Multidimensional Lightcuts

Bruce Walter Adam Arbree Kavita Bala Donald P. Greenberg
Cornell University*

Abstract

Multidimensional lightcuts is a new scalable method for efficiently rendering rich visual effects such as motion blur, participating media, depth of field, and spatial anti-aliasing in complex scenes. It introduces a flexible, general rendering framework that unifies the handling of such effects by discretizing the integrals into large sets of gather and light points and adaptively approximating the sum of all possible gather-light pair interactions.

We create an implicit hierarchy, the product graph, over the gather-light pairs to rapidly and accurately approximate the contribution from hundreds of millions of pairs per pixel while only evaluating a tiny fraction (e.g., 200–1,000). We build upon the techniques of the prior Lightcuts method for complex illumination at a point, however, by considering the complete pixel integrals, we achieve much greater efficiency and scalability.

Our example results demonstrate efficient handling of volume scattering, camera focus, and motion of lights, cameras, and geometry. For example, enabling high quality motion blur with 256× temporal sampling requires only a 6.7× increase in shading cost in a scene with complex moving geometry, materials, and illumination.

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

Keywords: motion blur, volume rendering, depth of field

1 Introduction

Multidimensional lightcuts is a new scalable system for efficiently rendering complex effects such as motion blur, participating media, depth of field, and spatial anti-aliasing, in scenes with complex illumination, geometry, and materials. These effects can be expressed as integrals over different domains, such as integrating over time for motion blur and over the camera's aperture for depth of field. Our approach first discretizes these integrals into a sum over sets of points and then provides a scalable and accurate method for estimating the total contribution from the resulting large point sets. This approach both unifies the handling of the different integral domains and allows us to solve simultaneously for multiple effects.

Our approach builds on the ideas and techniques of Lightcuts [Walter et al. 2005], which provided a scalable solution for computing the illumination at a single point from multiple complex sources. However we take a more holistic approach that considers the complete integrals over an entire pixel. This enables greater efficiency and scalability, as individual components may not need high accuracy as long as the total pixel estimate is accurate.

*Program of Computer Graphics & Department of Computer Science email:{bjw,arbree,kb,dpg}@graphics.cornell.edu

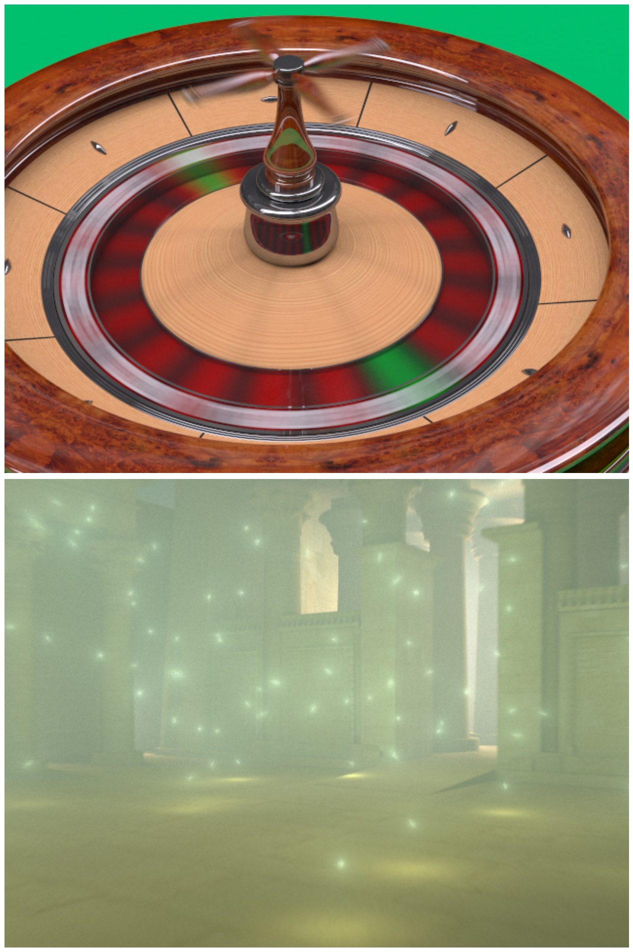


Figure 1: Results for the roulette and temple scenes demonstrating motion blur and participating media. See Figure 4 for statistics.

We first discretize the integrals into a computation over two point sets, *gather points* and *light points*, generated from the camera and light sources respectively. The integrals can be approximated by summing all the pairwise interactions between gather and light points, but evaluating each pair would be prohibitively expensive. Instead we create an implicit hierarchy, the product graph, over the gather-light pairs, and use it to adaptively select a cut that partitions the gather-light pairs into clusters. Similarly to Lightcuts, we use conservative cluster error bounds and a perceptual metric to select an appropriate cut. The contribution of each cluster is estimated by selecting representative pairs from the clusters.

Our results demonstrate our ability to simultaneously handle multiple effects (e.g., Figure 1) including temporal and spatial anti-aliasing, depth of field, specular surfaces, and participating media. The method scales extremely well to efficiently handle hundreds of millions of gather-light pairs per pixel. For example, for motion blur in our roulette scene we can achieve the quality of 256x temporal supersampling with only a 6.7x increase in shading cost, and are significantly faster than using the Metropolis algorithm (Figure 8).

Section 2 discusses related work. Section 3 introduces the product graph and describes our algorithm. Section 4 discusses how multidimensional lightcuts can be used to render rich visual effects. Section 5 gives results, and we conclude in Section 6. Appendix A covers some details about a refinement heuristic we use.

2 Previous Work

Our approach handles a wide range of visual effects. Here we focus only on reviewing work related to computing the shading integrals these effects require. We first discuss pure Monte Carlo approaches as a general solution, and then consider special optimizations for individual effects. See [Walter et al. 2005] for related work on the general problem of scalable rendering of complex illumination.

Monte Carlo rendering is the most general approach for computing the effects included in this paper; however even with optimized sampling patterns (e.g., [Mitchell 1991]) it often converges slowly. Two general approaches to reduce the cost of Monte Carlo approaches are path reuse and importance sampling.

Most closely related to our research are path reuse algorithms. Algorithms, like bidirectional path tracing [Lafortune and Willems 1993] and its variants [Bekaert et al. 2002; Havran et al. 2003], generate and reuse camera and light paths but have trouble dealing with the combinatoric explosion in potential path connections. Our system is able to more intelligently sample the space of potential connections. Metropolis [Veach and Guibas 1997] and variants [Cline et al. 2005] keep only a very small set of recent paths and use path mutations and perturbations to adaptively explore path space. However, their lack of global knowledge can limit their ability to efficiently sample complex effects.

