High Performance Graphics (2016)
Ulf Assarsson and Warren Hunt (Editors)

Photon Splatting Using a View-Sample Cluster Hierarchy

P. Moreau[1], E. Sintorn[2], V. Kämpe[2], U. Assarsson[2] and M. Doggett[1]

[1]Lund University, Sweden
[2]Chalmers University of Technology, Sweden

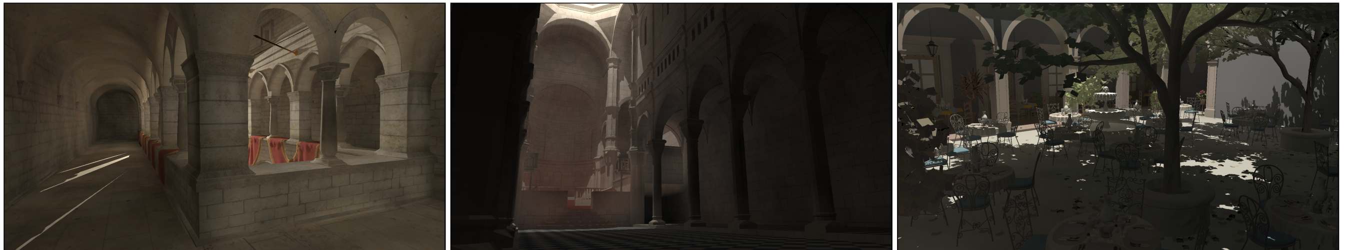


Figure 1: Views from Sponza, Sibenik, and San Miguel, rendered using our method with 200k photons and radius set up to produce a smooth image. The time taken to splat the photons is (left to right): 14 ms, 16 ms and 17 ms; full frame time is: 35 ms, 33 ms and 48 ms. The scenes were rendered at 1080p on an NVIDIA Titan X.

Abstract

Splatting photons onto primary view samples, rather than gathering from a photon acceleration structure, can be a more efficient approach to evaluating the photon-density estimate in interactive applications, where the number of photons is often low compared to the number of view samples. Most photon splatting approaches struggle with large photon radii or high resolutions due to overdraw and insufficient culling. In this paper, we show how dynamic real-time diffuse interreflection can be achieved by using a full 3D acceleration structure built over the view samples and then splatting photons onto the view samples by traversing this data structure. Full dynamic lighting and scenes are possible by tracing and splatting photons, and rebuilding the acceleration structure every frame. We show that the number of view-sample/photon tests can be significantly reduced and suggest further culling techniques based on the normal cone of each node in the hierarchy. Finally, we present an approximate variant of our algorithm where photon traversal is stopped at a fixed level of our hierarchy, and the incoming radiance is accumulated per node and direction, rather than per view sample. This improves performance significantly with little visible degradation of quality.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Global illumination algorithms attempt to simulate, in a physically based manner, how light is transported through a virtual scene. The goal is to estimate the radiance that is incident to each pixel of the image, which will be an aggregate of all possible light-transport paths that end up intersecting that pixel. Light-transport paths originate from virtual light sources and will undergo any number of reflections (where energy may be splatted or absorbed). There are several textbooks that discuss the common theory of global-illumination algorithms and the many different algorithms that numerically solve the underlying equations to produce photo-realistic images (see, e.g., [PH10, DBBS06]). These algorithms typically favour correctness over computation speed and can take minutes to hours to compute an image.

On the other end of the spectra are real-time, or interactive, global illumination algorithms. Depending on the use case, these algorithms have between one and a few hundred milliseconds to produce a plausible image. There exist a very large number of such algorithms, each with its own limitations and benefits. In this paper, we will explore one class of algorithms, commonly referred to as *photon splatting*, and suggest several novel improvements.

Photon splatting is a variant of the *photon-mapping* [Jen01] class

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.

of algorithms. In the photon-splatting variant of algorithms, photons are traced from the emitters and directly visualized, by accumulating the contribution to an outgoing radiance of each photon against the *view samples* (i.e. primary ray hit points) that it affects. Thus, a photon is given some artificial *region of influence*, usually a sphere or an ellipsoid, and the main difference between splatting algorithms is how this geometric shape is intersected with all view samples.

We propose a novel approach to real-time photon splatting for global illumination. For each frame, a 3D acceleration structure is built over the current view samples. We use the *view-sample cluster hierarchy* [SKOA14], which builds a hierarchy of clusters of view samples based on screen tiles. We show that this technique can be used to efficiently cull large clusters of view samples that lie outside of a photon's region of influence before considering individual view samples. Additionally, we show that having the view samples arranged in groups of clusters (that all lie close to each other), improves performance by stopping traversal as soon as a photon contains an entire node.

Another benefit is that under the assumption that nearby view samples are likely to have similar normals, we can utilize optimizations that cull photons intersecting a node, but originating from a direction such that they will not affect any of the contained view samples. If we accept the limitation that a photon affects all view samples equally (i.e. the surfaces have a constant BRDF and no distance-based smoothing kernel is used), we can also stop traversal as soon as all view samples contained in a node have normals such that they will be affected by the photon.

In all, these improvements result in an algorithm that can render a large number of photons with sufficiently large radii to produce smooth results at high frame rates. To further push performance in the direction where it could be used as a global-illumination solution for, e.g., video games, we propose a more approximate solution. Instead of testing each photon against each individual view sample, we accumulate *directional* flux for the hierarchy nodes. In the fastest version of this algorithm, where a leaf node at most contains 32×32 view samples, we obtain very high frame rates without significant loss in quality.

2. Related Work

There is a vast body of work concerned with interactive global illumination, and we refer the reader to an excellent recent survey [RDGK12] for a more complete introduction. In this section, we will briefly discuss only the previous work that is most relevant to our proposed method.

Traditional photon mapping, where a kd-tree is built over the photons to accelerate the *gathering* of nearby photons, has been accelerated on the GPU [ZHWG08, LSP*12], but has not been shown to be practical for real-time scenarios where the light source moves. Instead, it is common to *splat* photons onto the view samples. McGuire and Luebke [ML09] find the first bounce from the light by rasterization and trace the remainder of the path on the CPU. Photons are splatted by rendering spheres that enclose the photons' influence regions. Mara et al. [MML13] explore a number of faster approaches to splatting, which will be detailed in Section 3.2.

