware ray tracing, inferencing capabilities, and temporal filtering.

2 BACKGROUND AND RELATED WORK

With the original rendering equation [Kajiya 1986], path tracing emerged as the de facto Monte Carlo method for global illumination. While path tracing entire frames remains unattainable in real-time, ray tracing is commonly used for effects such as shadows, ambient occlusion or glossy reflections [Keller et al. 2019]. Ray tracing is classified as "embarrassingly parallel", but achieving optimal utilization of the GPU has proven challenging [Aila and Laine 2009]. Recent models thus contain dedicated ray tracing hardware [Haines and Akenine-Moller 2019; NVIDIA Corporation 2018].

Aside from path tracing, the most prominent alternatives for simulating global illumination include *metropolis light transport*, *photon mapping* and *instant radiosity* [Jensen 1996; Keller 1997; Veach and Guibas 1997]. Among these methods, instant radiosity stands out as an approach that easily maps to standard rasterization APIs, since all indirect illumination is approximated by VPLs. To compute their distribution, initial light sources emit multiple VPLs, whose path is traced throughout the scene. Each intersection with geometry may lead to another bounce or the termination of the path and final placement of the VPL. Based on this idea, many-light rendering has emerged as a new class of algorithms for image synthesis [Dachsbacher et al. 2014; Lin and Yuksel 2019; Prutkin et al. 2012; Ritschel et al. 2008; Segovia et al. 2007]. While using a higher number of VPLs yields superior results in most many-light methods, the implied effort for shading severely inhibits scalability.

To cap the number of VPLs that participate in shading, Walter et al. [2005] presented *lightcuts*. They generate a static binary light tree over all VPLs, where each inner node represents the lights in its leafs. These *representative lights* are obtained by recursively merging nodes with similar attributes. While the position of each representative is chosen randomly from one of its child nodes, its intensity equates to their sum to satisfy the preservation of radiant flux in arbitrary tree cuts. For a portion of the output image domain (*e.g.*, one pixel), they traverse the light tree and identify a lightcut that bounds the error caused by shading with the corresponding representatives. Visibility determination plays a key role, because fully occluded light sources and subtrees can be trivially rejected. Multidimensional lightcuts [Walter et al. 2006] extend the core algorithm to raise its efficiency in higher-order domains accounting for visual effects like depth-of-field, motion blur and participating media. Lightcuts interpolation [Rehfeld and Dachsbacher 2016] exploit the similarity between tree cuts of nearby shading points. Spherical virtual point lights [Hašan et al. 2009] introduces a spherical light type which addresses the loss of energy due to clamping and enables glossy lighting scenarios. Rich-VPLs [Simon et al. 2015] propose a new light type accounting for multiple incident light paths. Stochastic Light Culling [Tokuyoshi and Harada 2016] uses Russian Roulette for unbiased culling of VPLs in real-time.

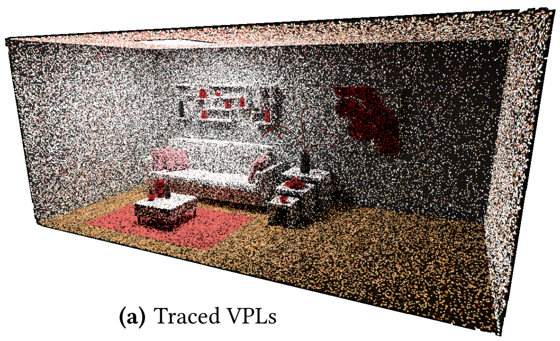
Due to the inherent sample correlation between cuts from the same static light tree, visible indicators of undersized lightcuts manifest as discretized penumbrae and isolated "blobs" of light, making them difficult to resolve with automated filtering. Stochastic lightcuts by Yuksel [2019] trade these structured artifacts for Monte Carlo noise. The selection of inner nodes' light positions is made via importance sampling of their respective leafs for each lightcut, thus rejecting the idea of precomputed representatives.

As lightcuts were originally devised as an offline method on the CPU, Hašan et al. proposed a GPU-friendly, matrix-based method for scalable many-light rendering [Hašan et al. 2007]. Several optimizations have been proposed that extend the original idea and further improve its performance [Huo et al. 2015; Ou and Pellacini 2011]. However, runtimes for rendering single frames with these techniques remain in the order of seconds to minutes.

Stochastic methods for image synthesis can often be enhanced by the application of reconstruction techniques on noisy or undersampled input. In many cases, a single sample per pixel suffices to produce high-fidelity output with state-of-the-art image reconstruction techniques [Dammertz et al. 2010; Li et al. 2012]. In this context, animations pose a particular challenge since reconstruction must take both spatial and temporal stability into account to preserve frame-to-frame coherence [Schied et al. 2017, 2018]. Recent work focusing on machine learning has shown that spatio-temporal denoising can largely be delegated to the run-time inference of pre-trained neural networks [Chaitanya et al. 2017].

3 STOCHASTIC SUBSTITUTE TREES

Our complete pipeline is outlined in Fig. 1: We first distribute VPLs according to IR and build our SST over the resulting VPLs. We then generate G-buffers using rasterization and for all fragments sample the illumination from the SST, yielding a noisy light image. With the help of G-buffers, we use a CNN for denoising and temporal filtering to ensure consistent shading over time. In the following, we give a detailed description of the substitute tree concept and its parallel construction. In Section 4, we describe the process of drawing samples from the SST and provide additional, implementation-specific details in Section 5.


(a) Traced VPLs

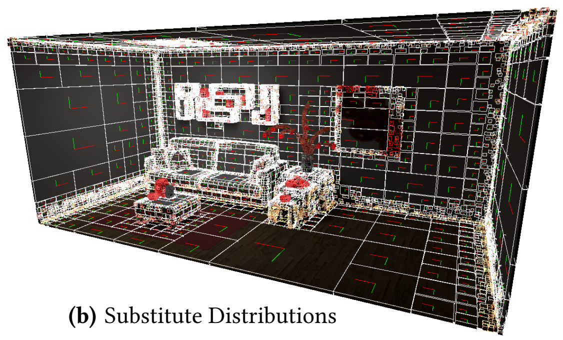

(b) Substitute Distributions

Figure 2: (a) 150 k VPLs distributed in the Pink Room with their RGB intensity. (b) While previous work uses discrete locations from the input VPLs for inner tree nodes, our stochastic substitute trees store continuous distributions of VPLs, defined by a substitute position and variance: red and green lines show the standard deviations on the surface; blue lines along the normal. Notice how large planar regions are captured by single distributions and variance along the normal is virtually undetectable.

