Stochastic Light Cut Exploration

for Real-Time Dynamic Global Illumination

Matthias Tavasszy

TU Delft

Stochastic Light Cut Exploration

for Real-Time Dynamic Global Illumination

by

Matthias Tavasszy

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday September 20, 2021 at 15:00 PM.

Student number:	4368401	
Project duration:	September 1, 2020 – September 1, 2021	
Thesis committee:	Prof. dr. E. Eisemann,	TU Delft, supervisor
	Dr. ir. R. Santbergen,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This report is the end result of my education at the TU Delft. It is written as a concise scientific paper, and does not contain a detailed report of the development process, many iterations, trials and missteps.

The past year has been a hectic, but interesting one. Finding a good balance between work and leisure was personally challenging, especially when working from home during the global pandemic. Despite this, I am proud of what I have learned and accomplished, and even more interested in the subject than before I started. Programming on the GPU at this scale (10.000+ lines of code) for the first time, I learned a lot about Vulkan and the intricacies of GPU programming, as well as how powerful the proper hardware can be when used right. It both daunts and excites me to think of how much there is left to explore. As per the famous quote of Aristotle, "The more you know, the more you know you don't know.".

I would like to thank my professor and supervisor Elmar Eisemann who has provided his expertise throughout the project. I would also like to give special thanks to Mathijs Molenaar, who has supported me both on the technical and motivational front for the past year, and was always ready to exchange ideas and help me improve the quality of my work. A special thanks as well to Markus Billeter, for helping me with Vulkan and my work in general.

I hope this work or the insights it provides may contribute to existing rendering engines to make global illumination more accessible, as well as that it may contribute to future research on this topic.

Matthias Tavasszy
Delft, September 2021

Contents

1 Introduction 1

2 Previous Work 3
- 2.1 Ray tracing . . . 3
- 2.2 Instant Radiosity and RSM . . . 3
- 2.3 Light Cuts . . . 3

3 Method Overview 5

4 Building the Light Tree 7
- 4.1 VPL Generation . . . 7
  - 4.1.1 Expected pVPL Contribution . . . 7
  - 4.1.2 pVPL selection . . . 7
- 4.2 Constructing the SST . . . 8
  - 4.2.1 VPL Sorting . . . 8
  - 4.2.2 Merging the STT . . . 8

5 Sampling the Light Tree 11
- 5.1 Shallow Light Cuts . . . 11
  - 5.1.1 Splitting Order . . . 12
  - 5.1.2 Expected Cluster Contribution . . . 12
  - 5.1.3 Light Cut Blending . . . 12
  - 5.1.4 Mock Cuts . . . 13
- 5.2 Tree Traversal . . . 14
- 5.3 Cluster Evaluation . . . 14

6 Implementation Details 15

7 Results 17
- 7.1 Comparison of Sampling Strategy . . . 18
  - 7.1.1 Frame time and Root Mean Squared Error . . . 19
- 7.2 Light Cut Blending . . . 19
- 7.3 Mock Cuts . . . 20
- 7.4 Upsampling . . . 20

8 Discussion 23

9 Conclusion 25

10 Future work 27

1 Introduction

Since the starting days of Computer Graphics, global illumination is a research topic of great interest. Global illumination, sometimes called indirect illumination, simulates the scattering of light between multiple surfaces. While light rays often move homogeneously right after being emitted from a light source, once they hit a surface the rays scatter in many directions, making their behaviour and interactions with the rest of the scene harder to simulate simultaneously. This scattering effect and possibly infinite amount of bounces makes simulating global illumination a complex task, and requires evaluation of light transfer between multiple surfaces before the final image can be determined. Due to its inherent complexity, it is an ongoing challenge to achieve higher quality images using global illumination with reasonable timings. In real-time interactive settings, the trade-off between quality and time becomes ever more apparent.

Often, the most computationally expensive part of simulating global illumination is checking for occlusion, where we verify whether light traveling between two points is blocked by any surface. To accelerate a simulation for global illumination, two obvious solutions exist: decrease the computation time necessary for individual occlusion checks, or design an algorithm in such a way that less occlusion checks are needed for a result of the same quality.

Recent developments in hardware-accelerated ray tracing [5] address the first point, decreasing the amount of time necessary for individual ray-geometry intersection checks used to check occlusion. However, while this enables real-time computation of physically correct shadows and reflections, it is not fast enough yet to allow for real-time global illumination. Approximations are still limited to low sample rates, which cause high amounts of noise. While noisy approximations may be filtered on the graphics card [18], on current-day hardware proper filtering can still cost an amount of time that can be considered prohibitive for real-time applications. Furthermore, any filtering solution still requires good input in order to deliver a high-quality outcome.

Decreasing the total amount of occlusion checks necessary may be addressed by only simulating light rays that have a high contribution to our image. Regions with a high expected contribution can be identified and prioritized for lighting evaluations using importance sampling. For example, by storing the light a surface receives directly from light sources, it is possible to map out the most important surfaces to sample indirect light from. This work proposes such a solution, using Virtual Point Lights (VPL) stored in a Stochastic Substitute Tree (SST) [18] combined with a hybrid sampling approach to allow for real-time evaluation of second-bounce global illumination for fully dynamic scenes. The use of Light Cuts [21] and Light Tree Traversal [23] in combination with SST allows for faster sampling and less noise in the final image when compared to similar methods. Next to this, we propose several improvements to the methods used as well as insights in techniques useful for GI pipelines in general.

The remainder of this report is built up as follows: Chapter 2 describes previous work related to global illumination. Chapter 3 discusses our proposed pipeline. In Chapter 4 and 5 we provide a detailed explanation of each step in the pipeline, and Chapter 6 highlights any further details in our implementation. We describe the evaluation of our method in Chapter 7, of which the results are discussed in 8. Chapter 9 provides a summary of the proposed work, and we provide an insight for possible future work in Chapter 10.

2

Previous Work

2.1. Ray tracing

An intuitive way of simulating light transport is by tracing individual light rays. Kajiya et al. [8] proposed this idea in form of a Monte Carlo method for computing global illumination, called path tracing. Here, the fact that light travels in a straight path in a continuous medium is exploited to express intersections between light and geometry as mathematically simple line-geometry intersections. While ray tracing allows for simulating effects such as realistic shadows, reflection, refraction and indirect illumination it exhibits slow convergence. Many solutions for accelerating ray tracing have been proposed over the years, like bi-directional path tracing, several types of acceleration structures and solutions like multiple importance sampling [19, 11, 4, 1]. In recent years ray tracing has received hardware support on modern graphics cards, enabling native hardware acceleration of ray-geometry intersections [5].

2.2. Instant Radiosity and RSM

Another approach to simulating global illumination is Instant Radiosity [10]. Here, scene illumination is cached in Virtual Point Lights, or VPLs. These VPLs are distributed onto surfaces by emitting them from light sources and path tracing them throughout the scene. Once distributed, global illumination can be determined by evaluating each VPL as a point light source, with its outgoing energy based on its traced path during distribution.

Instead of distributing VPLs using path tracing, Reflective Shadow Maps (RSM) [3] utilize shadow maps to generate second bounce VPLs directly. Since shadow maps only contain points on surfaces visible from the light source, any position represented in a shadow map can be used to generate a valid VPL representing bounced light. Because it is infeasible to sample from an entire shadow map for each pixel, a subset of random points are sampled with an increased sampling density close to the evaluated pixel.

Bidirectional Reflective Shadow Maps (BRSM) [15] recognize this random subsampling as a main shortcoming in RSM. Selecting the VPLs without considering their contribution to the final image leads to a suboptimal evaluation. Considering each position on a RSM as a potential VPL or pVPL, they weigh these using a cheap estimation of each pVPL's expected contribution to the final rendered image. These weights are used in a stochastic selection to pick a subset of VPLs from the pVPLs, which are used in the final evaluation. Each pVPL's expected contribution is approximated by sampling a limited number of surrounding surfaces and evaluating their illumination using the pVPL. Nearby surfaces are retrieved efficiently by projecting the pVPL into the camera G-Buffer and sampling nearby pixels, using screen-space distance as an approximation for world-space distance.

