To appear in the ACM SIGGRAPH 2005 conference proceedings

Lightcuts: A Scalable Approach to Illumination

Bruce Walter Sebastian Fernandez Adam Arbree Kavita Bala Michael Donikian Donald P. Greenberg
*Program of Computer Graphics, Cornell University**

Abstract

Lightcuts is a scalable framework for computing realistic illumination. It handles arbitrary geometry, non-diffuse materials, and illumination from a wide variety of sources including point lights, area lights, HDR environment maps, sun/sky models, and indirect illumination. At its core is a new algorithm for accurately approximating illumination from many point lights with a strongly *sublinear* cost. We show how a group of lights can be cheaply approximated while bounding the maximum approximation error. A binary light tree and perceptual metric are then used to adaptively partition the lights into groups to control the error vs. cost tradeoff.

We also introduce reconstruction cuts that exploit spatial coherence to accelerate the generation of anti-aliased images with complex illumination. Results are demonstrated for five complex scenes and show that lightcuts can accurately approximate hundreds of thousands of point lights using only a few hundred shadow rays. Reconstruction cuts can reduce the number of shadow rays to tens.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture;

Keywords: many lights, raytracing, shadowing

1 Introduction

While much research has focused on rendering scenes with complex geometry and materials, less has been done on efficiently handling large numbers of light sources. In typical systems, rendering cost increases linearly with the number of lights. Real world scenes often contain many light sources and studies show that people generally prefer images with richer and more realistic lighting. In computer graphics however, we are often forced to use fewer lights or to disable important lighting effects such as shadowing to avoid excessive rendering costs.

The lightcuts framework presents a new scalable algorithm for computing the illumination from many point lights. Its rendering cost is strongly sublinear with the number of point lights while maintaining perceptual fidelity with the exact solution. We provide a quick way to approximate the illumination from a group of lights, and more importantly, cheap and reasonably tight bounds on the maximum error in doing so. We also present an automatic and locally adaptive method for partitioning the lights into groups to control the tradeoff between cost and error. The lights are organized into a light tree for efficient partition finding. Lightcuts can handle non-diffuse materials and any geometry that can be ray traced.

Having a scalable algorithm enables us to handle extremely large numbers of light sources. This is especially useful because many other difficult illumination problems can be simulated using the illumination from sufficiently many point lights (e.g., Figure 1). We demonstrate three examples: illumination from area lights, from high dynamic range (HDR) environment maps or sun/sky models, and indirect illumination. Unifying different types of illumination within the lightcuts framework has additional benefits. For example, bright illumination from one source can mask errors in approximating other illumination, and our system automatically exploits this effect.

Figure 1: Bigscreen model: an office lit by two overhead area lights, two HDR flat-panel monitors, and indirect illumination. Our scalable framework quickly and accurately computed the illumination using 639,528 point lights. The images on the monitors were also computed using our methods: lightcuts and reconstruction cuts.

A related technique, called reconstruction cuts, exploits spatial coherence to further reduce rendering costs. It allows lightcuts to be computed sparsely over the image and intelligently interpolates between them. Unlike most interpolation techniques, reconstruction cuts preserve high frequency details such as shadow boundaries and glossy highlights. Lightcuts can compute the illumination from many thousands of lights using only a few hundred shadow rays. Reconstruction cuts can further reduce this to just a dozen or so.

The rest of the paper is organized as follows. We discuss previous work in Section 2. We present the basic lightcuts algorithm in Section 3, give details of our implementation in Section 4, discuss different illumination applications in Section 5, and show lightcut results in Section 6. Then we describe reconstruction cuts in Section 7 and demonstrate their results in Section 8. Conclusions are in Section 9. Appendix A gives an optimization for spherical lights.

2 Previous Work

There is a vast body of work on computing illumination and shadows from light sources (e.g., see [Woo et al. 1990; Hasenfratz et al. 2003] for surveys). Most techniques accelerate the processing of individual lights but scale linearly with the number of lights.

Several techniques have dealt explicitly with the many lights problem. [Ward 1994] sorts the lights by maximum contribution and then evaluates their visibility in decreasing order until an error

*email: {bjw,spf,arbree,kb,mike,dpg}@graphics.cornell.edu

bound is met. We will compare lightcuts with his technique in Section 6. [Shirley et al. 1996] divide the scene into cells and for each cell split the lights into important and unimportant lists with the latter very sparsely sampled. This or similar Monte Carlo techniques can perform well given sufficiently good sampling probability functions over the lights, but robustly and efficiently computing these functions for arbitrary scenes is still an open problem. [Paquette et al. 1998] present a hierarchical approach using light trees similar to the ones we will use. They provide guaranteed error bounds and good scalability, but cannot handle shadowing which limits the applicability. [Fernandez et al. 2002] accelerate many lights by caching per light visibility and blocker information within the scene, but this leads to excessive memory requirements if the number of lights is very large. [Wald et al. 2003] can efficiently handle many lights under the assumption that the scene is highly occluded and only a small subset contribute to each image. This subset is determined using a particle tracing preprocess.

Illumination from HDR environment maps (often from photographs [Debevec 1998]) is becoming popular for realistic lighting. Smart sampling techniques can convert these to directional point lights for rendering (e.g., [Agarwal et al. 2003; Kollig and Keller 2003]), but typically many lights are still required for high quality results.

Instant radiosity [Keller 1997] is one of many global illumination algorithms based on stochastic particle tracing from the lights. It approximates the indirect illumination using many virtual point lights. The resolvable detail is directly related to the number of virtual lights. This makes it a perfect fit with lightcuts, whereas previously it was largely restricted to quick coarse indirect approximations. [Wald et al. 2002] use it in their interactive system and added some clever techniques to enhance its resolution.

Photon mapping is another popular, particle-based solution for indirect illumination. It requires hemispherical final gathering for good results, typically with 200 to 5000 rays per gather [Jensen 2001, p.140]. In complex scenes, lightcuts compute direct and indirect illumination, using fewer rays than a standard hemispherical gather.

Hierarchical and clustering techniques are widely used in many fields. Well known examples in graphics include the radiosity techniques (e.g., [Hanrahan et al. 1991; Smits et al. 1994; Sillion and Puech 1994]). Unlike lightcuts, these compute view-independent solutions that, if detailed, can be very compute and storage intensive. Also since they use meshes to store data, they have difficulty with some common geometric flaws, such as coincident or intersecting polygons. Final gather stages are often used to improve image quality. The methods of [Kok and Jansen 1992] and [Scheel et al. 2001; Scheel et al. 2002] accelerate final gathers by trying to interpolate when possible and only shooting shadow rays when necessary. While very similar in goals to reconstruction cuts, they use heuristics based on data in a radiosity link structure.

Numerous previous techniques have used coherence and interpolation to reduce rendering costs. Reconstruction cuts' novelty and power comes from using the lightcuts framework. Like several previous methods, reconstruction cuts use directional lights to cheaply approximate illumination from complex sources. [Walter et al. 1997] use them for hardware accelerated walkthroughs of precomputed global illumination solutions. [Zaninetti et al. 1999] call them light vectors and use them to approximate illumination from various sources including area lights, sky domes, and indirect.