A large portion of recent work on interactive global illumination stems from the concept of Instant Radiosity [Kel97], which is similar to photon splatting in that particles are traced from the light source and stored at each bounce. Unlike photon splatting, these particles are then treated as *Virtual Point Lights* and store the outgoing radiosity (rather than incoming flux). The scene is then lit from all such VPLs. Reflective Shadow Maps [DS05] is a GPU based variant of this, in which the first bounce from the light source is calculated using rasterization. For each view sample, a stochastic subset of the generated VPLs is gathered for shading. Nichols and Wyman [NW09] suggest an alternative approach where the VPLs are instead scattered onto a multi-resolution buffer. These techniques do not consider visibility between VPL and view sample. This is addressed in a method called Imperfect Shadow Maps [RGK*08, REH*11], where highly approximative shadow maps are calculated in real time for each VPL.

We are only aware of a few real-time global-illumination methods that are production proven. One notable example is Voxel Cone Tracing [CNS*11] in which the scene is voxelized to a sparse voxel octree and a cone-marching algorithm gathers approximate incoming radiance. Another is Cascaded Light Propagation Volumes [KD10], where the light field is stored using spherical harmonics in a coarse discretization of the view frustum. Mara et al. [MMNL14] present a fast approximation to Global Illumination using deep G-Buffers, but since the illumination is view frustrum based, it does not produce a photon tracing of the scene every frame, leading to issues with off screen illumination.

The view-sample cluster hierarchies that we use in this paper were first suggested by Sintorn et al. [SKOA14], who use it to splat per-triangle shadow volumes. That paper, in turn, was inspired by the work of Olsson et al. [OBA12], who first suggested arranging view samples in clusters to speed up shading with many bounded light sources. In their work, the acceleration structure was built over the light sources, instead of the clusters.

3. Background

In this section, we will give an overview of the previous work that this paper directly builds upon.

3.1. Photon Mapping

Photon mapping, introduced by Jensen [Jen96], is a well established technique for global illumination. Photons are traced from the light source into the scene and are gathered from an acceleration structure to perform a radiance estimate when raytracing from the camera in a second pass. Usually, the photon map is not queried for the first vertex of a camera path, but secondary reflection rays are traced and where *they* hit, the photon map is used. This is called the *final-gather* step. Photon mapping has been extended in a large number of ways, and we refer the reader to [HJB*12] for a thorough overview.

3.2. Photon Splatting

For real-time global illumination, a final-gather pass is usually too costly, but tracing a number of photons through the scene to capture some indirect-illumination effects can be achieved at real-time frame

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.

rates. Direct lighting can be efficiently computed with the standard rendering pipeline (with light visibility handled by, e.g., shadow maps). By not storing photons at the first bounce from the light source, the remaining photons can be used to directly estimate indirect illumination.

Even so, the traditional photon-map density estimate is often too costly. An alternative approach is to give each photon an a priori region of influence (or *radius*) and to *splat* the photons onto the view samples. This is often referred to as *photon splatting* [SB97] and can be implemented on a GPU by rendering the photons as geometric objects onto the current depth buffer, calculating the photons' influence on each affected view sample and accumulating the results [LP03, ML09].

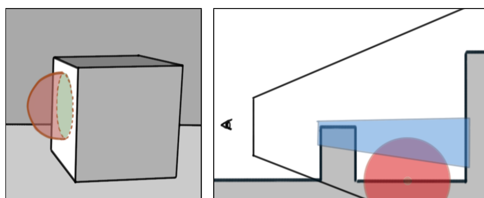


Figure 2: Photon Splatting. Left: when splatting photons using the rasterizer, view samples that lie behind the photon (red region) will not be culled. Right: A tiled renderer alleviates this, but tiles with depth discontinuities will still cause sub-optimal culling.

Mara et al. [MML13] evaluate a number of photon-splatting algorithms and find two algorithms to perform better than the rest. We will briefly explain these next. The first is 2.5D Photon Volumes that render 2D screen-aligned polygons [MM13] that represent the photons onto all the view samples that the polygon covers. The main problem with this method is that this can give rise to many unnecessary photon/view sample tests (see Figure 2).

The second method is Tiled Photon Splatting. This method associates a photon with a tile based on the tile's closest and furthest depth, reducing incorrect photon associations but still resulting in photons being incorrectly associated with tiles that have large depth ranges (see Figure 2). Also, the tile division is typically rather coarse, so that each tile will contain a long list of photons of which only a few affect each individual view sample.

4. Algorithm

In this section, we will describe our new photon-splatting algorithm in detail, beginning with the basic algorithm and then introducing some optimizations that are made possible by arranging the view samples in cluster hierarchies. Finally, we will discuss an approximate algorithm that is much faster, at the cost of a slight decrease in quality.

The basic algorithm consists of six passes:

1. **Render G-Buffer.** Using the standard pipeline, render the view sample positions, normals, and material properties to a G-Buffer texture.
2. **Generate Cluster Hierarchies.** Using the view-sample positions from the previous step, generate the cluster hierarchy (see Section 4.1).
3. **Photon Tracing.** A number of paths are traced from the light source, and at each bounce (except the first), a photon is stored to a list (see Section 4.2).
4. **Photon Splatting to Clusters.** Each photon is traversed through the cluster hierarchy, and when a node is enclosed, or a leaf node is found to be intersected, the photon ID is stored in a list unique to that node. (see Section 4.3)
5. **Radiance Estimate.** For each view sample, the lists of photons of the containing nodes are traversed and the contribution of each photon is accumulated. (see Section 4.4)
6. **Final Shading.** Direct lighting is computed in a full-screen pass, and the indirect lighting from the previous pass is added to obtain the final pixel color.

4.1. Generate Cluster Hierarchies

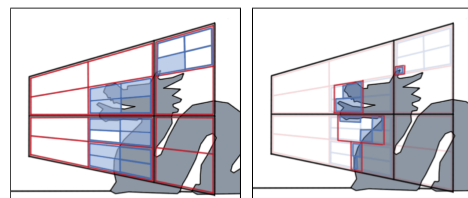


Figure 3: Left: Each cluster (cell in the 3D grid) is marked as occupied if it contains view samples. Then, parent nodes are marked as occupied recursively. Right: The bounding box of each cluster is calculated and propagated upwards in the tree.

We build a hierarchical view space 3D acceleration structure around the view samples using the technique presented by Sintorn et al. [SKOA14] and extending it for Photon Splatting. This addresses the issues of tiled photon splatting by creating much tighter bounds around view samples and allowing smaller groupings of view samples.

