Reflectance Filtering for Interactive Global Illumination in Semi-Glossy Scenes

David Maletz John Bowers Rui Wang
University of Massachusetts Amherst

Abstract

We introduce a method using reflectance filtering to simulate interactive global illumination in scenes that contain semi-glossy materials. We adopt a many-light approach to sample illumination onto a set of VPLs spread over the scene. We build a kd-tree of the VPLs, allowing for fast hierarchical integration of unshadowed indirect lighting at a shading point. Unlike previous approaches which assume the VPLs to be diffuse, we explicitly model the directional radiance of each VPL. We then use reflectance filtering to update the directional contributions of all nodes in the kd-tree. This allows for efficient handling of glossy to glossy reflection paths in semi-glossy scenes. We also propose using min-max tree cuts to further improve the efficiency of a many-light approach. To reduce spatial sampling cost, we introduce a new hierarchical image-space adaptive method, which combines level-independent sampling with bilateral interpolation to robustly preserve fine details as well as sharp edges. Our method performs entirely on the GPU, and thus is suitable for fully dynamic scenes.

Keywords: Interactive global illumination, glossy BRDFs, reflection filtering, many lights, spatial adaptive sampling, GPU

1 Introduction and Related Work

Interactive simulation of global illumination with complex reflection effects has been a long-standing challenge in computer graphics research. As described by the rendering equation [Kajiya 1986], the reflected light from any surface point potentially contributes illumination to all other surface points, leading to expensive computation. With the rapid progress in the GPU's computation power over the past several years, it is now possible to render multi-bounce diffuse and perfectly specular reflections at interactive rates [Wang et al. 2009]. However, existing techniques are largely focused on diffuse indirect lighting, and it remains a challenge to efficiently simulate scenes that contain many glossy materials. While we could use Monte Carlo path tracing or photon mapping to achieve an accurate solution, these methods require a long time to eliminate noise if glossy materials are present in the scene.

A large body of recent work in interactive global illumination adopts the idea of gathering from many lights [Walter et al. 2005; Hašan et al. 2007; Ritschel et al. 2009]. They first sample the scene's indirect illumination as many point lights called virtual point lights (VPLs); they then approximate indirect lighting by estimating the contributions from all VPLs to each shading point. These approaches are fast and suitable for parallel computation on the GPU. However, the VPLs are typically assumed to be diffuse in order to avoid keeping track of their directional radiance. While this still allows for glossy reflections in the final bounce, it cannot model glossy-to-glossy interreflection paths. A recent technique proposed by [Hašan et al. 2009] uses virtual spherical lights (VSLs) to overcome this limitation. It exploits GPU shadow mapping and provides high-quality rendering for glossy scenes; unfortunately, rendering requires a few minutes per frame.

In this paper we propose a new method to simulate global illumination in scenes that contain semi-glossy materials. We adopt a many-light approach to sample illumination onto a set of VPLs spread over the scene. We then build a kd-tree of the VPLs that represents clusters of lights. This allows for fast hierarchical integration of unshadowed indirect lighting at a shading point. Our main contribution is that we explicitly model the directional radiance of each point light with a single reflectance lobe. This is in contrast to previous work which assumes diffuse points lights. We then aggregate the directional radiance to interior nodes of the tree using *reflectance filtering*. This is computed via a simple convolution method [Toksvig 2005]. As a result, every node in the tree stores both a diffuse and a glossy radiance component. This allows us to easily extend standard many-light approaches to handle glossy-to-glossy reflection paths with little additional overhead.

Figure 1: *A screenshot captured at 7 fps with 768×512 res., glossy materials, 2 bounces of indirect lighting, and 2×2 supersampling.*

To compute indirect lighting at a shading point, we traverse the light tree in depth-first order and accumulate illumination hierarchically using a solid angle criterion. We follow a fixed traversal order to avoid maintaining a stack on the GPU. The standard traversal can be quite wasteful at the top levels of the tree, as these levels are almost always subdivided given their potentially large solid angles. We solve this problem by computing a min cut of the tree based on illumination importance. This enforces the traversal to begin on the min cut, skipping nodes above the min cut. Similarly we also compute a max cut of the tree to prevent the traversal from going arbitrarily deep into the tree. As our results show, the min-max cuts approach can greatly reduce thread divergence, improving the computation speed with minimal loss in rendering quality.

Finally, to avoid computing indirect lighting for every shading point, we introduce a new image-space adaptive method to reduce spatial sampling cost. We perform image subdivision in a top-down manner. Unlike traditional approaches, we check coherence at each level independently, and incorporate bilateral method [Sloan et al. 2007] to robustly preserve fine details as well as sharp edges. This approach also allows for efficient anti-aliasing without significantly increasing spatial sampling cost.

Our method is implemented entirely on the GPU using CUDA. Figure 1 shows an example of our rendering simulated at 7 fps. As no precomputation is required, our method is suitable for dynamic scenes. We currently ignore visibility in indirect lighting, but it's possible to incorporate a fast visibility approximation such as [Ritschel et al. 2008, 2009] to enable shadowed indirect lighting.

1.1 Related Work

Researchers have studied global illumination algorithms based on the rendering equation [Kajiya 1986] for decades. An overview of offline rendering techniques can be found in standard textbooks such as [Dutré et al. 2006].

Many-light Rendering. To achieve interactive global illumination, one common idea is to use a many-light formulation, which generates a large number of virtual point lights (VPLs) to sample indirect illumination, and shade visible surface points from these lights. Instant radiosity [Keller 1997] is the first method that introduced this idea, but its cost is linear to the number of VPLs as well as shading points. Many improvements have been studied since then. Lightcuts [Walter et al. 2005] use hierarchical summation and spatial adaptive sampling to improve the computation to sublinear cost. Matrix row-column sampling [Hašan et al. 2007] sparsely samples a large matrix representing the contribution from each VPL to every shading point. It exploits the low-rank property of such a matrix. Imperfect shadow maps [Ritschel et al. 2008] uses approximate visibility to accelerate rendering, and a recent improvement [Ritschel et al. 2009] uses micro rasterization buffer warped by BRDF importance to more accurately account for glossy reflections in the final bounce, while still maintaining interactive speed.

Existing many-light methods naturally fit the GPU rasterization model to achieve impressive rendering speed and quality. However, they typically assume VPLs to be diffuse, thus cannot model glossy-to-glossy interreflection paths. Consequently, if the scene materials have largely no diffuse components, the estimated indirect lighting will be extremely dark, leading to substantial energy loss. This is the case even if the materials have strong semi-glossy components, such as shown by Figure 1. A recent technique proposed by [Hašan et al. 2009] uses virtual spherical lights (VSLs) to overcome this limitation, by keeping track of photons instead of diffuse point lights. It provides high-quality rendering for glossy scenes, but is aimed for offline simulation where rendering takes a few minutes per frame.