3 The Lightcuts Approach

Given a set of point light sources $\mathbb{S}$, the radiance L caused by their direct illumination at a surface point $\mathbf{x}$ viewed from direction ω is a

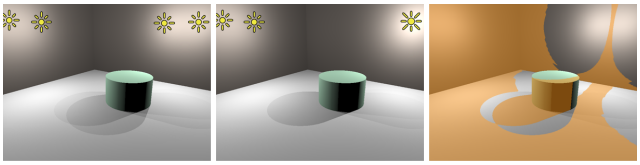


Figure 2: A simple scene with 4 point lights. (a) The exact solution. (b) Approximate solution formed by clustering the two lights on the right. (c) The orange region shows where the exact and clustered solutions are indistinguishable. Errors are typically largest near the lights and where their visibility differs.

product of each light's material, geometric, visibility and intensity terms summed over all the lights:

$$L_{\mathbb{S}}(\mathbf{x},\omega) = \sum_{i\in\mathbb{S}} \overbrace{M_i(\mathbf{x},\omega)}^{material}\ \underbrace{G_i(\mathbf{x})}_{geometric}\ \overbrace{V_i(\mathbf{x})}^{visibility}\ \underbrace{I_i}_{intensity} \tag{1}$$

The cost of an exact solution is linear in the number of lights since these terms must be evaluated for each light. To create a scalable, sublinear method, we need a way to approximate the contribution of a group of lights without having to evaluate each light individually.

Let us define a *cluster*, $\mathbb{C} \subseteq \mathbb{S}$, to be a set of point lights along with a representative light $j \in \mathbb{C}$. The direct illumination from a cluster can be approximated by using the representative light's material, geometric, and visibility terms for all the lights to get:

$$\begin{aligned} L_{\mathbb{C}}(\mathbf{x},\omega) &= \sum_{i\in\mathbb{C}} M_i(\mathbf{x},\omega)\, G_i(\mathbf{x})\, V_i(\mathbf{x})\, I_i \\ &\approx M_j(\mathbf{x},\omega)\, G_j(\mathbf{x})\, V_j(\mathbf{x}) \sum_{i\in\mathbb{C}} I_i \end{aligned} \tag{2}$$

The cluster intensity ($I_{\mathbb{C}} = \sum I_i$) can be precomputed and stored with the cluster making the cost of a cluster approximation equal to the cost of evaluating a single light (i.e. we have replaced the cluster by a single brighter light). The amount of cluster error will depend on how similar the material, geometric, and visibility terms are across the cluster. A simple example is shown in Figure 2.

Light Tree. No single partitioning of the lights into clusters is likely to work well over the entire image, but dynamically finding a new cluster partitioning for each point could easily prove prohibitively expensive. We use a global light tree to rapidly compute locally adaptive cluster partitions. A *light tree* is a binary tree where the leaves are individual lights and the interior nodes are light clusters containing the lights below them in the tree. A *cut* through the tree is a set of nodes such that every path from the root of the tree to a leaf will contain exactly one node from the cut. Thus each cut corresponds to a valid partitioning of the lights into clusters. An example light tree and three different cuts are shown in Figure 3.

While every cut corresponds to a valid cluster partitioning, they vary greatly in their costs and the quality of their approximated illumination. We need a robust and automated way to choose the appropriate cut to use locally. As the cuts will vary across the image, some points, or pixels, may use a particular cluster to reduce costs while others replace it with its children for increased accuracy. Such transitions could potentially cause objectionable image artifacts. To prevent this, we only use clusters when we can guarantee that the approximation error introduced by the cluster will be below a perceptual visibility threshold. Weber's law [Blackwell 1972] is a standard, well-known perceptual result that says the minimum perceptible change in a visual signal is roughly equal to a fixed percentage of the base signal. Under worst case conditions, humans can detect changes of just under 1%, though in practice,

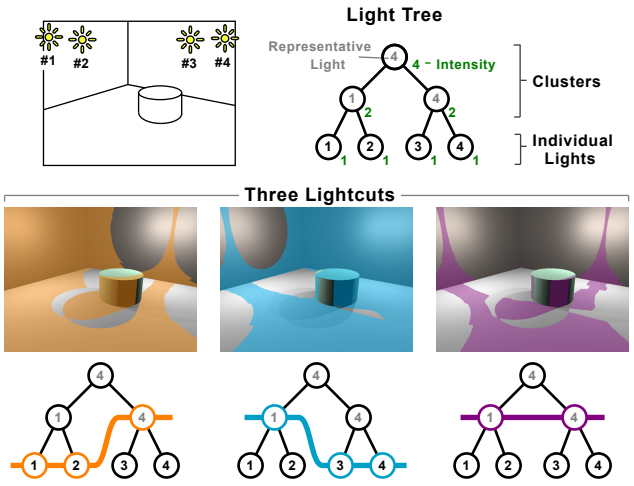


Figure 3: A light tree and three example cuts. The tree is shown on the top with the representative lights and cluster intensities for each node. Leaves are individual lights while upper nodes are progressively larger light clusters. Each cut is a different partitioning of the lights into clusters (the orange cut is the same as Figure 2). Above each cut, the regions where its error is small are highlighted.

the threshold is usually higher. In our experience, an error ratio of 2% results in no visible artifacts across a wide variety of scenes and was used for all our results. Changing this value can be used to vary the tradeoff between performance and accuracy.

Choosing Lightcuts. Using a relative error criterion requires an estimate of total radiance before we can decide whether a particular cluster is usable. To solve this difficulty, we start with a very coarse cut (e.g., the root of the light tree) and then progressively refine it until our error criterion is met. For each node in the cut we compute both its cluster estimate (Equation 2) and an upper bound on its error (Section 4.1). Each refinement step considers the node in the current cut with the largest error bound. If its error bound is greater than our error ratio times the current total illumination estimate, we remove it from the cut, replace it with its two children from the light tree, compute their cluster estimates and error bounds, and update our estimate of the total radiance. Otherwise, the cut obeys our error criterion and we are done. We call such a cut, a *lightcut*.

To make this process more efficient, we require that the representative light for a cluster be the same as for one of its two children. This allows us to reuse the representative light's material, geometric and visibility terms when computing that child. We use a heap data structure to efficiently find the cluster node in the cut with the highest error bound. If present in the cut, individual lights (i.e. light tree leaf nodes) are computed exactly and thus have zero error.

Our relative error criterion overestimates the visibility of errors in very dark regions. For example, a fully occluded point would be allowed zero error, but even at black pixels sufficiently small errors are not visible. Therefore, we also set a maximum cut size and, if the total number of nodes on the cut reaches this limit, stop further refinement. We chose our maximum cut size of 1000 to be large enough to rarely be reached in our results and then only in dark regions where the extra error is not visible.

4 Implementing Lightcuts

Our implementation supports three types of point lights: omni, oriented, and directional. Omni lights shine equally in all directions from a single point. Oriented lights emit in a cosine-weighted hemispherical pattern defined by their orientation, or direction of maximum emission. Directional lights simulate an infinitely far away source emitting in a single direction. All lights have an intensity I_i.

