Precomputed & Hybrid Variants of Lightcuts

Tim Condon Bruce Walter Kavita Bala

Cornell University

Abstract

Our extensions to multidimensional lightcuts improve rendering performance using precomputation and hierarchial approaches. The first approach precomputes static illumination of static geometry, but defers the remaining light path computations until render time. Once the dynamic content of a scene is known, the remaining light paths are resolved and a final image is computed. The impact of precomputation impacts performance in a view and scene-dependent manner. We also explore a hierarchial approach that computes coarse regional approximations for lighting, then progressively refines these approximations to an error-bounded threshold as necessary. Performance improvements for this hybrid method depend on the realization of fast visibility-testing, either through the use of hardware shadow mapping or packetized ray tracers.

1 Introduction

Lightcuts[Walter et al. 2005] and Multidimensional Lightcuts[Walter et al. 2006] render highly realistic scenes under complex lighting with guaranteed error bounds at speeds orders of magnitude faster than other algorithms of comparable quality. However, even when running in parallel on several machines, the lightcuts algorithms are far from interactive. With the goal of deploying Multidimensional Lighcuts (MDLC) as the final renderer in an interactive scene design and preview tool, we discuss two modifications intended to accelerate MDLC and allow it to render complex scenes in a few seconds.

The first method, which we call precomputed lightcuts, leverages precomputation to account for all fixed geometry, materials, and lighting for a fixed viewpoint of a semi-dynamic scene. A semi-dynamic scene consists of mostly-fixed geometry, as well as "sockets" - axis-aligned boxes whose content can be dynamic. These sockets are treated as black boxes during precomputation. Prior to rendering the scene, the contents of these boxes can be filled with arbitrary geometry and materials. At render time, the precomputed data provides illumination information where possible, and otherwise illumination is computed on the fly. The performance improvement from this precomputation is highly scene- and viewpoint-dependent, but for representative scenes, a three-fold acceleration is typical for full scene rendering. Further incremental changes - such as material updates - could be faster.

The second variant, which we call hybrid lightcuts, takes advantage of coherence by calculating coarse lighting for large regions of the image, then uses this regional information as a starting point to compute accurate, per-pixel illumination. The primary cost of this algorithm is visibility testing between the relatively small set of regional lights and each gather point in the final image. While our current implmentation offers little or no speedup, the results indicate a coherent occlusion testing method such as shadow maps or packet tracing would realize significant speedup without the need for precomputation.

2 Related Work

Offline Rendering: The most general solution to realistic image synthesis is Monte Carlo path tracing to solve the rendering equation [Kajiya 1986]. However, path tracing is slow, and generally very noisy. Enhancements that improve both performance and reduce noise include bi-directional path tracing [Lafortune and Willems 1993] and Metropolis [Veach and Guibas 1997], as well as a variety of importance sampling techniques to improve convergence, including [Lawrence et al. 2004][Lawrence et al. 2005][Clarberg et al. 2005][Burke et al. 2005][Talbot et al. 2005].

Photon mapping [Jensen 1996] can calculate indirect illumination both faster and with less noise than path tracing. It is general enough to handle caustics, participating media [Jensen and Christensen 1998], motion blur [Cammarano and Jensen 2002] and many other important effects. Photon mapping is often accelerated with irradiance caching [Ward et al. 1988].

Interactive Rendering: Instant Radiosity [Keller 1997] shoots particles from light sources and converts hits into indirect lights, which can then be rendered on graphics hardware as virtual point lights with cosine emission. Precomputed radiance transfer [Sloan et al. 2002][Ng et al. 2004] allows interactive previewing of static scenes (generally from a fixed viewpoint), but demands long precomputation times. Radiosity-based techniques such as Hierarchical Radiosity [Hanrahan et al. 1991] can compute b̈akedïllumination in the form of light maps or per-vertex data, albeit at a cost of significant precomputation. More recent advances, including [Bunnell 2005] and [Dachsbacher et al. 2007], calculate radiosity at interactive rates by replacing an explicit visibility calculation with an iterative solution using negative light.

