Eurographics Symposium on Rendering 2011
Ravi Ramamoorthi and Erik Reinhard
(Guest Editors)

Volume 30 (2011), Number 4

Importance Point Projection for GPU-based Final Gathering

David Maletz and Rui Wang

University of Massachusetts Amherst

Abstract

We present a practical importance-driven method for GPU-based final gathering. We take as input a point cloud representing directly illuminated scene geometry; we then project and splat the points to microbuffers, which store each shading pixel's occluded radiance field. We select points for projection based on importance, defined as each point's estimated contribution to a shading pixel. For each selected point, we calculate its splat size adaptively based on its importance value. The main advantage of our method is that it's simple and fast, and provides the capability to incorporate additional importance factors such as glossy reflection paths. We also introduce an image-space adaptive sampling method, which combines adaptive image subdivision with joint bilateral upsampling to robustly preserve fine details. We have implemented our algorithm on the GPU, providing high-quality rendering for dynamic scenes at near interactive rates.

1. Introduction

Interactive global illumination in dynamic scenes continues to present a great challenge in computer graphics. A popular technique in recent years is to perform final gathering from many point lights [WFA*05, HPB07]. The idea is to sample the scene's illumination as many virtual point lights (VPLs) [Kel97], then integrate the contributions from all VPLs to a shading pixel. One main advantage of this approach is that it enables gathering of VPLs, which is fast to compute and suitable for parallel computation. Moreover, [Chr08, REG*09] adopt a single point cloud to represent both the illumination and geometry, enabling very fast visibility calculation using point-based rasterization.

In this paper we present a new method for GPU-based final gathering of VPLs. As in [Chr08, REG*09], we adopt a single point cloud to represent directly illuminated scene geometry. We project and splat these points into each pixel's microbuffer, which stores the occluded incoming radiance field. Our main goal is to select points for projection based on importance – their approximated contributions to a shading pixel. Such an importance-driven method provides fast convergence speed and is suitable for GPU processing.

We start with random sampling to estimate the importance function, defined as each scene point's un-occluded contribution to a given shading pixel. We then draw points according to the estimated importance, and splat the selected points into the pixel's microbuffer. We compute the splat size adaptively using the importance value, providing an estimate of the solid angle subtended by the splat. The adaptive splatting requires no separate hole filling step in the microbuffers. To quickly compute the importance function, we partition the scene points into spatial clusters. Points within each cluster are treated uniformly randomly.

Experiments show that our method is simple, fast, and suitable for fully dynamic scenes. Moreover, our method allows the incorporation of additional importance factors such as glossy reflection paths, improving the efficiency of computing such effects. We implement our algorithms on modern GPUs, achieving high-quality rendering of dynamic scenes at $3 \sim 4$ seconds per frame (evaluated at every pixel) for a 512×512 image. Figure 1 shows several examples. We also provide a progressive version of the renderer to enable fully interactive scene manipulation. The progressive renderer exploits a new image-space adaptive sampling method, which combines adaptive image subdivision with joint bilateral upsampling to robustly preserve fine details and edges.

2. Related Work

Conventional global illumination methods, such as Monte Carlo ray tracing and photon mapping, provide high-quality results offline, but are usually too expensive for interactive applications. Precomputation based methods exploit offline computed datasets for fast online rendering, but provide limited support for dynamic scenes.

Point-based Global Illumination Extensive work has

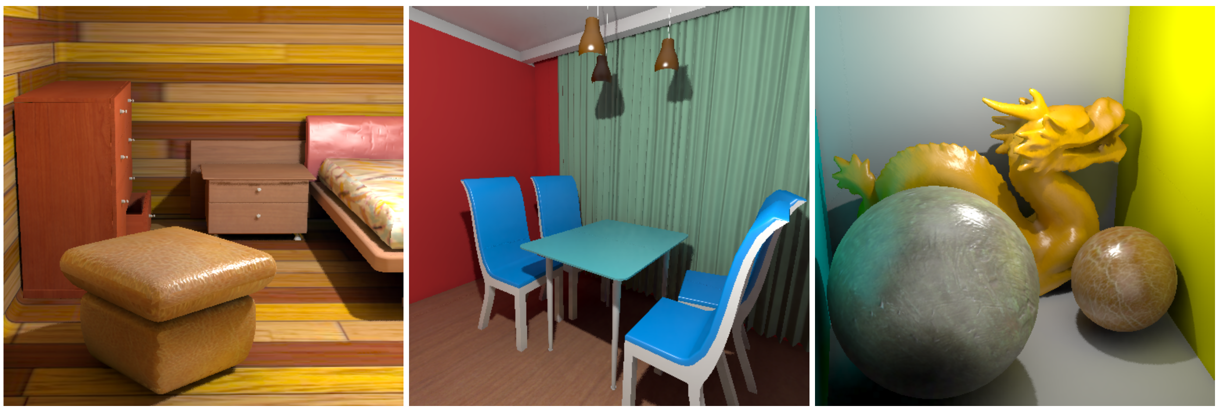


Figure 1: *Global illumination results using our method. These images are at* 512^2 *resolution and are computed within* $3 \sim 4$ *seconds on an NVIDIA 480 GTX. Shading is evaluated at every pixel. Note the realistic surface reflections and indirect shadows.*

shown that point-based representations are very suitable for global illumination due to their simplicity and intrinsic parallelism. Instant radiosity [Kel97] first proposed to treat indirect illumination as a set of virtual point lights (VPLs). The contribution of each VPL is computed using shadow mapping. Imperfect shadow maps [RGK*08] use a geometry point cloud to compute approximate shadow maps, significantly speeding up the visibility calculation. Recently, [NED11] presented screen space compensation to improve the accuracy in computing near-field VPL contributions. These methods typically employ up to 1000 VPLs. As shown by a perceptual study in [KFB10], scenes with glossy materials often require more than a few thousand VPLs.

A number of recent papers have also studied voxel or volume based approaches for global illumination. [NPW10] proposed screen-space voxelization for gathering illumination from area lights approximated as a set of VPLs; [KD10] introduced a lattice-based structure to store light propagation volumes; [THGM11] presented a voxel-based representation to accelerate visibility evaluation. These methods are fast and suitable for 3D games, but their renderings can miss fine details due to the limited resolution of voxels.

Lightcuts [WFA*05] represent indirect illumination as a hierarchical point cloud, allowing for VPL integration at sublinear cost. They solve visibility by using ray tracing. Matrix row-column sampling [HPB07] clusters VPLs by sampling their contributions to a subset of shading pixels. Visibility is solved using shadow mapping. A technique proposed in [HKWB09] extends VPLs to virtual spherical lights (VSLs), which are suitable for scenes with glossy materials. Most recently, [DKH*10] combine global and local lights to efficiently render high-rank illumination effects. These methods are accurate, but take a few minutes to render.