2.3. Light Cuts

To capture all the direct light in an entire scene, a high amount of VPLs is necessary. However, not every VPL will be relevant for every surface in our image. Light Cuts [21] proposes to build a light tree on top of the list of VPLs, which allows for quick retrieval for relevant VPLs for any given surface, as well as allowing clusters of VPLs to be evaluated at once by replacing them with a single point light representing the entire cluster. In the light tree, the VPLs are placed as leaves of the tree, and each inner node store a single point light representing

the cluster of VPLs of the leaf nodes in its subtree. Approximated clusters by a single representative point light results in strong sub-linear scalability w.r.t. the amount of VPLs. For each pixel in the image, a selection of tree nodes is made, called a light cut. This cut is chosen such that every path between the root and each of its leaves contain exactly one node in the cut. Approximating a VPL cluster as a point light introduces error, thus clusters are chosen such that the upper bound of the introduced error does not exceed 2%, a number chosen in relation with human perception of error.

When using a low amount of VPLs, evaluating the same representative point light for multiple pixels leads to sampling correlation between pixels. This shortcoming is recognized by the same authors in Multidimensional Lightcuts [22], where they propose to store a list of (typically 32) representative points for each cluster and randomly selecting one of these points during sampling. However, this greatly increases the memory footprint of each inner node, and while it decreases sampling correlation, it does not prevent it entirely.

Stochastic Lightcuts (SLC) [23] exchanges sampling correlation for noise by combining light cuts with a hierarchical importance sampling method. They traverse down the light tree from the node of an evaluated cluster, to stochastically find a representative leaf node instead of storing a fixed representative light for the cluster. Stochastic Substitute Trees (STT) [18] solve the correlation problem by ignoring exact VPL locations altogether. Instead, an inner node stores a substitute distribution of its cluster's VPLs, and this distribution can be sampled from during evaluation. To be able to apply this method in real-time, the authors do not compute any light cuts in their tree, and instead traverse the light tree once for every pixel, starting traversal from the root of the tree. The resulting undersampled image is then filtered using a pre-trained convolutional neural network. However, this filtering still takes a significant amount of time, much longer than generating the undersampled image itself.

A main restriction for applying Light Cuts in real-time applications is the generation of a light cut for every pixel. ManyLODs [7] propose to maintain cuts between frames, allowing for reuse of light cuts after initialization. Real-Time Stochastic Lightcuts [12] solve this differently, by sharing light cuts between pixels using interleaved sampling [20, 17]. This, as well as constructing the light tree as a fast perfect binary tree make the SLC algorithm GPU-friendly enough for real-time applications.

3

Method Overview

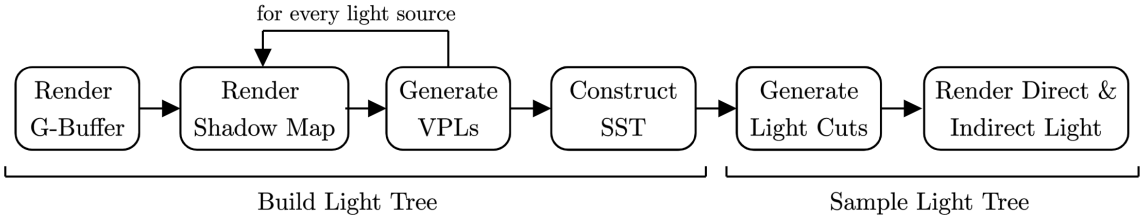


Figure 3.1: the high level pipeline

Our real-time pipeline for computing diffuse second-bounce global illumination is shown in Figure 3.1. We propose a pipeline which combines quick generation of VPLs stored in a Stochastic Substitute Tree (SST), coupled with an efficient sampling strategy that uses both shallow light cuts and stochastic tree traversal. An example is illustrated in Figure 3.2. While we are not the first to propose such a hybrid use of shallow light cuts and tree traversal [12], the novel combination with SST allows for even better performance compared to similar methods. Next to this approach, we propose improvements for BRSM generation as well as a new term for approximating expected cluster contribution, improvements for shallow light cut creation and a new light cut blending strategy which allows for efficient light cut sharing for multiple pixels.

The pipeline is run entirely on the GPU, and for each frame all VPLs, the SST and the light cuts are recomputed which allows for fully dynamic scenes. The pipeline can be divided in two phases: a light tree building phase (step 1-4), and a light tree sampling phase (step 5-6). Both phases will be discussed in more detail in chapters 4 and 5 respectively. First, we describe a high level overview of our proposed pipeline.

Using a deferred rendering approach, a G-Buffer [16] from the camera's perspective is rendered, storing material properties, normal, and position of the underlying geometry for each pixel. Using this, for each light source we render a Bidirectional Reflective Shadow Map (BRSM) [15] and generate a fixed amount of VPLs which are added to a global array. Once all VPLs are generated for every light source, we build a Stochastic Substitute Tree (SST) [18] from the VPLs by setting them as the leaf nodes of the tree and merging nodes in a bottom-up fashion.

After construction, we use the light tree to sample VPLs or VPL clusters for evaluation. We utilize a hybrid sampling method, combining shallow and easy to compute light cuts with stochastic tree traversal. Computing a light cut for each pixel is too costly however, thus we only compute and store a single light cut for each $k \times k$ pixel block. Like Real-Time Stochastic Lightcuts (RTSLC) [12], we use cut sharing to avoid step artifacts at the transitions between pixel blocks. RTSLC uses Interleaved Sampling [17, 20] for this purpose, where each pixel only evaluates a subset of the nodes in the shared light cut. However, this means that in order to maintain a certain amount of cluster evaluations per pixel, the shared cut needs to be larger, which is more costly to compute. We propose a simple yet effective stochastic light cut blending strategy which blends the light cuts of surrounding pixel blocks. This way, each pixel utilizes the full cut, allowing us to save time by being able to use smaller shared light cuts for evaluation.

At the final stage, each pixel uses the nodes of its blended light cut as a starting point for tree traversal. Starting at the respective cut node, for each node we stochastically determine whether to continue traversal

with either its left or right child node, until we reach a node with a cluster of sufficient quality for sampling. The probabilities for picking either child node are based on their expected contribution to the surface at the pixel we are currently evaluating. The substitute distributions in an SST allow for stronger criteria for finding clusters suitable for sampling, allowing us to terminate traversal higher in the tree instead of having to traverse all the way down to leaf nodes. This results in less time needed for traversal, as well as evaluating more VPLs in a single cluster. Once traversal terminates, the last node's cluster is evaluated and its contribution to the illumination of our pixel is added to the final image.

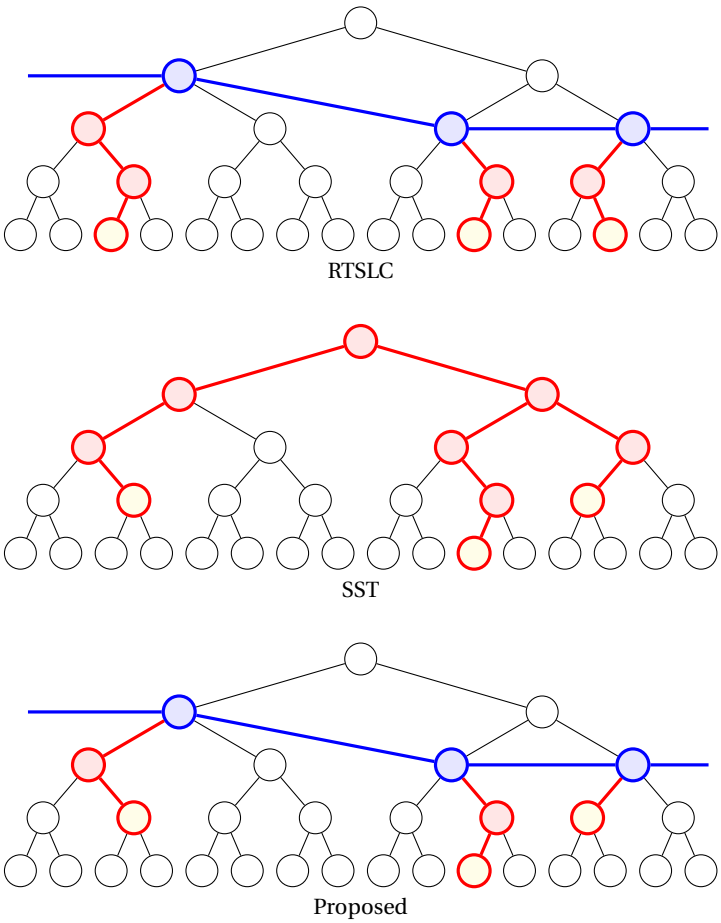


Figure 3.2: Comparison between tree traversal for RTSLC, SST, and the proposed hybrid method. The blue nodes represent a shallow light cut, red nodes indicates traversal and yellow nodes' clusters or VPLs are evaluated. RTSLC starts traversal from light cut nodes and traverses all the way down to its leaf nodes, while SST starts traversal at the root and returns clusters from inner nodes. Our method both starts at light cut nodes and returns inner node clusters, drastically decreasing traversal time for each pixel.