The cluster hierarchy divides the view frustum into a 3D grid with roughly cubical boxes. Each box is called a *cluster* and view samples are attached to one cluster. The next level of the hierarchy contains 32 clusters, a 32-bit word that indicates the occupancy of the child clusters, and a 32-bit node key (see Figure 3). Bounding boxes for each node are calculated and propagated up the tree.

4.2. Photon Tracing

This paper focuses on efficiently performing the radiance estimate, and our photon tracing is a straightforward GPU implementation using NVIDIA's Optix [PBD*10] framework. Our light sources are diffuse emitters with an angular cut-off. At each bounce, a new direction is chosen by importance sampling the BRDF (or the cosine term, for diffuse surfaces), and the new flux is calculated as:

$$\Phi' = \frac{f(\omega,\omega')}{p(\omega,\omega')} cos(\mathbf{n},\omega')\Phi, \qquad (1)$$

where f is the BRDF, p is the Probability Density Function (PDF), and Φ is the current flux of the photon. In order to maintain a roughly similar flux among stored photons, we then terminate the path with probability $R = min(1, \Phi'/\Phi)$ and set $\Phi = (1/R)\Phi'$ if the photon

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.

was not terminated. We keep tracing paths until we have reached some predefined number of photons.

In previous work [ML09, MML13], it has been suggested that the photon radius could be varied according to the path probability. This is important to avoid overblurring caustics, but we are mostly interested in scenarios where very few photons are stored to obtain real-time global lighting effects (avoiding caustics) and can generally keep the radius fixed.

4.3. Photon Splatting to Clusters

This step of the algorithm is actually broken into several passes to improve overall performance. Ultimately, each photon is intersected with the view-sample cluster hierarchy and inserted into a list at every node that is completely inside the photon, or leaf node that is intersected.

Since some photons will intersect many more nodes than others, there can be load-balancing issues. As suggested in [SKOA14], we therefore implement a pass that traverses down 3 levels and pushes the current photon ID and node key to a global list. To ensure the global list has good memory coherence, it is rearranged into local groups that are used in the following insertion passes to find the leaf intersections. When running the leaf intersection passes, a sufficient number of threads are started to fully utilize the GPU and each thread fetches jobs from the global list until it is empty.

At each cluster where photons intersect, an array is allocated to store the intersection photons. We calculate the required array size for each node in a first pass (traversing the hierarchy exactly as we do when inserting photons), and store the result in an array. We then use the prefix sum (calculated using cudpp [SHGO11]) of that array as a per-node index where the node's first photon shall be stored. An alternative approach to using arrays would be per-node linked lists. We have found, however, that while building these linked lists is not very expensive, iterating through them in the radiance estimation pass (Section 4.4) is very inefficient.

One thread per photon is launched on the GPU, and that thread will recursively test the photon against the nodes' bounding boxes. The traversal is implemented in an iterative fashion with a small stack in shared memory, which is initialized with the child mask from the root node. The main traversal loop then starts by looking at the top mask on the stack, and checking the first existing child node for intersection. That child node is cleared from the child mask on the stack, so as not to be tested again. When a photon is found to completely enclose a node, or the traversal reaches an intersected leaf node (cluster), the photon is appended to a list for that node and its child-nodes are not traversed.

When traversing the cluster hierarchy to find the sphere intersection we maintain a stack of child masks, and the current *node index* or *node key*. As each node is visited, the first bit of the 32-bit child mask becomes the active node, and the bit is removed from the mask and the remaining mask is pushed to the stack to track the remaining child nodes that still have to be processed. At a new node, the node key is used to fetch the child mask and bounding box of the current node from the corresponding global lists. Instead of storing the node key on the stack it can be stored in a single integer. When traversing to the i:th child of a node, the node key is simply shifted five bits to the left, and i is appended in the lower bits. When a node has been fully processed and the stack is popped, the node key is shifted back five bits to the right. For all but the final level, the node key is used as the immediate index in the list of bounding boxes. At the final level, to reduce the memory footprint, the node key is instead used to find an index to where the corresponding bounding box is in a compact list.

The algorithm differs from the method described by Sintorn et al. [SKOA14]. First, in their algorithm, a *warp* (32 threads) is started per primitive, and the intersection tests are done in parallel. We found that, because only a few of the subnodes are likely to be occupied, letting a single thread do all intersections for the occupied nodes is more efficient, at least in our case where the intersection tests are simple sphere/bounding box tests. Secondly, in their algorithm the bounding boxes were defined in *Normalized Device Coordinates* (NDC), and the hierarchy could be tested against each per-triangle shadow volume by, starting from the root, seeing which subnodes were occupied and testing the bounding boxes of each against the shadow-volume planes (also in NDC). In our case, the primitives to be tested are spheres, which are not simple geometric shapes in NDC, so we instead store the bounding boxes in view-space coordinates, which will be overly conservative at higher levels of the hierarchy.

Note that we consider photons to have a spherical influence region, rather than a squashed sphere as has been proposed in previous work [ML09]. Clipping the sphere by one or two planes is a simple modification to the algorithm but causes unwanted artifacts as we want to avoid a smoothing kernel in our radiance estimate (as explained below).

4.4. Radiance Estimation

When all photons have been traversed through the hierarchy, we have a list of photons per node that must be considered for all contained view samples. We start one thread per view sample and loop over the photons in each containing node's list. For each photon, we check if the view sample is actually within and, if so, we accumulate the reflected radiant intensity: $I = I + f(\omega, \omega_p)\Phi_p$ (where f is the BRDF, ω is the view direction, ω_p is the incoming direction of the photon and Φ_p is the flux carried by the photon). When all photons have been processed, we calculate the outgoing radiance as $L = I/\pi r^2$, where r is the photon radius (or, if we have photons of varying radii, the distance to the furthest photon).

When an insufficient number of photons are available in an area, the distinct photon splats will become visible on the surface. It is common practice to attempt to hide this by multiplying each photon's contribution with a distance-dependent smoothing kernel. In the scenarios we are focusing on, with smoothly varying diffuse inter-reflections, we found that it was more likely to reduce the image quality.

4.5. Normal Cones

If the view samples in a node all have normals such that the photons direction is not incident to any of their tangent planes, the node can be rejected immediately.

