The Irregular Z-Buffer and its Application to Shadow Mapping

Gregory S. Johnson [1,2] * William R. Mark [1] † Christopher A. Burns [1,2] ‡

Department of Computer Sciences [1] Texas Advanced Computing Center [2]
The University of Texas at Austin

Abstract

The classical Z-buffer algorithm samples a scene at regularly spaced points on an image plane. We present an extension of this algorithm called the *irregular Z-buffer* that permits sampling of the scene at arbitrary points on the image plane. The sample points are stored in a two-dimensional spatial data structure which is queried during rasterization. The irregular Z-buffer can be applied to shadow rendering, where we demonstrate that it eliminates the sampling artifacts previously associated with shadow mapping. We describe the extensions to modern graphics hardware necessary to support efficient operation of the irregular Z-buffer, and simulate the performance on the extended hardware. Our results indicate that the irregular Z-buffer can be used to produce hard shadows that are as accurate as those produced by a ray tracer or shadow volume-based renderer. We also find that shadow mapping on the irregular Z-buffer retains many of the performance and application-development simplicity advantages of standard shadow mapping.

CR Categories: I.3.1 [Computer Graphics]: Hardware Architecture—Graphics Processors; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Visible Line / Surface Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shadows;

Keywords: visible surface algorithms, shadow algorithms, real-time graphics hardware

1 Introduction

One of the fundamental computational tasks in three-dimensional graphics is the visible surface problem: to efficiently find the first intersection point between a ray and a collection of surfaces. In rendering, the visible surface problem is typically solved for many rays and a single collection of surfaces. Examples include visibility determination from an eye point, shadow computation, and computation of global illumination solutions.

Today, the visible surface problem is typically solved by either the Z-buffer algorithm [Catmull 1974] or by ray casting using a spatial acceleration structure [Whitted 1980]. The Z-buffer algorithm is appropriate for many rays that share a common origin and share a known pattern of directions. It does not require any preprocessing of the surfaces. In contrast, the ray casting algorithm solves the visible surface problem for the general case – one or more rays with arbitrary origins and directions – but requires preprocessing of the surfaces to build a spatial acceleration structure.

We present an algorithm that is more general than the classical Z-buffer, but does not require preprocessing of surfaces. This approach, which we refer to as the irregular Z-buffer, efficiently solves the visibility problem for rays that share a common origin but have arbitrary directions (Figure 2).

Our algorithm represents ray directions as points on an image plane, as in the classical Z-buffer approach. However, we explicitly store all of the sample locations in a two-dimensional spatial data structure rather than implicitly representing them with a regular pattern. The data structure can be any spatial data structure that supports efficient range queries, such as a k-d tree or a grid. Like the classical Z-buffer algorithm, we project triangles onto the image plane one at a time and then determine which samples lie inside a triangle. Unlike the classical algorithm, this determination is made by querying the two-dimensional spatial data structure. Finally, for each sample inside a triangle, we perform the standard Z comparison and update at the sample's address in a Z-buffer.

We demonstrate that the irregular Z-buffer can be used to render high quality shadows cast by a point light source, as seen from a

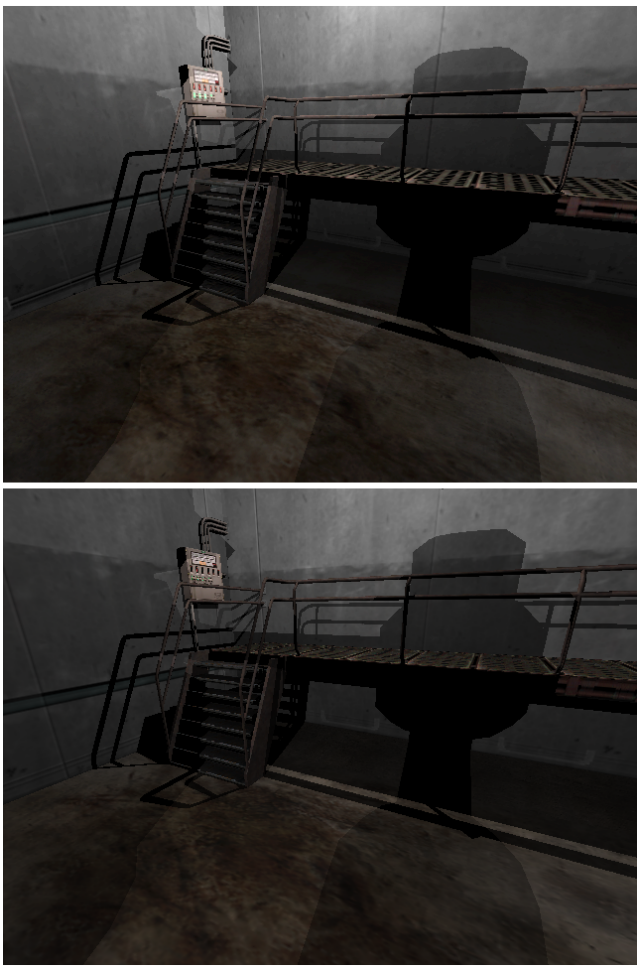


Figure 1: *Two images of a scene from Doom 3 (alpha). A screen capture directly from the game engine is shown at top. The game engine employs shadow volumes to render this image. The lower image is rendered by a software pipeline that implements irregular shadow maps.*