3.1 The Substitute Tree

Our substitute light trees are motivated by the original methods for lightcuts and, specifically, stochastic lightcuts [Walter et al. 2005; Yuksel 2019]. The inner nodes of light trees copy their geometric properties from one of the leaf nodes, *i.e.*, one of the original VPLs. This policy leads to noticeable lighting patterns when the number of VPLs is low. With stochastic lightcuts, the inner nodes' geometric properties are chosen via hierarchical importance sampling from one of the leaf nodes for each cut, eliminating the sampling correlation. In order to support dynamic light sources and animated scenes in real-time, we aim to keep the number of VPLs low to permit updating the tree each frame while minimizing visual artifacts.

To better approximate the sampling of the—in general smooth—indirect illumination, we do not use leaf VPL locations for the inner nodes of our light tree. Instead, we pretend as if the leafs were distributed more densely than they actually are by first merging VPLs into clusters and then approximating those clusters with fitting *substitute distributions* (see Fig. 2). Although we could theoretically use any probability distribution to describe the clusters, we aim to use a compact and efficient representation for sampling. We intuitively chose a normal distribution to describe the individual clusters and were content with the yielded results, hence we did not investigate further distributions for this work.

Similar to previous approaches, we construct the full light tree bottom-up and merge substitute clusters as we go. For traditional lightcuts, ideal merge candidates should be spatially close and have similar normals to minimize the view-dependent deviations from full shading when using representatives. For our approach, the situation is more complicated since we must ensure that estimated distributions align with the geometry of the scene. As describing distributions that follow non-trivial surfaces would be costly, we try to limit clusters to surfaces that can be described and sampled straightforwardly. The simplest surface, a plane, can be described by a location and normal. Incidentally, this corresponds to the required information for shading with a VPL. Following our previous considerations, we generate normally distributed samples on the plane defined by the cluster position and normal. A multivariate normal distribution described by a covariance matrix would also work, but incur greater memory and computational requirements.

3.2 Distribution Merging and Errors

To obtain a suitable substitute distribution for each inner node, we compute the intensity-weighted cluster position $\mathbf{p}$ and the positional variance σ^2. To have σ_x^2 and σ_y^2 represent the variance of lights on the plane surface, we transform their positions into a local plane coordinate system where the z-axis matches the cluster plane normal. Merged position and variance are computed using

$$\mathbf{p} = \frac{c_1 \cdot \mathbf{p_1} + c_2 \cdot \mathbf{p_2}}{c_1 + c_2} \quad \text{and} \tag{1}$$

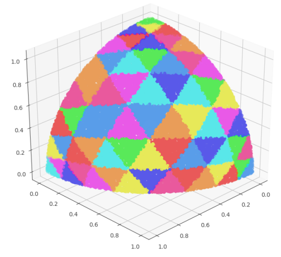
$$\sigma^2 = \frac{c_1 \cdot ({\sigma_1}^2 + \Delta'_{\mathbf{p1}} \odot \Delta'_{\mathbf{p1}}) + c_2 \cdot ({\sigma_2}^2 + \Delta'_{\mathbf{p2}} \odot \Delta'_{\mathbf{p2}})}{c_1 + c_2}, \tag{2}$$

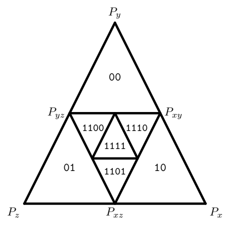
where c_1, c_2 are the child contributions, $\mathbf{p_1}$, $\mathbf{p_2}$ are the child cluster positions, ${\sigma_1}^2$, ${\sigma_2}^2$ are the child cluster variances, $\Delta'_{\mathbf{p1}}$, $\Delta'_{\mathbf{p2}}$ are the difference vectors from $\mathbf{p}$ to $\mathbf{p_1}$ and $\mathbf{p_2}$ in transformed cluster space and $\odot$ is the Hadamard product. The weights c_1, c_2 correspond to the luminance of the cluster's RGB light intensity. While we could compute a spread (and thus variance) alongside the VPL distribution and use this spread as initial variance, we have found that starting with simple VPLs where the variance is zero and the mean equals the position works equally well in practice and avoids complex spread computations. An example of a SST is given in Fig. 2(b).

Similar to regular lightcuts, the merged node's intensity is simply the sum of the child intensities and its extent is encoded by an axis-aligned bounding box that encloses all light sources in its subhierarchies. The merged normal is linearly interpolated based on the child contributions c_1, c_2. Hence, clusters with high weights (and thus high intensities) have a stronger influence on the outcome.

Note that strongly diverging normals will alter the cluster coordinate system and thus partially invalidate the local variance values during merging. To avoid corresponding artifacts, we therefore compute and store additional information on each cluster's suitability for sampling. During sampling, we then skip any distribution that does not fulfill the corresponding criteria. For evaluating the suitability of a substitute node for sampling, we use two indicators:

- The normal similarity metric $v = v_1 \cdot v_2 \cdot \max(\langle \mathbf{n_1}, \mathbf{n_2} \rangle, 0)$
- The variance σ_z along the plane normal


(a) Normal octant clustering



(b) Triangle bit encoding

Figure 3: (a) Normal clustering in an octant of the unit sphere using 3 refinement levels; (b) the corresponding bit values for two refinements.

For the normal similarity metric v, $\mathbf{n_1}$ and $\mathbf{n_2}$ as well as v_1 and v_2 capture normals and the normal similarities of the child clusters. The normal similarity metric serves as an indicator for both the similarity between the child cluster distribution orientations and the curvature of the surface. Furthermore, v is guaranteed to decrease when going up the tree. The variance σ_z indicates any deviations along the plane normal, *i.e.*, even if VPL normals point into the same direction, they may not lie on the same surface. For leaf nodes the normal similarity is initialized to one and the variance to zero. Given a set of suitable parameter thresholds, we can combine these indicators to form a joint condition for ensuring that merged substitute clusters are of sufficient quality for sampling:

$$v \geq v_{\mathtt{min}} \ \wedge \ \sigma_z \leq \sigma_{\mathtt{max}} \ . \tag{3}$$

3.3 Parallel Construction

Since we aim to rebuild the light tree in every frame, minimizing its construction time is essential. Traditional lightcuts follow an $O(n^2)$ approach to find pairs of lights/clusters that minimize the introduced error according to their metric. As this approach is incompatible with our real-time constraints, we follow the algorithm by Karras [2012] for building bounding volume hierarchies.