4

Building the Light Tree

This chapter describes the first phase of our pipeline (Figure 3.1), the construction of the light tree. The quality of the acceleration structure and its underlying data plays an essential role in the efficiency of data sampling and the quality of the final result.

4.1. VPL Generation

In the first step of the pipeline, light emitted by the light sources is stored in the scene using Virtual Point Lights. The VPLs are generated from an advanced version of Reflective Shadow Maps (RSM) [10], called Bidirectional Reflective Shadow Maps (BRSM) [15]. As opposed to RSM, BRSM takes the expected contribution to the final image of each shadow map position into account before sampling VPLs.

4.1.1. Expected pVPL Contribution

We only compute each pVPLs contribution to a few nearby surfaces, since computing their contribution to the full image is too time-consuming. To sample surrounding surfaces without having to resort to ray tracing, Ritschel et al. [15] propose to use the camera G-Buffer. By projecting a pVPL's location into the G-Buffer, we can sample positions near the projected coordinate and use the geometry stored in the G-Buffer at these locations. This strategy stems from the fact that if two points in 3D world space are close to each other, they will be close to each other in the projected 2D screen space as well. While the opposite does not always hold true, it is good enough for a quick approximation. We compute the diffuse part of the BRDF for each surface sampled from the G-Buffer, using the pVPL as light source. Occlusion is ignored to make the evaluation as quick as possible. The expected contribution of the pVPL is the sum of the evaluated illumination values from its sampled surfaces.

4.1.2. pVPL selection

Once the contribution of each pVPL is computed, we can stochastically select VPLs from the pVPLs. First, all pVPL contribution values are normalized such that all values of the pVPLs on the BRSM sum up to 1. The result is a 2D Probability Density Function (PDF) from which we can sample according to the pVPL contributions. In the original BRSM, this is done by determining the 2D Cumulative Density Function (CDF) using Summed Area Tables (SAT) [6]. To sample according to the distribution, one would have to invert the cumulative density. As a full inversion is costly, the inversion is done using a binary search over the CDF instead. Using uniform random numbers as input, binary search is performed n times both horizontally and vertically on the CDF to pick the locations of n pVPLs which will be added to the VPL array.

We propose using Hierarchical Warping (HW) [2] for pVPL sampling instead. Hierarchical Warping is a technique to transform a uniformly distributed set of point to match an underlying secondary distribution, and allows for the use of hardware-accelerated mipmapping instead of computing a SAT. Starting out with an initial point distribution, for each point the PDF is subdivided in four quadrants, and the point is warped according to the summed PDF values over each quadrant. After each warp, the quadrant the point ended up in is subdivided again, and the point is warped again between its new quadrants accordingly. This process is repeated until the subdivided quadrants become smaller than our PDF resolution. An example for the first warp iteration over a set of points is shown in Figure 4.1.

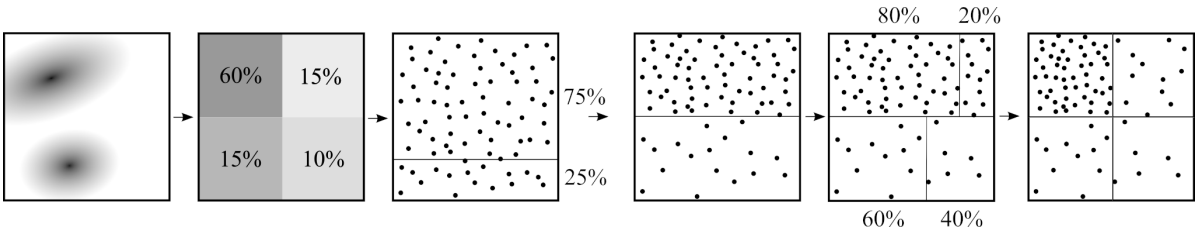


Figure 4.1: Hierarchical warping of uniformly distributed points over a 2D PDF. The PDF is mipmapped, and the relative intensities at the relevant level are used to warp points in a vertical and horizontal pass, refining the result with each mipmap level.

Hierarchical Warping still uses summed PDF values, however the main advantage over SAT is that hardware-accelerate mipmapping is faster than computing a SAT. While this improvement might seem trivial when both of these approaches are computed nearly instantly on modern graphics hardware, its optimization becomes relevant when the VPL generation pipeline steps have to be repeated multiple times for many light sources.

4.2. Constructing the SST

A Stochastic Substitute Tree or SST [18] is a type of light tree where the internal nodes store a substitute distribution of its cluster's VPLs instead of a representative light. This allows for sampling directly from the distribution when evaluating a VPL cluster, replacing sampling correlation for noise. Since VPLs are always located on surfaces, the substitute distribution is represented as the variance over a plane, which is aligned with the underlying VPLs. This plane stores a location, a normal and a 3D variance along the normal, its tangent- and its bi-tangent vectors. To build our SST in parallel, we first sort the VPL array such that geometrically similar VPLs lie close to each other in the array as well.

4.2.1. VPL Sorting

In Light Cuts [21], the light tree is built by using a greedy, bottom-up approach using a similarity metric to merge light clusters. Computing a similarity metric for every combination of VPLs is infeasible for our real-time requirement however, thus the light tree is built by exploiting spatial similarity between VPLs in the array directly. This is done by sorting the VPL array on keys constructed from a Morton encoding of the VPLs' 3D positions, encoded normals and the VPLs' ids. The id is appended to prevent any duplicate keys [18]. This sorting ensures that VPLs close to each other in 3D space will be relatively close to each other in the VPL list as well.

We sort the VPL array on the GPU using a bitonic sorting algorithm. While more efficient sorting algorithms exist when run on a single processor, bitonic sorting is one of the most efficient algorithms when run in parallel on the GPU. A GPU implementation allows for a time complexity of $O(log^2(n))$ in parallel time, where multiple cores work together on sorting the same array.

4.2.2. Merging the STT

Once the VPLs are sorted, the Stochastic Substitute Tree is created using a bottom-up approach starting with the VPLs as leaves of the tree, merging neighbouring nodes for each level up into the hierarchy. While the original SST implementation uses a more dynamically structured radix tree [9] for its internal structure, we found that using a perfect binary tree works better for our pipeline. A perfect binary tree allows for skipping a structure building phase entirely as well as several memory and logical optimizations further down the pipeline, like tighter iteration bounds for tree traversal and smaller memory footprints for nodes due to not having to store parent and/or child indices.

Tree node merging is done similarly to the original method proposed in SST [18]. A compute shader is dispatched with one thread for each pair of leaf nodes. Each thread keeps a pointer to the parent node of both of these leaf nodes and merges the two nodes into the parent node. The parent node's bounding box becomes the combined bounding box of its two children, the color is summed and the position, normal and variances are combined using the relative grayscale intensities of the child nodes as weights. The new position and variance are computed as shown in Equation (4.1) and (4.2). The normal information is merged similarly to the position, and the resulting normal is re-normalized after merging.

$$p = \frac{I_1 \cdot p_1 + I_2 \cdot p_2}{I_1 + I_2} \tag{4.1}$$

$$\sigma^2 = \frac{I_1 \cdot (\sigma_1^2 + \Delta'_{p1} \odot \Delta'_{p1}) + I_2 \cdot (\sigma_2^2 + \Delta'_{p2} \odot \Delta'_{p2})}{I_1 + I_2} \tag{4.2}$$

Here, I is the node's grayscale intensity, p is the cluster position, σ^2 is the variance, $\Delta'_{p1}, \Delta'_{p2}$ are the difference vectors from p to p_1 and p_2 in transformed cluster space. $\odot$ represents the element-wise (Hadamard) product.

As we will discuss in the next chapter (5), when traversing the tree it is useful to know how closely distributed the VPLs in the current node's cluster are to an actual flat surface. For this, two metrics are used: the variance σ_z along the plane normal, which indicates how 'flat' the distribution is along the normal, and the normal similarity metric v, which indicates how similar the surface orientations are for the cluster's internal VPLs. This metric is merged and stored in the internal node as well:

$$v = v_1 \cdot v_2 \cdot max(\langle n_1, n_2 \rangle, 0) \tag{4.3}$$

After the merged result is stored in the parent node, the thread moves up one level in the tree and repeats this process at its current node. However, while this thread has just finished merging one of the child nodes of the new current node, there is no guarantee that another thread has finished merging the other child node yet. We have to keep track of which thread visits the new parent node first, since we cannot be certain in which order the threads get executed on the GPU. For this purpose, we store an atomic counter for each node in the tree. If a thread is the first to arrive at a node, this means that the subtree under the other child has not finished merging yet. Thus, this thread updates the atomic counter and terminates. Only the second thread to visit the node can merge the information of its child nodes and continue up in the tree. This is repeated until the last thread reaches the root of the tree.