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.

The view samples that fall into one cluster are guaranteed to lie close to each other. Therefore, we make the assumption that they are also likely to have similar normals, and upon that assumption we attempt an optimization. When building the cluster hierarchy, starting with the leaf level, we compute the average normal and the maximum angle that a view sample's normal deviates from this average. We call this the *normal cone* of the cluster. We later refer to this optimisation as *cluster-cone*. This normal cone is similar to that used by Sederberg and Meyers [SM88] to bound Bézier normal vectors. At the next level in the hierarchy, we generate the average of all the subnode normal-cone directions and the maximum of the subnodes' deviation from that average *plus* the subnodes' normal-cone angle. In this way, we propagate the normal cone upwards in the hierarchy (see Figure 4).

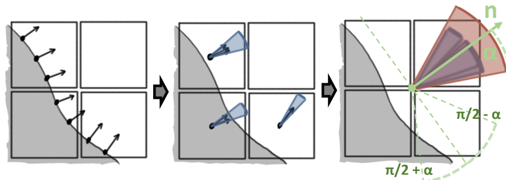


Figure 4: The normals of all view samples in a cluster are aggregated into a normal cone. The normal cones of all clusters are aggregated into normal cones for their parents.

With the normal cones in place, we can add a new simple rejection test to the photon traversal. Whenever a photon is found to intersect a node, we also check if the angle between the photon direction and the normal-cone direction is greater than $\pi/2+\alpha$, where α is the normal-cone angle. If so, the photon will not affect any of the contained view samples and does not need to be traversed further.

4.6. Trivially Accepting Photons

For diffuse surfaces, where the BRDF is constant, the contribution of a photon to a view sample within a node is either zero (if the photon is incoming from below the surface), or constant (for all other directions). Thus, when a photon encloses a node, and we know that all view samples within that node have normals on the same hemisphere as the incoming direction, the contribution for all view samples will be the same and we can accumulate this in the node instead.

Thus, if the angle between the photon's direction and the normal-cone direction is less than $\pi/2-\alpha$ (i.e. the photon is incident on all contained view samples' tangent planes), we simply add the photons flux to a per-node value. In the next pass, when estimating radiance, we add this flux multiplied by the BRDF to the view sample's accumulated intensity. This can greatly increase the performance of the radiance estimate pass, while the traversal performance remains nearly the same. This optimisation is referenced as *cluster-trivial*.

4.7. Directional Approximation

A further performance optimization, later referenced as *directional*, can be achieved by avoiding each view sample having to loop through all intersecting photons and instead accummulating a *directional* flux per intersected node. For each node, we store the incoming flux into a small number of buckets corresponding to discrete directions. When a photon is found to enclose a node, or intersect a leaf node, it adds its contribution to the bucket with the closest associated direction. When estimating radiance, we no longer have to process a list but simply evaluate the incoming flux from each direction, for all nodes that contain the view sample.

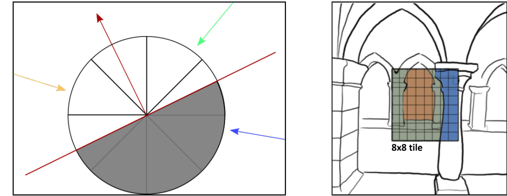


Figure 5: Left: a 2D representation of flux being stored per octant based on its incoming direction (yellow, green and blue arrows); the stored flux is later weighted to account for certain directions being below (greyed area) the view-sample's tangent-plane (the red plane, with its normal). Right: For each tile, there can be several clusters at different depths (uniquely colored in the image), and our method evaluates illumination per such cluster.

In our implementation, we store the flux incoming from each octant of the sphere (the number of directions used has a direct impact on memory consumption, as shown in Section 5). When shading, we only have the sum of the incoming flux from each octant. Parts of, or all of, the octant might lie below the view sample's tangent plane, and photons from those directions should not contribute. To remedy this overestimation of incoming flux, we could, for each direction, clip the corresponding octant of the unit sphere against the view sample's tangent plane. The ratio of the area that is below the tangent plane to the area that is above would give us a reasonable weight for this direction (see Figure 5). This calculation would be too expensive to perform for every view sample and direction, but the weights depend only on the view-space normal, so we could pre-compute it and store it in a small cubemap. In practice, however, we have found that a much simpler heuristic gives acceptable results. The contribution from each direction is simply weighted with $2\cos(n,d)$, where n is the view-space normal, and d is the direction in view space. We multiply by 2 to account for that the integral of the cosine is smaller (π) than the integral of the hemisphere (2π).

The proposed method only gives us an estimation of the incoming flux per cluster. If we use that result to shade each view sample, the result can be visibly blocky when the light-field changes quickly. To alleviate this, for diffuse materials, we store only the irradiance in a texture, and apply a depth- and normal-aware blur filter to that texture. The blurred irradiance is then multiplied by the diffuse BRDF in the final shading stage.

It is important to note that the proposed approximate method gives much better results than simply rendering the indirect illumination at a coarser resolution. E.g. a 32×32 tile can contain view samples from several distinct surfaces at different depths which will fall into different clusters. Our method will sample the irradiance for each of these, whereas a simple upsampling would pick only the one in the middle of the tile (see Figure 5).

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.

5. Results

All measurements have been made at a 1920x1080 resolution on a NVIDIA Titan X GPU with 12 GB VRAM using OptiX [PBD*10] 3.9.0 for tracing the photons in real-time and CUDA [NBGS08] 7.0 for every step of the algorithm, apart from generating the G-buffers, and the shadowmaps, as well as the blur pass, which are done using OpenGL. We limit the photon tracing to four bounces and do not store photons on the first bounce but instead use standard deferred shading to compute the direct lighting.

Three different scenes are used throughout this paper: Sponza, Sibenik and San Miguel (see Figure 1). For each of them, we have created a fly-through animation (see the accompanying videos). These fly-throughs do not include moving lights or geometry, but we have supplied additional videos showing dynamic scenes. The total frame time, including the time taken to trace photons is given in Figure 1. Updating the acceleration structures when an object moves adds less than a millisecond to these times.