Building the Light Tree. The light tree groups point lights together into clusters. Ideally, we want to maximize the quality of the clusters it creates (i.e. combine lights with the greatest similarity in their material, geometric and visibility terms). We approximate this by grouping lights based on spatial proximity and similar orientation.

We divide the point lights by type into separate omni, oriented, and directional lists and build a tree for each. Conceptually though, we think of them as part of a single larger tree. Each cluster records its two children, its representative light, its total intensity $I_{\mathbb{C}}$, an axis-aligned bounding box, and an orientation bounding cone. The cone is only needed for oriented lights. Although infinitely far away, directional lights are treated as points on the unit sphere when computing their bounding boxes. This allows directional lights to use the same techniques as other point lights when building light trees and, more importantly, later for bounding their material terms M_i.

Similarity Metric. Each tree is built using a greedy, bottom-up approach by progressively combining pairs of lights and/or clusters. At each step we choose the pair that will create the smallest cluster according to our cluster size metric $I_{\mathbb{C}}(\alpha_{\mathbb{C}}^2 + c^2\,(1-\cos\beta_{\mathbb{C}})^2)$, where $\alpha_{\mathbb{C}}$ is the diagonal length of the cluster bounding box and $\beta_{\mathbb{C}}$ is the half-angle of its bounding cone. The constant c controls the relative scaling between spatial and directional similarity. It is set to the diagonal of the scene's bounding box for oriented lights and zero for omni and directional lights.

The representative light for a cluster is always the same as for one of its children and is chosen randomly based on the relative intensities of the children. Each individual light is its own representative. Thus the probability of a light being the representative for a cluster is proportional to its intensity. This makes the cluster approximation in Equation 2 unbiased in a Monte Carlo sense. However once chosen, the same representative light is used for that cluster over the entire image. Tree building, by its very nature, cannot be sublinear in the number of lights, but is generally not a significant cost since it only has to be done once per image (or less if the lights are static).

4.1 Bounding Cluster Error

To use the lightcuts approach, we need to compute reasonably cheap and tight upper bounds on the cluster errors (i.e. the difference between the exact and approximate versions of Equation 2). By computing upper bounds on the material, geometric, and visibility terms for a cluster, we can multiply these bounds with the cluster intensity to get an upper bound for both the exact and approximated cluster results. Since both are positive, this is also an upper bound on the cluster error (i.e. their absolute difference).

Visibility Term. The visibility of a point light is typically zero or one but may be fractional (e.g., if semitransparent surfaces are allowed). Conservatively bounding visibility in arbitrary scenes is a hard problem, so we will use the trivial upper bound of one for the visibility term (i.e. all lights are potentially visible).

Geometric Term. The geometric terms for our three point light types are listed below, where $\mathbf{y}_i$ is the light's position and ϕ_i is the angle between an oriented light's direction of maximum emission and direction to the point $\mathbf{x}$ to be shaded.

Light Type	Omni	Oriented	Directional
$G_i(\mathbf{x}) =$	$\dfrac{1}{\|\mathbf{y}_i-\mathbf{x}\|^2}$	$\dfrac{\max(\cos\phi_i,\,0)}{\|\mathbf{y}_i-\mathbf{x}\|^2}$	1

(3)

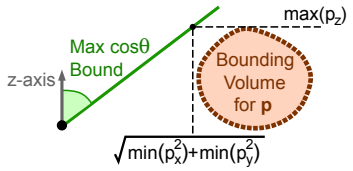


Figure 4: Bounding the minimum angle (and hence maximum cosine) to a bounding volume. See Equation 4.

The upper bound for directional lights is trivial since their geometric factor is always one. Omni lights are also easy. We just compute the minimum distance between the point **x** and the bounding volume of the cluster. Oriented lights are more complex because of the additional cosine factor. We could use the trivial cosine upper bound of one, but we prefer a tighter bound.

Let's start with the simpler problem shown in Figure 4. For any point $\mathbf{p} = [p_x, p_y, p_z]$, let θ be the angle between the vector from the origin to **p** and the z-axis. We want to find an upper bound on $\cos\theta$ over all points in some bounding volume. For any point **p**, we have $\cos\theta = \frac{p_z}{\sqrt{p_x^2+p_y^2+p_z^2}}$. To create an upper bound, we first replace the numerator by the maximum value of p_z within the bounding volume and then choose the p_x and p_y values to minimize or maximize the denominator depending on the sign of the numerator to get[1]:

$$\cos\theta \leq \begin{cases} \frac{\max(p_z)}{\sqrt{\min(p_x^2)+\min(p_y^2)+(\max(p_z))^2}} & \text{if } \max(p_z) \geq 0 \\ \frac{\max(p_z)}{\sqrt{\max(p_x^2)+\max(p_y^2)+(\max(p_z))^2}} & \text{otherwise} \end{cases} \quad (4)$$

To apply this to bounding $\cos\phi_i$ for oriented lights, we transform the problem as illustrated in Figure 5. First consider every point pair [**x**, $\mathbf{y}_i$] in the cluster and translate both points by $-\mathbf{y}_i$. This translates all the lights to the origin but spreads the point **x** across a volume with the same size as the cluster's bounding volume (the bounding volume's shape is the same but inverted). Second apply a coordinate transform that rotates the z-axis to match the axis of the cluster's orientation bounding cone. Now we can use Equation 4 to compute the minimum angle between the volume and the cone's axis. If this angle lies inside the bounding cone then we can only use the trivial upper bound of one, but if it lies outside then ϕ_i must be at least as large as this angle minus the cone's half-angle.

Material Term. The material term M_i is equal to the BRDF (Bidirectional Reflectance Distribution Function) times the cosine of the angle between the vector, $\mathbf{y}_i - \mathbf{x}$, and the surface normal at **x**. We have already described bounding the cosine of the angle to a bounding volume (Figure 4), so that only leaves the BRDF to be bounded.

Our current system supports three types of BRDF components: diffuse or lambertian, Phong [Phong 1975], and isotropic Ward [Larson 1992]. Multiple components can be summed when creating a BRDF. A diffuse BRDF component is simply a constant (i.e. does not depend on viewing or illumination directions) and so is trivial to bound. Phong components vary with the cosine of the angle between the vector to the light and the mirror reflection direction, raised to some exponent. We reuse the cosine bounding machinery already described to bound any Phong components. This same approach can be adapted to any similar BRDF component that is symmetric about an axis.

The isotropic Ward BRDF is not symmetric about any axis, because it is based on the half-angle (i.e. the angle between the surface normal and a vector halfway between the viewing and lighting vectors).

[1]Note if p_x ranges from -2 to 1 then $\max(p_x) = 1$ and $(\max(p_x))^2 = 1$ but $\min(p_x^2) = 0$ and $\max(p_x^2) = 4$.

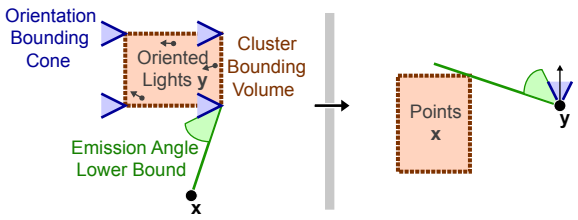


Figure 5: Conceptual transformation for bounding the emission angle (green) with respect to a cluster of oriented lights.