Importance sampling techniques improve convergence by intelligent sample placement. [Lawrence et al. 2004] demonstrates an importance sampling method for complex BRDFs based on a unifying factored representation. [Lawrence et al. 2005; Clarberg et al. 2005; Burke et al. 2005; Talbot et al. 2005] focus on multiple importance sampling of BRDFs and HDR environment maps, primarily to accelerate direct illumination. While importance sampling is powerful, it is hard to effectively importance sample the complex effects supported in this paper. Also, while the scalability of our method makes initial point selection less crucial, many importance sampling techniques could be incorporated into our initial point generation phase to further improve our performance.

Photon mapping has been extended to render effects such as participating media [Jensen and Christensen 1998] and motion blur [Cammarano and Jensen 2002]. However, these approaches limit themselves to reducing the cost of indirect illumination and rely on traditional techniques for direct illumination. Our lightcuts-based approach unifies the handling of direct and indirect and is more efficient in scenes with complex direct illumination.

Motion blur literature is reviewed in detail in [Sung et al. 2002] and [Damez et al. 2003] presents a comprehensive survey of global illumination techniques for dynamic scenes.

The motion blur system in Maya [Sung et al. 2002] uses an analytic visibility computation similar to [Korein and Badler 1983] to analytically sample visibility changes, but relies on supersampling for shading changes. Our approach greatly reduces supersampling costs and is complementary to their methods. The Reyes architecture [Catmull 1984; Cook et al. 1987] dices geometry into sub-pixel pieces called micro-polygons. Motion blur due to visibility changes is approximated by warping micro-polygons, but temporally changing shading still requires supersampling.

Several methods [Myszkowski et al. 2000; Myszkowski et al. 2001; Tawara et al. 2004] for accelerating the rendering of animated sequences produce an approximation of motion blur as a side effect of reusing shading information between frames. There are also image-based methods [Max and Lerner 1985; Wloka and Zeleznik 1996] that approximate motion blur by sacrificing correctness for speed. However, both types of methods focus on producing a qualitative, but not necessarily correct, approximation of motion blur.

Volumetric and participating media rendering techniques are surveyed in [Cerezo et al. 2005]. Interactive approaches, e.g., [Sun et al. 2005], make simplifying assumptions, such as only simulating single scattering or ignoring occlusion, to achieve performance at the cost of accuracy. [Premoze et al. 2004] accelerates rendering by precomputing a cache of approximate multiple scattering contributions; however, these precomputations require data structures per light and prohibit the use of complex illumination. Finally, [Pauly et al. 2000] extends Metropolis light transport for participating media, but still requires high sampling rates.

2.1 Lightcuts

Lightcuts [Walter et al. 2005] introduced a scalable solution for computing the illumination at a point from many complex sources including area lights, sun/sky models, high-dynamic range (HDR) environment maps and indirect illumination. This is effectively solving the rendering equation at a point x:

$$L(x,\omega) \quad = \quad \int_{\Omega} f_{\mathrm{r}}(x,\omega,\omega')L(x,\omega')d\omega'_{\perp} \tag{1}$$

where L is the radiance, ω is viewing direction, f_{r} is the BRDF, and Ω is the sphere of directions. By approximating the light sources as many point lights, Lightcuts discretizes this integral into the sum:

$$L_j \quad = \quad \sum_{i \in \mathbb{L}} M_{ji} G_{ji} V_{ji} I_i \tag{2}$$

The point j is being illuminated by the set of point lights $\mathbb{L}$. The contribution of each light i is the product of a material term, $M_{ji} = f_{\mathrm{r}} \cos\theta$, a geometric term G_{ji} that includes the lights emission distribution and distance, the visibility V_{ji} between the point and the light, and the light's intensity I_i. For simplicity, these are written as scalars, but may also represent RGB or spectral values.

Accurate approximations often require very large light sets $\mathbb{L}$ and Lightcuts provides a scalable solution for such cases. It organizes the lights into a light tree hierarchy and then adaptively selects cuts in the tree that partition the lights into clusters based on a perceptual metric and conservative bounds on the error in estimating the contribution of a cluster. Clusters are approximated by selecting and evaluating a representative light from the cluster.

An extension called reconstruction cuts exploits image coherence but only in regions which are sufficiently similar in material, orientation, and local occlusion. These are difficult to extend to more complex rendering problems and thus will not be considered here.

Lightcuts is very effective at handling high illumination complexity at a single point, but in this paper we will look at the larger problem of computing complex illumination integrated over entire pixels domains. We describe how to extend the following key concepts from Lightcuts for solving these larger problems: point-based discretization of the integral domain, point hierarchy and clustering, cut selection via conservative error bounds and a perceptual metric, and representative-based cluster approximation.

3 Multidimensional Lightcuts

Effects such as temporal blur, depth of field, volumetric effects and anti-aliasing can be expressed as integrals over multiple dimensions or domains [Cook et al. 1984]. For example, we may want to integrate over time, volume, camera aperture, and image plane:

$$pixel \quad = \quad \int_{time} \int_{volume} \int_{aperture} \int_{pixel\ area} L(x,\omega) \tag{3}$$

We could discretize the pixel integral into a set of points, evaluate the radiance L at each point, and sum the radiances to approximate such complex integrals. This approach has been used by many previous methods including supersampling and bidirectional path tracing. The problem is that a large number of points are often required for good approximations, which quickly becomes very expensive especially when the illumination is also complex.

Multidimensional lightcuts is a unified scalable point-based approach for rapidly and accurately approximating such multidimensional integrals. It provides an efficient and accurate algorithm for estimating the contribution from large point sets.

Multidimensional lightcuts first discretizes the illumination sources into a set of point lights $\mathbb{L}$, using the techniques of Lightcuts [Walter et al. 2005]. Then for each pixel it generates a set of gather points $\mathbb{G}$, by tracing rays from the eye or camera. The total pixel value is:

$$\begin{aligned} pixel &= \sum_{(j,i)\in\mathbb{G}\times\mathbb{L}} L_{ji} && (4)\\ &= \sum_{(j,i)\in\mathbb{G}\times\mathbb{L}} S_j M_{ji} G_{ji} V_{ji} \tau_{ji} I_i && (5)\end{aligned}$$

where, the M, G, V, and I terms are the material, geometry, visibility and intensity terms as in Equation 2, and:

- S_j is the strength of a gather point. We normalize the material term so that it integrates to one over direction space. Thus the strength gives the relative weight of each gather point in the integral. The sum of the strengths of all the gather points equals the average albedo over the pixel and is thus ≤ 1.
- τ_{ji} is a binary variable that checks if points i and j exist at the same time instant to ensure that we only evaluate interactions between points that exist at the same instant of time.