*e-mail: johnsong@cs.utexas.edu
†e-mail: billmark@cs.utexas.edu
‡e-mail: cslugg@cs.utexas.edu

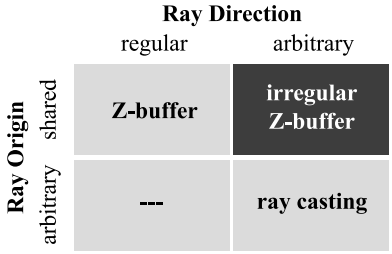


Figure 2: *A taxonomy of solutions to the visible surface problem, classified by the extent to which eye rays share origins and directions.*

known eye point. We propose changes to current graphics architectures that permit our algorithm to execute efficiently. Finally, we simulate the performance of shadow rendering using the irregular Z-buffer. Our results indicate that the approach is suitable for real-time use on our proposed architecture.

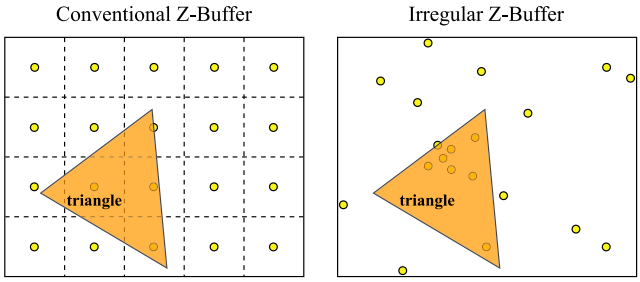


Figure 3: *The classical Z-buffer (left) samples a scene at regularly spaced points on an image plane. The irregular Z-buffer (right) samples a scene at arbitrary points on an image plane.*

2 The Irregular Z-Buffer

The Z-buffer algorithm processes scene geometry one polygon at a time. Each polygon is projected onto the image plane. An edge-walking or edge equation computation is used to determine which sample points (i.e. pixels) lie inside the projected polygon.

Answering this range query is simple when the positions of the sample points are implicitly defined by a simple formula, such as that for a regular grid (Figure 3, left side). Classical rasterization algorithms target this case and evaluate the formula directly – they do not explicitly store the location of each sample point.

However, if the locations of the sample points cannot be represented by a simple formula (Figure 3, right side), the standard rasterization approach does not work. In this paper we propose an alternative approach to handle this case. We explicitly store the points in a spatial data structure. During rasterization, a range query is performed on this data structure to determine which points are inside a triangle. We refer to this task as *irregular rasterization.*

The data structures and algorithms needed for efficient range queries are well understood [Samet 1990]. Range queries can be performed efficiently for points stored in two-dimensional space partitioning data structures such as BSP trees and quadtrees or their specialized variants such as k-d trees. If the points are somewhat evenly distributed, then grids are also efficient for this task.

The best choice of data structure depends on several factors. If the set of sample points changes often – as it does for the shadow mapping problem we examine later in this paper – then the cost of constructing the data structure is important. The construction cost can vary asymptotically depending on the choice of data structure, the overall distribution of sample points, and the order in which the points are inserted. Once the data structure has been built, the cost of range queries depends on these same factors as well. However, even algorithms with the same asymptotic cost have different effective costs that vary with usage patterns and the underlying hardware architecture. Finally, the storage requirements of the data structures vary and only certain ones can guarantee a fixed memory footprint regardless of point distribution.

For the shadow-mapping application on which we focus, the point distribution is somewhat even across large regions of the image plane, so we found that a hybrid data structure composed of a grid at the top level and a secondary data structure below it performs better than a simple k-d tree. We gathered statistics on two variants of the two-level structure: a grid of lists and a grid of k-d trees, both implemented with guaranteed storage bounds. We determined that the grid of lists (including optimizations described later) performed best for our data sets. The reasons for this result are that our average list size is small and that the point-in-area test at each step of list traversal is simpler than the area-to-area overlap test required at each step of k-d tree traversal. The difference was not overwhelming, and the outcome could be different for less balanced point distributions or a different architecture.

Constructing the grid-of-lists data structure is straightforward: we first determine which grid cell a point maps to, then prepend that point to the linked list for that cell. Irregular rasterization using this data structure is also simple. First, we determine which grid cell(s) the triangle overlaps. This test is just like conventional rasterization with the minor difference that we must consider any overlap between the triangle and a cell as a hit, rather than just an overlap between the triangle and the cell's center. At each cell that the triangle overlaps, we traverse the linked list to test all of the cell's points against each of the triangle's three edges. A point that passes all three edge tests is inside the triangle.