	Cluster	Directional	Tiled
Tiles z-Bounds	-	-	0.016
Cluster Hierarchy	97	97	-
Final Bounds	256	256	-
Normal Cone	24	-	-
Accum. Flux	73	582	-
Sub-Total	450	935	0.016
Jobs	2.4	2.4	-
Photons Array	60	-	28
Photon Map	0.16	0.16	0.16
Sub-Total	63	7.2	28
Total	513	942	28

Table 1: Memory-consumption (in MiB) breakdown for our cluster-trivial version, a directional version using eight regions and a tiled version using 32×32 tiles, all of them at a resolution of 1920×1080 and using 10k photons of radius 4 in Sponza.

5.1. Performance

We compare the splatting and shading performance of our method, with and without optimizations, to our own implementation of the tiled photon splatting presented by Mara et al. [MML13]. We have not implemented the stochastic selection of photons proposed by Mara et al., and therefore, we do not preload the shared memory with photons but interleave the loading of photons with the arithmetic operations of computing their contribution. This also simplifies the case when list sizes do not fit in shared memory. For the tiled algorithm we use a tile size of 32×32.

Figure 6 presents a comparison of each method's execution time (photon tracing and G-buffer generation is not included, as the time is similar for all methods) for a fly-through of each scene. The cluster-trivial method is on average about two times faster than the tiled method, and up to three at some peaks. The tiled method has a lower overhead compared to our methods, and will therefore be faster when few photons are being splatted (see Figure 6a, around frame 150). With the directional algorithm, we achieve another 2-3x speedup and the difference in image quality is minimal (see Figure 10, and Table 2). The directional algorithm also shows little variance in execution times, due to better load balancing. The directional algorithm stores radiance data starting at one level above the leaf nodes in the cluster hierarchy. The performance difference between the different optimizations added to our cluster version are presented in Figure 9.

When breaking down the execution times of the different methods into passes (see Figure 8), the tiled method is dominated by the final shading pass. The total frame time is dominated by the photon tracing, but it should be noted that the photon tracing code, which relies on OptiX, has not been optimized, nor has the deferred rendering pass; better performance results could therefore be expected for these steps.

5.2. Memory Consumption

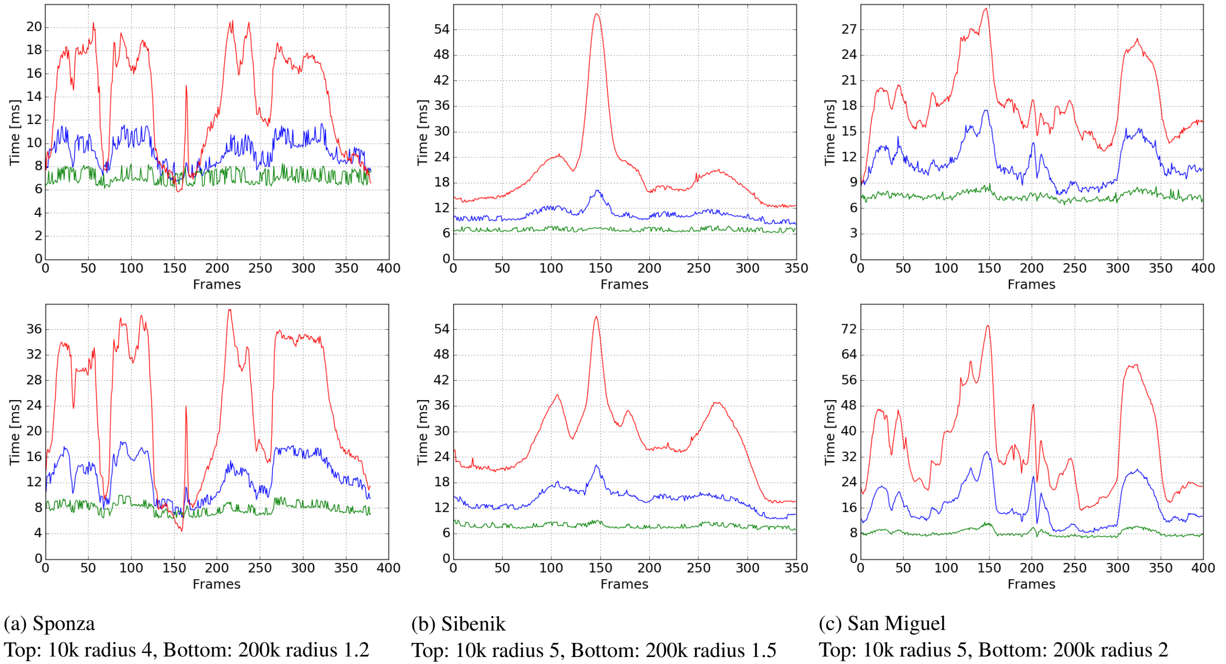
Our cluster method statically allocates memory for a dense hierarchy, even though we only build and use a sparse hierarchy per frame. We use a similar hierarchy to the 1080p hierarchy of Sintorn et al. [SKOA14] with six half-floats for each AABB and 32-bit childmask per node. In addition, we store one 32-bit word per node for the normal cone (theta and the normal cone angle are compacted to 8-bit values, whereas phi requires 16 bits), and 3 floats per node for the per-node flux, effectively doubling the memory consumption.

As explained in Section 4.3, we split the splatting pass into two sub-passes. The first sub-pass splats the photons down to a certain pre-defined level, from which the second sub-pass starts and continues splatting further down. This means that we need to store additional data, the *jobs* mentioned in Table 1, which is an array of pairs (a 32-bit value for the photon id, and another 32-bit value for the cluster key); as we group the jobs per cluster to improve data locality, two lists are needed in practice. The largest size required for this list during our experiments was 30M elements (requiring 458 MiB for both lists) for 50M photons of radius 0.2 in Sponza, and we simply pre-allocate a list of sufficient size. In order to reduce its memory footprint, the different sub-passes can be run several times on a smaller sized list.

The different methods presented here generate arrays of photons per cluster/tile. Those photons are stored in a compact format, amounting to 16 bytes per photon. Position and flux are both stored as three half-floats, the orientation as two 8-bit values (theta and phi), and the radius as a 16-bit value. We pre-allocate a sufficiently large array to hold these photons, and in our experiments it has never exceeded 102M elements (requiring 1.5 GiB when storing the full photon, rather than just its id) for 50M photons of radius 0.2 in Sponza.

In the directional method, we do not need to store individual photons per cluster, but we instead need to allocate memory for storing the accumulated flux for each direction of each node, starting at the level where the first splat sub-pass stops, down to the leaves. This amounts, for eight regions per node, to 24 floats per node, which translates to a total of 582 MiB for a 1080p resolution and starting the accumulation from level 4.