A related work is the reflective shadow maps [Dachsbacher and Stamminger 2005], which treat shadow map pixels projected from a single primary light as VPLs. Using image-space gathering, this method enables fast one-bounce indirect lighting. However, it is still limited to diffuse materials, and using shadow maps to parameterize VPLs makes it difficult to handle multiple primary lights.

In contrast to existing methods, we explicitly model the directional radiance of each VPL as a single reflectance lobe, and we use reflectance filtering to aggregate the directional radiance to clusters of lights. This allows us to efficiently handle glossy-to-glossy reflection paths, and more accurately simulate scenes where the materials have small diffuse components. The principle of this idea is similar to [Sillion et al. 1991] which uses spherical harmonics (SH) to model the emitted directional radiance. Although SH makes filtering easier due to its linearity, it requires many terms to model even moderately glossy materials.

GPU-based Photon Mapping. Photon mapping [Jensen 2001] simulates accurate multi-bounce indirect lighting, but its efficiency can degrade quickly when glossy materials are present. The first GPU-based photon mapping algorithm was introduced by [Purcell et al. 2003] using grid acceleration structures. [Zhou et al. 2008] implemented an efficient GPU-based kd-tree for interactive photon mapping; [Wyman and Nichols 2009] adaptively sample caustics photons to simulate high-quality caustics; [Wang et al. 2009] combine photon mapping with sparse irradiance sampling for global illumination on the GPU; [McGuire and Luebke 2009] propose an image-space photon mapping algorithm for fast global illumination by exploiting both the GPU and CPU. These methods achieve impressive results, but are not efficient for glossy scenes.

Caching and Splatting. Indirect lighting is typically smooth and thus makes a good candidate for adaptive caching. Irradiance caching (IC) [Ward et al. 1988] is a well-known technique that progressively caches diffuse irradiance samples and reuses them along the computation. Radiance caching [Gassenbauer et al. 2009] extends IC by caching directional as well as spatial radiance. These methods typically require sequential insertion of caching points, a step that's not yet suitable for the GPU. [Gautron et al. 2005] propose a GPU-friendly method by splatting radiance samples in image-space; but this method requires iterative refinement of caching samples and is not yet interactive. [Wang et al. 2009] compute irradiance samples in an independent pass to avoid iterations, but they only model diffuse-to-glossy reflections similar to most many-light methods.

Many techniques exploit image-space coherence by performing computation only on a subset of pixels selected via subdivision. Standard top-down subdivision stops whenever a coarser level patch is found coherent. This can easily miss small geometric details that pass coarser level tests but not finer level tests. Instead, [Nichols et al. 2009] introduced a more effective method where they simultaneously check every two levels in a single pass and store the results in an OpenGL stencil buffer. Our method differs in that we check each resolution level independently. Therefore ours is more robust in cases where two or more consecutive levels must all be subdivided. In addition, we use CUDA to avoid relying on the stencil buffer, allowing for better GPU utilization through thread compaction.

To reconstruct from sparse samples, linear interpolation does not work well around sharp edges. Bilateral sampling [Sloan et al. 2007] provides efficient non-linear interpolation by contributing a sample to its neighbors based on their geometric similarities. While this preserves sharp edges, current implementations use straightforward image subsampling (e.g. 4×4) without adaptation [Ritschel et al. 2009], which can easily lead to missing fine details. Our method combines bilateral methods with adaptive sampling to more robustly preserve both fine details and sharp edges.

Multi-resolution Filtering. The idea of multi-resolution filtering has been frequently used in graphics to reduce sampling noise or eliminate aliasing artifacts. [Křivánek and Colbert 2008] and [Yu et al. 2008] both exploit radiance filtering in the illumination to reduce Monte Carlo sampling noise. These methods filter diffuse illumination radiance. [Tan et al. 2005] create resolution dependent reflectance models for level-of-details (LOD) rendering. They model directional reflectance as Gaussian mixtures and store the results in mipmaps. [Han et al. 2007] introduced a frequency space filtering method for LOD rendering of normal maps. They model normal distribution functions as mixtures of vMF distributions, and compute reflectance as a convolution. We exploit the principle of reflectance filtering, but apply it in many-light problems to model directional radiance from a cluster of lights. For simplicity, we model filtered radiance as a single Phong lobe [Toksvig 2005], while it is possible to incorporate mixtures of lobes in future work.

2 Algorithms and Implementation

As in many-light rendering, we assume indirect illumination comes from a large set of VPLs denoted as $\mathcal{S}$. Each VPL in the set is associated with a position, a delta surface area, a normal, and material properties. A VPL can be infinitely far away, in which case it models a directional light with a delta solid angle. We focus on approximating *unshadowed* indirect lighting at a shading point caused by all VPLs $\mathcal{S}$. We assume the direct lighting comes from a few

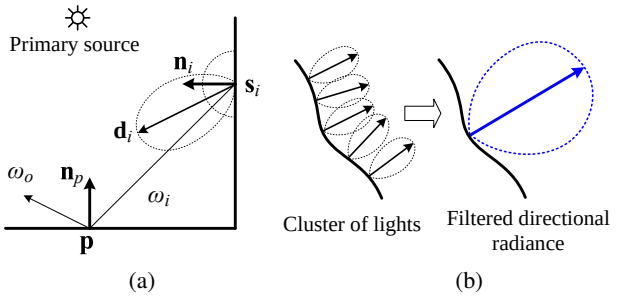


Figure 2: *(a) shows an diagram with our notation. The emitted radiance from a VPL $\mathbf{s}_i$ has both a diffuse and a directional components. (b) shows the filtered directional radiance.*

primary point lights, and is computed directly using GPU shadow mapping. All our scene materials use Phong BRDFs.

We model the directional (specular) radiance emitted at a VPL $\mathbf{s}_i$ as a single Phong distribution:

$$L_s(\mathbf{s}_i, \omega) = l_s \, \frac{\kappa + 1}{2\pi} (\omega \cdot \mathbf{d}_i)^\kappa \tag{1}$$

where $\mathbf{d}_i$ is the center direction of the lobe, l_s is the magnitude, κ is the Phong exponent, ω is an arbitrary direction, and $\frac{\kappa+1}{2\pi}$ is a normalization factor for Phong. For efficiency, we model the diffuse radiance of the VPL separately as $L_d = l_d$. So the total emitted radiance is simply $L_e = L_s + L_d$.