Alternatively, we could synchronize the threads after each iteration. However, with the low amount of atomic collisions between threads (max. 2 per atomic counter) the time impact of this approach is unnoticeable compared to the alternative.

5

Sampling the Light Tree

Now that we have our VPLs and SST set up, we can start rendering global illumination for our image. This chapter discusses the second phase of the pipeline (Figure 3.1), namely how we select the VPLs or VPL clusters that will be evaluated for each pixel using the SST.

First, we compute light cuts for our image. In the original Light Cuts[21], the size of the light cut is determined by the subset of light clusters which have an error bound below 2%. However, for real-time purposes such a low error bound is typically not achievable, given the amount of VPLs that can be handled at real-time rates, and the time it would require to compute such a deep cut. An error threshold larger than 2% leads to a more shallow light cut, which while easier to process results in noticeable artifacts in the final image. Real-Time Stochastic Light Cuts (RTSLC) [12] uses shallow light cuts, but pairs it with tree traversal starting from the cut nodes to reach deeper into the tree. This exchanges the sampling correlation from the lower cut for noise, which is easier to filter out. Figure 5.1 illustrates an example of this hybrid approach. We apply a similar technique, but are able to stop traversal earlier in the tree due to our light tree being an SST. In the following sections we will discuss the steps of the second and final phase in our pipeline: how light cuts are constructed, our improved tree traversal and how VPL clusters are evaluated. Here, we propose several improvements for generating light cuts and tree traversal with regards to both RTSLC and SST.

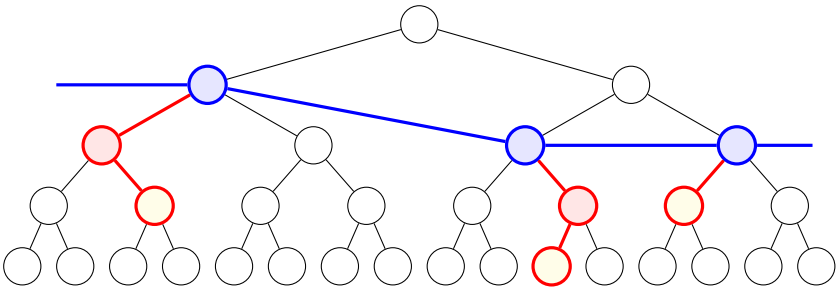


Figure 5.1: A shallow light cut (blue) combined with tree traversal (red). The clusters at the yellow colored nodes will be used for evaluation.

5.1. Shallow Light Cuts

Even with shallow light cuts, it is too costly to compute and store a light cut for every individual pixel. Therefore we compute only a single light cut for every $k \times k$ pixel block. Starting with the root node of the light tree, a cut is generated by iteratively replacing nodes by their child nodes until a maximum cut size is reached. The order in which the nodes are split is important, which is further discussed in the next subsection (5.1.1). If a child node's bounding box is completely behind the evaluated surface, this would mean that it would have zero contribution and is therefore not added to the cut.

In our case, we set the maximum light cut size to the per-pixel sample budget. Each cut node returns exactly 1 VPL cluster after traversal, and thus by evaluating an entire shallow cut the amount of clusters returned equals our per-pixel sample budget exactly. For example, Figure 5.1 illustrates a light cut for a sample

budget of 3 samples per pixel. A pseudocode representation of the stochastic blending algorithm is shown in Figure 5.3.

5.1.1. Splitting Order

The order in which the nodes are split determines which nodes get to be added before the cut is full, which has a significant impact on the quality of the cut. We determine a priority value for each node, and at each iteration select the node with the highest value for splitting.

We would like to sample more clusters from subtrees with a high expected contribution, which is done by including more nodes of these subtrees in our light cut. We split up the nodes with the highest expected contribution first. This makes for a more even distribution in subtree contribution of light cut nodes and a lowered variance of evaluated VPL clusters returned after traversal. We define our metric for expected cluster contribution in the next subsection, 5.1.2.

5.1.2. Expected Cluster Contribution

In Light Cuts [21], a cluster's illumination L_C and its approximation for shading point x, direction ω and representative point j are defined as:

$$\begin{aligned} L_C(x,\omega) &= \sum_{i \in C} M_i(x,\omega) \cdot G_i(x) \cdot V_i(x) \cdot I_i \\ &\approx M_j(x,\omega) \cdot G_j(x) \cdot V_j(x) \cdot \sum_{i \in C} I_i \end{aligned} \tag{5.1}$$

Here, $M_i(x,\omega)$ is the material term, $G_i(x)$ is the geometric term, $V_i(x)$ is the visibility term and I_i is the intensity term.

We base the definition of a cluster's expected contribution on this approximation. Like in the original work, $M_j(x,\omega)$ is defined as the integral of the material's BRDF within the solid angle of the cluster, $V_j(x)$ is set to 1 and $\sum_{i \in C} I_i$ is equal to the sum of intensities of the cluster's internal VPLs.

Light Cuts uses this approximation for determining the cluster error. However, we are interested in the cluster's average expected contribution instead. For this purpose, we define the geometric term $G_j(x)$ differently compared with the original. While its quadratic falloff term stays the same, we define the cosine term as the average clamped cosine instead of the upper bound cosine with regards to the cluster bounding box. Clamping the cosine ensures that we only count the part of the bounding box that is in front of the evaluated surface. Here, $G_j(x)$ is computed by averaging the clamped dot products of x's normal vector n_x and the normalized vectors directed from x's position p_x at the corners c of the cluster bounding box C_{BB}:

$$G_i(x) = \frac{1}{8} \sum_{c \in C_{BB}} max(\langle n_x, \frac{c - p_x}{||c - p_x||} \rangle, 0) \tag{5.2}$$

During light cut generation, we base this expected cluster contribution on the geometric properties at the center point of the $k \times k$ pixel block of the cut we are generating. These geometric properties are retrieved from the camera G-Buffer.

5.1.3. Light Cut Blending

As described in this section 5.1, we compute a single light cut for each $k \times k$ pixel block. However, if all the pixels in a pixel block use the same light cut, this results in transition artifacts at the borders of the blocks. Real-Time Stochastic Light Cuts [12] solve this by using Interleaved Sampling [17, 20], where $m \times m$ sub-blocks within the $k \times k$ block use only a subset of the nodes in its light cut. This lowers the amount of VPL clusters evaluated for a single pixel however. To correct for this, a deeper light cut with more nodes is necessary, which is more costly to compute. Therefore, instead of taking a subset of a single light cut, we choose to blend the light cuts of surrounding the pixel patches instead. This allows us to not have to construct light cuts larger than strictly necessary, and provides a smooth transition between pixel blocks.

We propose a fast stochastic light cut blending strategy. Using the cuts C_k from the four pixel blocks surrounding pixel x and with maximum cut size n, we iterate over node index $i = 1..n$. For each iteration, we stochastically select one of the cuts and add the node of the chosen cut C_s at index i to our blended cut. The

probabilities for each cut are based on the bilinear weights between the center of the cut's pixel block and point x.

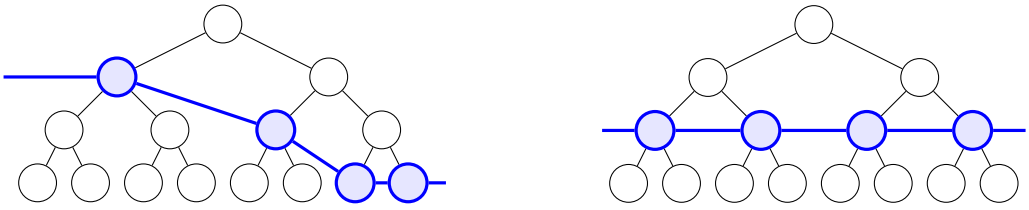
Furthermore, if the geometric properties at the light cut's pixel block center are too dissimilar from the geometry at x, the cut is not included in the blending process for this pixel. This similarity metric between two surface points x and y is defined as:

$$S(x,y) = \begin{cases} 1 & \text{if } \langle n_x, n_y \rangle > n_{min} \,\&\, \\ & dist(p_x, p_y) < d_{max} \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

where n is the surface normal, p is a surface point's position in world space, n_{min} is the minimum allowed normal dot product and d_{max} is the maximum allowed distance between the two surface points.

