Data-Parallel Hierarchical Link Creation for Radiosity

Quirin Meyer[1], Christian Eisenacher[1], Marc Stamminger[1], and Carsten Dachsbacher[2]

[1]University of Erlangen-Nuremberg
[2]University of Stuttgart

Abstract

The efficient simulation of mutual light exchange for radiosity-like methods has been demonstrated on GPUs. However, those approaches require a suitable set of links and hierarchical data structures, prepared in an expensive preprocessing step. We present a fast, data-parallel method to create links and a compact tree of patches. We demonstrate our approach for Antiradiance and Implicit Visibility. Our algorithm is able to create up to 50 M links per second on an Nvidia GTX 260, allowing fully dynamic scenes at interactive frame rates.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.1]: Parallel processing—Computer Graphics [I.3.6]: Graphics data structures and data types—Computer Graphics [I.3.7]: Radiosity—

1. Introduction

Interactive global illumination has been an elusive goal for a long time – and for fully dynamic scenes it still is. One method of choice is radiosity, which recently gained more attention again. Radiosity-like methods compute global illumination by simulating the mutual light exchange between the surface patches of a given scene. This works particularly well for static scenes and involves two main steps:

- Discretize the scene into a hierarchy of *patches* and establish *links* between them.
- Simulate *light transport* using these links.

While efficient parallel implementations have been demonstrated for the second step, link and patch creation is more difficult: It is a recursive process, requiring expensive ray casting operations to determine the mutual visibility between patches. Thus, in previous work, the link and patch creation was bound to be a costly preprocessing step on the CPU. This prevented the use of radiosity methods for fully dynamic, interactive scenes.

Recent approaches using Antiradiance [DSDD07] or Implicit Visibility [DKTS07] do not need visibility computations during hierarchical link and patch creation. While both papers only describe sequential algorithms, we propose a non-recursive and efficient data-parallel algorithm to create links and a patch hierarchy for these methods, running entirely on the GPU.

Our method is fast enough to create links and a compact patch hierarchy from scratch for every frame. Including the simulation of light transport, we can render dynamic scenes with indirect light at interactive frame rates (see Figure 1).

2. Related Work

There is a vast body of research on global illumination and excellent text books cover the area, e.g. [DBB06]. Two main approaches have been widely researched in the last decades: ray tracing based methods and radiosity.

There have been many attempts to use GPUs to speed up ray tracing. However, the hierarchical traversal of acceleration structures requires special adaptations, such as a stackless traversal or modified acceleration structures, e.g. [HSHH07, PGSS07], so that GPU ray tracers achieve a performance comparable to current CPU implementations. Generating those structures directly on the GPU often relies on space filling curves [AGCA08], and has recently been demonstrated for real-time kd-tree construction [ZHWG08]. Several methods are based on ray tracing, and cache information about the lighting situation. The most well-known representatives are irradiance caching [WH92], photon mapping [Jen01, HOJ08], and instant radiosity [Kel97]. In particular the latter class of methods gained attraction as it creates sets of virtual point lights (VPLs), and thus maps easily to graphics hardware, e.g. [DS05, DS06, LSK*07, RGK*08].
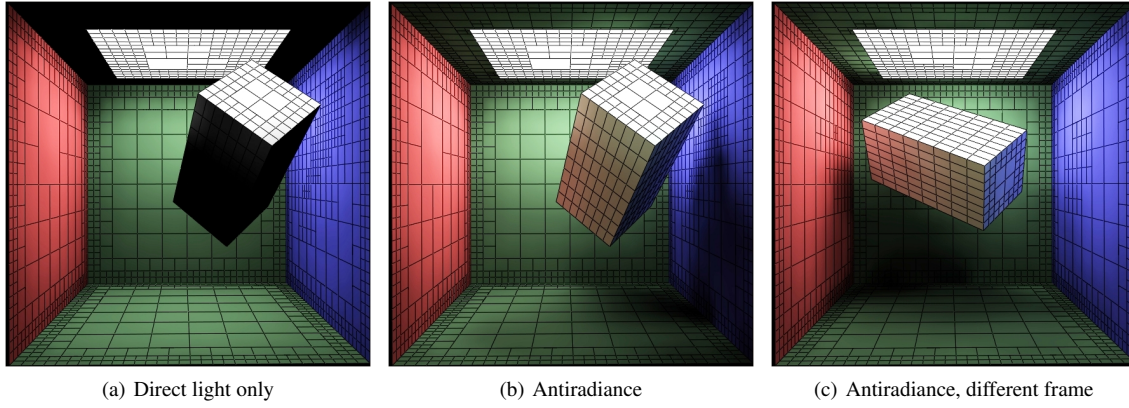
| (a) Direct light only | (b) Antiradiance | (c) Antiradiance, different frame |