Ignoring visibility, the reflected radiance at a shading point $\mathbf{p}$ in view direction ω_o due to illumination from $\mathcal{S}$ is computed by:

$$L_o(\mathbf{p}, \omega_o) = \sum_{\mathbf{s}_i \in \mathcal{S}} L_e(\mathbf{s}_i, -\omega_i) \, f_r(\omega_i, \omega_o) \, (\omega_i \cdot \mathbf{n}_p) \, \Delta\omega_i \tag{2}$$

where $\omega_i = \mathbf{p} \rightarrow \mathbf{s}_i$ is the direction from the shading point $\mathbf{p}$ to VPL $\mathbf{s}_i$, f_r is the shading point's BRDF, $\mathbf{n}_p$ is the normal, and $\Delta\omega_i$ is the delta solid angle of $\mathbf{s}_i$ observed at $\mathbf{p}$. See Figure 2(a) for an illustration of the notation.

Directly evaluating Eq. 2 is impractical as the number of VPLs can be very large. Instead, we apply hierarchical summation to reduce the computation to sublinear cost. Specifically we build a kd-tree of the VPLs such that the leaf nodes represent individual VPL and interior nodes represent clusters of VPLs. We use iterative k-means to create the tree, and our clustering accounts for both position and normal coherence of the points. For rendering, we traverse the tree from the root node, estimating the solid angle subtended by each node and check it against a given threshold. This determines if the contribution from the current node will be added as a whole and the traversal continues to the next node, or it needs to be subdivided and evaluated recursively.

The idea of using point-based illumination and hierarchical summation has been explored in numerous previous papers such as [Walter et al. 2005; Ritschel et al. 2009]. If the VPLs are assumed diffuse, as in much of previous work, each node of the tree only needs to store a diffuse radiance sum of its children. In our case, however, each node needs to store an additional directional radiance sum. therefore we need a fast way of aggregating directional radiance from the bottom of the tree all the way to the top. To achieve this, we use an efficient reflectance filtering method described below.

2.1 Reflectance Filtering

Overview. For simplicity, we model the total directional radiance of a cluster of VPLs as a single Phong distribution. This works well for semi-glossy materials, which contain no strong high frequencies. For materials with high specularities, it is necessary to use a mixture of Phong distributions instead. While this is a straightforward extension, we leave it for future work.

Figure 2(b) shows an example of reflectance filtering. Each VPL in the cluster contains a radiance lobe centered around a particular direction $\mathbf{d}_i$. When a shading point is sufficiently far away, the cluster can be treated as a single VPL, and the total directional radiance is simply the sum of each individual lobe. This can be thought of as a *filtering* of the individual lobes. Intuitively, the filtered radiance is now a single lobe with a wider angular spread than any individual lobe. We need to find out how to compute the filtered lobe.

When the individual lobes are uniform (i.e. with identical magnitude and exponent), computing the filtered lobe can be formulated as a convolution problem. This is similar to the idea of normal map filtering as introduced in [Toksvig 2005; Han et al. 2007]. Specifically, we model the distribution of lobe center directions $\mathbf{d}_i$ as a *reflection distribution function* (RDF). The RDF describes the distribution density of the lobe centers, and we use $\mathcal{R}(\omega)$ to denote it. As an example, if all the lobes are pointing to the same direction, the RDF will be a delta function. On the other hand, if the lobes are distributed uniformly randomly, the RDF will be a constant. Given the RDF, we can compute the filtered lobe as a convolution: $L_s(\omega) * \mathcal{R}(\omega)$. Here L_s denotes the individual lobe (Eq. 1).

To estimate the RDF, we use a method presented in [Toksvig 2005], which approximates the RDF as a spherical Gaussian. Assume that a cluster contains M lobes, each with a center direction $\mathbf{d}_i$. We first compute the *unnormalized* average of all $\mathbf{d}_i$'s as: $\bar{\mathbf{d}} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{d}_i$. The length of this average vector $|\bar{\mathbf{d}}|$ provides an idea of how widely distributed $\mathbf{d}_i$'s are. Specifically, we estimate the RDF as a spherical Gaussian with the following variance:

$$\sigma^2 = \frac{1 - |\bar{\mathbf{d}}|}{|\bar{\mathbf{d}}|} \tag{3}$$

Note that when all $\mathbf{d}_i$'s are identical, $|\bar{\mathbf{d}}| = 1$, hence $\sigma^2 = 0$. This gives a delta function which conforms with our observation before.

With this RDF definition, computing the filtered lobe becomes estimating the convolution of a Phong lobe with a spherical Gaussian distribution. As shown by [Toksvig 2005], a Phong distribution with exponent κ is actually closely approximated by a Gaussian with variance equal to $\frac{1}{\kappa}$. Note also that the convolution of two Gaussians is another Gaussian (whose variance is the sum of the two input variances), and resulting Gaussian can then be converted back to a Phong distribution. Putting everything together, the filtered lobe is a new Phong distribution with exponent $f_t \cdot \kappa$, where

$$f_t = \frac{1}{1 + \kappa\,\sigma^2} = \frac{|\bar{\mathbf{d}}|}{|\bar{\mathbf{d}}| + \kappa(1 - |\bar{\mathbf{d}}|)} \tag{4}$$

is called a 'Toksvig factor'. This factor scales down the original Phong exponent κ, creating a more diffuse lobe (Figure 2(b)).

Nonuniform Lobes. Note that the individual lobes within a cluster are not completely uniform: they differ in their magnitudes and Phong exponents. Some lobes may also have zero contributions because they represent VPLs that do not contain specular radiance. Ideally we should use a mixture of Phong distributions to model nonuniform lobes. In practice, however, because we assume semi-glossy materials whose BRDFs are relatively smooth, the difference in their exponents does not significantly affect the result of convolution. Therefore we filter nonuniform lobes by treating them as uniform. First, we compute the average exponent $\bar{\kappa} = \frac{1}{M} \sum_{i=1}^{M} \kappa_i$ and the average magnitude $\bar{l}_s = \frac{1}{M} \sum_{i=1}^{M} l_s^i$ of all non-zero lobes.