Many Lights Rendering: Lightcuts [Walter et al. 2005] converts all illumination into point light sources, then clusters these lights into a tree of lights. Each pixel is shaded by computing a cut across the tree. Starting from a cut consisting of the root of the light tree, the algorithm proceeds by recursively removing the maximum-error node from the working cut and adding its two children - this is called refinement. The algorithm halts when the cut's maximum error node is below some threshold.

Multidimensional lightcuts [Walter et al. 2006] further extends this idea with the notion of a gather-tree; a per-pixel tree of gather nodes. Cuts are now computed across light-gather pairs, and at any step of refinement, either a light or a gather cluster is refined. This allows for fast antialiasing, motion blur, and depth of field.

Matrix row-column sampling [Hasan et al. 2007] formulates the many-lights problem as a low rank matrix of gather-light interactions, then computes a reduced matrix consisting of representative lights chosen from the original matrix. These representative lights are used to shade the entire scene. The algorithm runs in a few seconds on the GPU. Tensor clustering [Hasan et al. 2008] can improve coherence for animated scenes with moving cameras and deformable objects.

3 Precomputed Lightcuts

Precomputed lightcuts augments multidimensional lightcuts with precomputation to accelerate rendering time for semi-dynamic scenes. Most geometry and material are assumed to be static, but we allow regions of the scene called sockets to be tagged; no assumption is made about the content of these regions, so their content can be fully dynamic. We represent a socket as an axis aligned bounding box.

The sockets provide information about which parts of the scene can change - and correspondingly, which regions are fixed. Therefore, we can precompute the lighting interactions between fixed scene elements, and defer computations that involve dynamic elements. Since lightcuts and its variants are based on virtual point lights (VPLs), computing illumination primarily involves shooting particles to create VPLs, then computing visibility between these VPLs and each gather point. In the case of particle shooting, VPLs can be fully precomputed if a particle's path involves only static elements, but only be partially computed if it intersects a socket. Likewise, visibility between static elements can be fully computed if no sockets lie between them, but visibility is indeterminate if a socket overlaps the path. Finally, no gather points can be computed for the sockets themselves until their content is instanced at render time. Performance is highly dependent on how much of the scene is static; if the probability of a random ray intersecting a socket is low, then most of its interactions can be precomputed and performance savings are impressive. Additionally, since gather points cannot be generated until a socket's content is instanced, performance also depends on camera viewpoint and the proportion of screen space occupied by sockets.

Therefore, in order for precomputation to improve performance, two conditions must be fulfilled:

1. The scene volume occupied by the dynamic regions is small relative to the rest of the scene

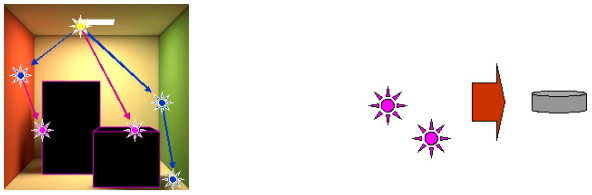
2. The screen-space projection of the dynamic regions is small relative to the screen space projection of the rest of the (static) scene

Our algorithm fully precomputes particles and paths which only interact with static elements, and partially computes particles and paths which interact with dynamic sockets. At render time, these dynamic paths are fully resolved, and a final image is computed. This proceeds in three stages of precomputation and four stages of rendering.