Directly evaluating all pair-wise interactions (g,l), where g is a gather point in $\mathbb{G}$, and l is a light point in $\mathbb{L}$, requires $|\mathbb{G}||\mathbb{L}|$ computations, which is prohibitively expensive. Instead, we apply adaptive estimation techniques like those used in Lightcuts. However, even explicitly constructing a hierarchy over the (g,l) pairs would require $O(|\mathbb{G}||\mathbb{L}|)$ work for each pixel, which is too expensive.

3.1 The Product Graph

We use an *implicit* construction of a hierarchy over the space of gather-light pairs. We first construct separate hierarchies over the gather points and the light points: the gather tree and light tree, respectively. Each node of these trees represents a cluster of all the gather or light points beneath them. The Cartesian *product graph* of the gather tree and light tree: $\mathbb{P} = Tree(\mathbb{G}) \times Tree(\mathbb{L})$, is then an implicit hierarchy on the set of all gather-light pairs, as illustrated in Figure 2. The root node of the product graph corresponds to the set of all gather-light pairs (pairing of the gather and light tree roots) while leaf nodes correspond to individual gather-light pairs (pairing of leaf nodes from the gather and light trees). Two nodes in the product graph have a parent-child connection if they correspond to the same node in one tree (gather or light) and to parent-child nodes in the other tree. Note that the product graph itself is not a tree as there can be multiple different paths from the root to a single leaf.

This implicit construction allows us to compute using a hierarchy of gather-light pairs without actually having to explicitly construct the full hierarchy. Instead, only the two, much smaller, gather and light trees are required. We denote a node in the product graph representing a cluster of gather-light pairs as $\mathbb{C}$.

3.2 Cuts in the Product Graph

Next we extend the notion of a *cut* to apply to our product graphs. A cut partitions the set of gather-light pairs into clusters and the goal

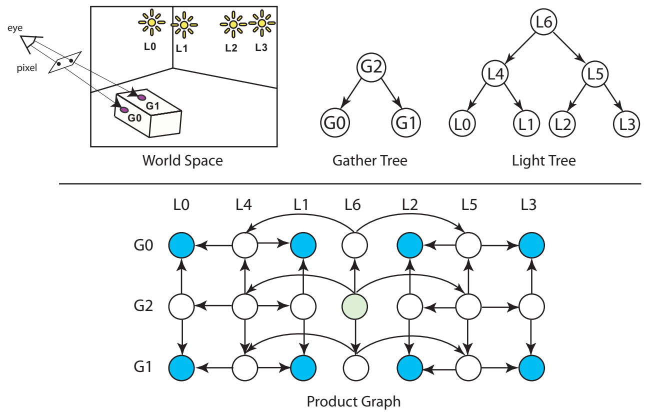


Figure 2: Product graph. Top left: scene with two gather points and four light points. Top right: gather and light cluster trees. Bottom: product graph of gather tree and light tree. Each product graph node corresponds to the pairing of a gather and light node and represents all pairwise interactions between points in their respective clusters. The light green node is the source/root, and the dark blue nodes are sinks/leaves in the product graph.

is to adaptively select a cut that will result in an accurate approximation of the pixel. In analogy with Lightcuts, we define a cut in the product graph as a set of nodes such that, for any leaf node, the set of all paths from the root to that leaf node will contain exactly one node from the cut. Thus every gather-light pair will be included in one and only one cluster on the cut.

We approximate the contribution of an individual cluster $\mathbb{C}$ by using a representative gather-light pair (g,l) as follows:

$$\tilde{L}_{\mathbb{C}} = M_{gl} G_{gl} V_{gl} \sum_{(j,i)\in\mathbb{C}} S_j \tau_{ji} I_i \qquad (6)$$

where the material, geometry, and visibility terms are evaluated at the representative (g,l). Note that we also require that g and l must exist at the same time instant; otherwise the estimate is ill-defined. Representative selection will be discussed in the next section.

To explain how to efficiently compute the sum in Equation 6, we need new notation. In our system, we discretize time into a fixed set of T time instants for any frame. Vectors of values over the time instants will be denoted with arrows like $\vec{V}$. Using unit time vectors, $\hat{e}_k$, we can express the k-th element of a time vector as $\vec{V} \cdot \hat{e}_k$. We can then represent all the strengths S and intensities I of the gather and light points as time vectors. For example, if light i exists at the second of four time instants and has intensity 5 then $\vec{I}_i = (0,5,0,0)$. The sum can then be computed using a dot product of time vectors:

$$\tilde{L}_{\mathbb{C}} = M_{gl} G_{gl} V_{gl} (\vec{S}_{\mathbb{C}} \cdot \vec{I}_{\mathbb{C}}) \qquad (7)$$

where $\vec{S}_{\mathbb{C}}$ is equal to the sum of the strength vectors for all the gather points in the cluster, which can be precomputed and stored in the gather tree. Similarly $\vec{I}_{\mathbb{C}}$ is the sum of the intensity vectors of the lights in the cluster and is cached in the light tree.

3.3 Representatives in the Product Graph

We can make the cluster approximation unbiased, in a Monte Carlo sense, by choosing cluster representative pair $(g,l) \in \mathbb{C}$ according to the probability:

$$p_{\mathbb{C}}(g,l) = \frac{(\vec{S}_g \cdot \vec{I}_l)}{(\vec{S}_{\mathbb{C}} \cdot \vec{I}_{\mathbb{C}})} \qquad (8)$$