5.1.4. Mock Cuts

Thin or detailed geometry might cause all the four cuts to be discarded due to bad geometric similarity. Evaluating x with any of these light cuts would give a clearly suboptimal result. However, recomputing complete lights cut for individual pixels is too costly. We propose a faster solution, by generating a 'mock cut' for the current pixel. This cut consists of all the nodes on the lowest possible level in the tree where the width at that level is no larger than the maximum per-pixel sample budget. This gives us a cheap cut as deep as possible in the light tree, while not exceeding our per-pixel sample budget. An example is illustrated in Figure 5.2 below.



(a) (b)

Figure 5.2: A light cut containing 4 nodes (a) and its mock cut (b), spanning all nodes on node level $floor(log_2(4)) = 2$.

While this is most likely not an optimal light cut for any surface, it is computed instantly and is a better fallback than other methods like averaging the unsuitable cuts or ignoring the cuts and starting traversal from the tree root.

Algorithm 1 Stochastic Light Cut Blending

```
for k ← 0...3 do
    p_k ← bilinear(C_k, x) * geometric_similarity(C_k, x)
p_sum ← Σ_{k=0}^{3} p_k
if p_sum ≠ 0 then
    for k ← 0...3 do
        p_k ← p_k / p_sum
    for i ← 0...n do
        C_s ← stochastic_select(C, p)
        evaluate(C_{s,i}, x)
else
    maxlvl ← floor(log2(n))
    C_m ← nodes_at_tree_level(maxlvl)
    for i ← 0...2^maxlvl do
        evaluate(C_{m,i}, x)
```

Figure 5.3: Stochastic Light Cut Blending: Weights for each cut are computed. If any weight is non-zero, weights are normalized and the cuts are blended. If all weights are zero, we evaluate the mock cut.

5.2. Tree Traversal

Traversal is done in a similar manner to SLC [12], with the main difference being that we use our expected contribution metric (section 5.1.2) as the geometric term instead of using the upper bound cosine. For each pixel, we traverse the tree multiple times starting from each of its light cut nodes once. During traversal, for each node we stochastically determine whether to continue traversal with either its left or right child node. The probability p to visit either child node is based on their relative expected contribution to the evaluated pixel, expressed as the weight w of each child node.

$$p_{left} = \frac{w_{left}}{w_{left} + w_{right}}, \tag{5.4}$$

$$w_C(x,\omega) = M_j(x,\omega) \cdot G_j(x) \cdot V_j(x) \cdot \sum_{i \in C} I_i \tag{5.5}$$

If a node is reached where $w_{left} = w_{right} = 0$, neither child node will have any contribution. Because backtracking is not feasible in a GPU shader, traversal is terminated and no cluster is evaluated.

For each traversal step, we check if the current node is suitable for sampling. If we would still like to guarantee the 2% error threshold, we could compute the upper bound error for each step and return the current cluster once its error bound is below the threshold. However, the error bound is too expensive to calculate for each step in the traversal, and would slow down our solution significantly. This is where the second advantage of a Stochastic Substitute Tree comes in to play, next to eliminating sampling correlation. The previous chapter (4.2) discusses the joint conditions for determining how close the node's cluster distribution is to a flat plane, using the variance σ_z along the plane normal and the normal similarity metric ν (4.3). If these values cross a certain user-defined threshold, the cluster is deemed to be of sufficient quality for sampling. This means traversal stops early and the node is evaluated. In our experiments, we noticed that $\nu_{min} = 0.7$ and $\sigma_{max} = 0.3$ gives a good balance between quality and speed.

$$\nu \geq \nu_{min} \wedge \sigma_z \leq \sigma_{max}. \tag{5.6}$$

Equation 5.6: Joint conditions for substitute distribution suitable for sampling.

This joint condition is computed at the time of building our SST and thus can be retrieved instantly during traversal, instead of having to be recomputed at each individual traversal step.

5.3. Cluster Evaluation

Clusters are evaluated using the approximation as described in Light Cuts [21] (Equation 5.1). To select a representative point for the cluster, we retrieve a 3D point from the node's substitute distribution. First we sample a normal distribution with the cluster's variance and mean position. The distribution is aligned with the substitute distribution's plane and its normal, thus the sampled point is first transformed back into world space, and clamped by the cluster's bounding box. The cluster is evaluated as a point light at this location with its color as the summed colors of all its internal VPLs. Visibility is evaluated using hardware-accelerated ray-tracing [5], where we perform a single shadow ray check between the surface point and the chosen representative point of the cluster.

6

Implementation Details

Our implementation was written from scratch in C++ using the Vulkan API. The Vulkan RTX extension was used for visibility checks.

Surface Sampling from the G-Buffer.
In the original BRSM implementation [15], surfaces for pVPL contribution estimation are sampled from pixels surrounding the projected pVPL position in the G-Buffer, with a probability falloff of $1/x^2$ to importance sample according to the light's quadratic falloff. We compute these G-Buffer sampling locations slightly differently. Instead of projecting a point and sampling from the nearby pixels, we sample a 3D point around the 3D pVPL coordinate, and project this point into the G-Buffer. The point is sampled with a random direction and random radius, where the distribution of the radius is based on a $1/x^2$ distribution in order to emulate the quadratic light falloff. To avoid singularities, the distribution is capped to 1.

BRSM Filtering.
Computing pVPL contributions with a low amount of surfaces results in a PDF with high amounts of noise. We filter the texture that stores pVPL contribution values using an edge-aware Gaussian kernel, using the same geometric similarity function as described in Equation 5.3. This results in a smoother PDF after normalization, while edges are preserved in the blurred texture.

Hammersley Distribution.
During sampling of the VPLs from the pVPLs (4.1.2), we initialize Hierarchical Warping (HW) with a randomized Hammersley distribution instead of a completely uniform random set of points, much like the original HW implementation [2]. This distribution prevents the initial points from 'clumping' together, and ensures a more even spread of VPLs across surfaces, which in turn improves sampling performance.

Multiple Lights.
In the case of multiple light sources the VPL generation process is repeated for each light, which uses constant memory. Ideally, each light would contribute an amount of VPLs relative to its contribution to the final image. While we did consider a more effective solution of analyzing all pVPLs of all lights together, the amount of memory that is needed for this outweighs the potential benefit. Therefore, in the case of multiple lights we just use the light source's brightness value relative to other lights as the amount of VPLs it should contribute to the global VPL array.

VPL sorting.
Each iteration of the sorting algorithm requires many data swaps, thus copying around complete VPL objects can quickly become a memory I/O bottleneck for the sorting algorithm. To circumvent this issue, an array of only the sorting keys paired with their respective VPL index are sorted. Sorting this array requires significantly less memory operations, since we only have to swap a single 64 bit key. Once the index sorting is complete, the actual VPLs are swapped within the list according to their new indices from the sorted index array.

Upsampling.
Due to its low-frequency nature, indirect illumination is a good candidate for being upscaled from a lower resolution rendering. We render indirect illumination at half the resolution and upsample it to full resolution using joint bilateral upsampling. We use a bilinear filter in combination with the geometric similarity metric defined in Equation 5.3. Rendering at a lower resolution reduces the amount of necessary cluster evaluations and thus the amount of visibility evaluations significantly. Since this is the most time consuming part of our pipeline, it brings major improvements for frame times.