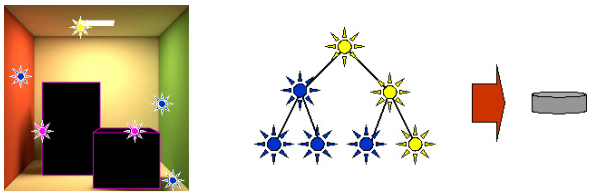
Precomputation begins with a particle trace; intersections with static geometry are converted into VPLs, and particles which intersect sockets are cached to disk (figure 1(a)). Next, we construct a light tree over the static VPLs and direct lighting (figure 1(b)). Finally, for each pixel that projects on to static geometry, we refine a light cut over the static light tree (figure 1(c)). Where possible, we resolve the visibility between light and gather points.

At render time, we continue propagating particles which intersected sockets during the first stage of precomputation to generate a tree of dynamic VPLs (figure 1(d)). We resolve visibility tests in cuts through the static light tree that could not befully resolved during precomputation (figure 1(e)). Next, we refine the static cuts against the tree of dynamic VPLs in order to account for illumination that passes through or bouces off a socket or its content (figure 1(f)). Finally, we construct a cut for pixels that project onto sockets over both the static and dynamic light trees (figure 1(g)).

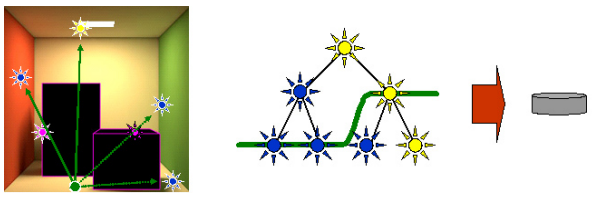
Precomputation



(a) Cache particles that intersect sockets to disk

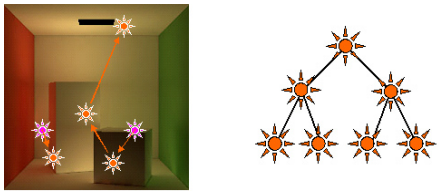


(b) Construct light tree from static lights and save to disk

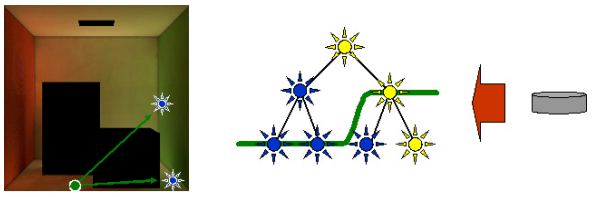


(c) Refine cut for static pixels over static lights and save to disk

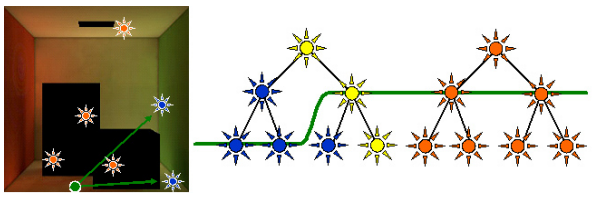
Render-Time



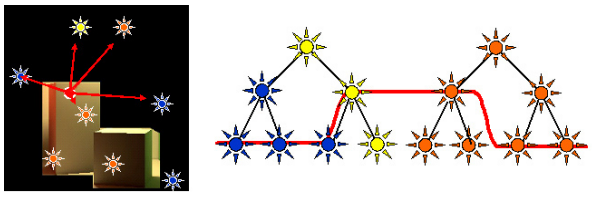
(d) Propagate cached particles through scene



(e) Resolve ambiguous visibility for static cuts



(f) Refine static pixels against dynamic lights to compute final shading



(g) Construct cut for dynamic socket pixels across both static and dynamic light trees

Figure 1: Algorithmic Stages of Precompued MDLC

3.1 Static Caching

Lightcuts is incremental in that given a list of lights for pixel shading, it is easy to add additional lights and reshade the pixel, and it is conservative in that adding more lights will only ever increase the accuracy of the rendered solution. This leads directly to our first improvement - namely, we can precompute cuts for every pixel that projects on to a static surface, then compress these cuts and save them to disk, knowing that additional lights or refinement can only improve accuracy.