We have also developed a reasonably cheap and tight way to bound the minimum half-angle to a cluster. Details are in [Walter 2005]. We tested all our bounds numerically to confirm that they are valid.

In principle, lightcuts can work with any BRDF, as long as there is a good method to bound its maximum value over a cluster. Delta components (e.g., a mirror) are handled by the standard ray tracing method of recursively tracing reflected and/or refracted rays.

5 More Illumination Applications

Once we have a scalable solution for robustly approximating the illumination from large numbers of point lights, we can apply it to other difficult problems. The examples included here are illumination from: area lights, high dynamic range environment maps, and indirect illumination. Integrating different illumination types in the lightcuts framework has many advantages. For example, lightcuts automatically reduce the accuracy of one component when the error will be masked by strong illumination from another component.

Area Lights. Illumination and soft shadows from area lights are difficult to compute, and standard techniques often scale poorly with the size and number of area lights. A common approach is to approximate each area light using multiple point lights, however, the number required varies considerably depending on the local configuration. Locations near area lights or in their penumbra require many point lights while others require few. Many heuristics have been proposed (e.g., subdivide down to some fixed solid angle), but these may not work everywhere and often require manual parameter tweaking. In our system, each light can be converted into a conservatively large number of points. The lightcut algorithm will automatically and adaptively choose the number of samples to actually use locally. Any diffusely-emitting area light can be approximated by oriented lights on its surface. For spherical lights, an even better method using omni lights is described in Appendix A.

HDR Environment Maps. High dynamic range environment maps are a popular way to capture illumination from real world environments and apply it to synthetic scenes. Computing accurate illumination, especially shadows, from them can be very expensive.

The most common approach is to convert the environment map into a discrete set of directional lights. One critical question is how many directional lights to use. Using too few causes image artifacts such as blocky shadows, while using too many increases rendering costs substantially. For example, [Agarwal et al. 2003] suggest using 300 directional lights as generally adequate. In our experience, 300 is sufficient for isolated objects, but rendering complete scenes with significant occlusion and/or narrow glossy BRDFs may require 3000 or more. The lightcuts approach can handle such large numbers of lights much more efficiently than previous approaches.

Indirect Illumination. Indirect illumination (also called global illumination) is desirable for its image quality and sense of realism, but is considered too expensive for many applications. Much research has gone into increasing their physical and mathematical ac-


Lightcut Image


Reference Image


Error Image

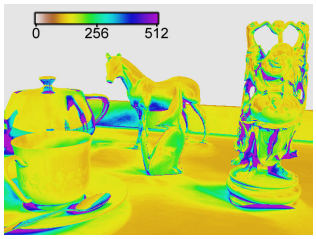

16 x Error Image

Point Lights	Avg Cut Size	Avg Shadow Rays	Time	Reference
4608	264	259	128s	1096s

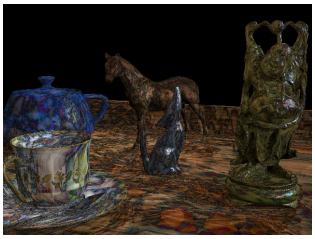
Figure 6: Kitchen scene with direct light from 72 area sources.


Lightcut Image


Reference Image



Cut Size (False Color)


16 x Error Image

Point Lights	Avg Cut Size	Avg Shadow Rays	Time	Reference
3000	(187) 132	(168) 118	54s	424s

Figure 7: Tableau scene illuminated by an HDR environment map. In parentheses are averages over only pixels containing geometry.

curacy, but there is also interest in lower cost approximations with the tradeoff of lower fidelity, as long as they do not introduce objectionable image artifacts (e.g., [Tabellion and Lamorlette 2004]).

Instant radiosity is one such method. It first tracks light particles as they probabilistically scatter through a scene. Then virtual point lights are created at these scatter locations such that their aggregate illumination simulates the effect of indirect illumination. There is a cost vs. quality tradeoff in choosing the number of virtual lights. Using more lights reproduces more detail in the indirect illumination, but also increases cost of evaluating so many lights. Prior results have mostly been limited to tens or at most hundreds of virtual lights (e.g., [Keller 1997; Wald et al. 2002]). With our scalable algorithm, we can use thousands or even millions of virtual lights to reproduce more detail in the indirect illumination.

Prior instant radiosity results were limited to diffuse-only indirect because of the small number of virtual indirect lights used. Freed from this restriction, our system can also include some glossy indirect effects. Only a BRDF's diffuse component is used when converting particle hits to virtual oriented point lights, but unlike prior systems, we use the full BRDF at the points they illuminate.

When using instant radiosity, one needs to be aware of its inherent limitations. It cannot reproduce some types of indirect illumination (e.g., caustics); other methods must be used if these are desired. It also has difficulties with short range and/or glossy indirect effects though using more virtual lights helps. The particle tracing is a stochastic process and thus there is considerable randomness in the positions of the virtual lights. The contribution of any particular light is largely noise, and it is only by combining the results over many lights that we get statistically reliable information. Without a noise suppression technique, this noise would be highly visible in locations whose illumination is easily dominated by one or a few lights (e.g., concave corners or sharp glossy reflections)

The noise suppression is typically accomplished by limiting the maximum contribution from a virtual light. This biases the results but is preferable to objectionable noise artifacts. [Keller 1997] used hardware that clamped values to a maximum of 255 and [Wald et al. 2002] had the user specify a minimum distance to use when computing the geometric factor of an indirect light. The latter does not work with non-diffuse BRDFs, so we apply a clamping threshold to the total contribution, $M_i G_i V_i I_i$, of individual indirect lights. We do not clamp the contribution of clusters since they will be handled by normal lightcut refinement if too bright. This clamping threshold can be user-specified[2] or automatically determined as follows.

Our automatic clamping limits each indirect light to contribute no more than a fixed percentage of the total result. Since this clamping threshold is computed from the same noisy particle data, we must take care that it doesn't accentuate the noise. We use half the error ratio parameter from the lightcut selection/refinement process (e.g., 1% if the error ratio is 2%). During lightcut refinement, we keep track of any indirect nodes on the cut that may require clamping. In our experience, automatic clamping requires negligible overhead, works at least as well as a manually-tuned clamping threshold, and is one less parameter for the user to worry about.

6 Lightcut Results

In this section we demonstrate the results of applying our lightcuts implementation to five scenes with varying illumination. All results use an error ratio of 2% and a maximum cut size of 1000 nodes. All images in this section have a resolution of 640x480 with one eye ray per pixel, however this sometimes requires shading multiple points due to reflections and transparency. Timings are for a single workstation with two 3 GHz Xeon processors, though our system is also capable of efficiently using multiple machines.

Error. Lightcuts conservatively bound the maximum error per cluster to ensure that individual cluster refinement transitions are not visible. Theoretically, the errors from many different clusters could still sum up to a large and visible error. In practice we have not found this to be a problem for two reasons. The per cluster error bounds are worst case bounds; the actual average cluster error is much less than our 2% bound. Also uncorrelated errors accumulate much more slowly than correlated errors would (i.e. $O(\sqrt{N})$ vs. $O(N)$). During tree building, we randomize the choice of representative lights specifically to ensure that the cluster errors will be

[2] $\frac{1}{1000}$ of the image white point is a reasonable starting point.