3 Shadow Rendering

Shadows provide important spatial cues and add to the realism of computer generated images. Shadow rendering algorithms are often placed into two categories: those for point light sources (hard shadows) and those for area light sources (soft shadows) [Assarsson and Akenine-Möller 2003; Chan and Durand 2003]. In this paper we focus on hard shadow algorithms. In particular, we consider the problem of rendering shadows in real-time for scenes containing moving and deforming objects. Even though this problem is conceptually simple and is of practical interest, existing approaches to it have significant shortcomings with respect to image quality or their performance on complex scenes.

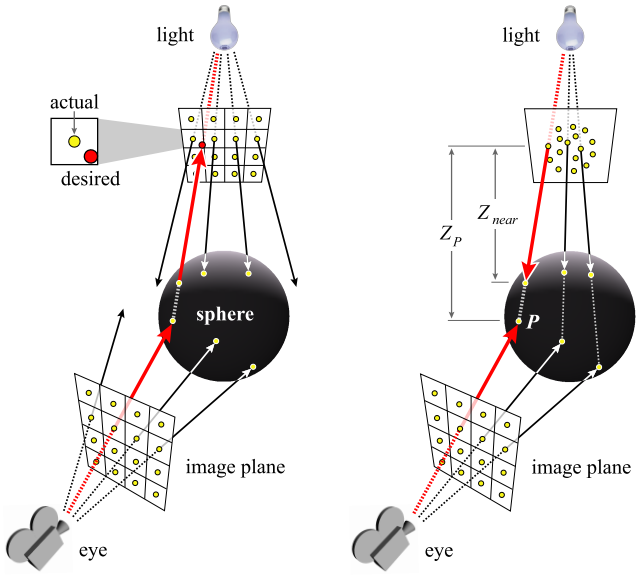
Object-space methods such as shadow volumes [Crow 1977; Everitt and Kilgard 2002] can achieve artifact-free hard shadows if implemented carefully. As a result, mainstream application developers such as id Software have chosen to use the approach in state-of-the-art video games like the upcoming Doom 3. The shadow volume technique creates and rasterizes extra polygons that demarcate the boundaries between volumes of space in and out of shadow in the scene. These extra polygons connect object silhouette edges (as seen from the light source) to infinity. The polygons are rasterized from the eye view into a stencil buffer, where a counter is incremented or decremented at each pixel to track the shadowed/non-shadowed status of the pixel. The polygons are often large, so the technique requires hardware with a very high fill rate. Even with recent hardware and software optimizations [NVIDIA Corp. 2003; McGuire et al. 2003], use of the technique requires that geometric scene complexity be held well below what would otherwise be possible.

Raytracing [Whitted 1980] and related techniques can accurately render a variety of global illumination effects including hard shadows. It is possible that real-time rendering systems will eventually adopt raytracing techniques. However, even with recent progress in this area [Wald et al. 2003], rendering performance remains inadequate for scenes containing deformable objects.

Shadow mapping [Williams 1978] and many of its variants [Hourcade and Nicolas 1985; Fernando et al. 2001] leverage existing Z-buffer hardware to render shadows with high performance for complex scenes. However, existing versions of the technique are prone to sampling and self-shadowing artifacts that are sufficiently serious to limit the technique's use in real applications.

Figure 4 (left) illustrates the shadow map algorithm. The scene is rendered first from the light position (yielding Z_{near} values) and then rendered from the eye position. Each pixel in the eye view is treated as a 3-space point positioned according to its X / Y position in the image plane and its Z value (from the depth buffer), and is transformed into light space. This transformation yields a point P in light space and a distance Z_P between P and the light-view image plane. The original eye-space pixel is considered to be in shadow iff $Z_{near} < Z_P$, using an estimated Z_{near} value. The algorithm estimates Z_{near} from the Z_{near} values of one or more light-view sample(s) that are nearest to the projection of point P onto the light-view image plane. This estimation step is the primary cause of artifacts produced by the technique as the estimation error is generally unbounded.

Most recent efforts to reduce these artifacts have taken one of two approaches. The first is to use additional information from object-space silhouette computations to reduce or eliminate estimation errors for the most common cases [Sen et al. 2003]. This approach seems to be the most successful at reducing the incidence of estimation artifacts, but sharp corners and details are often truncated or lost due to limited precision in the contours used to represent the silhouettes. Also, the need for object-space computation introduces additional complexity into the rendering system. The second approach is to adapt the sampling rate in the light-view image plane to the characteristics of the scene [Fernando et al. 2001; Stamminger and Drettakis 2002], thereby reducing the average distance between a projection of P and the nearest sample point. Fernando et al. [Fernando et al. 2001] replace the standard light view image with an adaptive image hierarchy. This focus on improving shadow quality through strategic placement of shadow map sample points is similar to our own, but we take this approach to its logical extreme by placing sample points in their ideal locations.