To start the hierarchy construction, we sort VPLs according to a spatial index using Morton codes. Additionally, we incorporate bits extracted from the normals, since we seek VPLs that point in similar directions to obtain candidates for suitable substitute distributions. Using polar coordinates to describe normal directions leads to non-uniform quantization, as bits cover differently-sized areas on the sphere. Thus, we first identify the octant the normal points to and then subdivide the octant as a triangle to yield approximately equally-sized areas [Taubin and Rossignac 1998], as shown in Fig. 3. Each octant is represented by three bits, followed by six bits for the subdivision level of the triangle. Our final code C for each VPL is composed of the Morton code M, normal code N and a unique id:

$$C = M \oplus N \oplus id, \tag{4}$$

where $\oplus$ is the bit concatenation operator. We thus sort primarily according to position, since VPLs lying on the same surface will be close to each other. Secondary sorting occurs according to normals, to ensure that VPLs with similar normals are prioritized merge targets. To circumvent any duplicate codes, a unique ID is concatenated at the end. In the final 64-bit code, M uses 30 bits and N uses 9 bits. The remaining bits are used for the unique id.

Following the initial sort according to (4), the tree is constructed bottom-up. For merging the inner node substitute distributions, one thread per leaf node is started. At each level, only one thread may continue to the next-higher parent. After computing the distributions and corresponding quality metrics, the node information is stored as an array of structs. For each node, we store:

- distribution variance, ID and child IDs, (3 floats + 3 ints)
- the world space position and normal, (2×3 floats)
- the bounding box, (2×3 floats)
- light color and intensity, (4 floats)
- a flag indicating if the distribution is suitable for sampling,

yielding a total structure size of 96 bytes (padded). Thus, the full node information can be written with six vector store instructions.

4 STOCHASTIC LIGHT SAMPLING

We sample our stochastic substitute tree by traversing it until we arrive either at a leaf VPL or a suitable inner node, in which case we draw the sample from its substitute cluster distribution. The number of suitable inner nodes depends on the scene geometry, since the surface planarity relates to the usability of our substitute distributions. For drawing multiple samples, we can simply run the traversal multiple times from the root. However, since the evaluation of a sample requires expensive visibility determination, we aim to maximize the quality gain using only one sample per pixel.

4.1 Tree Traversal

We define a cluster's illumination $L_{\mathbb{C}}$ for a shading point $\mathbf{x}$ from direction ω with the same terms as in the original lightcuts paper:

$$L_{\mathbb{C}}(\mathbf{x}, \omega) \approx M_j(\mathbf{x}, \omega) \cdot Gj(\mathbf{x}) \cdot V_j(\mathbf{x}) \cdot \mathbf{I_j}, \tag{5}$$

where for a tree node j, M_j is the material term, G_j is the geometric term, $V_j(\mathbf{x})$ represents visibility of $\mathbf{x}$ from the light source and $\mathbf{I_j}$ is the sum of VPL intensities in the subtree. In order to stochastically sample clusters based on their expected contribution to the shading of $\mathbf{x}$, we start traversal from the root and continually compute weights w_1, w_2 for the current node's children. Similar to stochastic lightcuts, the probability of choosing the ith child is given by $\frac{w_i}{w_1+w_2}$. If we enter a *dead branch* (*i.e.*, $w_1 + w_2 = 0$), we stop the traversal. While this wastes a light sample, backtracking within the shader is not feasible for our performance requirements. We compute w as an upper bound of $L_{\mathbb{C}}$ for each child by plugging in adequate upper bounds for those terms in (5) that are too expensive to evaluate. To map the RGB value of $\mathbf{I_j}$ to $\mathbb{R}^1$, we use its relative luminance in practice. V_j can be trivially set to 1, thus assuming no occlusions.

For the material term M_j, we use the upper bound of the cosine between ω and the surface normal at $\mathbf{x}$, multiplied by the BRDF. The upper bound of the cosine is computed w.r.t. the bounding box [Walter et al. 2005]. The diffuse BRDF is independent of the view or light direction, thus constant. For the glossy part of the BRDF, we use the following heuristic: We compute the perfect reflection $\mathbf{R}$ of the view vector along the surface normal at $\mathbf{x}$, *i.e.*, a mirror-like reflection. We then test if the ray $\mathbf{x} + t \cdot \mathbf{R}$ intersects with the cluster's bounding box. If so, we use $\mathbf{R}$ for the glossy BRDF computation. Otherwise, the BRDF is evaluated with a direction vector $\mathbf{R'}$ that actually does intersect the bounding box and minimizes the difference to $\mathbf{R}$.

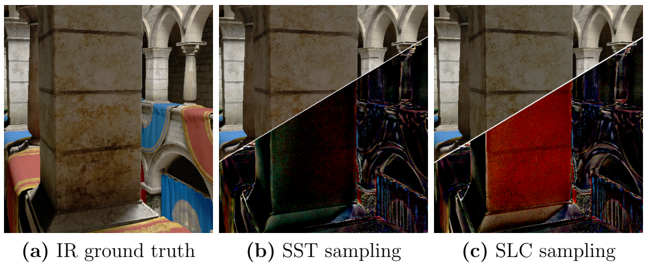


Figure 4: Comparing the accuracy of color bleeding in the final image. While SST (b) faithfully captures the soft reflection of the red fabric on the pillar as it is seen in the ground truth (a), it is almost entirely missing with SLC (c). The bottom right half of (b,c) shows the pixel difference $\times 10$.

Since we focus on omni-directional lights, G_j reduces to inverse quadratic attenuation. We note that, in order to guard against singularities in the geometric term during hierarchical importance sampling, stochastic lightcuts set $G_j = 1$ for omni-directional lights if $\mathbf{x}$ lies inside the cluster's bounding box, *i.e.*, ignoring the distance from the sample to the cluster. However, we found that this leads to missing color bleeding when using few samples per pixel, as shown in Fig. 4. Noticeable color bleeding at $\mathbf{x}$ is often due to VPLs that are close to $\mathbf{x}$ and thus—especially for lower levels of the tree—likely to be enclosed by the same bounding box as $\mathbf{x}$, leading to rejection of the distance term. This makes tree traversal more likely to end up in different subtrees, which do not contribute to the color bleeding and therefore increases the noise. We avoid singularities by simply bounding the distance from below against an ε term.