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.



(a) Sponza
Top: 10k radius 4, Bottom: 200k radius 1.2

(b) Sibenik
Top: 10k radius 5, Bottom: 200k radius 1.5

(c) San Miguel
Top: 10k radius 5, Bottom: 200k radius 2

Figure 6: Comparison of cluster, directional and tiled methods splatting time for the San Miguel, Sibenik and Sponza scenes using different radii and photons numbers. The red, blue and green curves correspond respectively to tiled, cluster-trivial and directional.

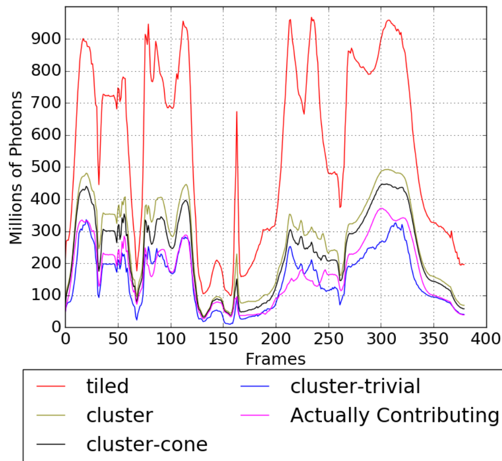


Figure 7: Sum of photons read in total during the shading pass for tiled shading and our method compared against the number of actually contributing photons. (case: 10k photons of radius 4 in Sponza)

5.3. Photon Splatting Efficiency

In Figure 7, we see that the tiled splatting reads many more photons during the shading pass than are actually contributing to the shading. Our method results in much fewer reads due to the much better spatial bounds. With our method, on average, less than 3% of the photons read from the lists are rejected by testing the view sample position against the photon influence sphere.

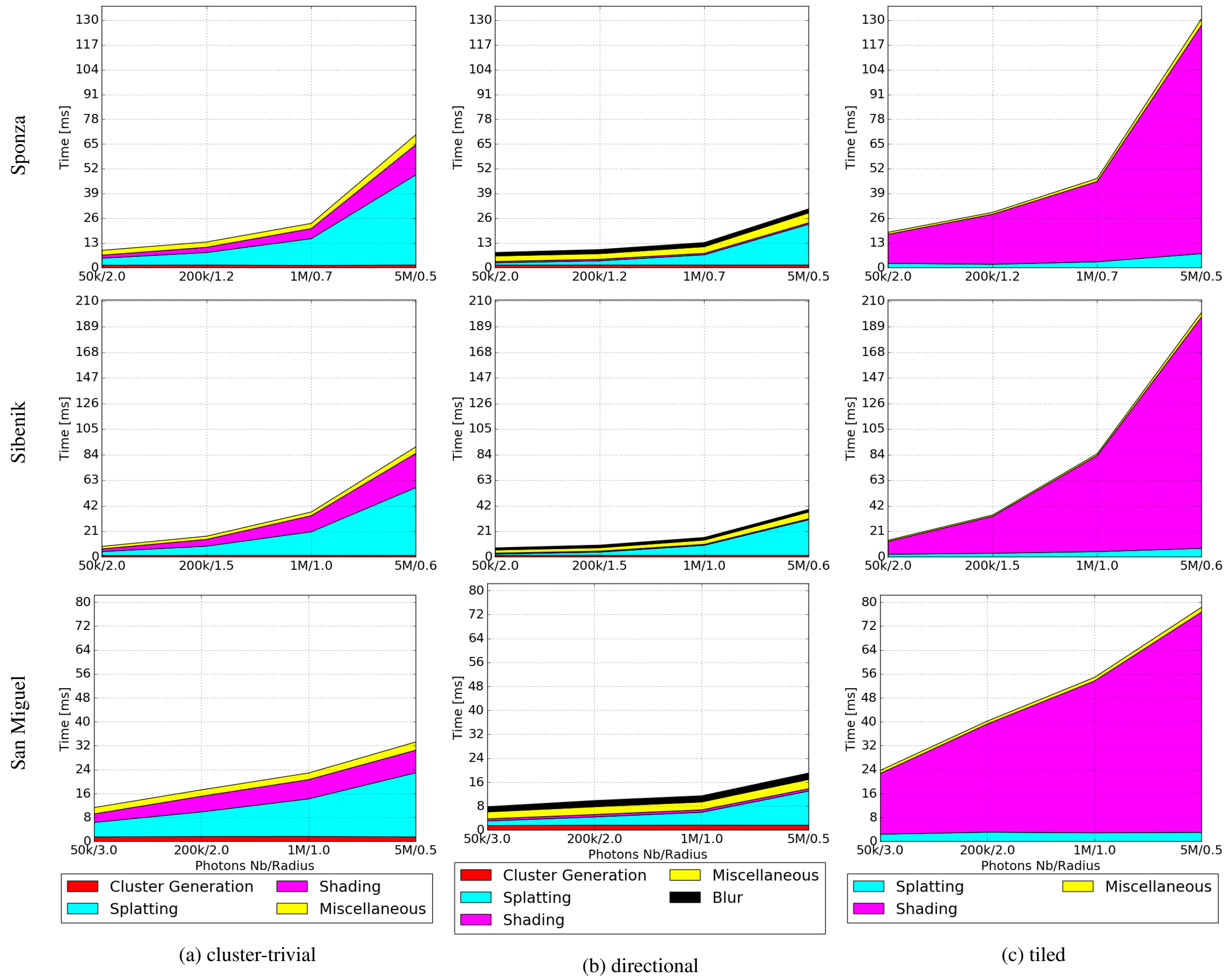
With the cluster-cone optimization (see Section 4.5) we discard even more photons and obtain more relevant lists (see Figure 7) and with the cluster-trivial optimization (see Section 4.5), we replace some of the insertions with accumulation of flux, which decreases the number of list reads to be even fewer than the number of photons that are actually contributing to the radiance.

5.4. Quality Evaluation

In order to assess the quality of our algorithms resulting images, we compute their SSIM [WBSS04] and PSNR mean score agains a path traced reference image generated using Embree [WWB*14]. The results are summarised in Table 2. The cluster and tiled methods have almost identical SSIM mean score, which is expected as they end up shading view samples with the same list of photons: only the way those lists are computed changes. Even though the directional method is an approximation, its SSIM mean score remains only slightly below the cluster score. For a visual comparison, Figure 10 presents the final image for the different methods in Sponza with 200k photons of radius 2.