Conventional Shadow Mapping Irregular Shadow Mapping

Figure 4: *Conventional (left) and irregular (right) shadow mapping. In the case of the former, the scene is rendered to a conventional Z-buffer from the light, and then from the eye. With the latter, the scene is rendered to a conventional Z-buffer from the eye, and to an irregular Z-buffer from the light.*

4 Irregular Shadow Mapping

Pseudocode for irregular shadow mapping is shown in Figure 5. The scene is first rendered from the eye point. As in conventional shadow mapping, pixels (at the Z values given by the Z-buffer) are transformed into light space, yielding P and Z_P. Unlike conventional shadow mapping, scene geometry is *then* rasterized to sample positions in the light view image plane *given by the projection of the transformed pixels*, yielding Z_{near}. As before, a pixel is in shadow iff $Z_{near} < Z_P$. Note that irregular shadow maps are view-dependent. Samples are computed in the shadow map plane precisely where required by pixels in the eye view. Therefore, no mismatch exists between the sampling rates or sample positions in eye and light space. Aliasing and self-shadowing are avoided, and no unnecessary samples are computed. Moreover, given points P prior to rasterization in light space, and the property that Z_{near} is always less than or equal to Z_P, we can maximize our use of the available Z-buffer precision.

Figure 6 plots the location of sample points within irregular shadow maps for the Doom 3 scene from Figure 1. The density of sample points varies significantly across the image plane, demonstrating the importance of adaptive and irregular sampling methods in this context.

Observe that irregular shadow mapping effectively mimics shadow generation by ray tracing. Points P match the intersection points between eye rays and scene geometry; and steps 2, 4 and 6 imitate light ray computation. Unlike ray tracing, irregular shadow mapping is an object-order algorithm, which means that primitives are processed in the order submitted by the application. In this way, our approach combines the image quality and sampling characteristics of ray-traced shadows with the system organization and performance characteristics of Z-buffer rendering.

4.1 Image Quality

We compare the quality of images produced by irregular shadow mapping to that of several other approaches. Figure 1 shows that images generated using irregular shadow mapping are visually indistinguishable from those produced by the shadow volumes technique. Figure 7 shows that irregular shadow mapping eliminates shadow aliasing artifacts commonly associated with conventional shadow mapping. In Figure 8 we use an L2 norm to compare quantitatively the image quality of our approach to that of three other approaches. Our quantitative comparison is made against ray-traced shadows and against two other shadow mapping algorithms that avoid object-space computations: conventional shadow mapping [Williams 1978] and adaptive shadow mapping [Fernando et al. 2001]. This figure illustrates that the number of shadow map samples required to attain high fidelity is much less than that required by these other shadow mapping techniques.

Our conventional and adaptive implementations include standard enhancements to reduce self-shadowing and shadow aliasing artifacts. These enhancements include percentage closer filtering (PCF) [Reeves et al. 1987], object IDs [Hourcade and Nicolas 1985] and orientation-dependent bias values like those computed by `glPolygonOffset` [OpenGL Architectural Review Board 2003].

	Conventional Shadow Mapping	Irregular Shadow Mapping
* 0	render light view to regular Z-buffer (yields Z_{near})	
1	render eye view to regular Z-buffer (gives points P)	render eye view to regular Z-buffer - depth values only (gives points P)
* 2	transform P into light space (gives Z_P) and project to light view image plane (P')	transform P into light space (gives Z_P) and project to light view image plane
* 3		insert planar points into 2D spatial acceleration structure
* 4		render light view to irregular Z-buffer locations given by accelerator (yields Z_{near})
5		render eye view to regular Z-buffer
* 6	select the light view sample nearest P' and shadow pixel if $Z_{near} < Z_P$	shadow pixel if $Z_{near} < Z_P$

Figure 5: *A comparison of conventional and irregular shadow mapping. Steps labeled with "*" are repeated per light source.*

PCF and `glPolygonOffset` rely on configurable parameters. We carefully hand tune these parameters for each image of each test scene to minimize the difference in L2 norms from the baseline ray-traced image. Additionally, the field of view used for rendering the conventional and adaptive shadow maps is tuned to maximize the effective resolution of these maps.

We used two scenes for our evaluations. The first is from Doom 3 (a leaked version widely available on the Internet). It consists of two light sources and 8548 triangles which the Doom 3 game engine renders using stenciled shadow volumes [Everitt and Kilgard 2002]. The second scene is our creation and is designed as a challenging test case for shadow mapping methods (including our own). This scene is composed of one light source with shadows, one detail light without shadows, and 16372 triangles spread among a knot-like shape and several spheres. The silhouette edges of the curved surfaces are prone to generating regions of high sample density in irregular shadow maps.