Kitchen


Tableau


Grand Central


Temple

Model	Polygons	Number of Point Lights				Per Pixel Averages		Tree	Image
		Direct	Env Map	Indirect	Total	(d+e+i) Lightcut Size	Shadow Rays	Build	Time
Kitchen	388552	(72) 4608	5064	50000	59672	(54+244+345) 643	(**0.8**%) 478	3.4s	290s
Tableau	630843	0	3000	10000	13000	(0+112+104) 216	(**1.1**%) 145	0.9s	79s
Grand Central	1468407	(820) 38400	5064	100000	143464	(73+136+480) 689	(**0.33**%) 475	9.7s	409s
Temple	2124003	0	5064	500000	505064	(0+185+437) 622	(**0.07**%) 373	44s	225s
Bigscreen	628046	(4) 614528	0	25000	639528	(83+0+222) 305	(**0.04**%) 228	46s	98s

Figure 9: Lightcut results for 640x480 images of our five scenes with all illumination components enabled. Average cut sizes include how many came from each of the three components: direct, environment map, and indirect. Average shadow rays per pixel is also shown as a percentage of the total number of lights. The aliasing (e.g., windows in Grand Central) is due to using only one eye ray per pixel here.

statistically uncorrelated. Essentially, lightcuts provide a stochastic error bound rather than an absolute one; large total errors are possible but very unlikely.

The kitchen model, shown in Figure 6, is based on part of an actual house. It contains 72 area sources, each approximated using 64 point lights for a total of 4608 point lights. Even a close examination reveals no visible differences between the lightcut result and a reference image that evaluated each point light exactly. The error image appears nearly black. Magnifying the errors by a factor 16 shows that, as expected, the errors are generally larger in brighter regions and consist of many discontinuities caused by transitions between using a cluster and refining it. The lightcut image took 128 seconds with an average cut size of 264 and 259 shadow rays per pixel, while the reference image took much longer at 1096 seconds with an average of 3198 shadow rays per pixel. Shadow rays are not shot to lights whose material or geometric terms are zero.

The tableau model, in Figure 7, has several objects with different glossy materials on a wooden tray and lit by a captured HDR environment map (the Kitchen map from [Debevec 1998], not related to our kitchen model). We used the technique of [Agarwal et al. 2003] to convert this map to 3000 directional lights for rendering. Again there are no visible differences between the reference image and the lightcut image. A visualization shows the per pixel cut size (i.e. the number of nodes on the lightcut). Cut size corresponds closely to rendering cost and tends to be largest in regions of high occlusion.

Scalability. As the number of point lights increases, lightcuts scale fundamentally better (i.e. sublinearly vs. linearly) than prior techniques. To demonstrate this, we varied the number of point lights in the kitchen and tableau examples above by changing the number of point lights created per area light and directional lights created from the environment map, respectively. Image times vs. number of point lights are shown in Figure 8 for both lightcuts and the naive (reference) solutions. We also compare to [Ward 1994] which we consider the best of the alternative approaches with respect to generality, robustness, and the ability to handle thousands of lights.

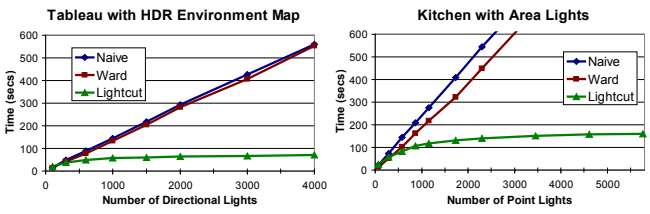


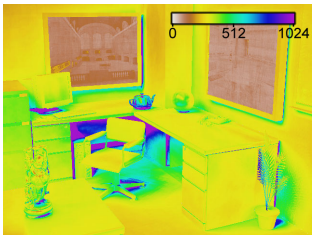
Figure 8: Lightcut performance scales fundamentally better (i.e. sublinearly) as the number of point lights increase.

Ward's technique computes the potential contribution of all the lights assuming full visibility, sorts these in decreasing order, and then progressively evaluates their visibility until the total remaining potential contribution falls below a fraction of the total of the evaluated lights (10% in our comparison). Since shadow rays (i.e.visibility tests) are usually the dominant cost, reducing them usually more than compensates for the cost of the sort (true for the kitchen and just barely for tableau). However, the cost still behaves linearly as shown in plots. Suppose we had 1000 lights whose unoccluded contributions would be roughly equal. We would have to check the visibility of at least 800 lights to meet a 10% total error bound[3], and this number grows linearly with the lights. This case is the Achilles' heel of Ward's technique, or any technique that provides an absolute error bound. Lightcuts superior scalability means that its advantage grows rapidly as the number of lights increase.

Mixed Illumination. We can use this scalability to compute richer and more complex lighting in our scenes as shown in Figure 9. Grand Central is a model of the famous landmark in New York City and contains 220 omni point lights distributed near the ceiling of the main hall and 600 spherical lights in chandeliers in the side hallways. The real building actually contains even more lights. Temple is a model of an Egyptian temple and is our most geometrically complex at 2.1 million triangles. Such geometric complexity

[3]Using the optimal value of 0.5 for the visibility of the untested lights.


Bigscreen Model



Cut Size (False Color)

Figure 10: Bigscreen model. See Figure 9 for statistics.

causes aliasing problems because we are using only one eye ray per pixel. The next section will introduce reconstruction cuts that allow us to perform anti-aliasing at a reduced cost. The bigscreen model (Figure 10) shows an office lit by two overhead area lights (64 points each) and by two large HDR monitors displaying images of our other scenes. The monitors have a resolution of 640x480 and were simulated by converting each of their pixels into a point light.

We've added a sun/sky model from [Preetham et al. 1999], which acts like an HDR environment map, to the kitchen, Grand Central, and temple. The sun is converted into 64 directional lights and the sky into 5000 directional lights distributed, uniformly over the sphere of directions. We also added indirect illumination to all the models using the instant radiosity approach. In the kitchen and Grand Central models only a fraction of particles from the sun/sky make it through the windows into the interior; many indirect lights end up on the outside of the buildings. With our scalable algorithm, we can compensate by generating more indirect lights. Temple uses the most indirect lights because its model covers the largest area.

With lightcuts the number of lights is not strongly correlated with image cost. Instead the strongest predictor of image cost is the degree of occlusion of the light sources. Thus the kitchen and particularly Grand Central are the most expensive because of the high degree of occlusion to their light sources especially the sun/sky. Although temple and bigscreen have more point lights and geometry, they are less expensive due to their higher visibility.

Shadow rays are the dominant cost in lightcuts and consume roughly 50% of the total time, even though this is the most optimized part of our code (written in C while the rest is Java). Computing error bounds consumes another 20% and 10% is shading. Various smaller operations including tree traversal, maintaining the max heap, updating estimates, etc. consume the rest. Tree building starts to be significant for the largest numbers of lights but there are much faster methods than our simple greedy approach.