4.2 Sampling the Substitute Distribution

In contrast to previous many-light methods, which usually focus on diffuse shading only, we also consider non-Lambertian BRDFs. Before traversing the tree, we decide whether to sample the diffuse or glossy portion of the BRDF, based on the magnitude of the respective material terms. For diffuse samples, we stop the traversal at the first usable substitute distribution, see (3). A suitable substitute distribution is often found early on, reducing the sampling depth significantly without any visible difference to the final illumination (see Fig. 5). For drawing a light position sample $\mathbf{p_S}$ from the distribution, we generate three normally distributed random variables, according to the cluster variance σ^2 in the local coordinate frame. After transformation to world coordinates, we clamp $\mathbf{p_S}$ by the node's bounding box and evaluate the incoming radiance according to Eq. (5). This implies visibility testing to determine the binary value for V_j at shading point $\mathbf{x}$, which we resolve through ray tracing. When sampling the glossy portion of the BRDF, we ignore the substitutes and only consider leaf VPLs as viable candidates, since highly glossy reflections would be overly blurred by picking substitute samples. While this leads to performance penalties for glossy surfaces, it is vital for ensuring high quality. We note that our substitute distributions introduce bias, since we generate VPLs which are not part of the traced light transport paths, which implies that drawn light positions may also be slightly inside geometry. Nonetheless, we did not encounter any obvious light leakage during our tests.

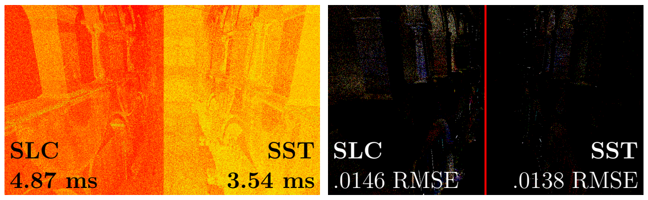


Figure 5: (left) The color-coded tree traversal depth (🟨 / 🟥 for low/high) is on average 20 for SLC, whereas SST only requires 15 steps on average for a 100 k VPL tree. (right) SST are both faster and achieve better image quality compared to the ground truth (difference $\times 10$).

5 IMPLEMENTATION

We implemented our approach on top of the NVIDIA Falcor framework [Benty et al. 2019] in DirectX 12 and CUDA. Ray tracing is done via hardware accelerated RTX. For diffuse lighting, we use the Lambertian model. For the glossy BRDF, we use Cook-Torrance reflectance [Cook and Torrance 1982].

VPL Generation. For VPL generation, we follow the core approach described by instant radiosity [Keller 1997]. We start by probabilistic sampling of the original light sources according to their size and flux. The number of VPL paths to trace for each light depends on its intensity, as well as the configured maximum number of bounces. To trace these rays, we use hardware supported DX12 ray tracing. At every hit, we generate a new VPL and append it to a global buffer. We use cosine-weighted sampling for the next bounce direction and early path termination via Russian Roulette.

Substitute Tree Generation. Substitute trees are built in parallel on the GPU with CUDA. Internal nodes are stored in the same buffer as the previously generated VPLs. The tree is constructed by a single kernel, decreasing the number of active threads while progressing up through the tree. Since the number of active nodes usually quickly falls below the core count of modern GPUs, the performance loss due to keeping inactive threads around is minimal.

Sampling. Substitute tree sampling is done by a ray tracing shader. We first fill a GBuffer with first hit shading information [Deering et al. 1988]. For each pixel, we traverse the tree to draw a VPL sample and cast a shadow ray from the VPL location $\mathbf{p_S}$ to the pixel's stored position in world space in the GBuffer. We offset the origin of the ray by a scene size dependent epsilon value, since VPL samples may be slightly inside geometry. To account for direct illumination, we randomly pick and sample one of the scene's light sources. Thus, we trace two shadow rays per pixel—one for tree sampling and one for direct illumination.

Denoising and Temporal Stability. Our approach produces Monte Carlo noise and thus, in theory, any Monte Carlo denoiser is applicable. In practice, we use a recurrent autoencoder (RDAE) with the same training procedure and loss function as stated in the original paper [Chaitanya et al. 2017]. To improve temporal stability, we apply the temporal accumulation step of the adaptive spatio-temporal variance-guided filter (A-SVGF) [Schied et al. 2017, 2018], using the pixel-wise intensity gradient from the RDAE output as temporal accumulation factor to control blending. This mitigates artifacts like ghosting and temporal lags, especially for animated