By converting polygons to surfels, [Bun05] approximate ambient occlusion on the GPU using a hierarchy of surfels to accelerate visibility computation. [Chr08] use a similar representation to compute final gathering in production quality renderings. A point cloud is created to represent directly illuminated scene geometry, and is hierarchically rasterized to a shading pixel's microbuffer. Micro-rendering [REG*09] uses a similar approach and achieves interactive rates by exploiting modern GPUs. They also presented importance-warped microbuffers to efficiently handle glossy BRDFs.

Instead of traversing a point hierarchy, our goal in this paper is to study an alternative formulation that uses importance point projection. The main benefits are its simplicity and improved support for fully dynamic scenes. Moreover, it allows the incorporation of both diffuse and glossy importance factors. Section 5 provides a more detailed discussion between our method with hierarchical point traversal.

GPU-based Photon Mapping Photon mapping [Jen01] is widely used to simulate multi-bounce indirect lighting. Since the first GPU-based photon mapper introduced in [PDC*03], a number of recent papers have demonstrated impressive results: [ZHWG08] presented a GPU-based kd-tree for interactive photon mapping; [ML09] introduced a fast image-space photon mapper for global illumination. These methods achieve fast computation speed by avoiding final gathering. Recently [WWZ*09] exploit sparse irradiance samples to reduce final gathering cost, but their computation of the irradiance samples is purely based on geometry changes. In contrast, our adaptive image subdivision method accounts for both geometry and radiance changes.

Importance Sampling reduces stochastic sampling noise by drawing samples from an importance function that approximates the integrand. An efficient importance function can be defined as the product of the illumination and BRDF. This is called bidirectional importance sampling [BGH05], which has been studied by several recent work [CJAMJ05, CETC06, CAM08, WÅ09]. These methods typically rely on ray tracing to compute each sample's visibility and thus remain offline; in contrast, we compute importance sampling on the GPU to allow for interactive rendering.

Caching and Interpolation Indirect lighting is typically

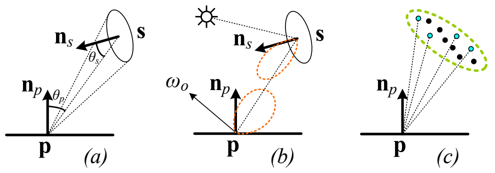


Figure 2: *(a) for diffuse surfaces, the importance of a point* **s** *is defined as its projected solid angle; (b) for glossy surfaces, we additionally include the BRDF into the importance value; (c) cluster importance is estimated by random sampling within the cluster.*

smooth, making it suitable for caching or adaptive sampling. [WRC88] progressively cache irradiance samples and reuse them during the computation. [TPWG02] proposed an object-space caching method suitable for dynamic scenes, but require parameterized objects. To reconstruct an image from sparse samples, joint bilateral upsampling [SGNS07] provides nonlinear interpolation that prevents blurring from crossing feature edges. However, a regular grid sampling pattern can lead to loss of fine details around small geometric features. Our method combines joint bilateral upsampling with adaptive image subdivision, which can robustly preserve fine details and features, as shown in Figure 10.

3. Algorithms

3.1. Overview

Point Representation. We use $\mathcal{S}$ to denote a point cloud sampled from a scene with directly illuminated radiance. Each point is associated with a position, normal, delta surface area, and radiance. The points can be generated using a variety of methods, such as instant radiosity, surface sampling, micropolygon subdivision, or photon mapping. In our case, we use Poisson disk surface sampling in [BWWM10] to generate points on scene surfaces. Due to the uniform distribution, all points are assigned the same delta surface area. Each point can emit diffuse as well as glossy radiance.

Microbuffers. Our goal is to project the points to each pixel's microbuffer, which stores the depth and color of the closest projected point. The microbuffer is essentially a small environment map observed at a pixel representing its incoming radiance field. Since we only need to represent the upper hemisphere, we use a hemi-octahedral map [WNLH06] to store the buffer as a 32×32 texture. Each pixel in the map is associated with a direction and a delta solid angle, and all pixels together cover a 2π solid angle.

Stochastic Sampling. To obtain the microbuffer, we could project all points. But this would be too slow as there are tens of thousands of points. It is also not necessary as the microbuffer size is much smaller than the total number of points. To improve efficiency, we draw samples stochastically from $\mathcal{S}$. A naive approach is to pick a point from $\mathcal{S}$ uniformly randomly, thus every point has an equal probability $p = \frac{1}{|\mathcal{S}|}$ to be selected ($|\mathcal{S}|$ is the point set size). But clearly, a more efficient way is to importance sample the points. In this case, every point is assigned an importance value, defined as its estimated contribution to a given shading pixel. This results in an importance function, which we can then use to draw samples from $\mathcal{S}$. Specifically, we define importance as a point's projected solid angle to a shading pixel. Intuitively, this ensures that points subtending large solid angles are more likely to be selected for projection, while those subtending small solid angles are selected less frequently.

3.2. Importance-Driven Point Projection

Clustering Points. Since the importance function has to be evaluated on the fly, computing it for all points is impractical. Instead, we partition the points into clusters, and assign a single importance value per cluster. Points within each cluster are treated with equal importance. This essentially approximates the importance function as piecewise constants, greatly reducing the evaluation and sampling cost. This approach is similar to [WÅ09], where they cluster lights based on the points' diffuse radiance energy. Unlike them, we do not consider the points' radiance values during clustering, because our points represent both illumination and geometry, thus a point that carries no emitted radiance may still be important as it can occlude other points. The partitioning of points can use k-means or any existing spatial clustering algorithm. By default we create 512 clusters. Once created, the clusters are treated independently.

Evaluating Per-Cluster Importance. Next, we need to estimate the importance of each cluster. We do so by drawing N_{rnd} random points from each cluster, and computing the sum of their individual importance, defined as the projected solid angle of the point:

$$p_k = \sum_{\mathbf{s}} \frac{\max(\cos\theta_{\mathbf{p}}, 0)\cdot|\cos\theta_{\mathbf{s}}|}{|\mathbf{s}-\mathbf{p}|^2}\Delta A_{\mathbf{s}} \qquad (1)$$

where $\mathbf{p}$ is the a shading pixel being considered, p_k is the estimated importance for cluster $\mathcal{C}_k$, $\mathbf{s}$ is a random point selected from $\mathcal{C}_k$, $\Delta A_{\mathbf{s}}$ is the point's delta surface area. See Figure 2(a)(c) for illustrations. Note that the absolute value of $\cos\theta_{\mathbf{s}}$ is taken because a point facing away from $\mathbf{p}$ still contributes importance, as it can cast indirect shadows. We typically use $N_{rnd} = 4$ random points per cluster, which works well in practice, and is fast to compute.