Figure 1: *We create hierarchical links and a compact patch tree at a rate of 50 M links per second on a Nvidia GTX 260. For moderately complex scenes (280 k links and 4k patches in 1(a)) we can create links from scratch for each frame. Including the simulation of mutual light exchange our method allows fully dynamic scenes with indirect light at 30-40 fps 1(b), 1(c).*

The light cuts method [WFA∗05] clusters light samples, e.g. VPLs, into a hierarchy to speed up rendering.

There have also been numerous attempts to port radiosity computations to the GPU. The main problem is the costly evaluation of a large number of mutual visibilities between surface patches. Earlier GPU radiosity solvers [CHL04, BSKS05] use rasterization inspired by the hemicube method. More recent approaches, such as [STK08], use ray tracing on the GPU to compute form factors combined with an asynchronous update mechanism for interactive rendering.

Recently, Dachsbacher et al. [DSDD07] introduced a reformulation of the rendering equation that replaces explicit visibility by a recursive computation with negative light or "Antiradiance". Dong et al. [DKTS07] use a directional discretization and store only one link to the closest patch per direction. Thus the visibility is constructed implicitly with the link hierarchy. While both approaches avoid direct visibility computation, they rely on a hierarchical link structure, which is generated sequentially on the CPU.

3. Data-Parallel, Hierarchical Links

Radiosity methods discretize the scene into a hierarchy of patches, generally using a quadtree, and connect patches that exchange light by links. Starting with a pair of root patches, an *oracle* decides whether sender and receiver see each other and are sufficiently small to be connected. If not, sender and/or receiver are subdivided and all combinations of child pairs are considered recursively.

After patch hierarchy and links are created, light exchange is simulated by iterating over all links and transporting light from the sender to the receiver side, simulating one bounce of light. This is repeated until the solution converges.

Antiradiance and Implicit Visibility discretize the (hemi-) sphere of directions into n *bins*, typically $n = 64$ to $n = 512$. This allows non-diffuse materials at the expense of n bins per patch. Each link transports light from one bin of the sender to one bin of the receiver (see inset). These interacting bins have to be determined during link creation (*bin search*).
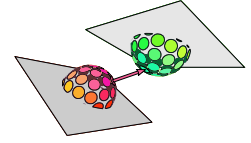
While there is a recursive parent-child dependency between patches, the whole process can be reformulated as a data-parallel breadth-first algorithm focusing on the links. As outlined in Figure 2, we first refine the links, and then create bins only for the linked patches and all their parents, resulting in a compact patch tree representation.
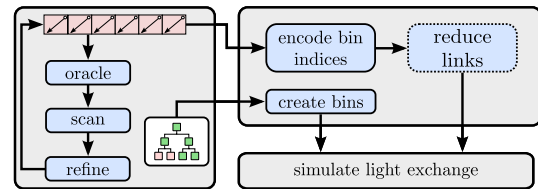

Figure 2: *We adaptively refine links breadth-first and mark the linked patches in a perfect tree. We use this information to create a compact patch hierarchy with directional bins. Optionally we eliminate multiple links per bin.*