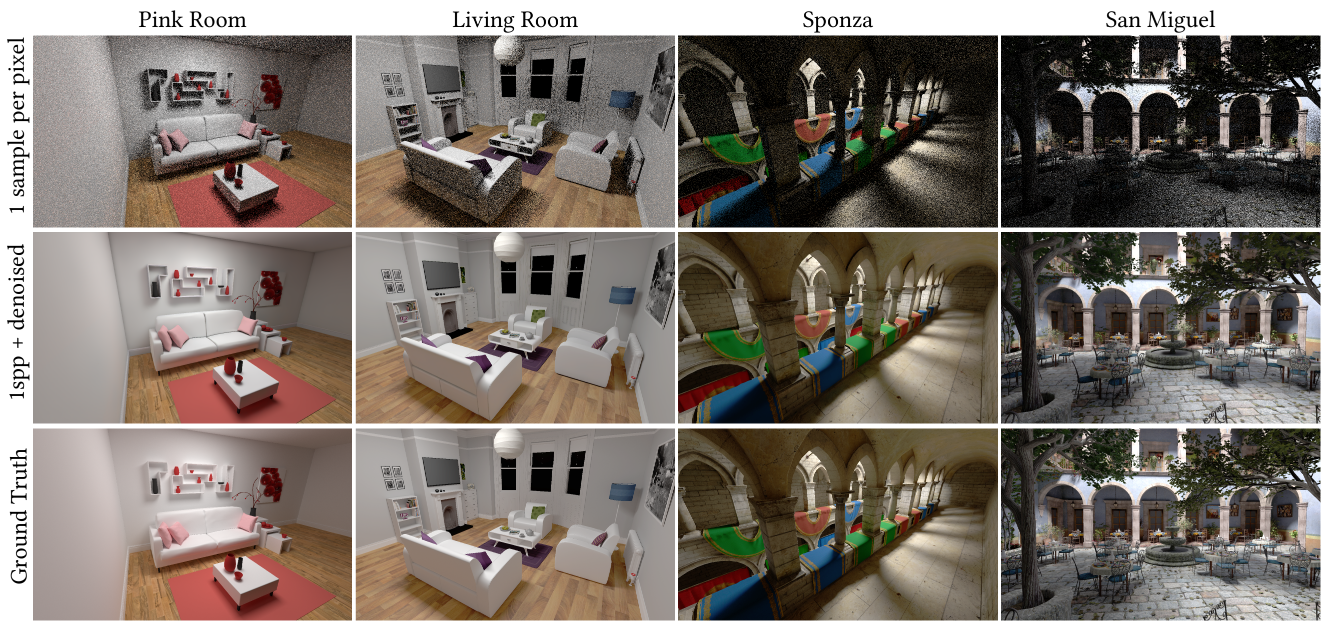


Figure 6: Sampling result of SST with 1 spp for 100k VPLs, denoised and ground truth. While there is some loss of indirect lighting (*e.g.* the wall in the Pink Room), which we attribute to the denoiser, SST overall produces high quality lighting.

scenes. To save time on inference, we reduce the number of features of RDAE, while still maintaining high quality. For downsampling, we use 24, 40, 54, 72, 96, and 96 features. For upsampling, we use (96, 96), (72, 72), (54, 54), (40, 40), (64, 32). As activation function we use ReLu, for upsampling we use the deconvolution operator. The network was modeled and trained with Tensorflow and TensorRT as inference engine. The training data was generated with the Sponza and San Miguel scene, with 500 captured frames per scene. Inference is executed on NVIDIA tensor cores in fp16 mode. While testing, we found that the trained denoiser generalized well to scenes which have not been part of the training sets. We further confirmed that our method is compatible with other denoisers by substituting our CNN-based denoiser with the SVGF [Schied et al. 2017] and comparing the visual results.

6 EVALUATION

We compare our stochastic substitute trees (SST) in detail to regular path tracing (PT) with varying number of bounces, stochastic lightcuts (SLC), and show the main differences w.r.t. the recent real-time method for Lighting Grid Hierarchies [Lin and Yuksel 2019] (LGH). All timings are generated on a GeForce RTX 2080Ti. We use $v_{\mathtt{min}} = 0.5$ and $\sigma_{\mathtt{max}} = 0.1$; choosing smaller $v_{\mathtt{min}}$ and greater $\sigma_{\mathtt{max}}$ leads the algorithm to accept clusters further up the tree and increases chances of sampling VPLs in mid-air or inside geometry.

The reference path tracer is based on NVIDIA's Falcor framework and uses *next event estimation* to improve convergence. The first hit is determined by a rasterized G-Buffer, additional bounces are computed with RTX. For quality comparisons, we use path tracing with 4000 samples per pixel (spp) and seven bounces as ground truth. We evaluate on four scenes with an image resolution of 1280×720 shown in Fig. 6. The Pink Room and Living Room are simple in terms of geometry, but offer interesting lighting situations and highly glossy surfaces. Sponza is slightly more complex in geometry and shows high amounts of color bleeding and shadowing through the corridors. Finally, San Miguel offers high geometric load and enforces diverse light paths. Fig. 6 shows that SST produces lighting close to the ground truth and accurately reconstructs the lighting under noisy conditions and complex setups. Some difficult-to-reach areas, like the table in the living room scene or the shelfs in the pink room, appear slightly darker, which is typical for VPL approaches.

For runtime evaluation, we separately look at SST construction, which we can run in every frame or when lighting changes; SST sampling, which rasterizes a G-buffer, samples the tree and performs direct lighting using RTX; denoising; and temporal filtering. For temporally stable lighting, we require ~100 k VPLs. Their distribution takes 0.27 ms to 0.85 ms, with an additional 0.9 ms for SST building. Sampling takes 2.7 ms to 7.3 ms, denoising 17.4 ms and temporal filtering 0.9 ms. Including VPL distribution and tree building, a full frame takes 22.17 ms to 27.35 ms. While we achieve 30fps overall, denoising is clearly the limiting factor, making up 63 % to 80 % of frame time. With GPU hardware set to further advance CNN performance, we expect this issue to lessen in the future.

6.1 Comparison

Table 1 shows the sample timings and final image quality for all tested approaches. SST is always faster and always achieves superior quality to SLC. We attribute our quality advantage to two factors: First, using substitute distributions approximates global illumination as if it were generated with more VPLs and thus can achieve image qualities for which other approaches require significantly higher VPL counts. Second, our node selection during tree traversal generalizes attenuation also to close-by VPLs, which have a higher contribution to localized lighting effects (see Fig. 4). Since the ground truth is generated with path tracing, PT variants may be slightly advantaged w.r.t. the quality comparison. Nevertheless, Table 1 shows that while PT with a single bounce is efficient, the

Table 1: Comparison between SST and SLC (100k VPLs) and path tracing with increasing number of bounces. While 1 bounce PT is faster than SST, its quality is lower, as it misses large amounts of indirect lighting. With increasing bounces, PT achieves comparable quality, but is considerably slower. To quantify the potential performance gains, we show the relative increase SSIM per invested ms **compared to the worst performing approach (higher is better, best in bold).**