Importance Sampling. Once we have the per-cluster importance p_k, we normalize it to a PDF, and further convert it to a CDF. To draw a sample, we generate a uniform random number, and use binary search to find where it falls in the CDF. This selects a cluster. Finally we pick a random point from that cluster as a final sample, as every point within the cluster is treated equally.

Projection and Adaptive Splat Size. Once a point is selected, we project it to the microbuffer, and splat its color if the center of the splat succeeds a depth test. To project,

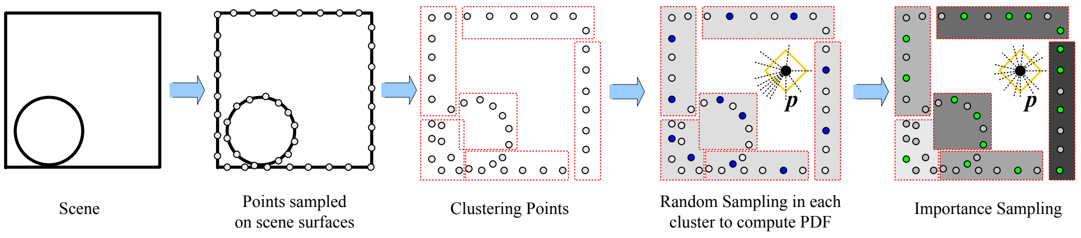

Scene
Points sampled on scene surfaces
Clustering Points
Random Sampling in each cluster to compute PDF
Importance Sampling
p
p

Figure 3: *Algorithm overview. Scene surfaces are sampled as many points, which are then clustered using k-means. At a shading pixel* **p***, we estimate each cluster's importance by drawing random points (blue dots) from the cluster; we then treat the importance as a PDF to draw importance samples (green dots). Both random and importance samples are projected into the microbuffer (yellow diamond). The shade of each box indicates cluster importance.*

we map the line-of-sight direction ($\mathbf{s}-\mathbf{p}$) to the microbuffer using hemi-octahedral mapping. Specifically, we transform ($\mathbf{s}-\mathbf{p}$) to the local coordinates at the shading pixel. Denote this transformed direction ω. We then compute:

$$\omega' = \frac{\omega}{|\omega.x|+|\omega.y|+|\omega.z|}, \quad \text{and} \begin{cases} t_x = \frac{1-\omega'.x+\omega'.z}{2} \\ t_y = \frac{1-\omega'.x-\omega'.z}{2} \end{cases} \tag{2}$$

The resulting $[t_x,t_y] \in [0,1]^2$ are the normalized 2D coordinates corresponding to a pixel location in the microbuffer.

The splat size (the number of microbuffer pixels covered by the projection) has to be carefully computed in order to minimize holes. Assume that a sample $\mathbf{s}$ comes from cluster $\mathcal{C}_k$: since an expected number of $N_s \cdot p_k$ samples will be drawn from this cluster, each sample shares $\frac{1}{N_s \cdot p_k}$ of the total surface area of the cluster. Therefore, we can estimate the solid angle represented by $\mathbf{s}$ as:

$$\Omega_\mathbf{s} = \frac{|\mathcal{C}_k| \cdot \Delta A_s}{N_s \cdot p_k} \cdot \frac{|\cos\theta_\mathbf{s}|}{|\mathbf{s}-\mathbf{p}|^2} \tag{3}$$

We further divide $\Omega_\mathbf{s}$ by the delta solid angle that the microbuffer pixel at $[t_x,t_y]$ represents, and the result gives the total number of pixels covered by the splat. We clamp the number to 1 if it is less than 1. Finally, the splat color and depth value are both written to the microbuffer in a square region covered by splat size.

Discussion. Note that the cluster importance p_k appears in the denominator of Eq. 3. This make sense intuitively, since a cluster with small importance is less likely to be sampled, thus any sample drawn from it must represent a larger support area. After projection, however, the solid angles represented by all samples should be roughly equal. Refer to Figure 4 for illustrations. Note that a pixel's nearby clusters are likely to obtain higher importance values, thus more samples, whether they carry illumination radiance or cast indirect shadows, will be drawn from them.

Glossy Importance. Because we estimate importance using point sampling, it is possible to include other factors. For example, for a glossy shading point, we should give higher importance to clusters that fall close to it reflection directions. This can be achieved by adding a BRDF term f_r to

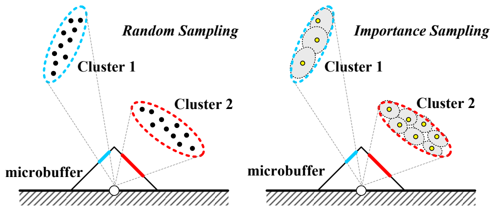


Figure 4: *At random sampling, cluster 1 is found to have smaller importance than cluster 2, thus is sampled less frequently during importance sampling. A sample drawn from cluster 1 will represent a larger support area, but the project solid angle will be similar to a sample drawn from cluster 2.*

Eq. 1, which now becomes:

$$p_k = \sum_\mathbf{s} \frac{\max(\cos\theta_\mathbf{p},0)\cdot|\cos\theta_\mathbf{s}|}{|\mathbf{s}-\mathbf{p}|^2} f_r(\mathbf{s}\to\mathbf{p},\omega_o)\Delta A_\mathbf{s} \tag{4}$$

Refer to Figure 2(b). In addition, we can further multiply it by the strength of the glossy radiance at the source point $\mathbf{s}$. This allows us to efficiently include glossy-glossy reflection paths. Note, however, as we use a small set of random points per cluster to sample the importance, this approach is only suitable for moderately glossy BRDFs, such as Phong with exponent less than 50. Highly glossy BRDFs require proper filtering to avoid aliasing, or should ideally be handled with an entirely different approach.

Summary. Figure 3 summarizes the steps of our algorithm. For efficiency, the points drawn from the random sampling step and importance sampling step are both projected into the microbuffer. This makes better use of the computation already required to evaluate the random samples. The splat size for a random point can be computed from Eq. 3 accordingly, with p_k equal to the inverse of the number of clusters.