Refining a node on a light cut requires testing visibility between the node's representative light and the gather point to which the cut belongs. If the path between the light and the gather point does not intersect a socket, visibility can be resolved statically. However, if this path passes through a socket, the visibility is unknown - the socket may be instanced with geometry that occludes this path, or it may not. In cases where visibility is unknown (ie, a visibility ray intersects a socket), we record this fact and recheck visibility at render time to determine whether to include the light in shading. Once we have resolved a cut as much as precomputation allows, it is compressed to a color value (summed from leaf node contributions), plus a list of nodes each with a 32-bit light index and an additional 32-bits to store a gather index (anti-aliasing dependent), a representative index, and a two-bit visibility flag (visible, not visible, or unknown). Cut size determines the length of this node list. The light tree is also saved to disk (using Java's serialization mechanism). At render time, if a pixel has a cached cut, we reconstruct the cut, resolve unknown visibility, and compute a pixel value. Otherwise, regular MDLC takes over and refines a cut from scratch.

The savings from this stage of precomputation depends highly on the degree to which our first assumption holds. If most of the image consists of dynamic regions, then very few pixels have cached cuts. If the image has no dynamic regions at all, the entire image will be precomputed, and rendering involves just pixel value computation.

3.2 Particle Tracing

During precomputation, we perform a particle trace to create indirect lights. Particles trace through the scene normally until terminated by Russian roulette or intersection with a socket. In the case of socket intersection, we save the particle's state (position, direction, color, and termination probability) to disk. At render time, once the content of the socket is known, we can restart these cached particles and finish their trace through the scene. We then construct a separate light tree over these cached particles.

3.3 Dynamic Refinement

The indirect sources generated by the render-time particle trace can affect any pixel in the scene, so every cut must be further refined to include these additional particles. Fortunately, lightcuts can limit refinement when provided with a base luminance value to threshold minimum pixel intensity; refinement halts when the algorithm detects further refinement will not yield perceptually meaningful changes. Thus, we compute an initial, minimal pixel color based on the precomputed cut (if it exists), or construct the cut and compute a pixel value (if it does not exist). In either case, we use this pixel value as a base radiance value and further refine the precomputed cut to account for the dynamic lighting introduced by the render-time particle trace.

3.4 Results

In order to test the performance of precomputed lightcuts, we rendered two scenes using both multidimensional lightcuts (MDLC) and precomputed multidimensional lightcuts with two sockets (PMDLC). Each scene was rendered on an Intel Xeon 8-core CPU at 2.8GHz per core, with 16 GB RAM and 8x threading. The results are summarized in Table 1.

Scene	**Precomputed MDLC**			**MDLC**
	Regional Cut		*Local Cut*	
	Time	*Memory*		
Cornell Box	13.8 sec.	315 MB	18.3 sec.	8.5 sec.
Kitchen	97.5 sec.	1.6 GB	26.0 sec.	94.6 sec.

Table 1: Each scene was rendered at 512 x 512 with 20,000 indirect lights

Precomputation	125 sec.
Cut reconstruction	7.3 sec.
Socket pixel refinement	14.6 sec.
Dynamic light refinement	8 sec.
Total (not including precomputation)	29.9 sec.

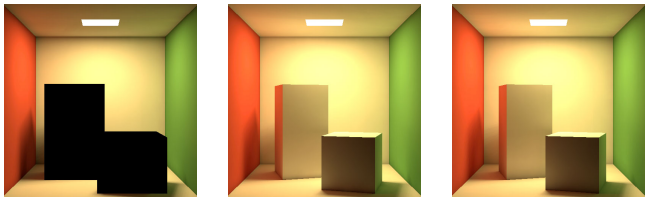
Table 2: Breakdown of rendering time by stage for the kitchen scene at 512 x 512 with 20,000 indirect lights

Performance is worse for the Cornell Box than for the kitchen because its large sockets comprise much of the image screen space. Furthermore, the scene's two sockets are placed directly underneath the primary light, so a majority of the indirect particles intersect the box and their propagation must be defered to render time. Increasing the number of indirect particles substantially reduces rendering performance for precomputed lightcuts with sockets. Table 2 provides a breakdown of rendering time for the kitchen scene by algorithmic phase.