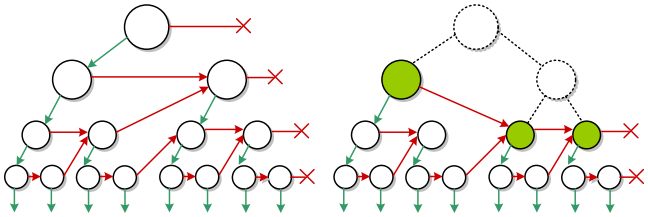


Figure 3: *(a) shows the traversal links. Red arrow indicates a skip link. (b) shows the change of skip links after applying a min cut.*

Second, to cope with lobes with different magnitudes, we compute $\bar{\mathbf{d}}$ as the average of lobe centers, but weighted by each lobe's magnitude (excluding zero lobes). This way, small lobes do not contribute much to the average lobe direction. Finally, we substitute $\bar{\kappa}$ and $\bar{l}_s$ in Eq. 1 to define a uniform lobe, and substitute $\bar{\kappa}$ and $\bar{\mathbf{d}}$ in Eq. 4 to compute the filtered lobe. This turns out to work well in practice.

Illumination Sampling. We generate VPLs by spreading sample points uniformly on the scene surfaces; we then build a kd-tree of the VPLs to form hierarchical clusters. For radiance sampling, we first sample direct illumination radiance at each VPL (leaf nodes). Because we use a Phong BRDF to model all materials in the scene, we can easily compute each VPL's diffuse and specular radiance due to direct lighting from the primary lights. We store the exponent and magnitude of each VPL's specular radiance lobe, and use GPU shadow mapping to account for occlusion in direct lighting.

Next, using the reflectance filtering algorithm described above, we update the filtered radiance at each interior node in the light tree. The diffuse radiance is simply the average diffuse radiance of all VPLs belonging to the cluster. For specular radiance, we compute the VPLs' average Phong exponent $\bar{\kappa}$, magnitude $\bar{l}_s$, and weighted average of their center directions $\bar{\mathbf{d}}$, then use Eq. 4 and 1 to estimate the filtered specular lobe. This process is analogous to building a mipmap of the illumination.

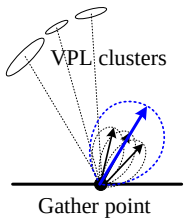
2.2 Illumination Gathering

To compute the indirect lighting at a shading point, we traverse the light tree to gather illumination. To avoid maintaining a stack on the GPU, we use a fixed-order traversal link similar to the method in [Carr et al. 2006]. Specifically, we lay out the tree nodes in a 1D array by following depth-first traversal order. At each node we compute and store a *skip link* which indicates where the traversal should continue to if the current node passes a solid angle test.

Figure 3(a) shows an illustration. A green arrow indicates the next node to visit if the solid angle test of the current node fails, thus the traversal needs to subdivide and visit the children nodes. This follows the natural layout order of the 1D array. A red arrow indicates the next node to visit if the solid angle test succeeds (therefore no need to subdivide). A cross indicates that the traversal is completed. The traversal links can be easily computed on the fly.

When the solid angle test succeeds at a node, we accumulate the illumination from the node by treating the whole cluster as a single VPL. We substitute the average position of the VPLs, the filtered diffuse radiance as well as specular radiance of the cluster to Eq. 2 to estimate the illumination contribution. We repeat this until the traversal of the tree is completed.

Multi-Bounce Indirect Lighting. To compute multi-bounce indirect lighting, we use the current set of VPLs to sample illumination; we then gather illumination at each VPL as a gather point instead of at every shading point. We use the same hierarchical accumulation as above. Note that we need to update the specular radiance lobe of each VPL due to multi-bounce lighting.This can be done by using the same reflectance filtering algorithms as before.



In the figure on the right, consider that at a gather point, each VPL cluster in the tree results in a reflected specular lobe centered around a different direction. We can merge these specular lobes into a single lobe using filtering. To do so, we compute the unnormalized weighted average of all reflected directions, and use Eq. 4 to compute the scale factor for the exponent. Note that each input lobe has the same exponent because they are reflected at the same point. We perform this computation for a number of iterations to include multi-bounce lighting at the VPLs. Finally, we perform a gathering at every shading point to simulate the total indirect lighting.

Min-Max Cuts. Our tree traversal results in different subdivisions for different shading points. While this adaptivity is excellent for accuracy, it also creates several efficiency problems. One is that the top several levels of the tree are almost always subdivided, given their potentially large solid angles. Thus, this portion of the traversal is quite wasteful. The second problem is divergent computation – some shading points only require visiting a few hundred nodes, while others require very deep subdivisions into the tree. This is true even for spatially coherent shading points. This divergent computation is ill-suited for the GPU. One solution to eliminate the divergence is to compute a *global cut* of the tree, forcing every shading point to use the same set of nodes. However, this solution loses the adaptivity and produces significant bias in rendering.

Our solution is to compute min-max cuts of the tree to reduce divergent computation while still preserving the ability for adaptive subdivision. We compute the cuts based on the normal variance of VPLs in a cluster. For example, every node on the min cut has a normal variance equal or below a specified threshold σ_1; the same is true for the max cut but with a different threshold σ_2. To implement the min-max cuts in parallel, we compute each node's normal variance while building the light tree; we then mark each node as *valid* if its normal variance is between σ_1 and σ_2, and invalid otherwise. Next we compact the tree to keep only valid nodes and discard invalid ones. For each valid node, we update its skip link to point to the next valid node that succeeds the current node in depth-first order. Figure 3(b) shows an example. To perform the update, we look at the original skip link: if it points to a valid node, then no change is made; otherwise, we follow its children nodes in standard depth-first order (the natural layout of the tree) until we find a valid node, and change the skip link to that node.

Note that the above parallel algorithm computes the min-max cuts implicitly: it essentially constructs a forest composed of min-cut parents and max-cut leaf nodes, and we never have to build the cuts explicitly. It runs very efficiently on the GPU since every node is processed in parallel.

2.3 Image Adaptive Sampling

Even with hierarchical summation, computing indirect lighting at every shading point is still very expensive. Fortunately the indirect lighting in most scenes tends to be smooth, so we can exploit the spatial coherence by computing indirect lighting at only a subset of screen pixels, reducing the overall computation cost. The sparsely sampled pixels are then used to reconstruct the remaining pixels using bilateral upsampling [Sloan et al. 2007]. Note that although the bilateral approach has been used in a number of interactive global illumination papers, these methods typically use straightforward coarse sampling, such as 4×4 sampling grids. As

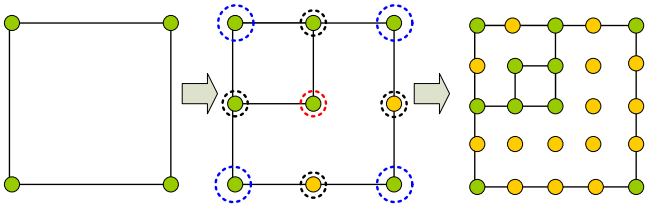


Figure 4: *Adaptive sampling. Green dots: sampled pixels; orange dots: reconstructed pixels using bilateral interpolation.*

a result, they can easily miss fine details that fall below or close to the coarse sampling resolution.