All four shadow algorithms are implemented within a single software framework [Kolb 1997]. Doing so ensures the consistency of non-shadow-related image features (i.e. blending, texturing, etc.) across images produced with any shadow type. The framework can be configured to mimic a real-time Z-buffer renderer, with depth values stored in a 24-bit floating point format. Alternatively, the framework can be configured as a recursive ray tracer, with all intermediate values maintained at 64-bit floating point precision.

5 Irregular Shadow Mapping: Data Structures

Our implementation of irregular shadow mapping stores sample locations in an enhanced version of the grid-of-lists data structure described earlier. Our primary enhancement is to use a hash-like function to map a high resolution "logical" grid into a smaller "physical" grid, thereby reducing the memory footprint.

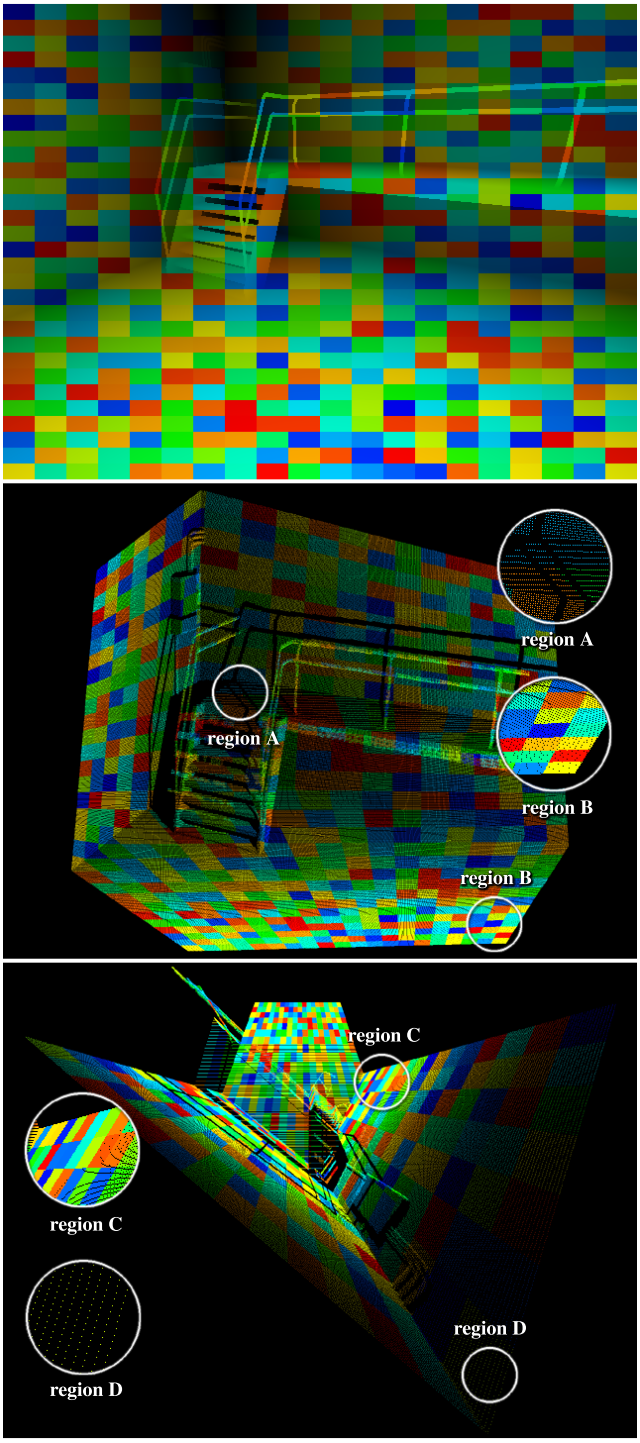


Figure 6: *The view from Figure 1 overlaid with colored tiles is shown at top. The lower images show the irregular shadow maps for the light sources behind and above the eye (center), and at the top of the right wall (bottom). We refers to these lights as "light 0" and "light 1" later in the paper. Colored points denote the tile containing the respective eye view pixel. Both maps contain regions of high (B, C) and low sample density (A, D).*