In this instance, approximately half the rendering time is spent constructing cuts for pixels that project onto the scene's sockets. The remaining time is split roughly in half between reconstructing precomputed cuts (primarily resolving visibility) and computing cuts in the dynamic light tree generated by propagating light particles deposited on sockets during precomputation.

4 Hybrid Lightcuts

In its original formulation, matrix row-column sampling[Hasan et al. 2007] starts from a matrix of sample-light pairs which is re-



(a) PMDLC Precomputation (b) PMDLC Final (c) MDLC

Figure 2: Cornell Box rendered with Precomputed MDLC and MDLC (sockets shown in black)

(a) PMDLC Precomputation (b) PMDLC Final (c) MDLC

Figure 3: Kitchen rendered with Precomputed MDLC and MDLC (sockets shown in black)

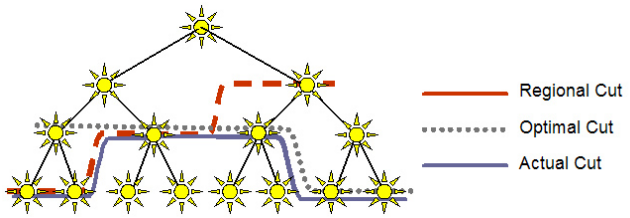


Figure 4: The actual cut size computed by Hybrid MDLC may be larger than the (optimal) cut computed by MDLC. This occurs when the regional cut refined regions of the light tree more than the specific pixel required

duced to a representative, smaller matrix. The lights of this new matrix are used to shade the image. But while matrix row-column sampling is fast, it lacks the bounds on error that lightcuts provides. It is in this sense which hybrid lightcuts is, in fact, a hybrid; it attempts to unify the speed of matrix row-column sampling with the quality guarantees of lightcuts. Reformulating matrix row-column sampling in the terminology of lightcuts allows us to describe the reduced matrix as a single global cut in the light tree used to shade each pixel. This natural correspondence suggests a unified approach which coarsely appoximates lighting using row-column sampling, then refines it to an error-bounded solution using lightcuts. Hybrid lightcuts proceeds in two phases. During the first phase, it divides the image space into several large regions and computes a coarse regional cut for each. In the second phase, pixel values are computed by starting from the regional cut,and refines further as necessary using the framework of lightcuts.

4.1 Regional Cuts

The granularity of subdivision in the image space typically affects both the number of regional cuts and the amount of per-pixel refinement required during the second phase. Generally speaking, a finer granularity of regional cuts require less refinement on a per-pixel basis, but introduces additional lights. In practice, the regional cut is computed on a reduced image; for example, assume we want 16 regional cuts (arranged as a 4 x 4 grid) for an image whose dimension are 512 x 512. Then, each region will be 128 x 128. To compute these regional cuts, we simply render a 4 x 4 image and save the 16 corresponding cuts.

While our measurements considered only uniform subdivision of the image space, the image could be paritioned hierarchically, with some or all regions being further subdivided into subregions. In this case, the regional cut would provide a starting point for subregional cut refinement, then proceed recursively until the region was the size of a single pixel.

4.2 Pixel Refinement

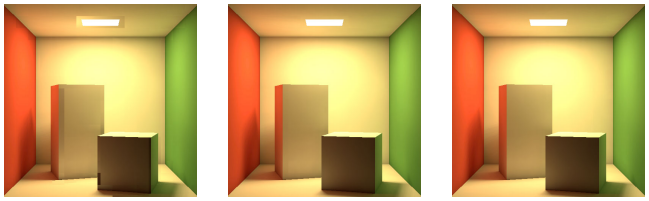
In order to refine a pixel, we start from the cut computed for its regional cut. First, we must resolve visibility between the pixel's gather point and each light in the regional cut. We compute error bounds and contributions for these nodes and insert them into a heap ordered by maximum error. Finally, we resume refinement as in lightcuts; we remove the maximum-error node and replace it with its two children recursively until the total error estimate is below threshold.