3.1. Link Refinement

We create unidirectional links between the original faces, encoded as a pair of quadtree coordinates, and place them into a *link queue* where all links are examined and split in parallel. This is iterated until they are sufficiently refined.
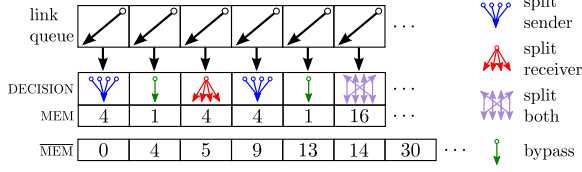
Figure 3: *The oracle examines each link in parallel. It determines how to refine the link, and the storage required. A parallel prefix scan converts the latter into array indices.*

Oracle: The oracle shown in Figure 3 examines each link independently, decides whether and how to refine it (DECISION), and computes storage required for the refined links (MEM). A parallel prefix scan [Ble90, HSO07] converts the latter into indices for the refinement kernel ($\overline{\text{MEM}}$). We cluster directions into bins with uniform solid angle Ω_{bin} as described in [DSDD07], and split patches whose solid angle, as seen from their partner patch, is larger than Ω_{bin}.

Refinement: The refinement kernel in Figure 4 uses the decisions of the oracle to refine all links in parallel. This is done by computing the quadtree coordinates for the child patches and storing them at the prepared indices. Refining links can imply refining one of the linked patches. We mark child patches in the perfect tree (TREE) and create bins for the marked patches when link creation is complete.
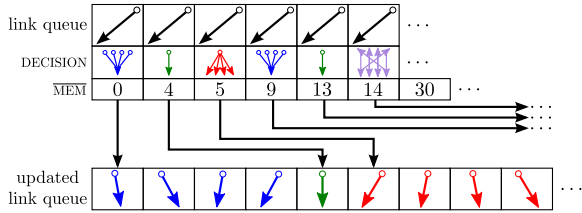

Figure 4: *Links are refined according to the oracles decision and stored at the indices prepared by the parallel prefix scan.*

3.2. Compact Patch Hierarchy and Link Recoding

Compact Patch Hierarchy: As patches with n bins consume considerable space, the hierarchy should only contain the required patches. During link refinement we mark the linked patches in TREE – a perfect tree stored breadth-first (see Figure 5). A simple scan is sufficient to compute a mapping into the compact patch hierarchy ($\overline{\text{TREE}}$). Note that TREE stores one `int` per node, while the compact patch hierarchy stores 4 kB per node for 256 bins. See Table 1.

Topology: After light exchange, we need to make the light distribution consistent throughout the patch hierarchy. First we *push* incident light from parent to child patches down to
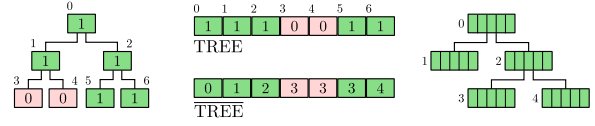

Figure 5: *Patch markers are stored breadth-first. A parallel prefix scan computes indices for the compact hierarchy.*

the leaf nodes. After the *local pass* convolved the light at the leafs with the BRDF, we *pull* exitant light from the children to the parents up to the root node. We simply use TREE to determine whether a given patch has children and look up their indices into the compact hierarchy in $\overline{\text{TREE}}$. Similarly we identify leaf nodes for the local pass.

Link Recoding: During link refinement, links between patches are encoded as a pair of quadtree coordinates. For the simulation of mutual light exchange we recode them using TREE. For each link we store the global indices of the linked bins in the compact patch hierarchy and the form factor (without visibility).

Bin Search: During link recoding we need to determine the actual bins linked. We use a simple hierarchical search. Despite the slightly more complex code and memory access patterns, link recoding time for 256 bins was reduced by a factor of more than three compared to a linear search.

3.3. Link Reduction

The idea of Dong et al. [DKTS07] is to remove all but the shortest link per bin to handle visibility implicitly. This removes the danger of concurrent writes to the same bin and promises increased performance.

To find the shortest link we write the distance between the linked patches into the alpha channel of its receiver bin during link recoding – if it is shorter than the stored value. During link reduction we simply mark longer links and remove them using a standard compaction algorithm [Ble90].