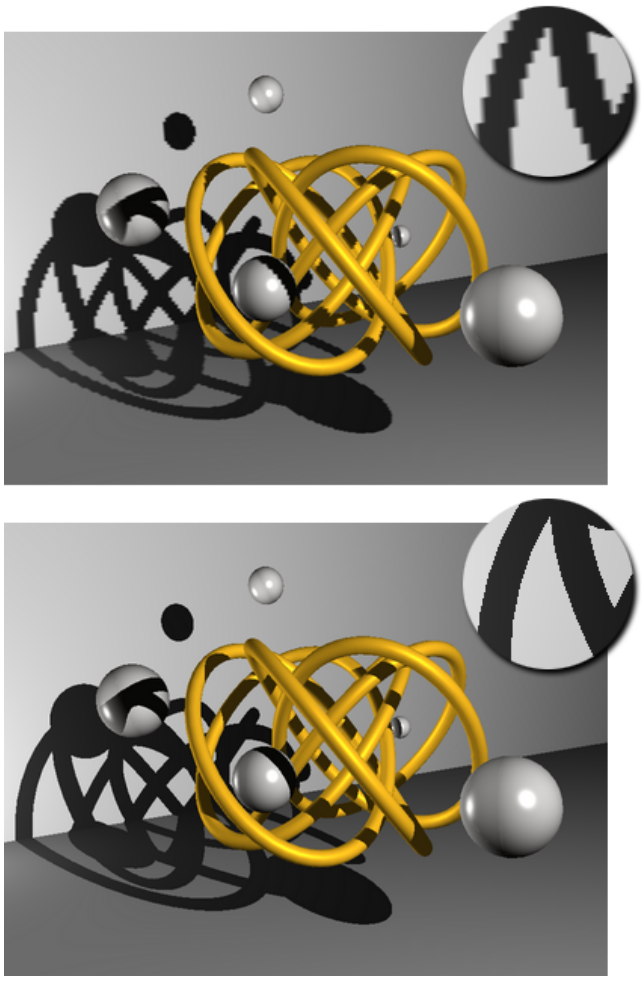


Figure 7: *A scene rendered with conventional shadow mapping and one sample per pixel is shown at top. The same scene rendered with irregular shadow mapping and one sample per pixel is shown at bottom.*

This structure is is illustrated in Figure 9. The logical grid overlays the light view image plane. Logical grid ($lgrid$) cell indices are mapped to indices in the physical grid ($pgrid$) using the hash-like function:

$$pgrid_i = (lgrid_i + \lfloor lgrid_j / pgrid_{width} \rfloor * tile_{width}) \% pgrid_{width}$$
$$pgrid_j = (lgrid_j + \lfloor lgrid_i / pgrid_{height} \rfloor * tile_{height}) \% pgrid_{height}$$

Each physical grid cell potentially forms the root of its own linked list. Shadow map samples projecting to the same cell become elements of the same list. Observe in Figure 6 that tiles of pixels in eye space remain largely intact in light space. Our hash-like function avoids folding contiguous regions of high sample density back onto themselves, while largely preserving any eye space - light space coherence. We exploit this coherence by tiling the physical grid in memory and processing fragments in tile order [McCormack et al. 1998].

This distinction between logical and physical grids (inspired by the work of Reinhard et al. [2000]), permits finer discretization of the shadow map plane than would otherwise be practical. Smaller cell sizes reduce the number and severity of hot spots (cells with

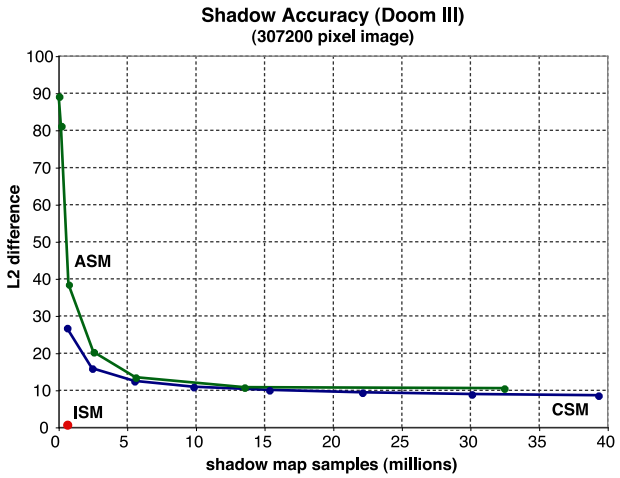


Figure 8: *A plot of the difference in L2 norms between images rendered with conventional (CSM), adaptive (ASM), and irregular (ISM) shadow maps and a reference image generated via ray tracing (one unjittered eye ray per pixel). The rendered scene is from Doom 3 (seen in Figure 1). The shadow map sample count represents the total over both light sources in this scene. Considerable effort was spent tuning CSM and ASM related parameters.*

significantly higher sample counts than other cells). Note that increasing the resolution of the logical grid relative to the physical grid yields a more even distribution of samples in the physical grid. However, increasing this ratio also increases the probability that the linked list at any particular physical grid cell will contain samples from distinctly different regions of the shadow map.

Construction of the data structure proceeds as follows. Eye view pixels are processed in tiles. The resulting spatial coherence yields memory access locality. Pixels in each tile are transformed into light space and projected onto the light view image plane. The index of the logical grid cell occupied by a projected point (shadow map sample) is mapped into the physical grid using the hash-like function shown previously. The sample is prepended to the linked list occupying this cell. Accelerator construction is parallelizable by assigning tiles of pixels to different processing elements and locking physical grid cells during list update.