A breakdown of performance for this phase is summarized in Table 4. Most of the algorithm's runtime (roughly 80%) is spent testing visibility between pixel gather points and the light nodes in the regional cut. This operation could be accelerated by instead iterating over each unique light across all regional cuts and testing visibility between it and points in the scene. This would be a fast operation given an implementation of coherenct visibility testing - specific examples include shadow mapping or packet ray tracing. Given a suitably efficient implementation of such an operation, this domination portion of the rendering cost could be substantially reduced.

Because each pixel is refined from the coarse regional cuts, the average cut size of a given pixel is expected to be larger than the corresponding pixel in lightcuts. Such a case can occur when a light source is visible to the gather point used to compute the regional cut but occluded from the local pixel. The cut then includes an unnecesary light, and the cut size grows correspondingly. This is demonstrated in 4; the regional cut fully refines the left side of the tree, but the optimal cut computed by regular MDLC does not refine the left-most nodes. Therefore, the actual cut is larger than the optimal cut. While this can only improve the accuracy of the solution, additional light computations increase the cost of computing the cut, and the improved accuracy is below perceptual threshold. The increase in cut size ranges from 20% (for 256 regional cuts) to 40% (for 4 regional cuts).

4.3 Results

In order to postulate accurate performance improvements based on a fast coherent occlusion testing algorithm, we rendered two scenes using multidimensional lightcuts (MDLC) and hybrid MDLC. Each scene was rendered on an Intel Xeon 8-core CPU at 2.8GHz per core, with 16 GB RAM and 8x threading. The results are summarized in Table 3.



(a) HMDLC Regional Cuts (b) HMDLC Final (c) MDLC

Figure 5: Cornell Box rendered with Hybrid MDLC and MDLC

The dominant cost (about 80% of the total render time) is spent reconstructing a regional cut for each pixel. This primarily consists of testing visibility between the light and the pixel's gather point. Accelerating this operation with coherent visibility testing has the potential to dramatically improve rendering performance. Furthermore, visibility testing is also the primary cost of refinemt.

Scene	Hybrid MDLC				MDLC
	Regional Cut	*Reconstruction*	*Refinement*	***Total***	
Cornell Box	0.5 sec	6 sec.	2.2 sec	8.7 sec.	8.5 sec.
Kitchen	0.3 sec	73.4 sec.	23.5 sce.	97.2 sec	94.6 sec

Table 3: Each scene was rendered at 512 x 512 with 20,000 indirect lights

Subdivision	Render Time			Unique Lights	Avg. Cut Size	
	Precomputation	*Render*	***Total***		*Regional*	*Local*
16 x 16	0.4	96.5	96.9	7559	767.7	926.6
8 x 8	0.3	104.0	104.3	5189	716.5	952.2
4 x 4	0.2	108.9	109.1	4171	888.9	1066.5
2 x 2	0.1	98.8	98.9	3292	823.0	1082.0
1 x 1	0.05	124.1	124.1	1048	1048.0	1309.0

Table 4: Kitchen scene rendered at 512 x 512 with 20,000 indirect lights. MDLC renders this scene in 94.6 sec.

(a) HMDLC Regional Cuts (b) HMDLC Final (c) MDLC

Figure 6: Kitchen rendered with Hybrid MDLC and MDLC

While refinement visibility testing is not coherent, it may still be optimized ray-tracing code. Finally, we consider several different granularities of image subdivision; these are recorded in Table 4.