To make cluster representative finding efficient, we split this prob-

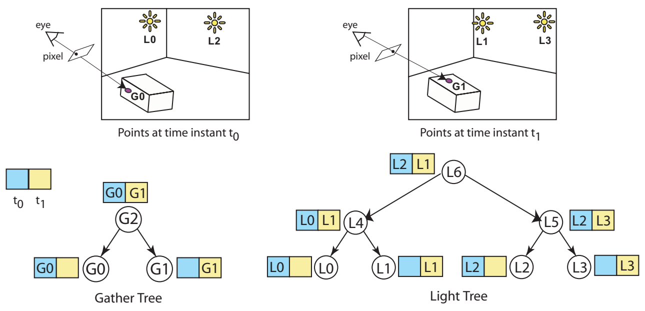


Figure 3: Multiple representatives for product graph with two time instants. Blue elements show the cached representatives for time t_0 and yellow elements show the representatives for time t_1

ability into three components as follows:

$$p_{\mathbb{C}}(g,l) = p_{\mathbb{C}}(t)\, p_{\mathbb{C}}(g|t)\, p_{\mathbb{C}}(l|t) \tag{9}$$

$$p_{\mathbb{C}}(t) = \frac{(\vec{S}_{\mathbb{C}} \cdot \hat{e}_t)(\vec{I}_{\mathbb{C}} \cdot \hat{e}_t)}{(\vec{S}_{\mathbb{C}} \cdot \vec{I}_{\mathbb{C}})} \tag{10}$$

$$p_{\mathbb{C}}(g\,|\,t) = \frac{(\vec{S}_g \cdot \hat{e}_t)}{(\vec{S}_{\mathbb{C}} \cdot \hat{e}_t)} \tag{11}$$

$$p_{\mathbb{C}}(l\,|\,t) = \frac{(\vec{I}_l \cdot \hat{e}_t)}{(\vec{I}_{\mathbb{C}} \cdot \hat{e}_t)} \tag{12}$$

where we first pick a time instant t and then select representative gather and light points based on this time instant. This method has several advantages. It guarantees that we will only pick representative pairs (g,l) that exist at the same time instant. Also since $p_{\mathbb{C}}(g|t)$ depends only on the gather points, the gather representatives for each time instant can be selected during the gather tree construction and cached for fast lookup. Similarly $p_{\mathbb{C}}(l|t)$ depends only on the light points and its representatives can be cached in the light tree. Thus each node in the gather and light trees stores a vector of T representatives. An example is shown in Figure 3.

Storing multiple representatives per node is also useful even when not including time ($T = 1$). Randomly choosing from a preselected list of representatives (e.g., 32 per cluster) creates a better distribution of shadow rays for cluster estimation as compared to always using the same representative for a gather or light cluster.

Representative sharing. As in Lightcuts, we can make the cut selection process more efficient by ensuring that parents always share each representative with one of their children. This allows the material, geometric, and visibility results for that representative to be reused during the cut refinement process. We enforce this during tree construction by always selecting the representative for a parent at time instant t from the representatives of its two children for t. This is done in both the gather and light trees.

3.4 Algorithm Overview

We are now ready to summarize the multidimensional lightcuts algorithm. First in a preprocess, discretize time into T instants, generate the light set $\mathbb{L}$ and build the light tree. Then for each pixel, perform the following. Trace rays from the eye or camera to generate the gather points $\mathbb{G}$ and build the gather tree. Initialize the cut with a coarse cut (e.g., the root node of the product graph). Then iteratively refine the cut:

1. Select the node in the cut with the largest error bound. If this error bound is less than the perceptual error threshold (Equation 15), terminate refinement.
2. Otherwise refine the node by removing it from the cut and replacing it with two of its children. This can be done by moving down one step in either the gather or light tree. The choice is made using the refinement heuristic from Appendix A.
3. Compute the cluster estimates (Equation 7) for new nodes and update the current overall pixel estimate. One child will share the parent's representative and reuse the same time instant and representative M, G, and V terms from its parent. The other child will randomly select a new time instant and representative pair from the cached lists in the trees.
4. Compute the conservative error bounds (Equation 13) for each of the two new nodes. Goto step 1.

For example in Figure 3, the cut might start with the root node (G_2,L_6) with representative (G_1,L_1) at time t_1. If light tree refinement was chosen we would replace it in the cut with (G_2,L_4) and (G_2,L_5). In this case (G_2,L_4) would reuse its parent's representative (G_1,L_1) while (G_2,L_5) would pick a new time instant and representative such as (G_0,L_2) at t_0. And the process would repeat.

Light Tree Generation. The light points and tree are constructed once per frame as a preprocess. The light sources in the scene such as area lights, sun/sky models, and HDR environment maps are first approximated using point lights. Next we emit light particles from the lights and record their hit locations. These locations are converted into virtual point lights that simulate indirect illumination. Finally the point lights are organized into a light tree. We use the same light types (omni, oriented, and directional) and same light tree building process as Lightcuts. However to support participating media we also allow particles to scatter in the volume and generate volume indirect lights. In our initial implementation with uniform phase functions, the volume lights are omni lights that are subject to clamping just like other indirect lights.

Each node in the light tree records its spatial bounding box, a bounding direction cone (if it contains oriented lights), an array with its representative for each time instant, and its intensity (as a time vector), which is simply the sum of the intensity vectors from its children. Storing the intensity and representative vectors is our largest memory overhead, but most nodes are low in the tree and have very sparse vectors that can easily be compressed.

Gather Tree Generation. For each pixel, we trace eye rays from the camera to generate a new set of gather points. Depending on the scene each eye ray may generate zero, one, or multiple gather points. We support both volume and surface gather points. Volume points record the local phase function, which is always uniform currently. Surface points record the local normal and bidirectional reflectance distribution function (BRDF). We currently support the same BRDF types as Lightcuts [Walter et al. 2005], namely diffuse, phong, Ward, and specular materials.

Because the gather tree must be rebuilt for each pixel, its construction must be fast. We use a simple top-down kd-tree construction instead of the greedy bottom-up construction used for light trees. During construction we map each gather point to a point in a 6D space and then iteratively split the points along the longest axis of their 6D bounding boxes until each cell contains no more than two points, thus creating a binary gather tree.

Points are grouped based on spatial and directional similarity (temporal similarity is not currently considered). The 6D coordinates used are the gather point's three spatial coordinates plus three coordinates based on the directionality of its material. Diffuse points map to points on the unit sphere based on the local surface normal, glossy surfaces map to vectors in their reflection direction with lengths between one and four based on the sharpness of the gloss. Volume points map to the directional origin. These directional coordinates are then scaled by 1/8 the diagonal of the scene's bounding box before kd-tree construction.

Similar to light nodes, each node in the gather tree records its spatial bounding box, a normal bounding cone (if it contains diffuse materials), a bounding cube-map (if it contains glossy surfaces), an array giving its representative for each time instant, and its strength (as a time vector), which is the sum of the strength vectors from its children. We will discuss the normal cone and cube-map next.