In practice, we are typically given a fixed number of **total** sample budget. Thus we need to decide how many of them should be allocated to the random sampling step (which evaluates the importance function) vs. the subsequent importance sampling step. Clearly an imbalanced allocation to either will cause the method to degrade to random sampling. We have found through experiments that an equal alloca-

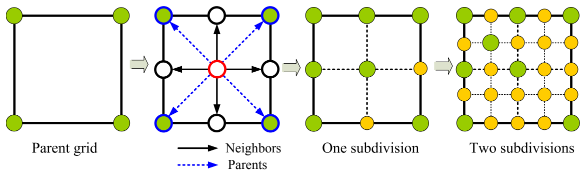


Figure 5: *Adaptive sampling. Green color indicates sampled pixels; orange color indicates interpolated pixels.*

tion of samples for both achieves the best rendering result in practice. Figure 7 provides a validation.

3.3. Image-Space Adaptive Sampling

Even though importance point projection is efficient, computing it for every shading pixel is still quite expensive, especially for high-resolution, anti-aliased (AA) images. Currently we can render a 512^2 image at every pixel (no AA) in $3 \sim 4$ seconds. To provide interactive rendering feedback, we need to further reduce the spatial sampling cost while the user manipulates the scene. This can be done by performing final gathering only at a subset of pixels. The sparsely sampled pixels are then used to reconstruct the other pixels using joint bilateral upsampling [SGNS07]. To select sample pixels, a standard way is to use a regular grid, such as 4×4. While this works well for simple scenes, it is often necessary to perform further sampling around edges and places with fine geometric details. See Figure 10 for an example.

We propose to compute samples adaptively. To begin, we rasterize a G-buffer, storing each pixel's position $\mathbf{p}_i$ and normal $\mathbf{n}_i$. We then use a top-down subdivision, starting with pixels on an initial 4×4 sampling grid. After these pixels are shaded, we examine pixels on the 2×2 grid. For each pixel on this grid, we compute the following coherence metric:

$$\max_i \left(\frac{|\mathbf{p}_i - \mathbf{p}_c|}{d/2} + \sqrt{2 - 2(\mathbf{n}_i \cdot \mathbf{n}_c)} \right) + \lambda \sum_{j,k} |L_o(\mathbf{p}_j) - L_o(\mathbf{p}_k)|$$

where $\mathbf{p}_c$ and $\mathbf{n}_c$ are the position and normal of the current pixel, i loops over its four neighbor pixels on the grid, and d is the length of the scene's bounding box diagonal. Here j and k are pairs of two out of the four parent pixels on the 4×4 grid, and L_o is the radiance computed at each parents pixel. The first term is inspired by [WWZ*09] and evaluates local geometric changes; the second term evaluates radiance changes. λ weighs the relative importance of the two. We found $\lambda = 8$ to work well practice. Refer to Figure 5 for an illustration of the samples. Note that including the radiance term L_o is important for the cases where high radiance changes are present on surfaces with low geometric changes, such as glossy highlights on a plane.

If the coherence metric is larger than a given threshold $\varepsilon = 0.3$, the current pixel must be sampled; otherwise it will be interpolated from the four parents. We perform this checking independently for every pixel, except for those that are already sampled in previous passes. We then use a parallel list compaction to collect all pixels that require sampling at the current subdivision level, and launch GPU threads to shade these pixels. Next, we launch GPU threads again to interpolate the un-sampled pixels, using joint bilateral upsampling [SGNS07].

We repeat the process for grid size 1×1. When anti-aliasing is enabled, it is further repeated until a desired sub-pixel level is reached. The error threshold ε is scaled by 2 every time we go down a level. Note that because the pixels are checked independently in each pass, even if a pixel is interpolated at a previous pass, it can still be requested for sampling at a later pass. This is important for robustly preserving edges and small details not discovered in the previous passes. Figure 9 shows the adaptive samples selected during convergence. Our results show that the samples can quickly capture edges, as well as areas with strong glossy reflections and under indirect shadows.

4. Implementation Details

We implement our algorithms on the GPU using NVIDIA CUDA 3.2. For random numbers, we precompute a texture storing $4\text{K}\times4\text{K}$ random numbers and reuse them on the fly.

Scene Points. For all scenes we generated 256K points uniformly distributed on the scene surfaces. The number of points is sufficient for our test cases. We have also tried 1M points, which did not produce any observable improvement in the rendering quality. We use a GPU-based Poisson disk sampling algorithm described by [BWWM10] to generate these points. For each point we store its position as 3 floats, and its normal, diffuse color, specular color, and Phong exponent as bytes. We also store its triangle ID and barycentric coordinates, which are used to update the points upon scene manipulation. Unless objects undergo significant deformation, we do not need to re-generate points; instead, we simply update their locations using their barycentric coordinates.

Materials. For simplicity we currently only support diffuse and Phong BRDFs. It is possible to include other BRDFs, as we make no assumption about the specific BRDF model. We also support bump maps and spatially varying BRDF parameters defined using textures. We do, however, restrict the glossiness of the BRDFs to be generally below 50, as highly glossy BRDFs cause artifacts in the sampling process and are better handled using a ray tracing based approach.

Primary Lights. We allow multiple point lights as the primary light sources. For each light we rasterize a 6×512^2 cube shadow map and use it to compute shadowed direct lighting at both the scene points and the shading pixels.

For other types of light sources, such as area lights and environment maps, we can perform stochastic sampling on the light source to estimate the direct lighting radiance at the

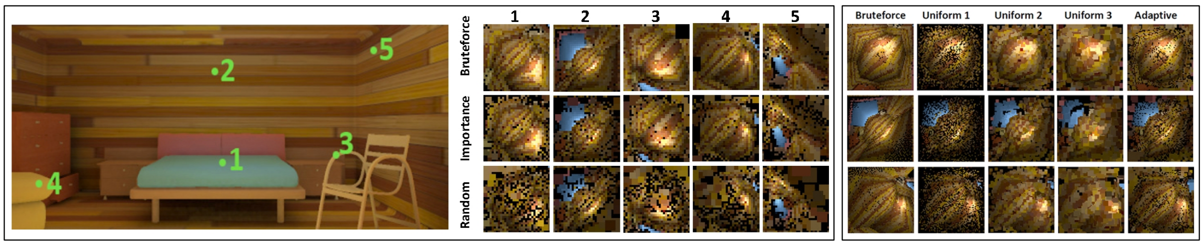


Figure 6: *The block of images on the left: comparisons of microbuffers generated at 5 selected pixels using importance-driven vs. random projection, both with 3K projected points; on the right: comparisons of microbuffers generated with our adaptive splats, non-adaptive (uniform) splats of 1, 2, and 3 pixels, and a brute-force reference solution.*

scene points. But the bigger challenge with them is to evaluate direct lighting at the shading pixels. Various soft shadowing techniques are available, and it remains our future work to include them into our implementation.