Bound Tightness. The ability to cheaply bound the maximum contribution, and hence the error, from a cluster is essential for our scalability. Using tighter (and probably more expensive) bounds might reduce the average cut size and potentially reduce overall costs. In Figure 11, we show the cut sizes that would result if we had exact bounds on the product of the geometric and material terms $G_i M_i$. This is the best we can hope to achieve without bounds on visibility. Overall, our bounds perform very well. Indirect lights are harder to bound due to their wider dispersal, but our bounds still perform well. Significant further gains will likely require efficient conservative visibility bounds. This is possible in many specific cases (e.g., see [Cohen-Or et al. 2003]), but an unsolved problem in general.

Model	Illumination	Point Lights	Avg Cut Size	
			Lightcut	Exact $G_i M_i$ Bound
Kitchen	Direct	4608	264	261
	+ Indirect	54608	643	497
Tableau	Env Map	3000	132	120
	+ Indirect	13000	216	153
Temple	Sun/Sky	5064	294	287

Figure 11: How tighter bounds would affect lightcut size.

7 Reconstruction Cuts

Reconstruction cuts are a new technique for exploiting spatial coherence to reduce average shading costs. The idea is to compute lightcuts sparsely over the image (e.g., at the corners of image blocks) and then appropriately interpolate their illumination information to shade the rest of the image. Although lightcuts allow the scalable computation of accurate illumination from thousands or millions of point lights, they still typically require shooting hundreds of shadow rays per shaded point. By taking a slightly less conservative approach, reconstruction cuts are able to exploit illumination smoothness to greatly reduce the shading cost.

The simplest approach would be to simply interpolate the radiances from the sparse lightcuts (e.g., Gouraud shading), but this would cause objectionable blurring of high frequency image features such as shadow boundaries and glossy highlights. Many extensions of this basic idea have been proposed to preserve particular types of features (e.g., [Ward and Heckbert 1992] interpolates irradiance to preserve diffuse textures and [Křivánek et al. 2005] use multiple directional coefficients to preserve low frequency gloss), but we want to preserve all features including sharp shadow boundaries.

To compute a reconstruction cut at a point, we first need a set of nearby samples (i.e. locations where lightcuts have been computed and processed for easy interpolation). Then we perform a top-down traversal of the global light tree. If all the samples agree that a node is occluded then we immediately discard it. If a node's illumination is very similar across the samples then we cheaply interpolate it using impostor lights. These are special directional lights designed to mimic the aggregate behavior of a cluster as recorded in the samples. Otherwise we default to lightcut-style behavior; refining down the tree until the errors will be small enough and using the cluster approximation from Equation 2, including shooting shadow rays.

Interpolating or discarding nodes, especially if high up in the tree, provides great cost savings. However when the samples straddle a shadow boundary or other sharp feature, we revert to more robust but expensive methods for the affected nodes. Because reconstruction cuts effectively include visibility culling, they are less affected by high occlusion scenes than lightcuts are.

Samples. A sample k is created by first computing a lightcut for a point $\mathbf{x}^k$ and viewing direction ω^k. This will include computing a radiance estimate $\tilde{L}_n^k$ at every light tree node n on the lightcut using Equation 2. For each node above the cut, we define $\tilde{L}_n^k$ as the sum of the radiance estimates of all of its descendants on the cut. Similarly, the total radiance estimate $\tilde{L}_\mathrm{T}^k$ is the sum over all nodes in the cut.

To convert a lightcut into a sample, we create *impostor directional point lights* (with direction $\mathbf{d}_n^k$ and intensity γ_n^k) for each node on or above the cut. Their direction mimics the average direction of incident light from the corresponding cluster or light, and their illumination exactly reproduces the radiance estimate $\tilde{L}_n^k$ at the sample point. Impostor lights are never occluded and hence relatively inexpensive to use (i.e. both visibility and geometric terms equal one).

For nodes on the cut, the impostor direction $\mathbf{d}_n^k$ is the same as from its representative light. For nodes above the cut, $\mathbf{d}_n^k$ is the average of the directions for its descendants on the cut, weighted by their respective radiance estimates. Using this direction we can evaluate the impostor's material term M_n^k and its intensity γ_n^k. We also compute a second intensity Γ_n^k based on the total radiance estimate and used to compute relative thresholds (i.e. relative magnitude of the node compared to the total illumination).

$$\gamma_n^k = \tilde{L}_n^k(\mathbf{x}^k, \omega^k) \,/\, M_n^k(\mathbf{x}^k, \omega^k) \tag{5}$$

$$\Gamma_n^k = \tilde{L}_\mathrm{T}^k(\mathbf{x}^k, \omega^k) \,/\, M_n^k(\mathbf{x}^k, \omega^k) \tag{6}$$

Occasionally we need an impostor directional light for a node below a sample's lightcut. These are created as needed and use the same direction as their ancestor on the cut, but with impostor's intensity diminished by the ratio between the node's cluster intensity $I_\mathbb{C}$ and that of its ancestor.

Exact sample size depends on the lightcut, but 24KB is typical. Samples store 32 bytes per node on or above the lightcut which includes 7 floats (1 for Γ_n^k and 3 each for $\mathbf{d}_n^k$ and γ_n^k) and an offset to its children's data if present. We decompose the image into blocks so that only a small subset of the samples is kept at any time.

Computing a Reconstruction Cut. Given a set of samples, we want to use them to quickly estimate Equation 1. Reconstruction cuts use a top-down traversal of the global light tree. At each node visited, we compute the minimum and maximum impostor intensities γ_n^k over the samples k, and a threshold τ_n, the lightcut error ratio (e.g.,2%) times the minimum of Γ_n^k over the samples. Then we select the first applicable rule from:

1. **Discard.** If $\max(\gamma_n^k) = 0$, then the node had zero radiance at all the samples and is assumed to have zero radiance here as well.
2. **Interpolate.** If $\max(\gamma_n^k) - \min(\gamma_n^k) < \tau_n$ and $\min(\gamma_n^k) > 0$, then we compute weighted averages of $\mathbf{d}_n^k$ and γ_n^k to create an interpolated impostor directional light. The estimate for this node is then equal to the material term for the interpolated direction times the interpolated intensity.
3. **Cluster Evaluate.** If $\max(\gamma_n^k) < \tau_n$ or if we are at a leaf node (i.e. individual light), then we estimate the radiance using Equation 2. This includes shooting a shadow ray to the representative light if the material and geometric terms are not zero.
4. **Refine.** Otherwise we recurse down the tree and perform the same tests on each of this node's two children.

While the above rules cover most cases, a few refinements are needed to prevent occasional artifacts (e.g., on glossy materials). First, we disallow interpolation inside glossy highlights. The maximum possible value for a diffuse BRDF is $1/\pi$. When computing the material term for an interpolated impostor, if its BRDF value is greater than $1/\pi$, then the direction must lie inside the gloss lobe of a material and we disallow interpolation for that node. Second, cluster evaluation is only allowed for nodes that are at, or below, at least one of the sample lightcuts. Nodes below all the sample lightcuts do not use interpolation, since they have no good information to interpolate. Third, if the result of a cluster evaluation is much larger than expected, we recursively evaluate its children instead.