3.5 Error Bounds

The cluster error bounds are upper bounds on the maximum error that can be introduced by approximating a cluster using its representative pair (Equation 7) and is computed as:

$$\left\|\tilde{L}_{\mathbb{C}} - L_{\mathbb{C}}\right\| \leq M^u_{\mathbb{C}} G^u_{\mathbb{C}} V^u_{\mathbb{C}} (\vec{S}_{\mathbb{C}} \cdot \vec{I}_{\mathbb{C}}) \tag{13}$$

where $M^u_{\mathbb{C}}$, $G^u_{\mathbb{C}}$, and $V^u_{\mathbb{C}}$ are upper bounds on the material, geometric, and visibility terms over all the gather-light pairs in cluster $\mathbb{C}$. For visibility we use the trivial upper bound of one. If the cluster only contains one gather point, then we directly use the upper bounds from Lightcuts since this is the case they considered and we support the same material and light types. However we need additional techniques to bound clusters with multiple gather points.

Given bounding boxes for the gather and light cluster, we can collapse the problem of bounding the interactions between two boxes into the simpler problem of point to box bounding using Minkowski sums. Given a set of point pairs $(\mathbf{x}_i, \mathbf{y}_i)$, we can translate each pair by $-\mathbf{x}_i$ to get $(\mathbf{o}, \mathbf{y}_i - \mathbf{x}_i)$ where $\mathbf{o}$ is the origin. Given bounding boxes on $\mathbf{x}_i$ and $\mathbf{y}_i$ it is easy to compute a bounding box for $\mathbf{y}_i - \mathbf{x}_i$. We can then evaluate the existing G bounds for this larger box.

Bounding M is more difficult because the gather points in a cluster may have different materials. Computing bounds for each point in a cluster individually would be expensive, instead we create a combined material bounding function. We split the materials into diffuse and glossy components which are bounded separately. We bound the diffuse component by keeping track of a bounding cone of all the point's surface normals in the cluster. This is very similar to the G bounding for clusters of oriented lights in Lightcuts.

Gloss components are more difficult because they depend on the local material properties, surface normal, and incident direction of the eye ray. Our solution is to discretize direction space using a cube-map decomposition. We use a 6x6 subdivision on each cube face for a total of 216 directional cells in the cube map. For each glossy gather point we build a cube map that stores its maximum material value over the set of directions corresponding to that cell. This is easily computed using the material bounds from Lightcuts.

Once converted to a cube map, we can compute the gloss bounding function over multiple points by taking the maximum value from the corresponding cells and storing it in another cube-map. To bound the maximum M value for a cluster, we project its bounding box onto the cube-map and take the maximum of the cells it touches. The cube-map has the advantage of being very general and easy to work with, but the disadvantage of limited resolution and being somewhat expensive to initialize for a gather point. Adaptive techniques, such as quadtrees or wavelets, would improve this.

Using the diffuse normal cones and gloss cube maps allows us to combine points which different material terms and quickly compute upper bounds on their material terms. Initially we used the cube maps for both diffuse and glossy components, but splitting them gave tighter bounds.

Indirect illumination and clamping. Our system is built on Lightcuts and uses the same Instant Radiosity [Keller 1997] approach for indirect illumination, which does not include caustics and requires some clamping to avoid objectionable noise artifacts.

The indirect clamping had to be modified to work with multiple gather points. The clamping threshold in Lightcuts is chosen such that no indirect light can contribute more than a fixed percentage (e.g., 1%) of the total illumination. This works well for a single gather point, but becomes problematic when there are multiple gather points. For example, an indirect light could contribute 0.1% to each of a hundred gather points for a total of 10% which would cause visible noise. Instead, we clamp the product of the material and geometric terms (e.g., $M_{ji}G_{ji}$ in Equations 5 or 7) for indirect lights by enforcing:

$$MG \leq \frac{L_a}{N\|I_i\|} \tag{14}$$

where L_a is the adaptation luminance for the image, N is a constant, and I_i is the intensity of an individual indirect light. The indirect light generation process is designed such that all indirect lights have the same intensity. This guarantees that the maximum contribution of an indirect light to a pixel over all gather points will be $\leq L_a/N$. This is easily shown because the sum of the strengths of the gather points for a pixel is equal to the average albedo over the pixel and hence ≤ 1 and the visibility term is also ≤ 1.

This metric requires knowing the adaption luminance for the image. L_a is typically computed as the geometric mean of the luminances in the image which can be estimated from a reduced size image (e.g., 32x24 pixels) computed as a preprocess or it can be specified by the user. We use $N = 200$ in our examples.

Perceptual threshold. Once an adaption luminance is known, it can also be used to modify the perceptual error criteria used to terminate cut refinement. Instead of requiring that each cluster error be less than $r\tilde{L}_{pixel}$, we require it be less than:

$$\left\|\tilde{L}_{\mathbb{C}} - L_{\mathbb{C}}\right\| < r\left(\tilde{L}_{pixel} + L_a/10\right) \tag{15}$$

where r is a constant (2% in our results) and $\tilde{L}_{pixel}$ is the estimated total for the current pixel. The factor of $L_a/10$ was estimated from the perceptual threshold vs. intensity plots for fixed adaption level in [Irawan et al. 2005]. This metric more accurately models our ability to perceive error in darker image regions and reduces unnecessary computation in these regions.

4 Applications

Multidimensional lightcuts are applicable to a wide range of rendering problems. In our implementation we demonstrate its effectiveness for temporal anti-aliasing, spatial anti-aliasing, camera depth of field, specular surfaces, and volume rendering.

Spatial anti-aliasing traces multiple eye rays spread over a pixel's area on the image plane. We currently use box filtering, but other filters such as Gaussian are easily supported.

Specular materials such as mirrors and glass are handled by tracing reflection and refraction rays whenever an eye ray hits them. Each specular ray then generates a new gather point.

Depth of field models cameras with non-zero apertures instead of the idealized pinhole cameras usually used in graphics. We implemented a thin-lens camera model where eye rays are distributed over the aperture and focused on a plane in front of the camera. Objects not on the focus plane are blurred, or out-of-focus. A large number of eye rays are required for good quality depth of field.

Temporal anti-aliasing requires handling both visibility and shading changes as objects, light sources, and cameras can all move or change over the frame time. We first discretize the frame interval into T time instants (using a truncated Gaussian distribution in our current system). Then we distribute the eye rays across these time instants with one eye ray per pixel per time instant in our examples. Our system handles multiple times correctly, preventing bleeding of light across different time instants. Good temporal anti-aliasing generally requires using many time instants per frame.