		Sampling time (ms)					RMSE denoised					SSIM denoised					Gain (% SSIM/ms)				
		SST	SLC	$PT_{1'}$	$PT_{2'}$	$PT_{3'}$	SST	SLC	$PT_{1'}$	$PT_{2'}$	$PT_{3'}$	SST	SLC	$PT_{1'}$	$PT_{2'}$	$PT_{3'}$	SST	SLC	$PT_{1'}$	$PT_{2'}$	$PT_{3'}$
diffuse	Pink Room	2.65	3.59	1.73	3.20	4.71	.036	.048	.133	.081	.053	.960	.951	.912	.956	.968	**5.22**	2.10	0.00	2.99	1.88
	Living Room	2.88	4.34	1.78	3.33	4.76	.026	.033	.070	.039	.024	.960	.956	.928	.965	.972	**2.91**	1.09	0.00	2.39	1.48
	Sponza	3.36	4.54	2.75	5.45	8.10	.017	.017	.029	.015	.012	.971	.967	.877	.967	.975	**15.41**	5.03	0.00	3.33	1.83
	San Miguel	5.77	6.83	8.86	17.81	27.18	.048	.049	.046	.045	.045	.872	.862	.873	.880	.881	**1.89**	0.00	0.54	0.16	0.09
+ glossy	Pink Room	4.24	5.08	1.88	3.62	5.33	.048	.059	.147	.092	.062	.947	.940	.900	.943	.958	1.99	1.25	0.00	**2.47**	1.68
	Living Room	4.11	5.14	1.91	3.77	5.27	.040	.047	.087	.051	.032	.946	.940	.910	.952	.966	1.64	0.93	0.00	**2.26**	1.67
	Sponza	3.99	5.31	2.95	5.82	8.67	.017	.017	.029	.015	.012	.970	.967	.876	.967	.975	**9.04**	3.86	0.00	3.17	1.73
	San Miguel	7.34	7.84	8.76	17.83	27.45	.072	.074	.065	.059	.058	.758	.752	.772	.793	.795	**2.40**	0.00	2.17	0.41	0.22

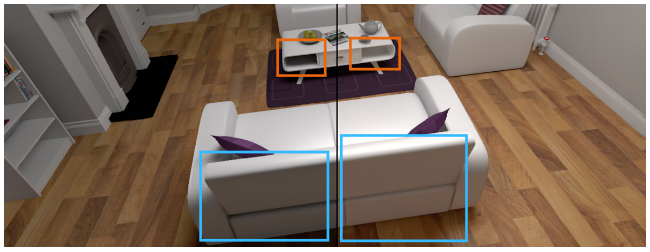


Figure 7: Path tracing with a single bounce (left) misses significant portions of indirect illumination. SST (right) distributes VPLs throughout multiple bounces (in a fraction of the trace time) and thus faithfully recovers the illumination.

image quality is clearly inferior to SST (see also Fig. 7). In all scenes, there are areas that require more than a single bounce to be properly lit. When the number of bounces is increased, PT catches up in terms of quality, but also requires more time than ours, *i.e.*, PT requires 2 to 3 bounces to achieve our quality while being 1.5× slower. San Miguel is an exception, where all approaches struggle to achieve good quality, showing the highest error among all scenes. However, substitute trees are faster than even single-bounce PT, as ray tracing is particuarly expensive due to the geometric load.

To quantify the potential gains of the different approaches, we list the SSIM gain per invested ms when switching to either technique from the lowest-quality approach, *e.g.*, switching from $PT_{1'}$ to SST for Pink Room, achieves an ∼.052 SSIM increase for the invested 1 ms. SST always yields the best gains for pure diffuse lighting. When using a non-Lambertian BRDF, scenes with highly glossy materials, like Pink Room and Living Room, work slightly better with path tracing. Sponza, featuring only moderately glossy materials, again works best with SST. In San Miguel, little gains can be achieved overall with either method.

Unfortunately we were unable to include LGH in this comparison as the LGH framework is not compatible with Falcor to produce identical scene setups. Nevertheless, we compare a view of Sponza directly between LGH and SST, as shown in Fig. 8: It is lit with indirect illumination generated from a single point light, placed at the center of the scene. The ground truth is generated by the LGH framework using instant radiosity. LGH significantly overestimates

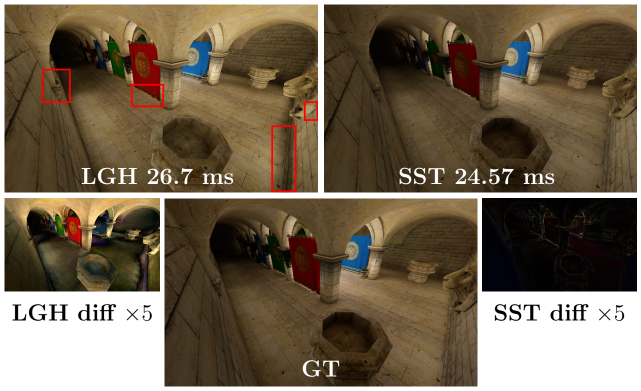


Figure 8: While LGH requires similar times as SST, it is further from the ground truth indirect illumination. LGH's shadowing heuristics overestimate the general illumination and predict high shadows around corners (red boxes).

lighting in some places, while being too dark in others. We attribute these issues to LGH's shadow heuristic, which is insufficient to distinguish between the brighter (but occluded) VPLs in the center of the scene and the less bright (but visible) VPLs in the corridors. Increasing the shadow ray count to 2 per pixel (the maximum supported by the framework) did not improve quality, but prolonged the frame time from 26.7 ms to 37.6 ms (from hierarchy generation to final image). For the entire pipeline, SST requires 24.57 ms and creates clearly superior results compared to LGH.

6.2 Scalability

Since we regularly (potentially in every frame) generate new VPLs and the substitute tree, performance of these steps is essential. Table 2 shows the timings for VPL generation and tree building with different scenes. The runtime for VPL generation is governed by the scene's complexity, while the duration of tree building depends only on the number of VPLs. About 100 k VPLs are needed for temporally stable lighting in our tests, resulting in ∼1.2 ms to 1.7 ms in total. Table 3 shows that the sample time also increases with tree depth. Sampling from the SST is more efficient as soon as the tree is not too shallow. For very small VPL counts (10k)—which imply temporally instable lighting conditions—SLC is slightly faster.

Table 2: Timings for VPL generation and substitute tree building with varying VPL counts. VPL generation depends on the scene geometry due to ray tracing. Tree building only depends on the VPL count. All times in ms**.**

	VPL generation				Tree building			
# VPLs	10k	100k	500k	1M	10k	100k	500k	1M
Pink Room	0.13	0.27	0.74	1.47	0.58	0.89	2.44	4.51
Living Room	0.13	0.30	0.83	1.54	0.54	0.89	2.38	4.43
Sponza	0.14	0.40	1.32	2.38	0.59	0.88	2.39	4.35
San Miguel	0.19	0.85	3.44	7.05	0.56	0.91	2.35	4.38

Table 3: Sampling timings in ms **with 1spp. For shallow trees (which are temporally not stable) SLC is slightly faster than sampling from SST. However, as the tree size increases, our substitute sampling clearly has the performance edge.**