We propose to use adaptive spatial sampling instead. To begin, we rasterize a deferred shading buffer, storing each shading pixel's position $\mathbf{p}_i$ and normal $\mathbf{n}_i$. We use a top-down subdivision of the image, and start with pixels on a coarse sampling grid, typically 16×16. Once these pixels are shaded, we then subdivide the image and examine 8×8 grids: for every pixel on this grid, we evaluate the following coherence error:

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

where i loops over the four nearby grid pixels, $\mathbf{p}_c$ and $\mathbf{n}_c$ are the position and normal of the current (center) pixel, and d is length of the scene's bounding box diagonal. Here j and k are pairs of two out of the four corner pixels on the parent (16×16) grid, and L_o is the luminance of the indirect lighting radiance computed at the parent pixels. In Figure 4, the red dotted line circles the current pixel being checked, the black circles its four nearby pixels at the same level, and the blue circles the four parent pixels.

Intuitively the first term ($\max_i$) gives a conservative error estimate for the position and normal, and the second term gives the error in sampled radiance from the parent level. λ is a normalization factor that determines the relative importance of the two terms. We found $\lambda = 16$ to work well in all our examples. If the coherence error is above a given threshold ($\varepsilon = 0.3$ by default), the current pixel has to be sampled; otherwise the current pixel will be interpolated from the parent later. After this checking, we collect all pixels that require sampling by using a parallel list compaction. We then launch CUDA threads on these pixels to compute indirect lighting. This allows for better utilization of the GPU since no thread will be idle.

We repeat the above process for the remaining grid levels (4×4, 2×2, and 1×1). We scale the coherence error threshold ε by 2 every time we go down one level. Note that the first term in the above equation is important for preserving small geometric details. This is because even if the parent level reports coherent color, this term can still enforce additional sampling if there are significant geometric changes detected at a lower level.

Reconstruction. Once the adaptive sampling step is completed, we reconstruct the image by filling in remaining pixels using bilateral interpolation [Sloan et al. 2007]. Standard bilinear interpolation simply interpolates pixels from samples using bilinear weights. This is problematic around sharp edges. Bilateral interpolation computes the contribution of a sample to a point by additionally considering the geometric similarity between the two. This prevents the interpolation across sharp boundaries.

Our reconstruction algorithm works level by level, starting from the coarsest level. At each level, we fill in pixels at the current level that do not already have a sampled value. This is done using interpolation from the four parents in the higher level grid (note that a pixel that falls on the edge of a higher level grid will only be interpolated from two parent pixels). To interpolate, we use the bilateral upsampling formula from [Sloan et al. 2007], where the weight for a contributing pixel is computed as the product of the standard bilinear weight, the positional weight, and the normal weight. Because we perform interpolation top-down, at each level the parent pixels must all exist, whether they were sampled or interpolated.

2.4 Implementation Details

Generating VPLs. We generate 16K VPLs by directly spreading sample points on the scene surfaces, and use a CPU-based point repulsion algorithm to uniformly distribute the point. This step is similar to [Cheslack-Postava et al. 2008]. While we could also use reflective shadow maps [Dachsbacher and Stamminger 2005] to generate VPLs, directly sampling the scene makes it convenient to handle multi-bounce indirect lighting and multiple primary light sources.

For each VPL we store its position, normal, delta surface area, and material properties. In addition, we store the triangle it exists on and the barycentric coords. When an object is dynamically transformed during rendering, we can use the barycentric coords to quickly recompute the associated data of each VPL. Therefore we do not need to re-sample VPLs on the fly. This works well for rigid body motion and moderate deformations.

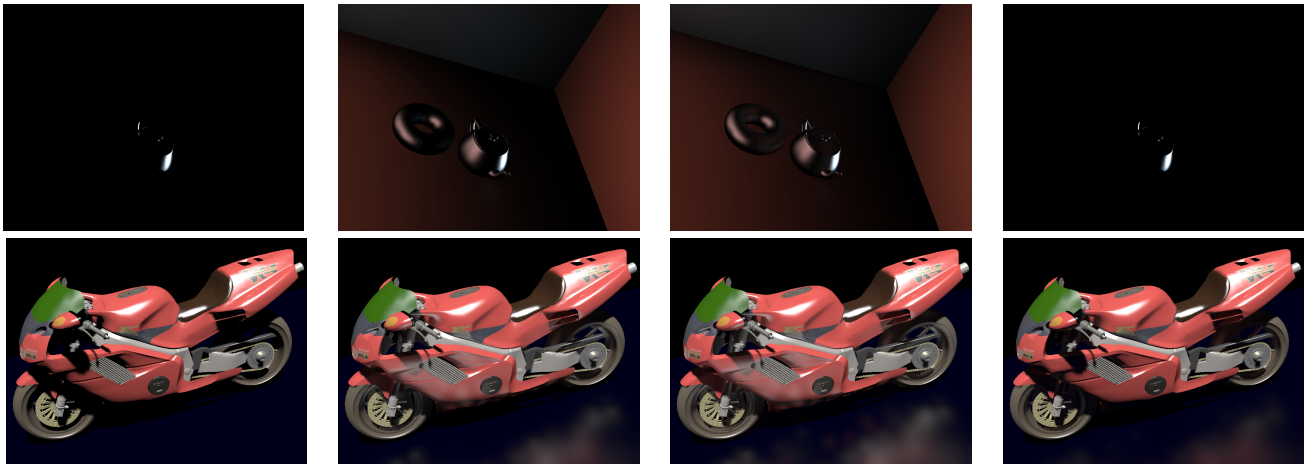
Building the KD-tree. Once the VPLs are generated, we build a kd-tree by by iteratively splitting the entire set of VPLs into subsets. We start from the root node, representing the entire set of VPLs, and split it into two equally sized subsets. We achieve the splitting using k-means with two clusters and the following distance metric:

$$dist(i,c) = \frac{|\mathbf{p}_i - \mathbf{p}_c|}{d/2} + \frac{\sqrt{2 - 2(\mathbf{n}_i \cdot \mathbf{n}_c)}}{4} \tag{5}$$

which calculates the position and normal differences between a point i and a cluster center c. We initialize two random cluster centers c_1 and c_2; the for every point we compute $dist(i, c_1) - dist(i, c_2)$, and sort the results. The sorting moves points that are closer to c_1 to the front of the array and points that are closer to c_2 to the back. We then divide the array in the middle, re-calculate the two cluster centers, and repeat the process for a few iterations to let the cluster centers converge. Finally, the first and second halves of the array become the children of the current node, and we repeat the same process on the subsets until it gets to individual leaf nodes.