7

Results

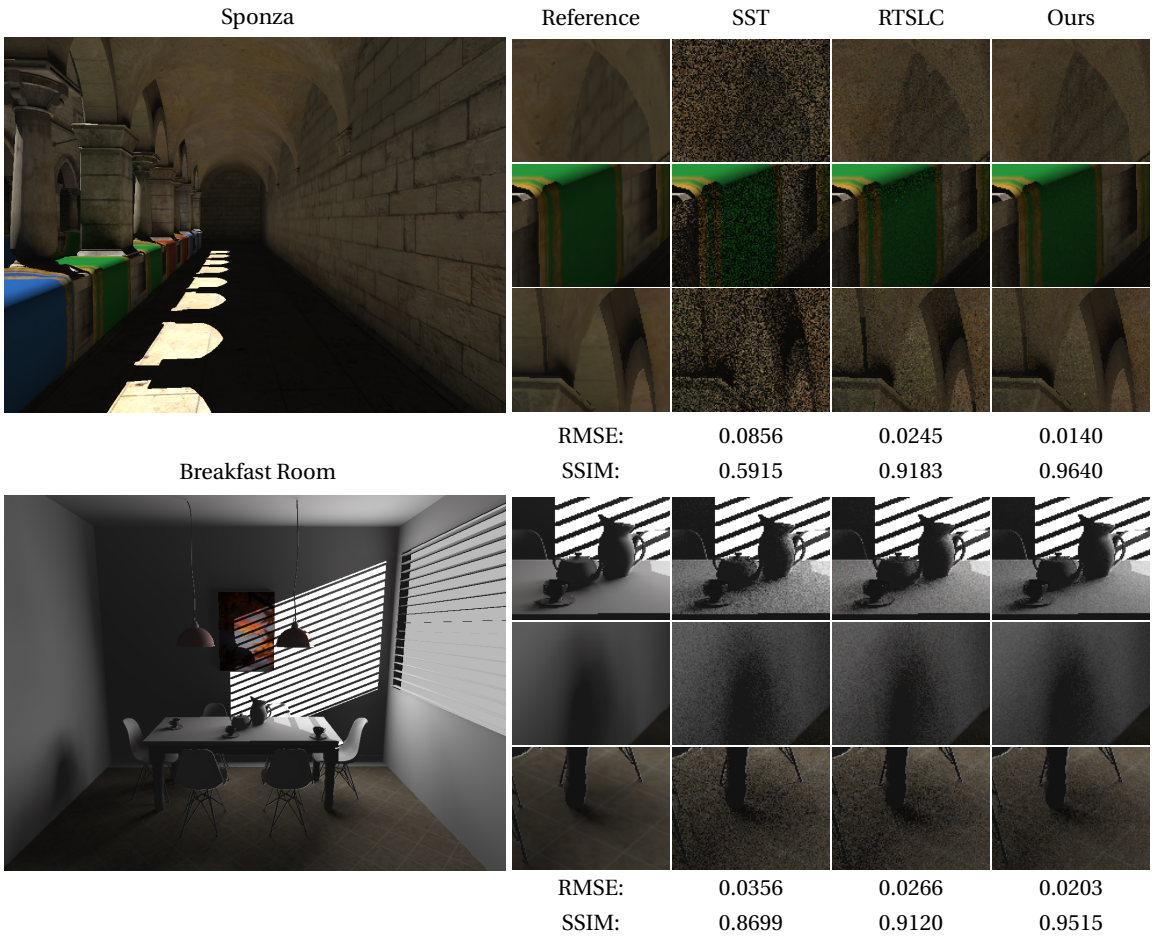


Figure 7.1: Equal-time comparisons of SST, RTSLC and our proposed method against the ground truth.

We evaluate our proposed method and the individual improvements for varying parameters. Our sampling strategy described in Chapter 5 is compared with Stochastic Substitute Trees (STT) [18] and Real-Time Stochastic Lightcuts(RTSLC) [12] against the ground truth. The ground truth is computed by evaluating the complete array of VPLs. Note that while SST is designed for low sample rates using a denoising step afterwards, we do not apply denoising or upscaling to any of the results, in order to accurately evaluate and compare the sampling strategies.

Our method is evaluated on two scenes, the Sponza scene and the Breakfast Room scene [13]. Each image is rendered at a 1280×720 native resolution, without GI upscaling (Chapter 6). The entire pipeline is executed

on the GPU. Implementation is done in C++ using the Vulkan graphics API, with hardware-accelerated ray tracing for visibility checks. All results are generated on a NVidia RTX 2060 Super GPU. For our method we use a light cut block size of 8×8 pixels. Each scene contains a single directional light source with a shadow map (and BRSM) resolution of 512×512. Unless stated otherwise, we render both scenes with a VPL count of 4096. The parameters for early traversal termination (5.6) are set to $\upsilon_{min} = 0.7$ and $\sigma_{max} = 0.3$. Decreasing υ_{min} or increasing σ_{max} leads to clusters being selected that are not flat enough, leading to increased possibility of points being sampled inside geometry.

Table 7.1 shows the individual timings of rendering both the Sponza scene and the Breakfast Room scene. While we do not show it here, about three-quarters of the time of the VPL generation is spent filtering the pVPL values, and about 80% of the time in rendering global illumination is used for visibility checks using hardware-accelerated ray tracing. The rest is used mainly by traversing the light tree.

time(ms)	Render G-Buffer	Render SM	Generate VPLs	Sort VPLs	Build SST	Compute Light Cuts	Render GI	Render Direct Light
Sponza	0.40	0.20	0.74	0.40	0.02	0.36	15.00	0.39
Breakfast Room	0.87	0.53	0.42	0.39	0.02	0.34	20.85	0.63

Table 7.1: Individual timings for both Sponza and Breakfast Room.

7.1. Comparison of Sampling Strategy

Figure 7.1 shows how STT, RTSLC and our proposed method compare against the ground truth. Here, we only evaluate the sampling strategy of each method. For comparison we look at both the Root Mean Square Error (RMSE) and the Structural Similarity Index (SSIM). The light tree is computed identically for each method, and light cuts and light cut blending are the same for RTSLC and our proposed method. For the Sponza scene, STT, RTSLC and our method evaluate 14, 13 and 32 samples for an average frame time of 17.88ms, 18.02ms and 17.52ms respectively. For the Breakfast Room scene, this was 16, 16 and 32 samples with an average frame time of 24.45ms, 24.03ms and 24.32ms respectively. Our algorithm runs faster, thus allowing for more samples evaluated in the same amount of time. A main contributor to this fact is that our proposed method spends less time traversing the light tree before finding suitable clusters to sample from, as shown in Figure 7.2.

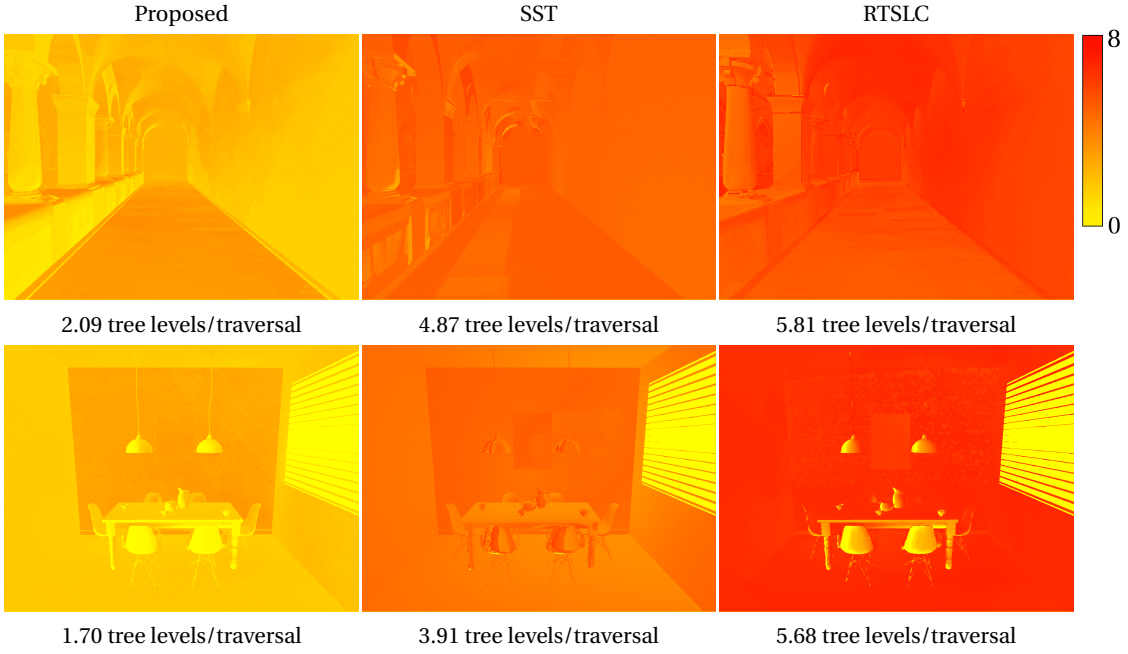


Figure 7.2: Average amount of traversal steps per pixel for a tree with 12 levels (2^{12} VPLs). Less tree levels visited per traversal results in faster retrieval of VPL clusters.

Here we show the average amount of tree traversal steps for each pixel for all three methods. As we can see, SST and RTSLC traverse around double and triple the amount of tree levels per pixel on average respectively. As previously illustrated in Figure 3.2, SST starts traversal at the root of the light tree and stops in the tree, while RTSLC starts in the tree but only stops at its leaf nodes. The combination of both, starting traversal in the tree and stopping in the tree saves many traversal steps when compared to the other two methods, which saves time and thus allows us to sample more clusters in equal time.