Image Blocks. The image is first divided into 16x16 pixel blocks which do not share sample information. This keeps storage requirements low and allows easy parallel processing. To process each block, we initially divide it into 4x4 pixel blocks but may divide it further based on the following tests.

To compute a block, we compute samples at its corners, shoot eye rays through its pixels, and test to see if the resulting points match the corner samples. If using anti-aliasing, there will be multiple eye rays per pixel. A set of eye rays is said to match if they all hit surfaces with the same type of material and with surface normals that differ by no more than some angle (e.g., 30 degrees). We also use a cone test to detect possible local shadowing conditions that require block subdivision (e.g., see Figure 12). For each eye ray, we construct a cone starting at its intersection point and centered around the local surface normal. If the intersection point for any other eye ray lies within this cone, then the block fails the cone test. The cone test uses true geometric normals and is unaffected by shading effects such as bump maps or interpolated normals.

If the eye rays do not match and the block is bigger than a pixel, then we split the block into four smaller blocks and try again. If the block is pixel-sized, we relax the requirements, omit the cone test, and only require that each eye ray match at least two nearby samples. If there are still not enough matching samples, then we compute a new sample at that eye ray.

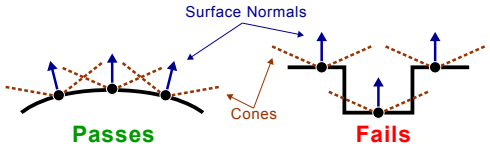


Figure 12: Block cone test. (Right) fails because the upper two points lie within the cone of the lower point. (Left) passes because no points lie within the cones defined by the other points.

After block subdivision we compute a color for each remaining eye ray using reconstruction cuts. For blocks larger than a pixel we use the four corners as the set of nearby samples and use image-space bilinear interpolation weights when interpolating impostors. Eye rays within pixel-sized blocks use their set of matching samples and interpolation weights proportional to the world-space inverse distance squared between the surface point and the sample points.

8 Reconstruction Cut Results

In this section we present some results of applying the reconstruction cuts technique to our scenes. Result images and statistics are shown in Figure 13. These results are adaptively anti-aliased [Painter and Sloan 1989] using between 5 and 50 eye rays per pixel. The sparse samples are computed using lightcuts with the same parameters as in Section 6. Most of the shading is done using reconstruction cuts that interpolate intelligently between the samples.

By exploiting spatial coherence, reconstruction cuts can shade points using far fewer shadow rays than lightcuts. While a lightcut requires a few hundred shadow rays, on average a reconstruction cut uses less than fourteen in our results. In fact, most of our shadow rays are used for computing the sparse samples (lightcuts) even though there are 15-25 times more reconstruction cuts. This allows us to generate much higher quality images, with anti-aliasing, at a much lower cost than with lightcuts alone. For the same size images, the results are both higher quality and have similar or lower cost than those in Section 6. Moreover for larger images, rendering cost increases more slowly than the number of pixels.

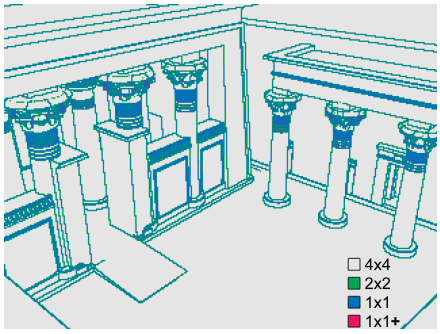
Samples are computed at the corners of adaptively sized image blocks (between 4x4 and 1x1 pixels in size) and occasionally within a pixel when needed (shown as 1x1+ in red). As shown for the temple image, most of the pixels in all images lie within 4x4 blocks. An average of less than one sample per pixel is needed even with anti-aliasing requiring multiple eye rays per pixel. We could allow larger blocks (e.g., 8x8) but making the samples even sparser where they are already sparse has less benefit and can be counterproductive because reconstruction cut cost is strongly related to the similarity of the nearby samples.

Because the accuracy of reconstruction cuts relies on having nearby samples that span the local lighting conditions, there is a possibility of missing small features in between samples. While this does happen occasionally, it is rarely problematic. For example in a few places, the interpolation smooths over the grooves in the pillars of the temple, but the errors are quite small and we have found that people have great difficulty in noticing them. Future improvements in the block refinement rules could fix these errors.

A Metropolis solution and its magnified (5x) differences with our result are shown in Figure 14. Metropolis [Veach and Guibas 1997] is considered the best and fastest of the general purpose Monte Carlo solvers. Its result took 13 times longer to compute than ours and still contains some visible noise. The main differences are the noise in the Metropolis solution and some corner darkening in our

Kitchen

Tableau



Temple (Reconstruction Block Size)

Grand Central

Temple

Model	Point Lights	Per Pixel Averages		Avg Shadow Rays Per Pixel		Avg Per Reconstruction Cut		Image Time	
		Eye Rays	Samples	Samples	Recon Cuts	Shadow Rays	Interpolations	1280x960	640x480
Kitchen	59672	5.4	0.29	143	50	**9.1**	14.4	672s	257s
Tableau	13000	5.4	0.23	50	41	**10.6**	17.8	298s	111s
Grand Central	143464	6.9	0.46	225	93	**13.3**	11.5	1177s	454s
Temple	505064	5.5	0.25	91	52	**9.4**	6.0	511s	189s
Bigscreen	639528	5.3	0.25	64	24	**4.6**	15.0	260s	98s

Figure 13: Reconstruction cut results for 1280x960 images (except where noted) of our scenes. Images are anti-aliased using 5 to 50 eye rays per pixel adaptively. Samples (lightcuts from Figure 9) are computed sparsely and most shading is computed using reconstruction cuts to interpolate intelligently. Reconstruction cuts are much less expensive and require many fewer shadow rays than lightcuts. Bigscreen image is shown in Figure 1.

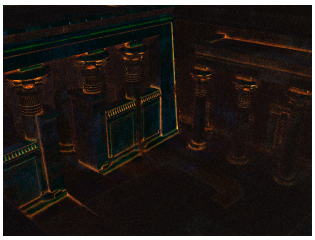
result. The latter is due to instant radiosity not reproducing very short range indirect illumination effects.

9 Conclusion

We have presented lightcuts as a new scalable unifying framework for illumination. The core component is a strongly sublinear algorithm for computing the illumination from thousands or millions of point lights using a perceptual error metric and conservative per cluster error bounds. We have shown that it can greatly reduce the number of shadow rays, and hence cost, needed to compute illumination from a variety of sources including area lights, HDR environment maps, sun/sky models, and indirect illumination. Moreover it can handle very complex scenes with detailed geometry and glossy materials. We have also presented reconstruction cuts that further speed shading by exploiting coherence in the illumination.

There are many ways in which this work can be improved and extended. For example, extending to additional illumination types (e.g., indirect components not handled by instant radiosity), integrating more types of light sources including non-diffuse sources and spot lights, developing bounds for more BRDF types, more formal analysis of lightcuts stochastic error, further refinement of the reconstruction cut rules to exploit more coherence, and adding conservative visibility bounds to further accelerate lightcuts.