Irregular rasterization is similarly straightforward. Our selection of a grid as the top level data structure enables us to leverage many components of existing rasterizers. Scene geometry is rasterized to the logical grid as normal, except that any cell crossed by a primitive is processed further. The primitive is tested against each element in the linked list of the cell. Z interpolation and a depth test are performed as usual for samples found to fall within the primitive. Viewport clipping is used to restrict rasterization to the region of the logical grid populated by samples. A stencil mask is used to discard fragments associated with logical grid cells devoid of samples. Rasterization is parallelizable by assigning logical grid cells covered by a given primitive to different processing elements. This approach is similar to parallel rasterization in existing hardware.

5.1 Architectural Support

Irregular shadow mapping is particularly appropriate for real-time rendering, as it provides image quality comparable to ray traced shadows and stenciled shadow volumes, with the computational efficiency of shadow maps. However, irregular shadow mapping is unlikely to run efficiently on current graphics hardware. An effi-

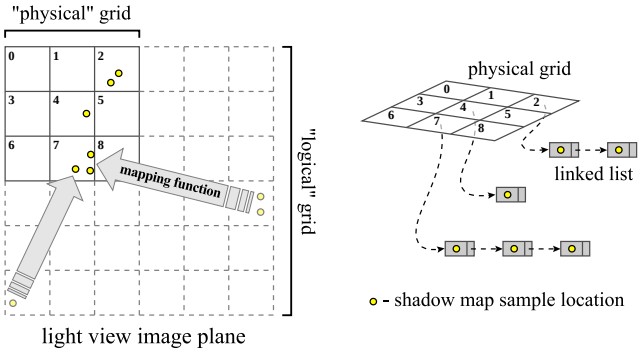


Figure 9: *The data structure used for irregular shadow mapping. The light view image plane is overlaid with a "logical" grid. The logical grid has higher resolution than the actual stored ("physical") grid. A hash-like function maps points in the logical grid into cells in the physical grid. Each cell of the physical grid potentially serves as the root of a linked list. The linked list contains the points mapping to a given cell, for later use during rasterization.*

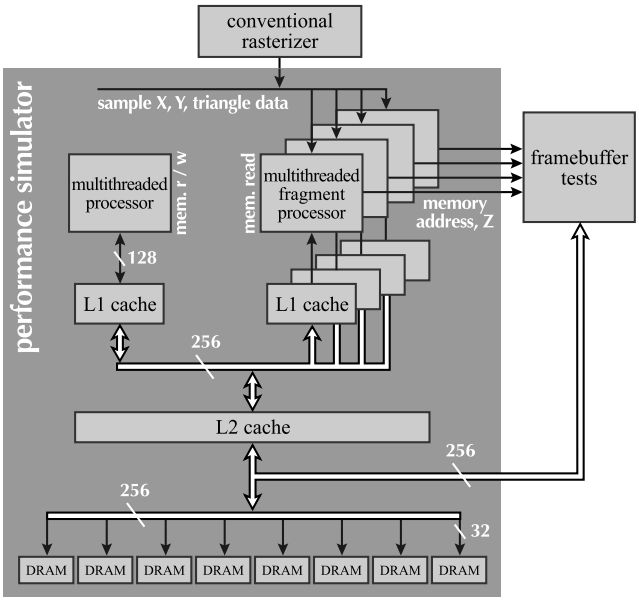


Figure 10: *An architecture with components which enable efficient execution of the irregular Z-buffer. The large gray box denotes architectural components included in our simulation.*

cient implementation of the technique requires additional functionality not supported in these GPUs. Our proposed architecture is illustrated in Figure 10, and combines an enhanced GPU with a multithreaded CPU-like processor (denoted "multithreaded processor") on the same die. This additional processor is similar to that found in some commercial embedded processor chips [Levy 2002], network processors [Adiletta et al. 2002], and experimental architectures [Caşcaval et al. 2002].

Irregular shadow mapping consists of two phases which use different parts of the architecture. Accelerator construction requires read / write access to the data structure in memory, and executes on the CPU-like processor. The construction kernel contains 40 instructions and consumes 8 live four-vector registers. Irregular rasterization requires read-only access to the data structure, and utilizes the graphics pipeline with its enhanced fragment processors. The respective kernel contains 14 instructions and consumes 8 live four-vector registers.

During irregular rasterization, the application sends polygons down the graphics pipeline just as it would during conventional Z-buffer rendering. The fixed-function rasterizer generates a fragment (hereafter *grid fragment*) for each logical grid cell overlapped by a triangle. Fragments covering logical grid cells marked empty in the stencil mask are discarded. Our enhanced fragment processor transforms the grid fragment's logical grid coordinates into physical grid coordinates, and traverses that physical grid cell's linked list of samples. Each sample in the cell is tested against the triangle's edge equations [Pineda 1988]. If a sample lies inside the triangle, the fragment processor evaluates the Z interpolation equation at the sample location and generates an output *framebuffer fragment* with the computed Z value. The fragment processor sets the address of the framebuffer fragment to the sample's index value. This index is the x / y position of the eye-view pixel in the image plane corresponding to the sample. Finally, each framebuffer fragment passes through the fixed-function Z-test unit, where the usual read / modify / write Z buffer comparison is performed.