By using more photons and of smaller radius, our experiments show that the final image quality improves, as regular photon mapping would. This is supported by the SSIM and PSNR scores listed in 2, at least for the cluster and tiled methods. Combined with the scaling of our algorithm, performance-wise (see Figure 8), for "higher quality" setups, we expect the cluster-trivial method to perform favorably on newer hardware, without any additional modifications. Figure 11 compares the final image for our cluster-trivial optimisation in Sponza between a low-quality setting (10k photons of radius 4) and a high-quality one (50M photons of radius 0.2).

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.



(a) cluster-trivial (b) directional (c) tiled

Figure 8: Breakdown of total frame time into its various components, for our cluster-trivial and directional methods, as well as Mara et al. tiled, inside the Sponza scene using the same viewpoint as in Figure 1. *Miscellaneous* groups buffer clearing, texture mapping and unmapping.

5.5. Reflections

Our method can be used to render non-diffuse BRDFs, but we cannot use the cluster-trivial optimization (as the incoming direction of each photon is important for view-dependent BRDFs). Figure 12 shows an example of glossy reflections. Note that, while curved objects look convincing, we do not shoot enough photons (nor small enough photons) to capture glossy reflections from e.g. a flat floor.

6. Discussion and Limitations

We have shown that view-sample cluster hierarchies can be used to perform fast radiance estimates in interactive settings where photons are few enough to be traced per frame, and large enough to provide a smooth result. Additionally, we have shown that an approximate version of our method can produce convincing diffuse inter-reflection images at around 10 ms per frame, bringing global illumination closer to use in real-time applications like video games.

Like most realtime GI algorithms (including production proven algorithms like Voxel Cone Tracing and Light Propagation Volumes), the most obvious drawback of Photon Splatting is the existence of light leakage. While clearly visible in all of our renderings, we have not found these artifacts too disturbing, however, and are convinced that photon splatting (at the low cost we obtain) is a usable solution for global lighting phenomena. As we focus on a small number of large photons, high frequency phenomenas, like caustics, are poorly represented, but still supported by our method. More convincing results might be attained by a higher number of smaller photons. Similarly, glossy surfaces are supported by our method, but results remain poor, even with a higher number of small photons, as it does for traditional photon mapping.

We have illustrated efficient splatting in the context of photon splatting, but the same method could be applied to VPL based algorithms, and we are eager to explore if there may be other uses (e.g. ambient occlusion).

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.

Scene	Photons Nb	Radius	Cluster		Directional		Tiled	
			SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Sponza	10k	4.0	90	25	91	27	90	24
	50k	2.0	91	26	90	25	92	27
	200k	1.2	93	30	92	27	93	30
	1M	0.7	94	32	92	27	94	32
	5M	0.5	94	32	92	27	94	32
	50M	0.2	95	33	89	25	95	33
San Miguel	10k	5.0	75	22	81	25	76	22
	50k	3.0	82	26	83	26	81	25
	200k	2.0	84	26	83	26	83	26
	1M	1.0	85	27	82	26	84	27
	5M	0.5	85	28	82	26	85	27
	50M	0.2	86	28	83	27	85	28
Sibenik	10k	5.0	91	28	87	27	92	27
	50k	2.0	88	31	85	28	89	31
	200k	1.5	94	32	89	28	93	32
	1M	1.0	95	34	90	28	94	34
	5M	0.6	96	36	94	30	95	36
	50M	0.2	96	38	93	30	96	38

Table 2: SSIM (in %) and PSNR (in dB) results for various setups across the three test scenes using the cluster-trivial and the directional methods against a path traced reference image generated using Embree.

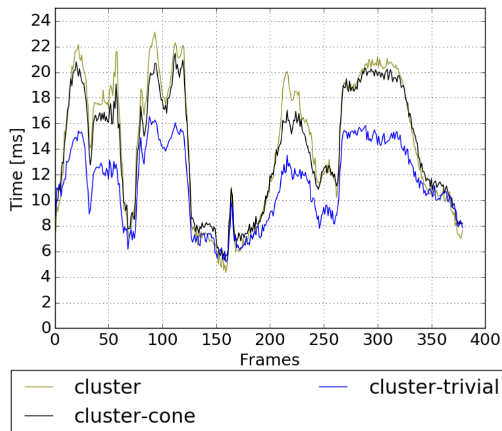


Figure 9: Total execution time for our method with and without optimizations. Included timings are splatting, shading and other essential steps, e.g., building the cluster hierarchy. Tracing of photons and deferred shading are not included. (case: 200k photons of radius 1.2 in Sponza)

Acknowledgements We would like to thank Jacob Munkberg and Jon Hasselgren for their critical and helpful comments about the paper. We use Mehdi Rabah's SSIM implementation [Rab]. The Sponza scene is created by Frank Meinl, Sibenik is by Marko Dabrovic, and San Miguel is by Guillermo M. Leal Llaguno. All scenes are freely available at Morgan McGuire's Computer Graphics Archive [McG]. Pierre and Michael thank the Swedish Research Council under grant 2014-5191 and ELLIIT for funding. Erik, Viktor and Ulf are supported in part by the Swedish Research Council under grant 2014-4559.

References

[CNS*11] Crassin C., Neyret F., Sainz M., Green S., Eisemann E.: Interactive indirect illumination using voxel cone tracing: A preview. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 207–207. 2

[DBBS06] Dutre P., Bala K., Bekaert P., Shirley P.: *Advanced Global Illumination*. AK Peters Ltd, 2006. 1

[DS05] Dachsbacher C., Stamminger M.: Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2005), I3D '05, ACM, pp. 203–231. 2

[HJB*12] Hachisuka T., Jarosz W., Bouchard G., Christensen P., Frisvad J. R., Jakob W., Jensen H. W., Kaschalk M., Knaus C., Selle A., Spencer B.: State of the art in photon density estimation. In *ACM SIGGRAPH 2012 Courses* (New York, NY, USA, 2012), SIGGRAPH '12, ACM, pp. 6:1–6:469. 2

[Jen96] Jensen H. W.: Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96* (London, UK, UK, 1996), Springer-Verlag, pp. 21–30. 2

[Jen01] Jensen H. W.: *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001. 1

[KD10] Kaplanyan A., Dachsbacher C.: Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D '10, ACM, pp. 99–107. 2

[Kel97] Keller A.: Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 49–56. 2

[LP03] Lavignotte F., Paulin M.: Scalable photon splatting for global illumination. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South*

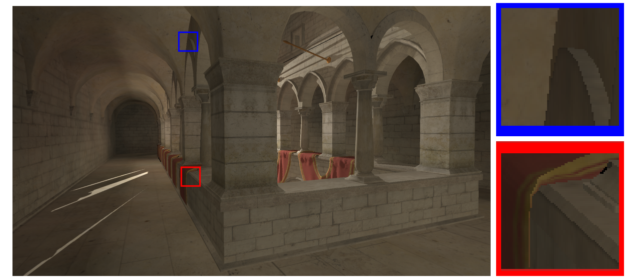
© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.