As we might expect, cut size increases with coarser regional refinement, but total unique lights used across all regional cuts decreases. The number of unique lights is particularly important, as it determines the number of lights whose visibility must be tested during pixel refinement.

5 Future Work

In the implementation of precomputed lightcuts, we currently store an entire cut for each pixel, but in some cases we could reduce this to just a pixel color. While this would improve rendering performance marginally, computing a pixel value from a cut is not a dominant cost, and would significantly reduce the storage needed to retain the precomputed results.

In cases where geometry is fixed but material can change, more agressive precomputation may be possible. Because visibility is fixed, we could construct a single conservative cut that would suffice for both a highly diffuse and a highly specular material. Though such a conservative cut would be more expensive to calculate than a minimal cut, the performance advantage of avoiding cut refinement should offset any such overhead.

Implementing a coherent occlusion testing algorithm for hybrid lightcuts would provide tangible performance improvement numbers, and should be an immediate order of business. Determining optimal regional subdivision, or subdividing hierarchically, could tune the performance of this algorithm.

The code upon which these results were based was written in Java, and profiled to address bottlenecks, but more careful tuning could likely reap significant performance gains.

6 Conclusion

We have presented two variants of lightcuts and MDLC which improve rendering performance without sacrificing quality. The first achieves its speedup by precomputing illumination for static scene elements completely, and resolves the remainder of the illumination once dynamic elements have been instanced. The second approach is a hybrid between lightcuts and matrix row-column sampling. Given fast shadow mapping hardware or packetized ray tracing, it promises dramatic speedup without a need for precomputation.

Acknowledgments

We would like to thank the team members of Autodesk Labs Project Showroom for fruitful discussions related to problem specification, usage patterns, and system requirements. The kitchen model was originally developed by BHI Media for Autodesk, Inc. as part of Autodesk Labs Project Showroom technology preview, and is used with permission from Autodesk, Inc.

References

BUNNELL, M. 2005. Dynamic ambient occlusoin and indirect lighting. In *GPU Gems 2*, M. Pharr, Ed.

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

CAMMARANO, M., AND JENSEN, H. W. 2002. Time dependent photon mapping. In *13th Eurographics Workshop on Rendering*.

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.

DACHSBACHER, C., STAMMINGER, M., DRETTAKIS, G., AND DURAND, F. 2007. Implicit visibility and antiradiance for interactive global illumination. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings) 26*, 3 (August).

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

HASAN, M., PELLACINI, F., AND BALA, K. 2007. Matrix row-column sampling for the many-lights problem. In *To appear in Proceedings of SIGGRAPH 2007*, Computer Graphics Proceedings, Annual Conference Series.

HASAN, M., VELAZQUEZ-ARMENDARIZ, E., PELLACINI, F., AND BALA, K. 2008. Tensor clustering for rendering many-light animations. In *Eurographics Symposium on Rendering*.

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.

JENSEN, H. W. 1996. Global illumination using photon maps. In *Proceedings of the Eurographics workshop on Rendering techniques '96*, 21–30.

KAJIYA, J. T. 1986. The Rendering Equation. *SIGGRAPH '86 20*, 4 (August), 143–150.

KELLER, A. 1997. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 49–56.

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.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2004. Triple product wavelet integrals for all-frequency relighting. *ACM Transactions on Graphics 23*, 3 (Aug.), 477–487.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of SIGGRAPH 2002*, 527–536.

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

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Computer Graphics Proceedings, Annual Conference Series, 1997 (SIGGRAPH 1997)*.

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, 1098–1107.

WALTER, B., ARBREE, A., BALA, K., AND GREENBERG, D. P. 2006. Multidimensional lightcuts. In *To appear in Proceedings of SIGGRAPH 2006*, Computer Graphics Proceedings, Annual Conference Series.

WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. 1988. A Ray Tracing Solution for Diffuse Interreflection. vol. 22, 85–92.