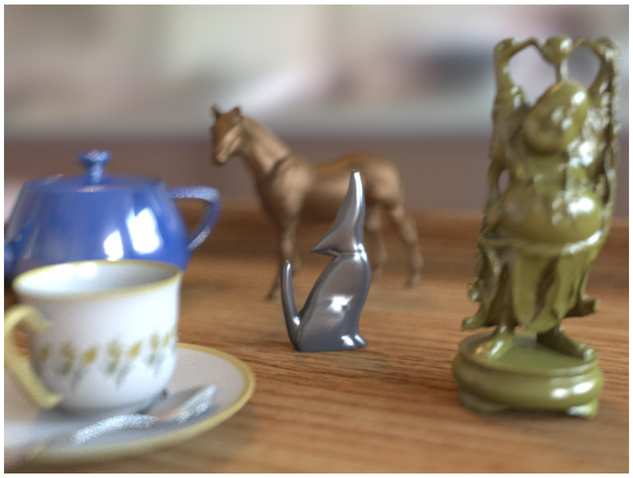
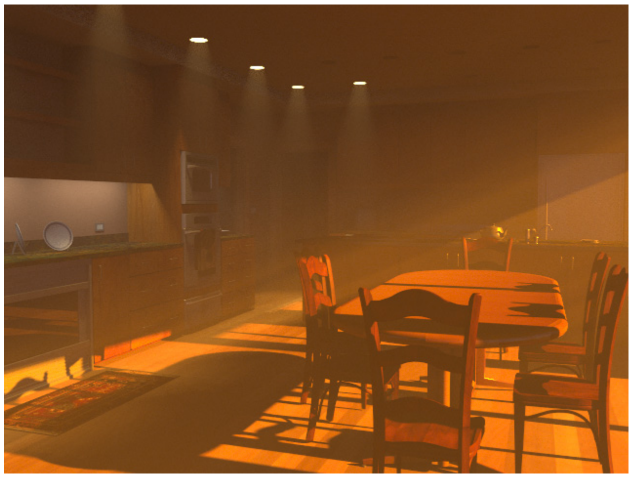


Tableau (Depth of Field)


Kitchen (Participating Media)

Model	Polygons	Light	Per Pixel Averages					Light Build	Image
		Points	Eye Rays	Gather Points	Gather-Light Pairs	**Cut Size**	Total Rays	Preprocess	Time
Tableau	630843	13000	256	180	234 000	**447**	721	1.8s	702s
Roulette	151752	23000	256	306	7 047 430	**174**	581	2.4s	590s
Kitchen	388552	55189	32	100	5 518 900	**936**	969	7.4s	705s
Temple	2124003	94168	256	1282	114 149 280	**821**	1077	9.8s	1740s

Figure 4: Results for 640x480 images of our four scenes. The polygon and point light counts are shown for each scene. We also give per pixel averages for the number of eye rays and gather points. This results in a very large number of gather-light pairs that potentially need to be evaluated per pixel. Cut size is the number of pairs that we actually evaluated to compute each pixel and is far smaller with our scalable algorithm. We also show the total number rays used per pixel which includes eye, specular, and shadow rays. Lastly we give the preprocess time to build the light tree and the time to generate each image after the preprocessing. See Figure 1 for roulette and temple images.

Participating media causes absorption and scattering along the length of the rays as well as at surfaces. We modified our particle and eye rays to include volume attenuation and generate volume scattering events based on the density of the media. The light tree can thus include volume indirect lights to account for out-scatter, while we generate volume gather points along the eye rays to account for in-scatter. The spacing of the volume scattering events is proportional to the density of the media such that all the volume gather points have equal strength.

Our implementation currently supports uniform media and a uniform scattering phase function, though extensions to other densities and phase functions are possible. We include bounds on the minimum attenuation when computing cluster error bounds based on the minimum distance between the gather and light point's bounding boxes. If the volume density is low or its extent is small then the trivial upper bound of one works reasonably well.

5 Results

In this section we present results from our initial multidimensional lightcuts implementation for four scenes with different effects. All times are for a dual processor workstation with 3.8 GHz Xeon processors. All the code is in Java, and relatively unoptimized, except for the ray-geometry intersection testing which is written in C++. Results are for 640x480 images with a 2% error threshold and a maximum cut size of 2000. Statistics for each scene are given in Figure 4 and animations are shown in the accompanying video.

Depth of field. The tableau image demonstrates depth of field using a thin lens camera focused on the coyote figure and an f-stop of 16. We used 256 eye rays per pixel distributed over the camera's aperture for depth of field and spatial anti-aliasing. The scene contains several complex objects with different glossy materials. It is lit by an HDR environment map (the Kitchen map from [Debevec 2002]) and indirect illumination simulated using 3000 directional and 10000 indirect point lights respectively.

Tableau uses an average of 180 gather points per pixel (some eye rays did not hit any geometry) generating 234,000 gather-light pairs which potentially contribute to a pixel. Our technique evaluated an average of only 447 pairs per pixel, allowing us to rapidly compute a high quality anti-aliased image with depth of field.

Motion blur. The roulette scene demonstrates temporal anti-aliasing. The inner part of the wheel is rotating while the outer part is stationary. Images without motion and with slower motion are shown in Figure 5. The wheel contains complex geometry including 38 numbered bins with mirror-like metallic dividers. The scene is lit by an HDR environment map (the Uffizi map) and indirect illumination using 3000 directional and 20000 indirect lights. We used 256 time instants and 256 eye rays per pixel for this image to handle motion blur in reflections as well as in primary visibility and shading. Reflections of stationary objects like the HDR map correctly remain sharp while reflections of moving objects are blurred. You can also see a reflection of the central handle reflected in the tops of the spinning metallic dividers. These effects can be seen more clearly in the accompanying video.

Participating media. The kitchen scene demonstrates volume effects from participating media, including occlusion and multiple scattering. The kitchen is filled with light smoke or haze and lit by a sun/sky model, twenty interior lights, and indirect illumination using 2129, 2560, and 50000 point lights respectively. We used 32 rays per pixel for spatial anti-aliasing and an average of 68 volume gather points per pixel to capture inscatter along eye rays. This creates over 5 million potential gather-light pairs to be checked, but an average of only 936 pairs per pixel are evaluated to capture complex volume effects like the light shafts from the sun.

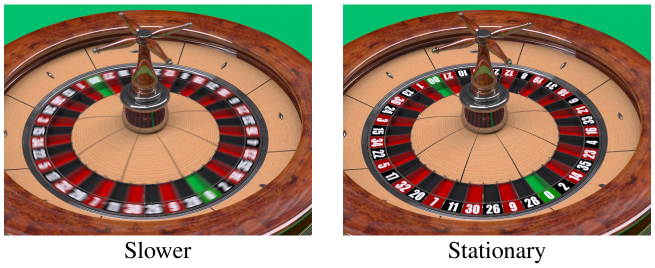


Figure 5: Roulette wheel with slower motion and stationary.