Clustering Scene Points. As described in Section 3.2, we partition the scene points into 512 spatial clusters in order to estimate the importance PDF. To do so, we apply straightforward k-means clustering with a fixed number of 8 iterations. The distance between two points is calculated using a metric defined in [WWZ*09], which considers both the position and normal of a point. This metric is inspired by irradiance caching [WRC88]. With 256K scene points and 512 clusters, the k-means can quickly converge on the GPU in a few milliseconds. Thus its overall cost is very small. Note that other spatial clustering schemes can be adopted as well.

Microbuffers. We use 32×32 microbuffers, and compute each microbuffer in a single CUDA block. This allows the entire microbuffer to be stored in shared memory, providing fast access speed. The microbuffer itself requires 3 bytes for storing each of the diffuse and glossy incoming radiance, and 1 ushort for storing depth. The total is 8KB per microbuffer.

The reason to separate glossy from diffuse component is for better image-space interpolation in textured areas. In those areas, direct color interpolation of pixels will lead to loss of texture details. Instead, we interpolate the diffuse and glossy radiance separately, and multiply them with the textured reflectance values to produce the final color. Note that the diffuse and glossy components can use different textures.

To estimate cluster importance, we launch 512 threads, corresponding to 512 clusters. Each thread k independently draws 4 random points from its cluster $\mathcal{C}_k$ and evaluates p_k (Eq.4). These points are immediately projected to the microbuffer. Then a parallel prefix sum is used to build the CDF. Following this, each thread proceeds to draws 4 importance samples using the CDF, and projects them to the microbuffer using splat sizes estimated via Eq. 3. So in total we project (4+4)×512=4K points to the microbuffer. Every time a point is projected to the microbuffer, we use `atomicMin` to update the color and depth values, ensuring correctness upon concurrent writes. Finally we multiply each microbuffer pixel with the BRDF value, and perform a parallel reduction to return the total reflected color.

For robustness when calculating delta solid angles, we use an approximated disk-to-point solid angle formula:

$$\Omega_\mathbf{s} = \frac{2\pi A_\mathbf{s} \cos\theta_\mathbf{s}}{A_\mathbf{s} + 2\pi|\mathbf{s}-\mathbf{p}|^2} \tag{5}$$

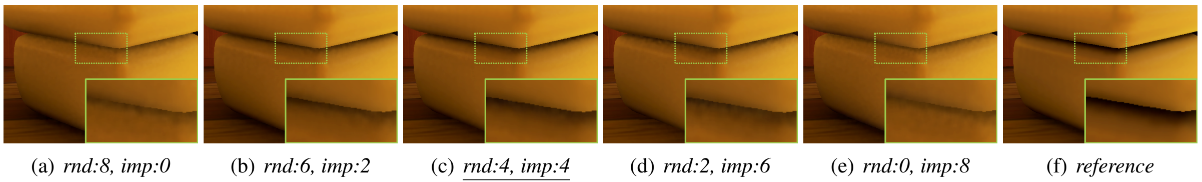
This prevents $\Omega_\mathbf{s}$ from becoming arbitrarily large when $\mathbf{p}$ and $\mathbf{s}$ are very close to each other.

Glossy-Glossy Reflections. Since our approach is based on point sampling, we can achieve glossy-glossy reflections by performing a direct lighting calculation at a scene point when it is requested for projection. Clearly the computation cost depends on the number of primary lights. In practice, we assume that there are no more than 4 primary light sources and hence we can store the glossy reflected lobes together with the scene points. To do so, we run a separate pass that calculates shadowed direct lighting on the scene points, including a diffuse radiance and up to 4 glossy reflected lobes. For each lobe we store its center direction and the Phong exponent. These will then be used during point projections.

Multi-Bounce Indirect Lighting. We enable multi-bounce indirect lighting by treating the scene points as shading points, and use the same microbuffer algorithm to update the diffuse radiance at each point. This allows us to include an additional bounce of indirect illumination in the final shading. Note, however, we do not update the glossy reflection lobes at the scene points, as doing so would cause the number of glossy lobes to increase exponentially. Instead, we only update the diffuse radiance of the scene points. Nonetheless, this approach provides satisfactory multi-bounce illumination results in most cases.

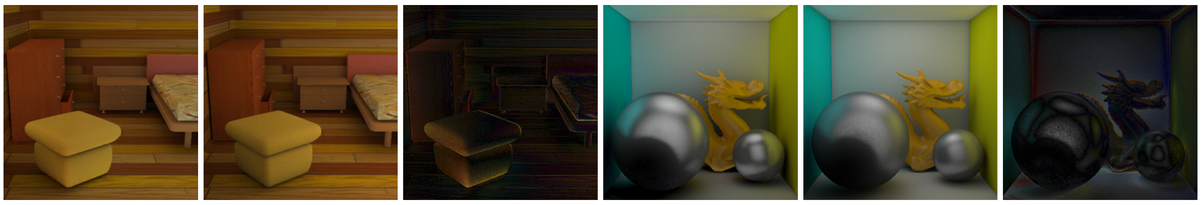
Progressive Rendering. Using image-space adaptive sampling, we can easily enable progressive rendering to improve the user interaction speed. During adaptive sampling, the user can manipulate any part of the scene, and the current adaptive sampling step will be interrupted if the user starts to edit the scene. As soon as the user stops moving, the program will continue to compute the remaining pixels, allowing the full indirect light buffer to be filled in over time. Typically the rendering quality converges in a second, while full frame rendering (i.e. evaluated at every pixel) takes $3 \sim 4$ seconds.

When higher quality rendering is desired, we enable super-sampling to provide anti-aliasing. In this case, the



(a) *rnd:8, imp:0* (b) *rnd:6, imp:2* (c) *rnd:4, imp:4* (d) *rnd:2, imp:6* (e) *rnd:0, imp:8* (f) *reference*

Figure 7: *Comparing allocation of samples for the initial random sampling (rnd) vs. the subsequent importance sampling (imp) step. Here the total sample budget is 8 per thread. The renderings only show indirect lighting component. We choose an equal allocation of both as it typically provides the best result, shown in (c). An imbalanced allocation can degenerate the algorithm towards random sampling, reducing the sampling efficiency and rendering quality. In particular, (a) is equivalent to stratified uniform random sampling with 512 stratas, and (e) is equivalent to un-stratified uniform random sampling.*



(a) Ours (b) Reference (c) 2× diff (d) Ours (e) Reference (f) 2× diff

Figure 8: *Comparison of indirect lighting result computed using our method vs. a ray traced reference. For each example we show a 2× difference image to highlight the places of errors.*

sampling will proceed to subpixel level. Due to adaptive sampling, the most important pixels (usually those around edges, indirect shadows, and glossy surfaces) are selected and shaded first. Thus the rendering usually converges quickly. This allows the rendering cost to grow sublinearly with respect to the total number of super-sampled pixels. To keep track of the status of each pixel, the alpha component is set to 1 for sampled pixels and 0 for interpolated pixels. The program will then overwrite interpolated pixels in subsequent sampling passes.