As reported by Dong et al., situations might occur, where child patches have shorter links than their parent in a given direction. Due to the hierarchical nature of the approach, this inconsistency leads to missing shadows, as incident light from the parent patch will be pushed to the children. They propose an additional consistency step combined with pushing the receiver end of all links to the leaf patches.

We skip both and handle consistency during the push phase of the light transport instead: When pushing incident light towards the child nodes, we compare the minimum length stored at each bin. If the parent's link for the given bin is shorter, we propagate the parent's light and update the minimum length of the child. As we need to store four values per bin for alignment reasons, this comes with no additional storage and memory bandwidth cost.

Bins	A_{min}	Compact Tree		Antiradiance					Implicit Visibility				
		N_P	MB	N_L	LC	$IT1$	$IT2$	$IT3$	N_L	LC	$IT1$	$IT2$	$IT3$
64	$1/16\ m^2$	528	0.52	13k	1.2	2.8	3.2	3.5	6 k	1.4	2.7	3.0	3.2
	$1/64\ m^2$	1620	1.58	32k	2.0	3.1	3.8	4.4	15 k	2.3	2.9	3.3	3.7
	$1/256\ m^2$	3980	3.89	70k	2.7	3.8	4.9	6.1	32 k	3.1	3.2	4.1	4.7
	$1/1024\ m^2$	10016	9.78	151k	5.1	5.2	7.7	10.2	63 k	5.9	4.2	5.7	7.2
256	$1/16\ m^2$	556	2.17	66k	1.9	3.4	4.3	5.2	27 k	2.5	3.0	3.4	3.9
	$1/64\ m^2$	2220	8.67	204k	4.2	5.3	7.9	10.5	79 k	5.7	3.9	5.1	6.2
	$1/256\ m^2$	7604	29.7	488k	9.7	10.0	16.7	23.5	187 k	13.0	6.7	10.1	13.5
	$1/1024\ m^2$	21212	82.9	1090k	24.2	20.9	37.0	53.1	411 k	29.8	13.5	22.4	31.1

Table 1: *We create links and compact patch hierarchy for a Cornell box (Figure 6) with side length 1m. We obtain N_P patches with minimal area A_{min} and N_L links. We need LC ms for creation and IT ms for one to three light exchanges on a GTX 260.*

4. Results

To test our approach, we create links and patch hierarchy for a Cornell box with side length 1m, shown in Figure 6. Using a closed scene with simple geometry makes it easy to study the performance of our algorithm and the impact of different parameters on the quality of the radiosity solution: We are able to control the number of patches and links by defining a minimum patch area A_{min} and the flat faces intensify artifacts common to hierarchical radiosity, like contact shadows (compare Figures 6 (c) and (f)) or banding (see Figure 7).

We simulate mutual light exchange with Antiradiance or Implicit Visibility and present results for 64 and 256 bins. We allow the oracle to refine links until the mutual solid angles are less than Ω_{bin}, but skip link refinement if it would produce patches smaller than A_{min}.

Table 1 lists timings for link creation (*LC*) on an Nvidia GTX 260, running CUDA 2.1 on Windows XP. The compact patch hierarchies contain N_P patches that exchange light over N_L links. Example hierarchies are shown in Figure 6. Overall we are able to create up to 50 M links per second.

For our largest example, 256 directional bins and refinement until $A_{min} < 1/1024\ m^2$, we present a detailed breakdown for the time spent at the various stages in Table 2. The timings for the link refinement kernels are accumulated over all iterations needed. The time for "Recode Links" is for computing form factors, looking up the patch index into the compact hierarchy and storing a recoded link in order to clarify the cost for bin and minimum search.

Kernels operating on links process all links independently and do not need shared memory, hence we assign one thread for each link. Kernels creating topology information, allocate one thread for each node of the perfect quadtree. Different block sizes have negligible impact on performance.

For ease of implementation we store one `float4` (4×4 bytes) per bin and one `float4` per quadtree coordinate. For our largest example we need about 83 MB for 21 k patches and 32 MB for 1088 k links.