7.1.1. Frame time and Root Mean Squared Error

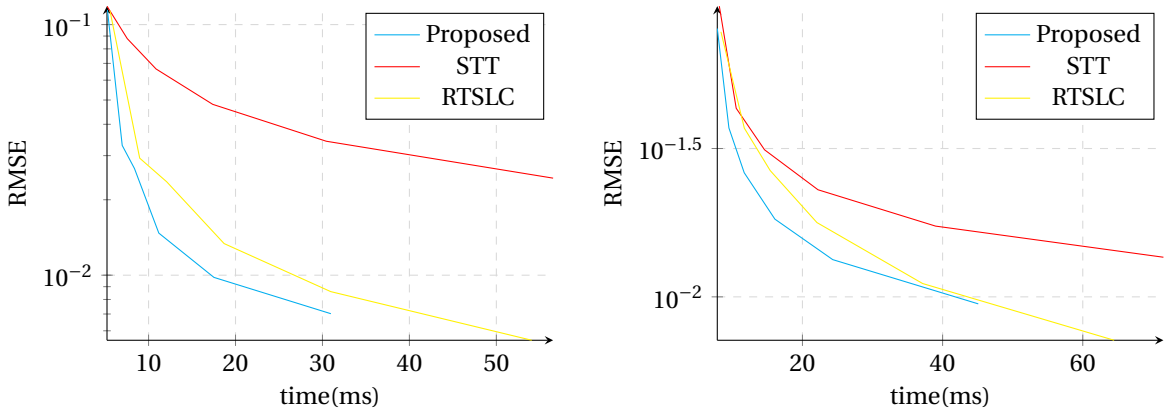


Figure 7.3: RMSE vs. frame time for Sponza (left) and Breakfast Room (right).

To show how RMSE scales with frame time, we evaluate both RMSE and frame time for an increasing number of samples per pixel. We use the same light cut, light cut blocks and blending for all methods, and only vary the strategy used to find clusters in the light tree. The results are illustrated in Figure 7.3. As we can see, our proposed method outperforms both similar methods, having a lower RMSE for the same frame times for all tested cases.

7.2. Light Cut Blending

Light cut blending (section 5.1.3) allows for light cuts to be shared by blocks of pixels without having transitioning effects at the borders, as well as discarding unsuitable light cuts based on geometric similarity. Figure 7.4 illustrates the importance of light cut blending at edges and fine geometry.

Figure 7.4: No light cut blending (top) vs. light cut blending (bottom)

By sharing light cuts instead of computing them for each pixel, we introduce additional error to our image.

Figure 7.5 illustrates the effect of light cut block size on both the RMSE and frame time:

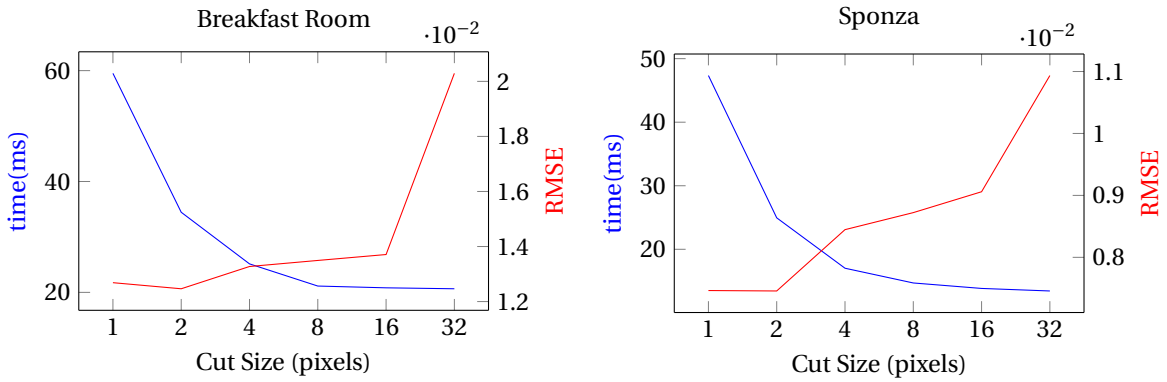


Figure 7.5: RMSE and frame times for different Light Cut block sizes for the Breakfast room scene and Sponza scene respectively

These plots show that by sharing light cuts for 8×8 sized pixel patches, frame time is nearly a third of when a light cut is computed for each pixel individually, with only a small effect on the RMSE. Visually, the light cut blocks are nearly unnoticeable until sizes above 16×16 pixels, mainly thanks to the fact that we have a good fallback for when there is no suitable surrounding light cut for geometry in-between pixel blocks.

7.3. Mock Cuts

It might occur that none of the four light cuts of the pixel blocks surrounding a pixel are suitable for its underlying geometry. In this case, we create a new, simple cut as substitute for a fully computed light cut (5.1.4). Figure 7.7 shows the mock cut and its alternatives. Simply averaging the unsuitable light cuts creates dark patches on the blinds, seen on the lower and upper part of the window. The mock cut, traversal from root and recomputed light cut all effectively solve this issue, however the mock cut is the only solution that has no impact on computation time.

Average surrounding cuts (24.23ms)	Mock cut (24.15ms)	Traversal from root (25.58ms)	Recompute light cut (28.14m)s

Figure 7.6: Different fallback solutions for unsuitable surrounding light cuts

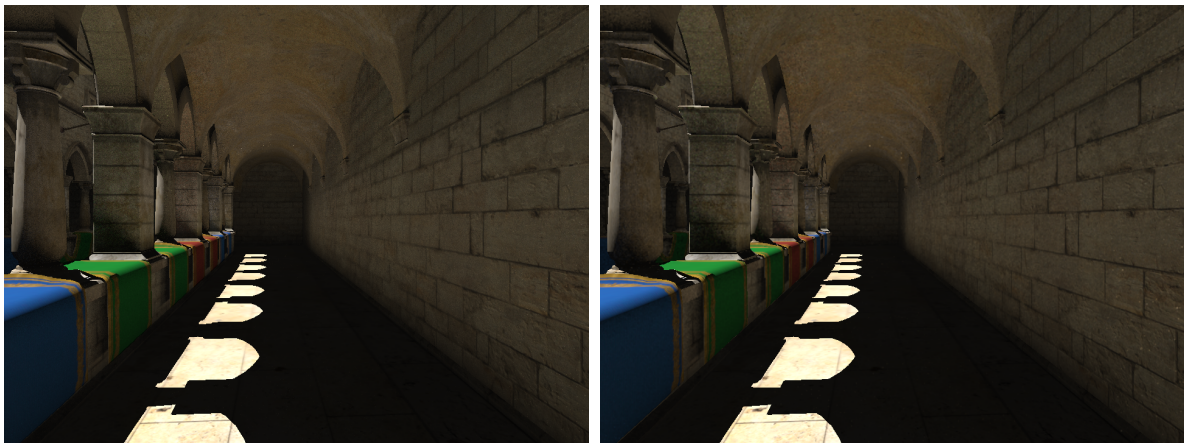
7.4. Upsampling

For better frame rates, we can render global illumination at a lower resolution, and upsample it using joint bilateral upsampling (6). The table below shows the effect on RMSE and timings for the Sponza scene:

Setting	RMSE	SSIM	time(ms)
1280x720, full GI resolution	0.0137	0.9453	17.52ms
1280x720, half GI resolution	0.0140	0.9547	6.96ms
1920x1080, full GI resolution	0.0132	0.9677	28.64ms
1920x1080, half GI resolution	0.0133	0.9825	10.95ms

Table 7.2: RMSE and timings for bilateral joint upsampling.

As we can see, the effect on RMSE is negligible while frame times are nearly a third by rendering global illumination at half the resolution. Interestingly, the SSIM increases when upsampling an image. This might be due to the blurring effect of bilateral joint upsampling, where sharp noise gets blurred out. Visually, he effect of upsampling is slightly noticeable, especially on surfaces where direct light is absent. Due to bilateral joint upsampling, noise can turn into more noticeable 'blobs' whereas at native resolution noise has a grain-effect (Figure 7.7). At higher frame rates this noise is less noticeable however, and applying a temporal filter would be faster at lower resolutions as well.



full GI resolution (17.52ms) half GI resolution, upsampled (6.96ms)

Figure 7.7: Global illumination rendered at full resolution and its upsampled, half-resolution counterpart.