Post-process Filter. We apply a 5×5 joint bilateral filter on the computed indirect lighting buffer as a post-process step. This generally reduces stochastic sampling noise and works very effectively for diffuse scenes. For scenes with many glossy objects, the sampling noise is usually more pronounced, but we cannot use a larger filter to attenuate noise because that would blur out reflection details. Instead, we rely on anti-aliasing to provide a higher quality image.

5. Results and Discussions

Our results are tested on a PC with Intel Core i7-920 CPU and NVIDIA 480 GTX graphics card. Unless specified otherwise, all images and videos are captured at a default resolution of 512×512 with no anti-aliasing. For high-quality rendering, we turn on 2×AA, which provides nicer quality but at slower performance.

Performance. By default we turn on progressive rendering mode to provide better user interaction. The rendering speed is about 2∼3 fps, and the full frame rendering (i.e. all pixels are shaded, none interpolated) is achieved in 3∼4 seconds. Figure 1 shows three example images. Note the realistic surface reflections and the indirect shadows.

With 2×AA, the frame rate drops to 1 fps. Note that due to our adaptive sampling, the image quality typically converges in a few seconds, and hence full frame rendering is usually not necessary to obtain a high-quality rendering. Since we use point-based illumination, the rendering performance is generally insensitive to the scene complexity, although the indirect shadows can take longer to converge in scenes with high depth complexity.

Microbuffers. We first examine the quality of the microbuffers generated using our method. Our reference is a brute force solution that projects all scene points into the microbuffer. In Figure 6 left, we show 5 selected pixels from the bedroom scene. For each point, we compare our importance-driven point projection vs. uniform random point projection. Both are computed by drawing 3K samples from the scene points. Comparison results are shown in the middle image. Note how the importance-driven method better matches the reference, while the random method leaves many gaps that need to be filled with large adaptive splats.

To see how effective the adaptive splats are, in Figure 6 right, we compare microbuffers generated using adaptive splats vs. uniform splats of 1, 2, and 3 pixels. Small uniform splats (e.g. 1) do not cover the microbuffers well, leaving many holes; large uniform splats (e.g. 3) overfill the buffer, causing one splat to spill to adjacent pixels. In contrast, adaptive splats can effectively overcome these issues.

In Figure 7 we examine the rendering quality when allocating different number of samples for the initial random sampling step vs. the subsequent importance sampling step.

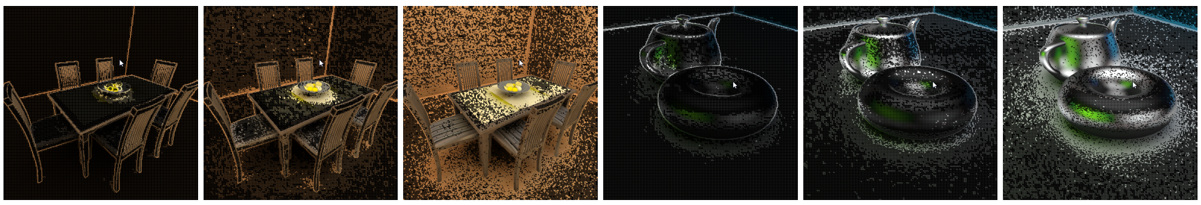
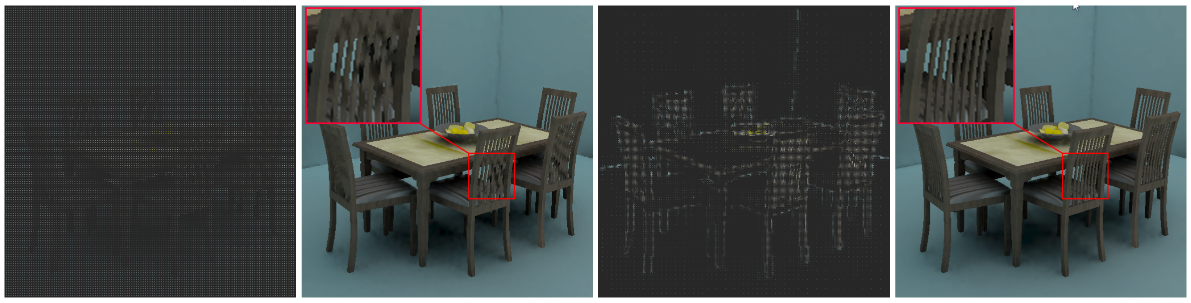


Figure 9: *Image adaptive samples selected at different time stamps as the rendering progresses. Note how the samples quickly capture areas of depth discontinuities and radiance changes (such as indirect shadows and glossy highlights).*



(a) Regular 4×4 (16,384 samples) (b) Reconstructed image (c) Adaptive (15,182 samples) (d) Reconstructed image

Figure 10: *This example compares reconstruction quality using a regular 4×4 sampling vs. our adaptive sampling method. (a) shows the regular sampling grid which contains 16,384 samples; (c) shows adaptive sampling grid with 15,800 samples. (b) and (d) show the reconstructed images, both interpolated using joint bilateral upsampling. Clearly adaptive sampling is more effective at capturing detailed geometric features in this scene, such as the stripes on the chairs. In contrast, regular sampling produces noticeable sampling artifacts, and leads to temporal aliasing when the view or lighting changes.*

We set the total sample budget to 8 per thread. If too many samples are devoted to the initial sampling, there won't be enough samples to exploit the evaluated importance function; on the other hand, if too few samples are devoted to the initial sampling, the accuracy of the importance function will degrade. We have found that an equal number of both achieves the best result. Note that (a) and (e) are both equivalent to uniform random sampling (one stratified and one un-stratified). This is because (a) draws only random samples per-cluster, and (e) constructs an importance function that's constant everywhere. Also note that as all samples are projected into the microbuffer, the computation cost remains about the same as long as the total sample budget is the same.

Validation. To verify the rendering quality, we used ray tracing instead of microbuffers to generate reference images of two scenes. Results are shown in Figure 8. The bedroom scene contains only diffuse objects, while the Cornell box scene contains both diffuse and glossy objects. We only show indirect lighting. Note that our result looks qualitatively similar to the reference. Some differences are observable, mainly in the tone of the color and the indirect shadows around edges. The color difference is primarily caused by color quantization (i.e. 8-bit), and the indirect shadow difference is primarily caused by the limited resolution of the microbuffers. We also show a 2× difference image on the right, which highlights the places of errors.