	Stage	AR	IV
Link Refinement	Oracle	5.5	5.5
	Scan	1.2	1.2
	Refine	8.0	8.0
	Mark Used Patches	2.9	2.9
Bin Encoding	Compute Topology	0.3	0.3
	Recode Links	5.1	5.1
	Bin Search	1.2	1.2
	Write Minimum	-	3.6
Reduce Links	Mark Longer Links	-	0.8
	Compact Links	-	1.2
Total		24.2	29.8

Table 2: *Detailed timings of the link creation in ms (1088 k links Antiradiance (AR), 411 k links Implicit Visibility (IV)).*

The actual light transport is implemented in CUDA. For Antiradiance we use the global `atomicAdd()` to deal with concurrent writes to the same bin. Currently those are limited to integer arithmetic but the available dynamic range was sufficient for our test scene. To search the shortest links for Implicit Visibility we use the global `atomicMin(__float_as_int(distance))`.

5. Discussion

We have presented a complete system for global illumination with radiosity on the GPU: Link and hierarchy creation, light transport and rendering. It allows dynamic scenes with indirect lighting at interactive frame rates.

5.1. Performance

Link Creation: A direct comparison of our results with the original papers is difficult. The link creation for Antiradiance [DSDD07] was designed as offline preprocess involving clustering. Dong et al. [DKTS07] presented only open scenes with environment lighting and did not elaborate on the number of links they created.
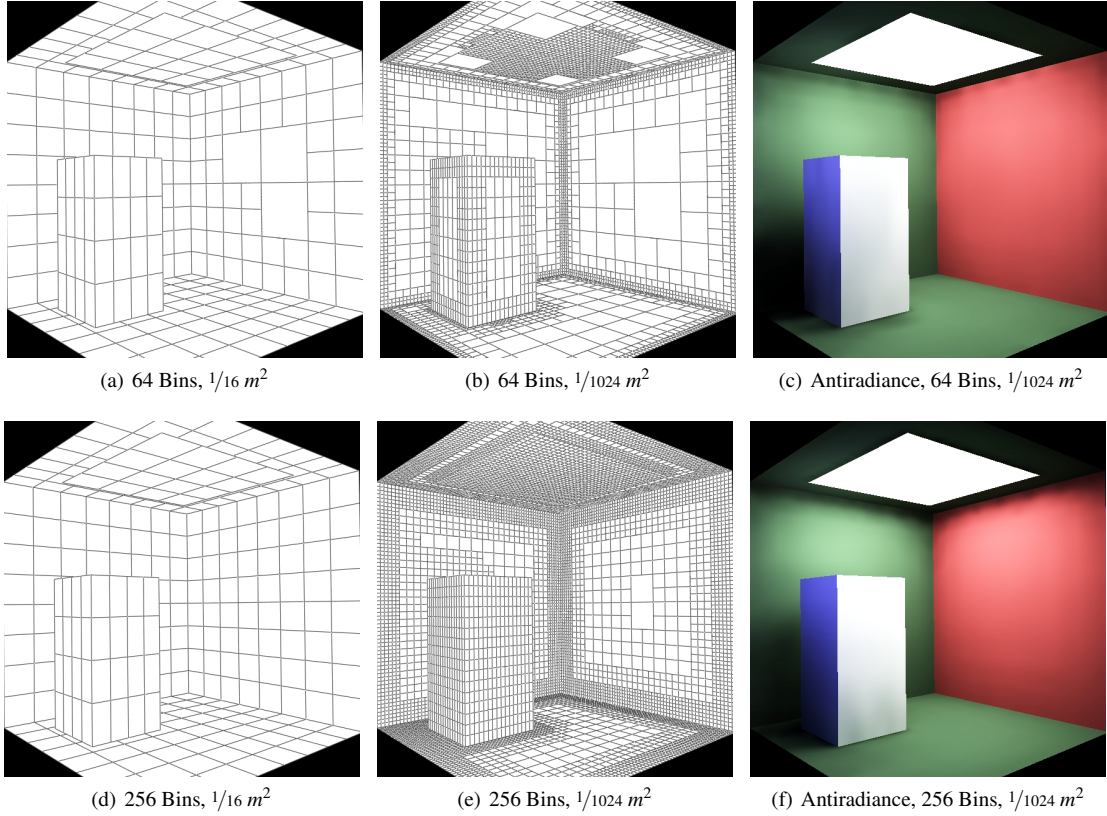
(a) 64 Bins, $^1/_{16}\ m^2$