(a) cluster-trivial

(b) directional

(c) tiled

Figure 10: View from Sponza rendered using our cluster-trivial method, our directional method, and Mara et al. tiled method with 200k photons of radius 1.2.



(a) 10k photons of radius 4 in 10 ms

(b) 50M photons of radius 0.2 in 290 ms

Figure 11: Comparison between a "low-quality" setting and a "high-quality" setting in Sponza; the time are those from cluster-trivial.

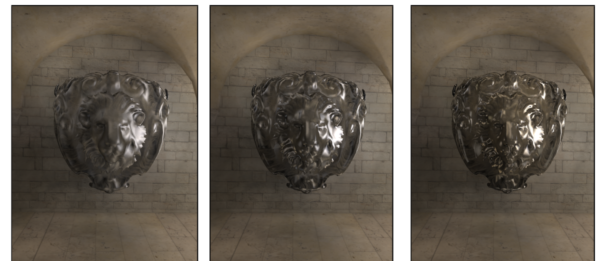


Figure 12: Glossy materials of varying roughness rendered with our method. 50k photons with radius 1. Total time per frame: 34 ms.

East Asia (New York, NY, USA, 2003), GRAPHITE '03, ACM, pp. 203–ff. 3

[LSP*12] Li S., Simons L., Pakaravoor J. B., Abbasinejad F., Owens J. D., Amenta N.: kann on the gpu with shifted sorting. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics* (Aire-la-Ville, Switzerland, Switzerland, 2012), EGGH-HPG'12, Eurographics Association, pp. 39–47. 2

[McG] McGuire M.: Computer graphics archive. http://graphics.cs.williams.edu/data Accessed on 2016/03/29. 9

[ML09] McGuire M., Luebke D.: Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the 2009 ACM SIGGRAPH/EuroGraphics conference on High Performance Graphics* (New York, NY, USA, August 2009), ACM. 2, 3, 4

[MM13] Mara M., McGuire M.: 2d polyhedral bounds of a clipped, perspective-projected 3d sphere. *Journal of Computer Graphics Techniques (JCGT) 2*, 2 (August 2013), 70–83. 3

[MML13] Mara M., McGuire M., Luebke D.: Toward Practical Real-Time Photon Mapping: Efficient GPU Density Estimation. In *Interactive 3D Graphics and Games 2013* (March 2013). 2, 3, 4, 6

[MMNL14] Mara M., McGuire M., Nowrouzezahrai D., Luebke D.: *Fast Global Illumination Approximations on Deep G-Buffers*. Tech. Rep. NVR-2014-001, NVIDIA Corporation, June 2014. 2

[NBGS08] Nickolls J., Buck I., Garland M., Skadron K.: Scalable parallel programming with cuda. *Queue 6*, 2 (Mar. 2008), 40–53. 6

[NW09] Nichols G., Wyman C.: Multiresolution splatting for indirect illumination. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 83–90. 2

[OBA12] Olsson O., Billeter M., Assarsson U.: Clustered deferred and forward shading. In *HPG '12: Proceedings of the Conference on High Performance Graphics 2012* (2012). 2

[PBD*10] Parker S. G., Bigler J., Dietrich A., Friedrich H., Hoberock J., Luebke D., McAllister D., McGuire M., Morley K., Robison A., Stich M.: Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics* (August 2010). 3, 6

[PH10] Pharr M., Humphreys G.: *Physically Based Rendering, Sec-*

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.

ond Edition: From Theory To Implementation, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. 1

[Rab] RABAH M.: C++ implementation of SSIM. http://mehdi.rabah.free.fr/SSIM/ Accessed on 2016/03/29. 9

[RDGK12] RITSCHEL T., DACHSBACHER C., GROSCH T., KAUTZ J.: The state of the art in interactive global illumination. *Comput. Graph. Forum 31*, 1 (Feb. 2012), 160–188. 2

[REH*11] RITSCHEL T., EISEMANN E., HA I., KIM J. D., SEIDEL H.-P.: Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. *Computer Graphics Forum (presented at EGSR 2011)* (2011). 2

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 129:1–129:8. 2

[SB97] STÜRZLINGER W., BASTOS R.: Interactive rendering of globally illuminated glossy scenes. In *Proceedings of the Eurographics Workshop on Rendering Techniques '97* (London, UK, UK, 1997), Springer-Verlag, pp. 93–102. 3

[SHGO11] SENGUPTA S., HARRIS M., GARLAND M., OWENS J. D.: Efficient parallel scan algorithms for many-core gpus. In *Scientific Computing with Multicore and Accelerators*, Kurzak J., Bader D. A., Dongarra J., (Eds.), Chapman & Hall/CRC Computational Science. Taylor & Francis, Jan. 2011, ch. 19, pp. 413–442. 4

[SKOA14] SINTORN E., KÄMPE V., OLSSON O., ASSARSSON U.: Per-triangle shadow volumes using a view-sample cluster hierarchy. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2014), I3D '14, ACM. 2, 3, 4, 6

[SM88] SEDERBERG T. W., MEYERS R. J.: Loop detection in surface patch intersections. *Computer Aided Geometric Design 5*, 2 (1988), 161 – 171. 5

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (April 2004), 600–612. 7

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph. 33*, 4 (July 2014), 143:1–143:8. 7

[ZHWG08] ZHOU K., HOU Q., WANG R., GUO B.: Real-time kd-tree construction on graphics hardware. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 126:1–126:11. 2

© 2016 The Author(s)
Eurographics Proceedings © 2016 The Eurographics Association.