Image adaptive sampling. Our adaptive sampling algorithm considers both local geometric changes and luminance changes, thus it efficiently budgets samples in areas of sharp features or strong radiance changes. In Figure 9 we show the adaptive sample points selected at different time stamps during the rendering. Note how the samples quickly capture depth discontinuities as well as indirect shadows and glossy reflections, making the sampling process more efficient.

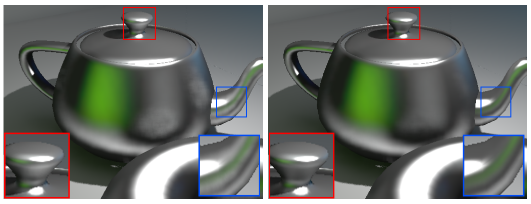


Figure 11: *Comparisons of enabling vs. disabling glossy-glossy reflections. The inlets show zoomed in views of two spots with self-reflections on the teapot.*

Figure 10 shows an example that compares our adaptive sampling method with a straightforward regular 4×4 sample pattern as used in in [SGNS07, REG*09]. For this example, we set the adaptive sampler to start from an 8×8 initial grid and progressively refines using the algorithm described in Section 3.3. Both images (indirect lighting only) are reconstructed with joint bilateral upsampling. We found that while a regular sample pattern works well for smooth geometry, it often causes aliasing artifacts when fine geometric details are present, such as the stripes on the chair models. These artifacts become more pronounced when the view or lighting changes. In contrast, our adaptive sampling method can

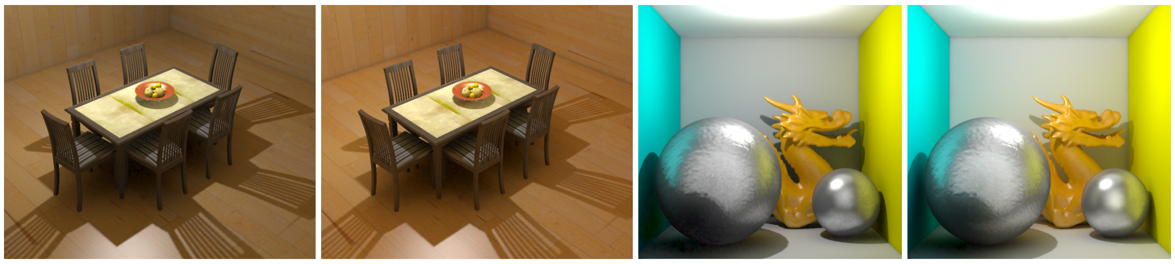


Figure 12: *The dining room scene and the Cornell box scene. Here we compare global illumination with one and two bounces of indirect lighting. Note how the additional bounce adds more color bleeding and brightens up the indirect shadowed regions.*

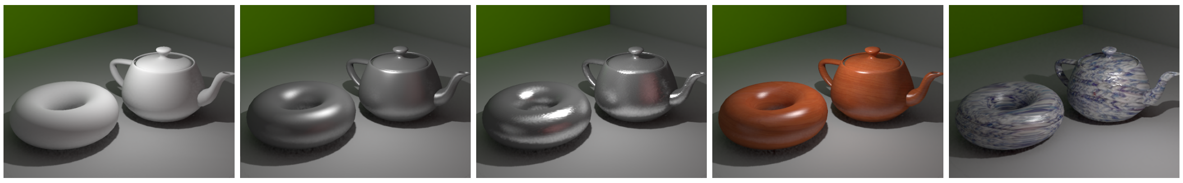


Figure 13: *The teapot-torus scene edited in real-time with several different materials, textures and bumpmaps.*

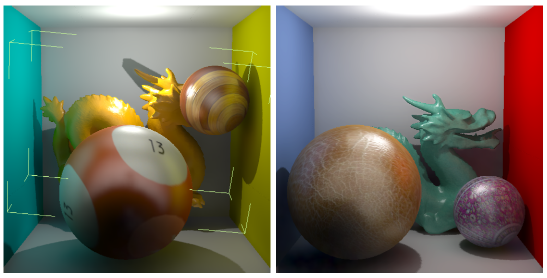


Figure 14: *This example shows run-time scene manipulation, including object, lighting, and material changes.*

quickly snap to areas that demand more sampling, thus is more efficient at the same or even less number of samples.

Multi-bounce Indirect Lighting. We enable multiple diffuse bounces by applying the same final gathering algorithm on the scene points. The final bounce of reflection (at shading pixels) is still glossy. It generally takes several seconds to update the diffuse radiance at all scene points. Once it's done, the user can change viewpoints at will, but moving the light source, objects, or changing material parameters will incur a new round of scene points update. Figure 12 shows our results for the dining room scene and the Cornell box scene. Note how the additional bounce adds more color saturation, and brightens up the indirect shadowed regions.

Glossy-glossy Reflections. As we allow the scene points to carry glossy reflected lobes, we enable glossy-glossy reflection effects. Figure 11 shows a comparison between renderings with the glossy lobes at the scene points enabled vs. disabled. Note the differences in the glossy highlights reflected from the teapot, especially the self-reflections.

Scene Manipulation. Our algorithm supports dynamic BRDF editing with bump maps and spatially varying BRDF parameters defined by textures. Figure 13 shows the teapot-torus scene edited in real-time with several different materials. We also support online manipulation of scene objects. Upon manipulation, the direct lighting radiance at every scene point will be updated, and we re-cluster the points. These steps take less than 12ms. Figure 14 shows two snapshots captured during an editing session where the objects, primary light, and materials have all been edited.

Comparisons to [Chr08] and [REG*09]. Our method differs from [Chr08] and [REG*09] mainly in that we replace the hierarchical point traversal with importance point sampling and projection, which provides compatible efficiency but is conceptually simpler and easier to implement on the GPU. Using point sampling also makes it possible to account for glossy source radiance at the scene points. In addition, as shown in Figure 10, our adaptive image sampling algorithm is better suited for scenes with fine geometric details.

On the other hand, our method is prone to stochastic sampling noise, which is not an issue with hierarchical point traversal. This may cause severe artifacts when the scene's depth complexity increases, particularly if a cluster contains points from multiple layers of geometry. In this case we may need to increase the number of clusters, or improve the clustering algorithm to avoid creating clusters that contain complex geometry. Also, we have not yet included the BRDF-warped microbuffers [REG*09], which can more efficiently handle glossy reflections at the shading points. Finally, we currently do not employ a separate raytracing pass, as suggested in [REG*09], to compensate for near-field contributions. As seen from Figure 7, our sampling method is fairly efficient at capturing near-field illumination and occlusions. Nonetheless, using raytracing compensation can further improve the efficiency of our algorithm.