Note that because the clustering considers the normal coherence, VPLs with different normals are likely to be split into different clusters in the upper levels of the tree. This benefits the reflectance filtering algorithm because VPLs with coherent normals are likely to have coherent specular radiance lobes. We implement the tree building entirely on the GPU. Most steps use mipmap style parallel reduction; the sorting is performed using a GPU global sort, but we add a large enough per-cluster offset to make sure that the sorting of each cluster does not mix with other clusters.

Sample Direct Lighting. We sample shadowed direct lighting on all VPLs using GPU shadow maps. For each VPL we compute a diffuse radiance and a specular radiance lobe using the VPL's material property. For scenes that contain multiple primary lights, there will be multiple specular lobes. In this case, we immediately apply a reflectance filtering step to merge all lobes into a single lobe. Next, we follow the tree structure to compute the filtered diffuse values and specular lobes. These values are recomputed every time the lighting changes or an object moves in the scene. Finally, we also compute the normal variance at each cluster in order to create min-max cuts as described in Section 2.2. This step only needs to be recomputed every time an object moves.



Direct illumination | With 1-bounce indirect | With 2-bounce indirect | Without glossy lobes

Figure 5: *Comparison between direct lighting, 1 and 2 bounces of indirect lighting, and without storing glossy lobes. The materials in the teapot scene have no diffuse components, therefore the indirect lighting comes entirely from glossy-to-glossy reflections.*

Scene	Adaptive Sampling		Bilateral Sampling		Full Sampling		Min-max Cuts				Full Tree				All Leaf Nodes	
	FPS	#Samples	FPS	#Samples	FPS	#Samples	FPS	Min	Max	Avg	FPS	Min	Max	Avg	FPS	#Nodes
Kitchen	9	38012	12	49152	1	786432	9	690	994	843	1	1839	8191	6379	0.5	16384
Dining Room	16	18909	14	49152	1	786432	16	522	1020	870	2	835	7694	4981	0.5	16384
Teapot Torus	16	11472	14	49152	1	786432	16	528	1023	906	3	498	8178	4697	0.5	16384
Apartment	12	34362	14	49152	1	786432	12	544	948	784	1	2326	8270	5977	0.5	16384
Piano	17	9475	13	49152	1	786432	16	521	1022	821	3	283	8263	4921	0.5	16384
Motorbike	14	18437	14	49152	1	786432	15	512	981	685	3	170	7901	3107	0.5	16384

Table 1: *From left to right: fps and # of samples for adaptive sampling, straightforward bilateral sampling, and full sampling (at every pixel); fps and min/max/avg cut size when enabling min-max cuts, disabling min-max cuts (full tree), and brute force with all VPLs (leaf nodes).*

Compute Indirect Lighting. We apply our adaptive image sampling algorithm as described in Section 2.3. At each sampling level, we collect all pixels that need to be computed, and launch CUDA threads to compute indirect lighting. Due to the use of min-max cuts, we only need to traverse the portion of the light tree marked as valid. This allows us to skip many upper level tree nodes since they are likely to be subdivided anyways; it also prevents the traversal to go arbitrarily deep into the tree, thus reducing thread divergence.

For all tests we use a default solid angle threshold of 0.02. We estimate the solid angle of a cluster by treating it as a distant patch with an area equal to the total surface area of all VPLs in the cluster. If the cluster is very close to a shading point, we instead treat the cluster as a disc and look up in a precomputed 2D texture to obtain an accurate disc-to-point solid angle. We can straightforwardly add multi-bounce indirect lighting by gathering illumination at VPLs for one or two bounces, before gathering them at the shading points.

3 Results and Discussions

All timings are reported on an Intel 1.86 Ghz CPU with an NVIDIA GeForce 280 GTX GPU. We implement our algorithms using NVIDIA's CUDA 2.2. All screenshots and videos were captured at 1024×768 resolution, unless specified otherwise. We achieve interactive rates for all test scenes (see Table 1). By default we enable min-max cuts and image adaptive sampling. Table 1 lists the rendering performance, statistics and comparisons. We typically use 0.02 as the solid angle threshold for tree traversal, and 0.3 as the coherence error threshold for image adaptive sampling. The normal variances we apply to compute min-max cuts are: $\sigma_1 = 40.0$ and $\sigma_2 = 9.0$. We generate $N = 16K$ VPLs for all examples. All materials consist of a diffuse BRDF and a specular Phong BRDF with exponent between [5,60].

As our rendering algorithm runs entirely on the GPU with no precomputation, the user is allowed to make flexible changes while navigating around the scene. These changes include manipulating the primary light sources, scene materials, and scene geometry. See our attached video for demonstration.

Glossy Interreflections. In Figure 5 we compare renderings with direct lighting, direct plus one and two bounces of indirect lighting, and without storing glossy lobes at each tree node. The teapot scene consists of materials that have no diffuse BRDFs, hence the indirect lighting comes entirely from glossy-to-glossy reflections. If we were to model each VPL as a diffuse point light, we would have lost the indirect lighting completely. The motorbike scene also demonstrates the effects of preserving glossy lobes of VPLs.

Image Adaptive Sampling. Figure 6 compares our adaptive sampling method vs. standard bilateral sampling (i.e. 4×4 uniform subsampling). Note that at about the same performance, adaptive sampling produces noticeably better results on both geometry edges (the motorbike scene) and specular reflections (the kitchen and apartment scenes). Figure 6 also visualizes the adaptive sample points computed for these images. Our results have comparable image quality with full sampling, which performs sampling at every pixel but is significantly slower (1 fps). In contrast, standard bilateral sampling is visibly worse. Note that despite the fact that adaptive sampling uses fewer sample points, it can have a higher computation cost, due to the adaptive sampling overhead. Nonetheless, as the image resolution increases, the benefit of adaptive sampling becomes more obvious, as the cost typically grows sublinearly. This makes it more efficient for anti-aliasing.