Combined effects. The temple scene is our most complex scene, with over 2 million polygons. It includes both participating media and temporal anti-aliasing with both moving camera and moving light sources. The scene simulates a foggy night and is lit by a sky model, 5 interior lights, 150 small moving volume lights meant to be luminous fireflies or faeries, and indirect illumination. Volume lights are simulated as moving omni lights with a modified falloff function. We used 256 time instants and 256 eye rays per pixel for temporal and spatial anti-aliasing. We also used an average of 1026 volume gather points per pixel to capture the volume in-scatter including the inscatter from the small moving lights. This creates over a hundred million potential gather light pairs per pixel, but we only needed to evaluate an average of 821 pairs per pixel.

Although the temple scene is much more complex, with many more polygons, light points, and gather points, its cost only increases by a factor of two to three over the other scenes due to our scalable algorithm. Figure 6 shows the pixel cut sizes for the roulette and temple scenes as false color images.

Scalability. To demonstrate how our algorithm scales with increasing numbers of gather points, we plot cut sizes and image times for the roulette scene with varying numbers of eye rays per pixel in Figure 7. Multidimensional lightcuts is compared to two other methods. The first is the original Lightcuts algorithm (without reconstruction cuts), which computes an independent lightcut at each gather point. Unified cut is a simplified version of our algorithm which does not use gather clusters or a gather tree. Instead, it computes a cut for each individual gather point, but considers them as part of a unified cut for refinement and termination purposes.

The cut size results show dramatically better scaling for our algorithm with increasing gather points. Adding more gather points has only a small effect on cut size once we include gather clusters and the gather tree, whereas cut size increases much faster in methods without gather clusters. Image time scaling is also much better for our algorithm though not quite as flat as the cut size curve. However by plotting the time to just trace the eye rays without any shading, we can see that most of the added time is due to the cost of eye rays; actual shading costs increase very slowly. The time increased by only a factor of 11 when going from 1 to 256 eye rays per pixel, and if we exclude the time to trace the eye rays, then the shading

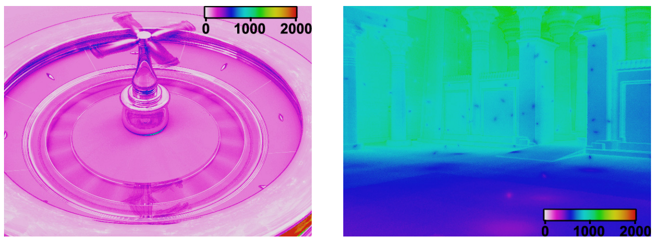


Figure 6: False color images showing the cut sizes per pixel for the roulette and temple scenes from Figures 1 and 4.

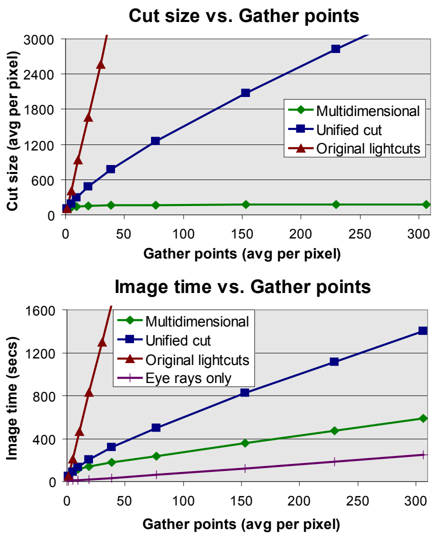


Figure 7: Variation in cut size and image time with the number of gather points for three algorithms. Multidimensional lightcuts greatly reduces the cost of shading multiple gather points per pixel.

cost increased by only a factor of 6.7—much smaller than the 256 times increase in shading points required to capture the effect.

We also show a side-by-side comparison of our result to an image computed using the Metropolis algorithm[Veach and Guibas 1997]. The images are visually very similar. The main differences are that the Metropolis result still has some residual visual noise, even though it took 15 times longer to compute than our result, and it is slightly brighter (about 5% on average) because it includes some indirect paths that cannot be handled by the Instant Radiosity-style indirect approximation used in our system (e.g., caustic paths).

6 Conclusions

Multidimensional lightcuts is a scalable rendering technique for rich visual effects such as motion blur, participating media, depth of field and spatial anti-aliasing in scenes with complex illumination. It unifies handling of these effects by discretizing the rendering integrals into many gather-light pairs and then accurately approximating their total contribution using only a tiny fraction of the pairs.

We use an implicit hierarchy, the product graph, on the gather-light pairs and adaptively partition them into clusters based on conservative cluster error bounds and a perceptual metric. This allows us to accurately estimate the total integral while cheaply approximating the individual clusters, and allows us to scale to very large numbers of gather-light pairs (hundreds of millions per pixel). For example,

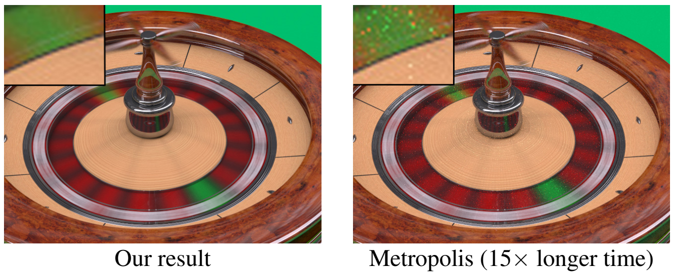


Figure 8: Comparison of our result with a result computed using Metropolis. The images are very similar, but Metropolis takes 15 times longer and still has residual noise [see inset zoom].

we increased the sampling rate in a complex scene by 256x while only increasing the shading cost by less than 7x.

We have also shown that multidimensional lightcuts can be much faster than alternative Monte Carlo methods that are typically used for complex scenes and effects, and computes images that are visually good approximations of the exact solutions.

Future Work. There are many areas of future research worth exploring. Our technique uses conservative error bounds, thus developing good bounds for more types of functions will let us handle a wider range of lights, materials, scattering functions, and volume densities. More sophisticated perceptual error metrics, beyond Weber's law, could further reduce cut sizes. Adaptive or on-the-fly generation of gather and light points could reduce the initial number of points we need to generate and further improve robustness. Gather points could be used to solve more rendering effects, such as using additional gather points to automatically compensate for the bias introduced by indirect clamping. Also complementary techniques could be integrated for the indirect paths not currently handled, such as caustics.