6. Limitations and Future Work

To summarize, we have presented an efficient method using importance point projection for GPU-based final gathering.

Our method provides a viable alternative to existing work. Our method is not real-time yet, so it is not immediately applicable to 3D games. However, it serves as a practical tool for design applications, as it provides realistic rendering feedbacks in just a few seconds.

Our work has several limitations that remain to be addressed in future work. First, like other point-based methods, when dealing with highly glossy materials, the number of scene points can quickly become a limiting factor that leads to reflection aliasing artifacts. We plan to incorporate techniques such as [DKH*10], which combine local and global virtual lights to more efficiently handle glossy reflections. Second, scenes with complex geometry will reduce the sampling efficiency of our algorithm, leading to spatial and temporal aliasing artifacts. We plan to address this issue by separating the illumination into near-field and far-field components, and use different approaches to handle each component. Finally, we plan to study how importance point projection can be used to simulate other effects such as translucency, participating media, and hair rendering.

Acknowledgments We would like to thank EGSR anonymous reviewers for valuable comments and suggestions. This work was supported by NSF grant CCF-0746577.

References

[BGH05] BURKE D., GHOSH A., HEIDRICH W.: Bidirectional importance sampling for direct illumination. In *Proceedings of Eurographics Symposium on Rendering* (2005), pp. 147–156. 2

[Bun05] BUNNEL M.: Dynamic ambient occlusion and indirect lighting. *GPU Gems 2 2* (2005), 223–233. 2

[BWWM10] BOWERS J., WANG R., WEI L.-Y., MALETZ D.: Parallel poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graph. 29* (2010), 166:1–166:10. 3, 5

[CAM08] CLARBERG P., AKENINE-MÖLLER T.: Practical product importance sampling for direct illumination. *Computer Graphics Forum 27*, 2 (2008), 681–690. 2

[CETC06] CLINE D., EGBERT P. K., TALBOT J. F., CARDON D. L.: Two stage importance sampling for direct lighting. In *Proceedings of Eurographics Symposium on Rendering* (2006), pp. 103–113. 2

[Chr08] CHRISTENSEN P.: *Point-based approximate color bleeding*. Tech. rep., Pixar Tech. Memo #08-01, 2008. 1, 2, 9

[CJAMJ05] CLARBERG P., JAROSZ W., AKENINE-MÖLLER T., JENSEN H. W.: Wavelet importance sampling: efficiently evaluating products of complex functions. *ACM Trans. Graph. 24*, 3 (2005), 1166–1175. 2

[DKH*10] DAVIDOVIČ T., KŘIVÁNEK J., HAŠAN M., SLUSALLEK P., BALA K.: Combining global and local virtual lights for detailed glossy illumination. *ACM Trans. Graph. 29* (2010), 143:1–143:8. 2, 10

[HKWB09] HAŠAN M., KŘIVÁNEK J., WALTER B., BALA K.: Virtual spherical lights for many-light rendering of glossy scenes. *ACM Trans. Graph. 28* (2009), 143:1–143:6. 2

[HPB07] HAŠAN M., PELLACINI F., BALA K.: Matrix row-column sampling for the many-light problem. *ACM Trans. Graph. 26*, 3 (2007), 26. 1, 2

[Jen01] JENSEN H. W.: *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., 2001. 2

[KD10] KAPLANYAN A., DACHSBACHER C.: Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of I3D* (2010), pp. 99–107. 2

[Kel97] KELLER A.: Instant radiosity. In *Proc. of SIGGRAPH '97* (1997), pp. 49–56. 1, 2

[KFB10] KŘIVÁNEK J., FERWERDA J. A., BALA K.: Effects of global illumination approximations on material appearance. *ACM Trans. Graph. 29* (2010), 112:1–112:10. 2

[ML09] MCGUIRE M., LUEBKE D.: Hardware-accelerated global illumination by image space photon mapping. In *Proc. of HPG '09* (2009), pp. 77–89. 2

[NED11] NOVÁK J., ENGELHARDT T., DACHSBACHER C.: Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *ACM I3D* (2011), pp. 119–124. 2

[NPW10] NICHOLS G., PENMATSA R., WYMAN C.: Interactive, multiresolution image-space rendering for dynamic area lighting. *Computer Graphics Forum 29*, 4 (2010), 1279–1288. 2

[PDC*03] PURCELL T. J., DONNER C., CAMMARANO M., JENSEN H. W., HANRAHAN P.: Photon mapping on programmable graphics hardware. In *Proc. of Graphics Hardware* (2003), pp. 41–50. 2

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph. 28*, 5 (2009), 1–8. 1, 2, 8, 9

[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 (2008), 1–8. 2

[SGNS07] SLOAN P.-P., GOVINDARAJU N. K., NOWROUZEZAHRAI D., SNYDER J.: Image-based proxy accumulation for real-time soft global illumination. In *Proc. of Pacific Graphics 07* (2007), pp. 97–105. 3, 5, 8

[THGM11] THIEDEMANN S., HENRICH N., GROSCH T., MÜLLER S.: Voxel-based global illumination. In *ACM I3D* (2011), pp. 103–110. 2

[TPWG02] TOLE P., PELLACINI F., WALTER B., GREENBERG D. P.: Interactive global illumination in dynamic scenes. *ACM Trans. Graph. 21*, 3 (2002), 537–546. 3

[WÅ09] WANG R., ÅKERLUND O.: Bidirectional importance sampling for unstructured direct illumination. *Comput. Graph. Forum 28*, 2 (2009), 269–278. 2, 3

[WFA*05] WALTER B., FERNANDEZ S., ARBREE A., BALA K., DONIKIAN M., GREENBERG D. P.: Lightcuts: a scalable approach to illumination. *ACM Trans. Graph. 24*, 3 (2005), 1098–1107. 1, 2

[WNLH06] WANG R., NG R., LUEBKE D., HUMPHREYS G.: Efficient wavelet rotation for environmentmap rendering. In *Proc. of Eurographics Symposium on Rendering* (2006), pp. 173–182. 3

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *Proc. SIGGRAPH '88* (1988), pp. 85–92. 3, 6

[WWZ*09] WANG R., WANG R., ZHOU K., PAN M., BAO H.: An efficient GPU-based approach for interactive global illumination. *ACM Trans. Graph. 28*, 3 (2009), 1–8. 2, 5, 6

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