(b) 64 Bins, $^1/_{1024}\ m^2$

(c) Antiradiance, 64 Bins, $^1/_{1024}\ m^2$

(d) 256 Bins, $^1/_{16}\ m^2$

(e) 256 Bins, $^1/_{1024}\ m^2$

(f) Antiradiance, 256 Bins, $^1/_{1024}\ m^2$

Figure 6: *We test our algorithm for link creation with various parameters on the shown Cornell box. The illumination for the image on the right is simulated with three bounces of Antiradiance and rendered with splatting.*

However, we believe our approach to hierarchy creation on the GPU is very competitive. *Transferring* 1 M links plus topology information from the CPU to the GPU takes 13.2 ms on our system, compared to *creating* the same information on the GPU in 24.2 ms.

Implicit Visibility: At moderate additional cost we can avoid concurrent writes to the same bin. However this accounts only for a fraction of the reduced cost, about 2 ms for 1 M links. Most savings result from the reduced number of links. In combination with the fact that Implicit Visibility needs one bounce less (i.e. 2 bounces for indirect shadows) it is about twice the speed of Antiradiance.

CUDA Atomics: Global atomics are surprisingly fast with current drivers and only about 15% slower then non-atomic versions in our case. Surprisingly our CUDA implementation of the global pass is almost 5× faster than using the hardware blend stage via OpenGL on the same card. This is not an entirely fair comparison, as the blend stage operates with floats and can perform link filtering [DSDD07] at little additional cost, but using global atomics seems to be an interesting option, as they are much simpler to implement.

5.2. Quality

Implicit Visibility vs. Antiradiance: For our closed room test scene, Implicit Visibility turned out to be very sensitive to discretization and produced many sharp quantization artifacts. Antiradiance tended to overblur illumination but degenerated with less objectable artifacts when the number of bins and patches was reduced. Overall we have the subjective impression that both methods are tied in terms of image quality for a given computational budget.

Push down links: While Dong et al. suggest to push the receiver end of the created links to the leaf patches to avoid scatter operations on the GPU, we observe that our implementation spends only 7% of the total time per bounce scattering data. However, we found that pushing down the receiver side one or two levels, if child patches exist, drastically reduced banding artifacts, common to hierarchical radiosity methods, at little extra cost: The number of patches and bins remains unchanged, but while it almost quadruples the link count, the time for link creation only doubles as the push down is very simple. Most surprisingly, quadrupling the number of links does not even double the time for light

exchange. By pushing down links, we essentially cluster links with neighboring receivers and the same sender. This improves memory locality during light transport, and neighboring threads even distribute light from the same sender patch, sharing the same global memory read. The reduced banding even allows us to use a reduced number of patches for comparable visual quality as Figure 7 demonstrates.
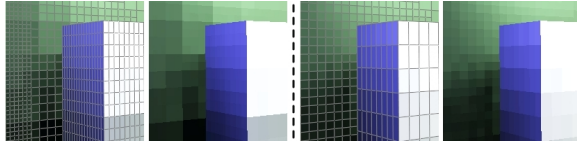

Figure 7: *Hierarchical radiosity methods are prone to banding artifacts. By pushing down links on the receiver side we can use fewer patches with better visual quality (left: 21 k patches @ 13 fps; right, 7.6 k patches @ 30 fps; both rendered with Antiradiance, 3 bounces and 256 bins).*

6. Conclusion and Future Work

We have presented a method to produce links for hierarchical radiosity methods in parallel, and generate around 50 M links per second for Antiradiance and 15 M links per second for Implicit Visibility. This allows moderately complex dynamic scenes with global illumination at 30 to 40 fps.