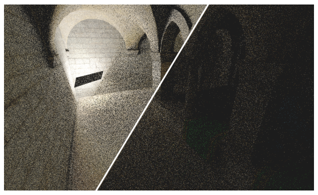
		10k		100k		500k		1M	
		SST	SLC	SST	SLC	SST	SLC	SST	SLC
diffuse	Pink Room	2.22	**2.10**	**2.65**	3.59	**2.88**	5.45	**3.10**	6.79
	Living Room	2.27	**2.17**	**2.88**	4.34	**3.36**	6.63	**3.73**	8.04
	Sponza	2.77	**2.74**	**3.36**	4.54	**3.78**	6.12	**3.92**	7.00
	San Miguel	**4.81**	5.19	**5.77**	6.83	**6.64**	8.51	**7.30**	9.39
+ glossy	Pink Room	3.90	**3.62**	**4.24**	5.08	**5.02**	7.08	**5.18**	9.14
	Living Room	3.68	**3.27**	**4.11**	5.14	**5.03**	7.01	**5.24**	8.86
	Sponza	3.36	**3.26**	**3.99**	5.31	**4.58**	6.99	**4.73**	8.02
	San Miguel	6.45	**6.29**	**7.34**	7.84	**8.92**	9.85	**10.04**	11.80

For 100k VPLs, SST is already 1.35× faster. Our performance edge increases with the tree size, indicating a better scalability of substitute trees. When sampling glossy reflections, the performance difference is less pronounced, as glossy sampling is more complex and we traverse the tree further. Nevertheless, SST is still faster for all tree sizes above 10k VPLs. Our performance edge comes from two factors. First, we stop our traversal early, as outlined in Fig. 5. Second, SLC's random sampling of VPL locations leads to scattered memory accesses, while SST only requires the (coherent) fetching of distribution information and thus less memory bandwidth.

6.3 Discussion and Limitations

So far, we only reported our results on image quality for individual still frames. However, our experiments have shown that denoising the result of our sampling stage with temporal filtering produces temporally stable global illumination in animated scenes, where we rebuild the SST each frame with sufficiently many VPLs. Thus, our approach supports dynamic light sources and scenes while maintaining good frame-to-frame coherence. Due to the difficulties of establishing temporal stability via conventional illustrations and for lack of a universally accepted quality metric, we kindly refer the reader to our supplemental video for an adequate demonstration.

Compared to path tracing, one particular advantage of SST is that it can handle difficult-to-reach light sources due to its VPL distribution in the style of instant radiosity, as shown in Fig. 9(a). Although SST does support glossy BRDFs, mirrors and glossy reflections pose a challenge, as sampling the tree will in general not sample the perfect reflection direction and thus degrade sharpness


(a) Occluded Light Source

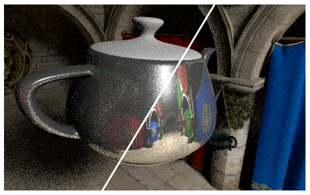

(b) Highly Glossy Material

Figure 9: (a) An area light source, pointed at a corner, is sampled well with SST (left). Path tracing (right) has difficulties connecting the rays to the light source, which leads to undersampling. (b) While SST (left) supports glossy BRDFs, highly glossy surfaces are challenging to resolve with 1spp, as the reflection directions are usually not sampled and thus more noise remains compared to path tracing (right).

(see Fig. 9(b)). As a remedy, one can trace an additional ray for samples that hit a mirror and sample the tree from the next hit.

Working with VPLs, using heuristically driven importance sampling, and estimating VPL distributions leads to a biased approach. Additionally, as is common with VPL approaches, we avoid singularities in the geometric term by clamping, which results in slightly darkened corners. However, focusing on the creation of convincing real-time rendering content, being unbiased is not our main goal.

7 CONCLUSION

We have presented a new, scalable many-light rendering approach for simulating global illumination in real-time in animated scenes. Our stochastic substitute trees approximate the actual global illumination by converting clusters of input VPLs into suitable probability distributions. While we can benefit from the same advantages as stochastic lightcuts, sampling with our tree is more efficient since traversal can be stopped early. Furthermore, our substitute distributions generate higher-quality lighting than approaches that directly sample from the underlying VPLs. Our sampling method compares favorably to stochastic lightcuts and the recently proposed Lighting Grid Hierarchy in terms of quality and runtime. We can usually capture indirect illumination more accurately with a single bounce than path tracing and report higher gains for image quality metrics. By combining our sampling method with CNN denoising and temporal filtering, we can generate temporally stable approximations of global illumination in real-time, in fully dynamic scenes.

Our approach supports highlights and produces agreeable results for semi-glossy surfaces. However, highly glossy surfaces and sharp, mirror-like reflections may still produce noise and are better handled with path tracing. Looking forward, we believe that tracing additional rays for those hits may combine the best of both worlds, leading to hybrid VPL sampling and path tracing. Our work can be downloaded from *https://github.com/wotatz/sst-demo*.

ACKNOWLEDGMENTS

This research was supported by the German Research Foundation (DFG) under GrantNo.: STE 2565/1-1, the Austrian Science Fund (FWF) under GrantNo.: I3007 and the Research Cluster "Smart Communities and Technologies (Smart CT)" at TU Wien. Test scenes were kindly provided by Morgan McGuire's Computer Graphics Archive [McGuire 2017].

REFERENCES

Timo Aila and Samuli Laine. 2009. Understanding the Efficiency of Ray Traversal on GPUs. In *Proceedings of the Conference on High Performance Graphics 2009 (HPG '09)*. ACM, New York, NY, USA, 145–149. https://doi.org/10.1145/1572769.1572792

Nir Benty, Kai-Hwa Yao, Lucy Chen, Tim Foley, Matthew Oakes, Conor Lavelle, and Chris Wyman. 2019. The Falcor Rendering Framework. https://github.com/NVIDIAGameWorks/Falcor https://github.com/NVIDIAGameWorks/Falcor.

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4, Article 98 (July 2017), 12 pages. https://doi.org/10.1145/3072959.3073601

R. L. Cook and K. E. Torrance. 1982. A Reflectance Model for Computer Graphics. *ACM Trans. Graph.* 1, 1 (Jan. 1982), 7–24. https://doi.org/10.1145/357290.357293

Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák. 2014. Scalable Realistic Rendering with Many-Light Methods. *Comput. Graph. Forum* 33, 1 (Feb. 2014), 88–104. https://doi.org/10.1111/cgf.12256

Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik P. A. Lensch. 2010. Edge-avoiding À-Trous Wavelet Transform for Fast Global Illumination Filtering. In *Proceedings of the Conference on High Performance Graphics (HPG '10)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 67–75. http://dl.acm.org/citation.cfm?id=1921479.1921491

Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. 1988. The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics. *SIGGRAPH Comput. Graph.* 22, 4 (June 1988), 21–30. https://doi.org/10.1145/378456.378468

Eric Haines and Tomas Akenine-Moller. 2019. *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*. Apress, Berkely, CA, USA.