Min-max Cuts. For tree traversal, our min-max cuts can greatly reduce thread divergence and eliminate wasteful traversals at the top levels of the tree. When disabling the min-max cuts (i.e. us-

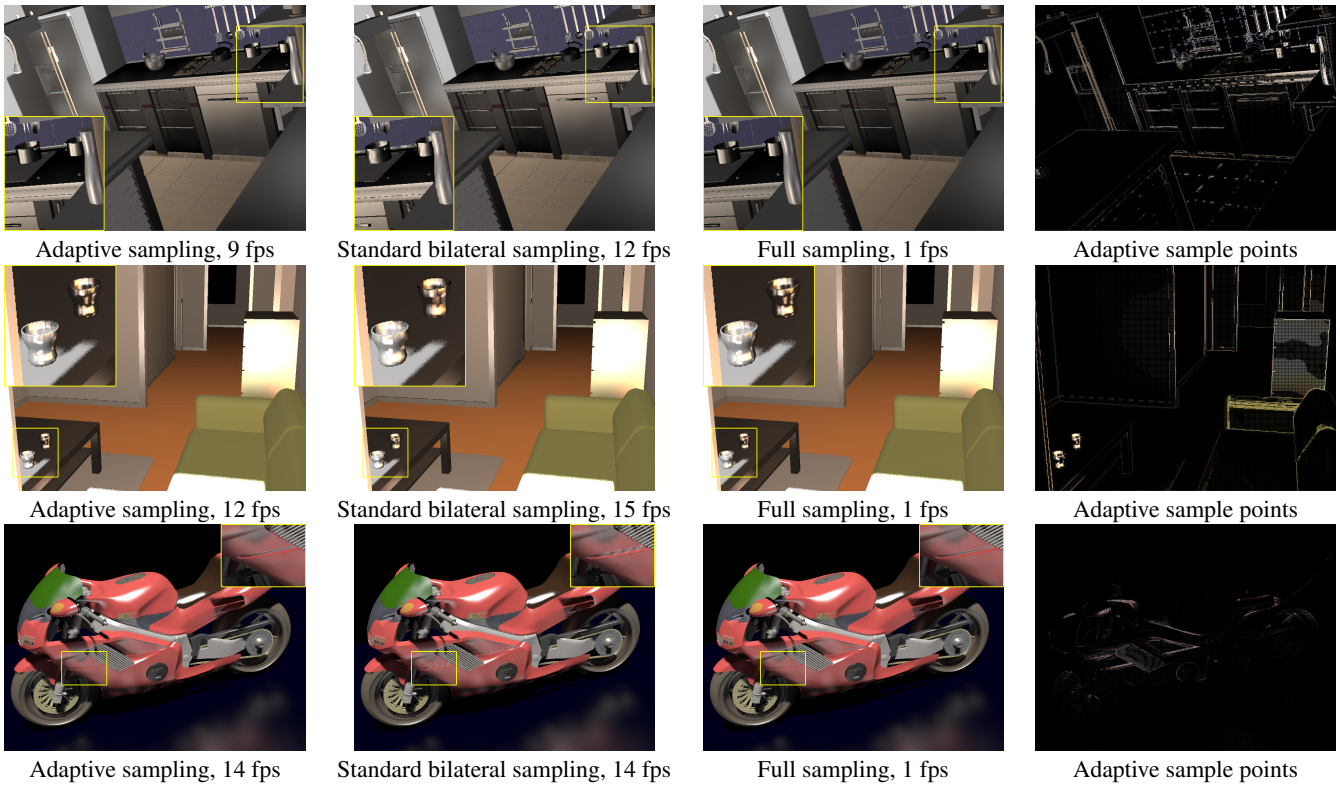


Figure 6: *Comparing adaptive sampling vs. straightforward bilateral sampling and full sampling at every shading point.*

ing the full tree), we found that most threads only need to visit a few hundred nodes, but some threads require visiting 6000∼8000 nodes. This divergence can significantly decrease the performance of the GPU, to a point where we would rather do brute force summation over all individual VPLs. Table 1 shows the performance comparisons. For the min-max cuts method, we also list the minimum/maximum/average number of nodes visited per shading point.

We observe that when the BRDFs are smooth, the min-max cuts method provides comparable image quality with the brute force method that sums over all individual VPLs. On the other hand, when the BRDFs are sharp, the min-max cuts method can lose details in specular highlights, due to clamping of tree traversal at the max cut. This can be seen in the first row of Figure 7, where we compare the min-max cuts with the brute force method. Note the specular reflections on the metal bowl in the kitchen scene, and the reflections of the keys in the piano scene. These specular highlights are largely missing with our method. In addition, as can be seen around the spotty reflections in the brute force images, 16K VPLs (drawn in the brute force images) are often insufficient for highly specular BRDFs. This causes artifacts where we can see reflections of individual VPLs. This could be improved by increasing the number of VPLs or a better VPL distribution. However, for surfaces with semi-glossy specularity, our method provides renderings that are overall very close in to the brute force method, but at 20 times the performance.

Multi-bounce Indirect Lighting. The second row of Figure 7 compares the differences between direct lighting only, with one, two, and three bounces indirect lighting added. The room of the scene has a specular BRDF component, therefore the indirect lighting on the piano exhibits noticeab glossy-to-glossy reflections.

Teaser Image. Figure 1 is an image of the kitchen scene rendered at 7 fps with two bounces of indirect lighting. We apply 2×2 supersampling to reduce aliasing artifacts on the image edges. This scene is rendered at 1536×1024 resolution with adaptive sampling, then downsampled to 768×512 resolution. Due to the adaptive sampling, the increase in sampling resolution does not significantly decrease the frame rates.

4 Conclusions and Future Work

In this paper, we present an efficient method using reflectance filtering to simulate interactive global illumination in scenes that contain semi-glossy materials. Unlike previous approaches, we explicitly model the directional radiance of each VPL using a single Phong lobe, then use reflectance filtering to update the directional radiance of a cluster of VPLs. We also propose using min-max tree cuts to further improve the efficiency of standard tree traversal. To reduce spatial sampling cost, we introduce a new hierarchical image-space adaptive method, which combines level-independent sampling with bilateral interpolation to robustly preserve fine details as well as sharp edges. Our method is implemented entirely on the GPU using NVIDIA's CUDA, thus is suitable for dynamic scenes.

In future work, we plan to extend our approach to model the directional radiance with a mixture of Phong lobes, thereby improving the accuracy of our method in case of highly glossy materials. We would also like to incorporate fast visibility approximation to compute shadowed indirect lighting. Finally, we would like to improve the quality of our renderings by using a more efficient anti-aliasing method other than straightforward supersampling.

References

CARR, N. A., HOBEROCK, J., CRANE, K., AND HART, J. C. 2006. Fast gpu ray tracing of dynamic meshes using geometry images. In *Proc. of GI '06*, 203–209.