With the push down of links we presented a simple method to reduce banding artifacts common to hierarchical radiosity methods at moderate additional cost.

While the compact patch hierarchy and push down of links allows to reduce memory consumption to some degree, the memory required for the bins is still the most limiting factor. However not all bins are linked: For 21 k patches and 256 bins per patch we generate 411k links (Implicit Visibility) for 5.4 M bins, indicating considerable opportunities to save memory.

Overall, link creation is about as fast as one light transport. An intriguing idea we want to explore, is to combine both for BF-refinement. Also we experimented only with subdividing faces. While the extension to the "surfel hierarchy" of Dong et al. should be straight forward, an interesting question would be how to handle self shadowing while exchanging light only on the upper levels of the hierarchy.

References

[AGCA08] AJMERA P., GORADIA R., CHANDRAN S., ALURU S.: Fast, parallel, GPU-based construction of space filling curves and octrees. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games* (2008), pp. 1–1. 1

[Ble90] BLELLOCH G. E.: *Vector Models for Data-Parallel Computing*. MIT Press, 1990. 3

[BSKS05] BARSI A., SZIRMAY-KALOS L., SZIJÁRTÓ G.: Stochastic glossy global illumination on the GPU. In *Spring Conference on Computer Graphics 2005* (May 2005), pp. 187–193. 2

[CHL04] COOMBE G., HARRIS M. J., LASTRA A.: Radiosity on graphics hardware. In *Graphics Interface 2004* (May 2004), pp. 161–168. 2

[DBB06] DUTRE P., BALA K., BEKAERT P.: *Advanced Global Illumination*. AK Peters, 2006. 1

[DKTS07] DONG Z., KAUTZ J., THEOBALT C., SEIDEL H.-P.: Interactive global illumination using implicit visibility. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (2007), pp. 77–86. 1, 2, 3, 4

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (2005), pp. 203–231. 1

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (2006), pp. 93–100. 1

[DSDD07] DACHSBACHER C., STAMMINGER M., DRETTAKIS G., DURAND F.: Implicit visibility and antiradiance for interactive global illumination. *ACM Transactions on Graphics 26*, 3 (July 2007), 61:1–61:10. 1, 2, 3, 4, 5

[HOJ08] HACHISUKA T., OGAKI S., JENSEN H. W.: Progressive photon mapping. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers* (2008), pp. 1–8. 1

[HSHH07] HORN D. R., SUGERMAN J., HOUSTON M., HANRAHAN P.: Interactive k-d tree GPU raytracing. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games* (2007), pp. 167–174. 1

[HSO07] HARRIS M., SENGUPTA S., OWENS J. D.: Parallel prefix sum (scan) with CUDA. In *GPU Gems 3*, Nguyen H., (Ed.). Addison Wesley, Aug. 2007. 3

[Jen01] JENSEN H. W.: *Realistic Image Synthesis using Photon Mapping*. A. K. Peters, Ltd., 2001. 1

[Kel97] KELLER A.: Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), pp. 49–56. 1

[LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007* (2007), pp. 277–286. 1

[PGSS07] POPOV S., GÜNTHER J., SEIDEL H.-P., SLUSALLEK P.: Stackless kd-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum 26*, 3 (Sept. 2007), 415–424. 1

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers* (2008), pp. 1–8. 1

[STK08] SCHMITZ A., TAVENRATH M., KOBBELT L.: Interactive global illumination for deformable geometry in CUDA. *Computer Graphics Forum 27*, 7 (2008). Pacific Graphics 2008 Conference Proceedings. 2

[WFA*05] WALTER B., FERNANDEZ S., ARBREE A., BALA K., DONIKIAN M., GREENBERG D. P.: Lightcuts: a scalable approach to illumination. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (2005), pp. 1098–1107. 2

[WH92] WARD G. J., HECKBERT P. S.: Irradiance gradients. In *Eurographics Workshop on Rendering* (1992), pp. 85–98. 1

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