Miloš Hašan, Jaroslav Křivánek, Bruce Walter, and Kavita Bala. 2009. Virtual Spherical Lights for Many-Light Rendering of Glossy Scenes. In *ACM SIGGRAPH Asia 2009 Papers (SIGGRAPH Asia '09)*. Association for Computing Machinery, New York, NY, USA, Article Article 143, 6 pages. https://doi.org/10.1145/1661412.1618489

Miloš Hašan, Fabio Pellacini, and Kavita Bala. 2007. Matrix Row-column Sampling for the Many-light Problem. *ACM Trans. Graph.* 26, 3, Article 26 (July 2007). https://doi.org/10.1145/1276377.1276410

Yuchi Huo, Rui Wang, Shihao Jin, Xinguo Liu, and Hujun Bao. 2015. A Matrix Sampling-and-recovery Approach for Many-lights Rendering. *ACM Trans. Graph.* 34, 6, Article 210 (Oct. 2015), 12 pages. https://doi.org/10.1145/2816795.2818120

Henrik Wann Jensen. 1996. Global Illumination Using Photon Maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. Springer-Verlag, London, UK, UK, 21–30. http://dl.acm.org/citation.cfm?id=275458.275461

James T. Kajiya. 1986. The Rendering Equation. *SIGGRAPH Comput. Graph.* 20, 4 (Aug. 1986), 143–150. https://doi.org/10.1145/15886.15902

Tero Karras. 2012. Maximizing Parallelism in the Construction of BVHs, Octrees, and K-d Trees. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics (EGGH-HPG'12)*. Eurographics Association, Goslar Germany, Germany, 33–37. https://doi.org/10.2312/EGGH/HPG12/033-037

Alexander Keller. 1997. Instant Radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 49–56. https://doi.org/10.1145/258734.258769

Alexander Keller, Timo Viitanen, Colin Barré-Brisebois, Christoph Schied, and Morgan McGuire. 2019. Are We Done with Ray Tracing?. In *ACM SIGGRAPH 2019 Courses (SIGGRAPH '19)*. ACM, New York, NY, USA, Article 3, 381 pages. https://doi.org/10.1145/3305366.3329896

Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012. SURE-based Optimization for Adaptive Sampling and Reconstruction. *ACM Trans. Graph.* 31, 6, Article 194 (Nov. 2012), 9 pages. https://doi.org/10.1145/2366145.2366213

Daqi Lin and Cem Yuksel. 2019. Real-Time Rendering with Lighting Grid Hierarchy. *Proc. ACM Comput. Graph. Interact. Tech. (Proceedings of I3D 2019)* 2, 1, Article 8 (2019), 17 pages. https://doi.org/10.1145/3321361

Morgan McGuire. 2017. Computer Graphics Archive. https://casual-effects.com/data `https://casual-effects.com/data`.

NVIDIA Corporation. 2018. *NVIDIA Turing GPU Architecture.* https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

Jiawei Ou and Fabio Pellacini. 2011. LightSlice: Matrix Slice Sampling for the Many-lights Problem. *ACM Trans. Graph.* 30, 6, Article 179 (Dec. 2011), 8 pages. https://doi.org/10.1145/2070781.2024213

Roman Prutkin, Anton Kaplanyan, and Carsten Dachsbacher. 2012. Reflective Shadow Map Clustering for Real-Time Global Illumination. (2012).

Hauke Rehfeld and Carsten Dachsbacher. 2016. Lightcut Interpolation. In *Proceedings of High Performance Graphics (HPG '16)*. Eurographics Association, Goslar, DEU, 99–108.

T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. 2008. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Trans. Graph.* 27, 5, Article 129 (Dec. 2008), 8 pages. https://doi.org/10.1145/1409060.1409082

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R. Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal Variance-guided Filtering: Real-time Reconstruction for Path-traced Global Illumination. In *Proceedings of High Performance Graphics (HPG '17)*. ACM, New York, NY, USA, Article 2, 12 pages. https://doi.org/10.1145/3105762.3105770

Christoph Schied, Christoph Peters, and Carsten Dachsbacher. 2018. Gradient Estimation for Real-time Adaptive Temporal Filtering. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2, Article 24 (Aug. 2018), 16 pages. https://doi.org/10.1145/3233301

B. Segovia, J.C. Iehl, and B. Peroche. 2007. Metropolis Instant Radiosity. *Computer Graphics Forum* (2007). https://doi.org/10.1111/j.1467-8659.2007.01065.x

Florian Simon, Johannes Hanika, and Carsten Dachsbacher. 2015. Rich-VPLs for Improving the Versatility of Many-Light Methods. *Comput. Graph. Forum* 34, 2 (May 2015), 575–584. https://doi.org/10.1111/cgf.12585

Gabriel Taubin and Jarek Rossignac. 1998. Geometric Compression Through Topological Surgery. *ACM Trans. Graph.* 17, 2 (April 1998), 84–115. https://doi.org/10.1145/274363.274365

Yusuke Tokuyoshi and Takahiro Harada. 2016. Stochastic Light Culling. *Journal of Computer Graphics Techniques (JCGT)* 5, 1 (March 2016), 35–60. http://jcgt.org/published/0005/01/02/

Eric Veach and Leonidas J. Guibas. 1997. Metropolis Light Transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 65–76. https://doi.org/10.1145/258734.258775

Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. 2006. Multidimensional Lightcuts. *ACM Trans. Graph.* 25, 3 (July 2006), 1081–1088. https://doi.org/10.1145/1141911.1141997

Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. 2005. Lightcuts: A Scalable Approach to Illumination. *ACM Trans. Graph.* 24, 3 (July 2005), 1098–1107. https://doi.org/10.1145/1073204.1073318

Cem Yuksel. 2019. Stochastic Lightcuts. In *High-Performance Graphics (HPG 2019)*. The Eurographics Association. https://doi.org/10.2312/hpg.20191192