8

Discussion

Our evaluation (Chapter 7) shows that our proposed method outperforms similar methods in all tested cases. As expected, the combination of SST with light cuts and tree traversal allows us to find VPL clusters using less traversal steps, and thus sample more clusters in the same amount of time. While RTSLC comes very close in terms of RMSE for a similar amount of samples, our method is faster since it does not have to traverse the light tree down to its leaves for every sample. While we do not use a radix tree like in the original SST implementation [18], the sampling strategy of STT performs the worst of all three methods in this context. While this method is easier to implement, it performs relatively poorly when the tree is sampled multiple times for a single pixel. We expect this is due to the fact that at higher levels of the light tree information is too ambiguous for intelligent traversal decisions, thus disturbing traversal at the higher levels of the tree. Using a shallow light cut, samples are more evenly distributed between subtrees which allows for a more even exploration of the light tree.

Results show that light cut blending allows us to share light cuts between pixel blocks without noticeable artifacts, using instantly generated mock cuts as fallback. We show that sharing light cuts between pixels and rendering global illumination at a lower resolution result in significantly improved frame rates, while visual quality is not affected much.

There are however some limitations to our method. First of all, our current method does not support glossy or specular BRDFs. This can be added to our model the same way as in SST [18], by storing the glossy component and reflective direction for VPLs and using an altered traversal algorithm to find glossy VPL clusters during sampling.

Our light cut blending strategy does not necessarily produce valid light cuts, and may cause duplicate sampling of VPL clusters. Other strategies for interpolating light cuts that result in a valid light cut do exist [14], however we did not find a need for this in our use case. Due to the small scale of our light cut blocks and the fairly homogeneous light cut constructions between neighboring pixel blocks, stochastic blending works well enough to remove any transitioning artifacts at pixel block borders.

Second-bounce VPLs account for a large part of the indirect illumination, but its limitations are noticeable when testing in poorly-lit or mostly occluded parts of a scene. Its alternative, path traced VPLs, reach deeper into the scene but take more time to distribute. This is a trade-off between speed and lighting quality, which may differ between applications.

Even with these limitations, our method provides a good and efficient approximation of second-bounce illumination at real-time frame rates, which can be included in the pipeline of any interactive application.

9

Conclusion

In this work we present a solution for real-time second bounce diffuse global illumination for fully dynamic scenes. By combining efficient VPL generation using Bidirectional Reflective Shadow Maps (BRSM) with a hybrid sampling method that uses both shallow light cuts and light tree traversal, we are able to sample VPL clusters more efficiently than our predecessors, leading to lower noise in our final image at similar frame times. In summary, our main contributions include:

- Improved VPL sampling from Bidirectional Reflective Shadow Maps by utilizing hardware-accelerated mipmapping.
- A novel combination of Stochastic Substitute trees with Stochastic Lightcuts, which utilizes shallow light cuts in combination with tree traversal and SST for quick sampling of VPL clusters.
- A new geometric term for evaluating a VPL cluster's average expected contribution, used both in tree traversal and shallow light cut generation.
- A stochastic light cut blending strategy for seamless cut sharing between pixels.
- Mock cuts, an effective fallback for light cut sharing in case none of the surrounding light cuts are suitable for a pixel's underlying geometry.

We show how these techniques combined in a single streamlined pipeline and discuss its strengths as well as its limitations. The new method allows for quicker evaluation of second-bounce global illumination than its competitors, with little post-processing filtering necessary.

10

Future work

This paper presents a robust method for dynamic real-time global illumination. While our results are better compared to similar methods, there are several opportunities for further research.

First of all, it would be beneficial to be able to reuse parts of the generated structures between frames. For example, some VPLs could be reused or warped to fit a new distribution instead of being recomputed every frame. This would lead to better temporal stability, as well as save computation time.

Our current method is restricted to second-bounce global illumination only. One could generate or propagate VPLs generated from a BRSM, for example by adding the possibility for path tracing starting from primary VPL locations, or even using screen-space techniques to place new VPLs at locations with many primary VPLs in close proximity.

In our method, light cut blocks have a constant size, something which we noticed was not always necessary. A dynamic grid would allow for a dynamic light cut resolution throughout the scene, saving computation time at large constant surfaces or increasing detail at fine geometry and edges. For this purpose, stochastic blending might not be suitable, and the strategy for interpolating light cuts between light cut points would have to be re-evaluated.

Finally, many applications assume a constant sample rate for each pixel in the image. We notice that while rendering global illumination, its contribution to areas which are already lit directly is almost completely unnoticeable. By recognizing brightly lit areas before rendering global illumination, we could save time by assigning less samples to these areas, resulting in more time left for poorly lit areas. Research can be done in optimizing the expected signal to noise ratio between pixels by using a dynamic per-pixel sample budget, which could lead to significant improvements in image quality and/or frame times.

Bibliography

[1] K. Breitung. “Ripley, B.D., Stochastic simulation”. In: *Statistical Papers* 30 (1 1989). ISSN: 0932-5026. DOI: 10.1007/bf02924321.

[2] Petrik Clarberg et al. “Wavelet importance sampling”. In: *ACM Transactions on Graphics* 24 (3 2005). ISSN: 0730-0301. DOI: 10.1145/1073204.1073328.

[3] Carsten Dachsbacher and Marc Stamminger. “Reflective shadow maps”. In: 2005. DOI: 10.1145/1053427.1053460.

[4] Tim Foley and Jeremy Sugerman. “KD-tree acceleration structures for a GPU raytracer”. In: vol. 2005. 2005. DOI: 10.1145/1071866.1071869.

[5] Eric Haines and Tomas Akenine-Möller. *Ray tracing gems: High-quality and real-time rendering with DXR and other APIs*. 2019. DOI: 10.1007/978-1-4842-4427-2.

[6] Justin Hensley et al. “Fast summed-area table generation and its applications”. In: *Computer Graphics Forum* 24 (3 2005). ISSN: 14678659. DOI: 10.1111/j.1467-8659.2005.00880.x.

[7] Matthias Holländer et al. “ManyLoDs: Parallel many-view level-of-detail selection for real-time global illumination”. In: *Computer Graphics Forum* 30 (4 2011). ISSN: 14678659. DOI: 10.1111/j.1467-8659.2011.01982.x.

[8] James T. Kajiya. “RENDERING EQUATION.” In: *Computer Graphics (ACM)* 20 (4 1986). ISSN: 00978930. DOI: 10.1145/15886.15902.

[9] Tero Karras. “Maximizing parallelism in the construction of bvhs, octrees, and k-d trees”. In: 2012. DOI: 10.2312/EGGH/HPG12/033-037.

[10] Alexander Keller. “Instant radiosity”. In: 1997. DOI: 10.1145/258734.258769.

[11] Eric P. Lafortune and Yves D. Willems. “Bi-Directional Path Tracing”. In: *Proc. SIGGRAPH* (1993). ISSN: 1098-6596.

[12] Daqi Lin and Cem Yuksel. “Real-Time Stochastic Lightcuts”. In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3 (1 2020). DOI: 10.1145/3384543.

[13] Morgan McGuire. *Computer Graphics Archive*. https://casual-effects.com/data. July 2017. URL: https://casual-effects.com/data.

[14] Hauke Rehfeld and Carsten Dachsbacher. *Lightcut interpolation*. 2016.

[15] Tobias Ritschel et al. “Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes”. In: *Computer Graphics Forum* 30 (8 2011). ISSN: 14678659. DOI: 10.1111/j.1467-8659.2011.01998.x.

[16] Takafumi Saito and Tokiichiro Takahashi. “Comprehensible rendering of 3-D shapes”. In: *Computer Graphics (ACM)* 24 (4 1990). ISSN: 00978930. DOI: 10.1145/97880.97901.

[17] B. Segovia et al. “Non-interleaved deferred shading of interleaved sample patterns”. In: 2006. DOI: 10.1145/1283900.1283909.

[18] Wolfgang Tatzgern et al. “Stochastic Substitute Trees for Real-Time Global Illumination”. In: 2020. DOI: 10.1145/3384382.3384521.

[19] Eric Veach and Leonidas J. Guibas. “Optimally combining sampling techniques for Monte Carlo rendering”. In: 1995. DOI: 10.1145/218380.218498.

[20] Ingo Wald et al. “Interactive global illumination using fast ray tracing”. In: 2002.

[21] Bruce Walter et al. “Lightcuts: A scalable approach to illumination”. In: vol. 24. 2005. DOI: 10.1145/1073204.1073318.

[22] Bruce Walter et al. “Multidimensional lightcuts”. In: 2006. DOI: 10.1145/1179352.1141997.

[23] Cem Yuksel. “Stochastic Lightcuts for Sampling Many Lights”. In: *IEEE Transactions on Visualization and Computer Graphics* (2020). ISSN: 1077-2626. DOI: 10.1109/tvcg.2020.3001271.