Metropolis Solution



5 x Difference

Figure 14: Comparison with metropolis image for temple.

Acknowledgments

Many thanks to the modelers: Jeremiah Fairbanks (kitchen), Will Stokes (bigscreen), Moreno Piccolotto, Yasemin Kologlu, Anne Briggs, Dana Getman (Grand Central) and Veronica Sundstedt, Patrick Ledda, and the Graphics Group at University of Bristol (temple). This work was supported by NSF grant ACI-0205438 and Intel Corporation. The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U. S. Government.

References

AGARWAL, S., RAMAMOORTHI, R., BELONGIE, S., AND JENSEN, H. W. 2003. Structured importance sampling of environment maps. *ACM Transactions on Graphics 22*, 3 (July), 605–612.

BLACKWELL, H. R. 1972. Luminance difference thresholds. In *Handbook of Sensory Physiology*, vol. VII/4: Visual Psychophysics. Springer-Verlag, 78–101.

COHEN-OR, D., CHRYSANTHOU, Y. L., SILVA, C. T., AND DURAND, F. 2003. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics 9*, 3, 412–431.

DEBEVEC, P. 1998. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 189–198.

FERNANDEZ, S., BALA, K., AND GREENBERG, D. P. 2002. Local illumination environments for direct lighting acceleration. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, 7–14.

HANRAHAN, P., SALZMAN, D., AND AUPPERLE, L. 1991. A rapid hierarchical radiosity algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, vol. 25, 197–206.

HASENFRATZ, J.-M., LAPIERRE, M., HOLZSCHUCH, N., AND SILLION, F. 2003. A survey of real-time soft shadows algorithms. In *Eurographics*, Eurographics, Eurographics. State-of-the-Art Report.

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

KELLER, A. 1997. Instant radiosity. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 49–56.

KOK, A. J. F., AND JANSEN, F. W. 1992. Adaptive sampling of area light sources in ray tracing including diffuse interreflection. *Computer Graphics Forum (Eurographics '92) 11*, 3 (Sept.), 289–298.

KOLLIG, T., AND KELLER, A. 2003. Efficient illumination by high dynamic range images. In *Eurographics Symposium on Rendering: 14th Eurographics Workshop on Rendering*, 45–51.

KŘIVÁNEK, J., GAUTRON, P., PATTANAIK, S., AND BOUATOUCH, K. 2005. Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics*.

LARSON, G. J. W. 1992. Measuring and modeling anisotropic reflection. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26, 265–272.

PAINTER, J., AND SLOAN, K. 1989. Antialiased ray tracing by adaptive progressive refinement. In *Computer Graphics (Proceedings of SIGGRAPH 89)*, vol. 23, 281–288.

PAQUETTE, E., POULIN, P., AND DRETTAKIS, G. 1998. A light hierarchy for fast rendering of scenes with many lights. *Computer Graphics Forum 17*, 3, 63–74.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Commun. ACM 18*, 6, 311–317.

PREETHAM, A. J., SHIRLEY, P. S., AND SMITS, B. E. 1999. A practical analytic model for daylight. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 91–100.

SCHEEL, A., STAMMINGER, M., AND SEIDEL, H.-P. 2001. Thrifty final gather for radiosity. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, 1–12.

SCHEEL, A., STAMMINGER, M., AND SEIDEL, H. 2002. Grid based final gather for radiosity on complex clustered scenes. *Computer Graphics Forum 21*, 3, 547–556.

SHIRLEY, P., WANG, C., AND ZIMMERMAN, K. 1996. Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics 15*, 1 (Jan.), 1–36.

SILLION, F. X., AND PUECH, C. 1994. *Radiosity and Global Illumination*. Morgan Kaufmann Publishers Inc.

SMITS, B., ARVO, J., AND GREENBERG, D. 1994. A clustering algorithm for radiosity in complex environments. In *Proceedings of SIGGRAPH 94*, Annual Conference Series, 435–442.

TABELLION, E., AND LAMORLETTE, A. 2004. An approximate global illumination system for computer generated films. *ACM Transactions on Graphics 23*, 3 (Aug.), 469–476.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 65–76.

WALD, I., KOLLIG, T., BENTHIN, C., KELLER, A., AND SLUSALLEK, P. 2002. Interactive global illumination using fast ray tracing. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, 15–24.

WALD, I., BENTHIN, C., AND SLUSALLEK, P. 2003. Interactive global illumination in complex and highly occluded environments. In *Eurographics Symposium on Rendering: 14th Eurographics Workshop on Rendering*, 74–81.

WALTER, B., ALPPAY, G., LAFORTUNE, E. P. F., FERNANDEZ, S., AND GREENBERG, D. P. 1997. Fitting virtual lights for non-diffuse walkthroughs. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 45–48.

WALTER, B. 2005. Notes on the Ward BRDF. Technical Report PCG-05-06, Cornell Program of Computer Graphics, Apr.

WARD, G. J., AND HECKBERT, P. 1992. Irradiance gradients. In *Third Eurographics Workshop on Rendering*, 85–98.

WARD, G. 1994. Adaptive shadow testing for ray tracing. In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, Springer-Verlag, New York, 11–20.

WOO, A., POULIN, P., AND FOURNIER, A. 1990. A survey of shadow algorithms. *IEEE Computer Graphics and Applications 10*, 6 (Nov.), 13–32.

ZANINETTI, J., BOY, P., AND PEROCHE, B. 1999. An adaptive method for area light sources and daylight in ray tracing. *Computer Graphics Forum 18*, 3 (Sept.), 139–150.

A Spherical Lights

Any diffuse area light can be simulated using oriented lights on its surface. For spherical lights, we have developed a better technique that simulates them using omni lights (which are a better match for far field emission). Placing omni lights on the surface of the sphere would incorrectly make it appear too bright near its silhouette as compared to its center. Similarly a uniform distribution inside the volume of the sphere would exhibit the reverse problem. However, by choosing the right volume distribution inside the sphere, we can correctly match the emission of a spherical light.

$$d(\mathbf{x}) = \frac{1}{\pi^2 R^2 \sqrt{R^2 - r^2(\mathbf{x})}} \tag{7}$$

The normalized point distribution $d(\mathbf{x})$ is defined inside a sphere of radius R where $r^2(\mathbf{x})$ is the squared distance from the sphere's center. The beauty of this distribution is that it projects to a uniform distribution across the apparent solid angle of the sphere when viewed from any position outside the sphere, which is exactly the property we need. We also need a way to generate random points according to this distribution. Given three uniformly distributed random numbers, ξ_1, ξ_2, ξ_3, in the range $[0,1]$ we can compute points with the right distribution for a sphere centered at the origin using:

$$\begin{aligned} x &= R\sqrt{\xi_1}\cos(2\pi\xi_2) \\ y &= R\sqrt{\xi_1}\sin(2\pi\xi_2) \\ z &= \sqrt{R^2 - x^2 - y^2}\sin(\pi(\xi_3 - 1/2)) \end{aligned} \tag{8}$$

The omni lights generated from spherical lights behave exactly like normal omni lights except that when computing their visibility factors, they can only be occluded by geometry outside the sphere (i.e. their shadow rays terminate at the surface of the sphere).