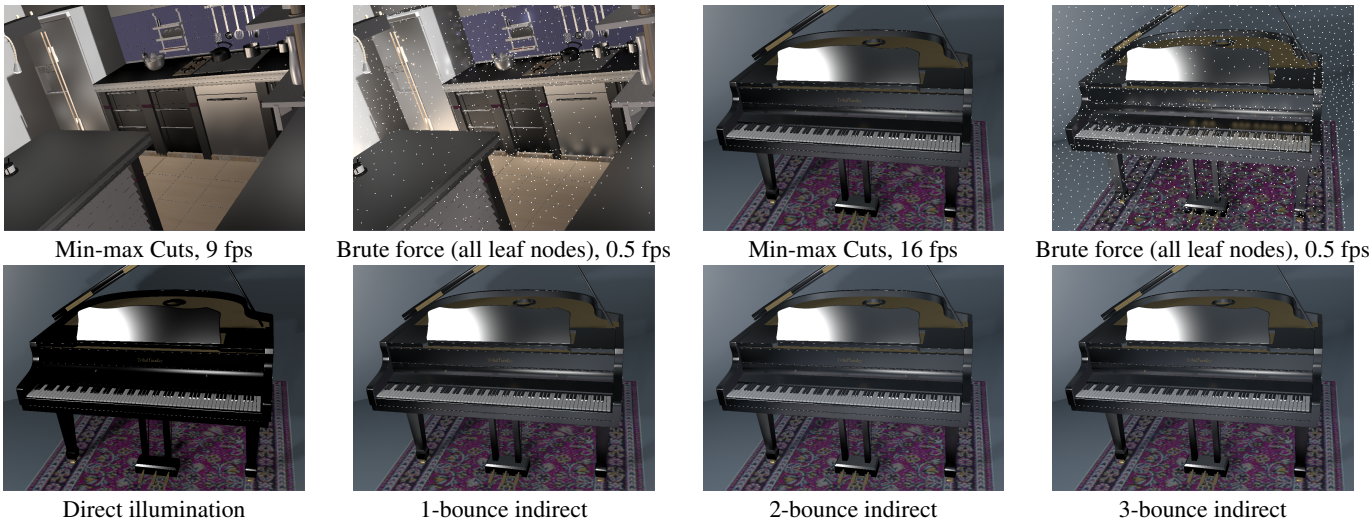


Figure 7: *First row: comparison of min-max cuts with a brute force method that sums over all individual VPLs; second row: comparison of direct lighting with one, two, and three bounces of indirect lighting added.*

CHESLACK-POSTAVA, E., WANG, R., AKERLUND, O., AND PELLACINI, F. 2008. Fast, realistic lighting and material design using nonlinear cut approximation. *ACM Trans. Graph. 27*, 5, 1–10.

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *Proc. of I3D '05*, 203–231.

DUTRÉ, P., BALA, K., AND BEKAERT, P. 2006. *Advanced Global Illumination, second edition*. A. K. Peters, Ltd.

GASSENBAUER, V., KRIVÁNEK, J., AND BOUATOUCH, K. 2009. Spatial directional radiance caching. *Comput. Graph. Forum 28*, 4, 1189–1198.

GAUTRON, P., KRIVÁNEK, J., BOUATOUCH, K., AND PATTANAIK, S. N. 2005. Radiance cache splatting: A gpu-friendly global illumination algorithm. In *Rendering Techniques*, 55–64.

HAN, C., SUN, B., RAMAMOORTHI, R., AND GRINSPUN, E. 2007. Frequency domain normal map filtering. *ACM Trans. Graph. 26*, 3, 28.

HAŠAN, M., PELLACINI, F., AND BALA, K. 2007. Matrix row-column sampling for the many-light problem. *ACM Trans. Graph. 26*, 3, 26.

HAŠAN, M., KŘIVÁNEK, J., WALTER, B., AND BALA, K. 2009. Virtual spherical lights for many-light rendering of glossy scenes. *ACM Trans. Graph. 28*, 5, to appear.

JENSEN, H. W. 2001. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd.

KAJIYA, J. T. 1986. The rendering equation. In *Proceedings of SIGGRAPH '86*, 143–150.

KELLER, A. 1997. Instant radiosity. In *Prof. of SIGGRAPH '97*, 49–56.

KRIVÁNEK, J., AND COLBERT, M. 2008. Real-time shading with filtered importance sampling. *Computer Graphics Forum 27*, 4.

MCGUIRE, M., AND LUEBKE, D. 2009. Hardware-accelerated global illumination by image space photon mapping. In *Proc. of HPG '09*, 77–89.

NICHOLS, G., SHOPF, J., AND WYMAN, C. 2009. Hierarchical image-space radiosity for interactive global illumination. *Comput. Graph. Forum 28*, 4, 1141–1149.

PURCELL, T. J., DONNER, C., CAMMARANO, M., JENSEN, H. W., AND HANRAHAN, P. 2003. Photon mapping on programmable graphics hardware. In *Proc. of Graphics Hardware*, 41–50.

RITSCHEL, T., GROSCH, T., KIM, M. H., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. 27*, 5, 1–8.

RITSCHEL, T., ENGELHARDT, T., GROSCH, T., SEIDEL, H.-P., KAUTZ, J., AND DACHSBACHER, C. 2009. Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph. 28*, 5, to appear.

SILLION, F. X., ARVO, J. R., WESTIN, S. H., AND GREENBERG, D. P. 1991. A global illumination solution for general reflectance distributions. In *Proc. of SIGGRAPH '91*, 187–196.

SLOAN, P.-P., GOVINDARAJU, N. K., NOWROUZEZAHRAI, D., AND SNYDER, J. 2007. Image-based proxy accumulation for real-time soft global illumination. In *Proc. of Pacific Graphics 07*, 97–105.

TAN, P., LIN, S., QUAN, L., GUO, B., AND SHUM, H.-Y. 2005. Multiresolution reflectance filtering. In *Rendering Techniques*, 111–116.

TOKSVIG, M. 2005. Mipmapping normal maps. *Journal of Graphics Tools 10*, 3, 65–71.

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: a scalable approach to illumination. *ACM Trans. Graph. 24*, 3, 1098–1107.

WANG, R., WANG, R., ZHOU, K., PAN, M., AND BAO, H. 2009. An efficient gpu-based approach for interactive global illumination. *ACM Trans. Graph. 28*, 3, 1–8.

WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. 1988. A ray tracing solution for diffuse interreflection. In *Proc. SIGGRAPH '88*, 85–92.

WYMAN, C., AND NICHOLS, G. 2009. Adaptive caustic maps using deferred shading. *Comput. Graph. Forum 28*, 2, 309–318.

YU, X., WANG, R., AND YU, J. 2008. Interactive glossy reflections using gpu-based ray tracing with adaptive lod. *Comput. Graph. Forum 27*, 7, 1987–1996.

ZHOU, K., HOU, Q., WANG, R., AND GUO, B. 2008. Real-time kd-tree construction on graphics hardware. *ACM Trans. Graph. 27*, 5, 1–11.