Acknowledgments

We would like to thank Jeremiah Fairbanks (kitchen), Veronica Sundstedt, Patrick Ledda, and the Graphics Group at University of Bristol (temple) for allowing us to use their models. This work was supported by NSF grants ACI-0205438, CCF-0539996, and Intel Corporation.

References

BEKAERT, P., SBERT, M., AND HALTON, J. 2002. Accelerating path tracing by re-using paths. In *EGRW '02*, 125–134.

BURKE, D., GHOSH, A., AND HEIDRICH, W. 2005. Bidirectional importance sampling for direct illumination. In *EGSR '05*, 147–156.

CAMMARANO, M., AND JENSEN, H. 2002. Time dependent photon mapping. In *EGRW '02*, 135–144.

CATMULL, E. 1984. An analytic visible surface algorithm for independent pixel processing. In *SIGGRAPH '84*, 109–115.

CEREZO, E., PEREZ-CAZORLA, F., PUEYO, X., SERON, F., AND SILLION, F. 2005. A survey on participating media rendering techniques. *The Visual Computer 21*, 5, 303–328.

CLARBERG, P., JAROSZ, W., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Wavelet importance sampling: efficiently evaluating products of complex functions. *ACM Transactions on Graphics 24*, 3, 1166–1175.

CLINE, D., TALBOT, J., AND EGBERT, P. 2005. Energy redistribution path tracing. *ACM Transactions on Graphics 24*, 3, 1186–1195.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. In *SIGGRAPH '84*, 137–145.

COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The Reyes image rendering architecture. In *SIGGRAPH '87*, 95–102.

DAMEZ, C., DMITRIEV, K., AND MYSZKOWSKI, K. 2003. State of the art in global illumination for interactive applications and high-quality animations antialiasing. *Computer Graphics Forum 22*, 1, 55–77.

DEBEVEC, P. 2002. Image-based lighting. *IEEE Computer Graphics & Applications 22*, 2 (March-April), 26–34.

HAVRAN, V., DAMEZ, C., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2003. An efficient spatio-temporal architecture for animation rendering. In *EGSR '03*, 106–117.

IRAWAN, P., FERWERDA, J. A., AND MARSCHNER, S. R. 2005. Perceptually based tone mapping of high dynamic range image streams. In *EGSR'05*, 231–242.

JENSEN, H. W., AND CHRISTENSEN, P. H. 1998. Efficient simulation of light transport in scenes with participating media using photon maps. In *SIGGRAPH '98*, 311–320.

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

KOREIN, J., AND BADLER, N. 1983. Temporal anti-aliasing in computer generated animation. In *SIGGRAPH '83*, 377–388.

LAFORTUNE, E. P., AND WILLEMS, Y. D. 1993. Bi-directional path tracing. In *Compugraphics '93*, 145–153.

LAWRENCE, J., RUSINKIEWICZ, S., AND RAMAMOORTHI, R. 2004. Efficient BRDF importance sampling using a factored representation. *ACM Trans. Graph. 23*, 3, 496–505.

LAWRENCE, J., RUSINKIEWICZ, S., AND RAMAMOORTHI, R. 2005. Adaptive numerical cumulative distribution functions for efficient importance sampling. In *EGSR '05*, 11–20.

MAX, N. L., AND LERNER, D. M. 1985. A two-and-a-half-d motion-blur algorithm. In *SIGGRAPH '85*, 85–93.

MITCHELL, D. P. 1991. Spectrally optimal sampling for distributed ray tracing. In *SIGGRAPH '91*, 157–164.

MYSZKOWSKI, K., ROKITA, P., AND TAWARA, T. 2000. Perception-based fast rendering and antialiasing of walkthrough sequences. *IEEE Transactions on Visualization and Computer Graphics 6*, 4, 360–379.

MYSZKOWSKI, K., TAWARA, T., AKAMINE, H., AND SEIDEL, H.-P. 2001. Perception-guided global illumination solution for animation rendering. In *SIGGRAPH '01*, 221–230.

PAULY, M., KOLLIG, T., AND KELLER, A. 2000. Metropolis light transport for participating media. In *EGRW '02*, 11–22.

PREMOZE, S., ASHIKHMIN, M., RAMAMOORTHI, R., AND NAYAR, S. 2004. Practical rendering of multiple scattering effects in participating media. In *EGSR '04*, 52–63.

SUN, B., RAMAMOORTHI, R., NARASIMHAN, S. G., AND NAYAR, S. K. 2005. A practical analytic single scattering model for real time rendering. *ACM Transactions on Graphics 24*, 3, 1040–1049.

SUNG, K., PEARCE, A., AND WANG, C. 2002. Spatial-temporal antialiasing. *IEEE Transactions on Visualization and Computer Graphics 8*, 2, 144–153.

TALBOT, J., CLINE, D., AND EGBERT, P. 2005. Importance resampling for global illumination. In *EGSR '05*, 139–146.

TAWARA, T., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2004. Exploiting temporal coherence in final gathering for dynamic scenes. In *Proceedings of the Computer Graphics International*, 110–119.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *SIGGRAPH '97*, 65–76.

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: A scalable approach to illumination. *ACM Transactions on Graphics 24*, 3 (Aug.), 1098–1107.

WLOKA, M. M., AND ZELEZNIK, R. C. 1996. Interactive real-time motion blur. *The Visual Computer 12*, 6, 283–295.

A Refinement heuristic

When refining a node in the cut, a refinement heuristic is used to choose between gather and light tree refinement; this heuristic consists of several different components. If the gather and light bounding boxes overlap, then the node with the larger bounding box is refined. Otherwise we try to guess which refinement type will shrink the material M^u and geometric G^u error bounds the fastest by estimating the minimum possible M^u and G^u through gather or light refinement alone. This is computed during the process of computing the M and G bounds for the cluster.

The material term consists of a diffuse and glossy term. We estimate the largest possible reduction in the diffuse material term by estimating the decrease in angular size of the clusters: for gather refinement we use $\cos(\theta_{cone})\, d_g/d_{min}$ and for light refinement we use d_l/d_{min}, where θ_{cone} is the half angle of the normal bounding cone for the gather node, d_g and d_l are the diagonals of the gather and light bounding boxes, and d_{min} is the minimum distance between the bounding boxes.

Similar rationales are used in bounding the gloss term using the cube map, and the cosine in the geometric term. Putting all the estimates together gives a crude but cheap estimate of how much either gather or light refinement could reduce M^u and G^u and we choose the one with the greatest estimated reduction in error bounds. However, long sequences of only one refinement type are usually undesirable, so we also add a small term to favor the other refinement type after repeated sequences of only one refinement type.