As we've indicated, efficient support for irregular rasterization requires enhancing the capabilities of current graphics pipelines and their fragment processors [Microsoft Corporation 2003; Beretta et al. 2003]. First, the rasterizer needs a mode where it outputs a grid fragment when a triangle overlaps *any* portion of a pixel, not just its center as is the case now. Several setup / rasterizer designs can be modified to support this option, either by changing the pixel walking algorithm or by moving the triangle edges outward by $\sqrt{2}/2$ of pixel spacing. Additionally, the fragment processor must support conditional branching and have access to the homogeneous equations describing the edges and Z interpolation [Olano and Greer 1997] of the fragment's triangle. Finally, the fragment processor must be able to conditionally output a framebuffer fragment as many times as needed, to a different computed address (framebuffer location) each time. This computed-address capability is often referred to as a scatter operation and has proven to be useful for a wide variety of purposes in stream processors [Kapasi et al. 2002].

6 Simulation and Analysis

Any new technique targeted at real-time graphics applications must run fast as well as produce images of the desired quality, but evaluating the performance of new techniques like ours is challenging. Typically, an algorithm's performance is measured in one of two ways: at a high level by counting arithmetic operations used for some data set, or at a low level by implementing the algorithm on current hardware and timing its execution. For our technique, neither approach is adequate. Memory hierarchy effects such as cache misses and associated latencies have become too important for simple operation counting to yield accurate performance estimates for memory-intensive techniques. However, current architectures do not support our technique well, so timing an implementation of it on these architectures would not be very informative.

Instead, we have constructed a high-level performance simulator for the architecture we proposed in the previous section. Our simulator models the key performance characteristics of the architecture – particularly memory hierarchy effects – but does not attempt to model every detail that would be necessary to determine exact execution times. In other words, the primary purpose of our simulator is to evaluate algorithms rather than to evaluate architectures. We simulate the memory system performance in the greatest detail, the processor core performance at a medium level of detail, and the fixed-function graphics units at a functionality level only.

Our memory hierarchy simulator is adapted from an existing

Scene	Avg. Depth Complexity	Logical Grid Size	Physical Grid Size	Avg. Cell List Length	Stencil Buffer Size	Grid Fragments Pre / Post-Stencil	Framebuffer Fragments Pre / Post Z-Test
Doom light 0	3.43	1024 x 1024	512 x 512	5.32	641 x 692	1.60M / 1.26M	4.36M / 1.22M
Doom light 1	2.30	1024 x 1024	512 x 512	24.47	413 x 327	0.23M / 0.11M	3.05M / 1.27M
Knot light 0	2.98	1024 x 1024	512 x 512	16.06	209 x 938	0.57M / 0.41M	3.52M / 0.51M

Table 1: *Statistics captured during irregular rasterization for each light view in our scenes. Each light view contains 1,310,720 samples.*

event queue based simulator [Burger et al. 1999]. We discarded the out-of-order CPU core simulation components and replaced the DRAM subsystem simulator with our own. In our simulator, writes to one cache are incoherent with reads from another as automatic coherence is complex and unnecessary in the context of our algorithm. Our DRAM subsystem simulator honors all bank and channel timing restrictions (e.g. precharge times) specified by a memory manufacturer's data sheet [Micron Technology, Inc. 2003]. The configuration of the simulated memory system is shown in Table 2.

Our performance simulator uses a single model for the CPU-like processor core and the fragment processor core, except that the fragment processor is forbidden to write to memory and it supports stream inputs and outputs. The processors are multithreaded with zero cycle context switches and round-robin scheduling, allowing the processor to remain active even when most threads are waiting on cache misses or locks. Each processor is simple – it issues one scalar or 4-wide vector operation each cycle. We use the NV_vertex_program2 instruction set [Brown and Kilgard 2003], which includes conditional branch instructions but no integer arithmetic. We augment this instruction set with instructions for lock and unlock (using one of 256 on-chip locks), 128-bit load and store (indirect addressing OR'd with one of two base registers), and logical shifts. We assume that all instructions except load, store, and lock complete in one cycle; i.e. we do not model pipeline stalls caused by branch misprediction or arithmetic unit latency. These potential stalls would be partially or wholly covered by multithreading and are less important for the short pipeline we are assuming (6 stages @ 1.2GHz in 90nm) than they are for most modern 15+ stage CPUs.

				Latency (cycles)		
Cache	Size	Assoc.	Line Size	Hit	Miss Min.	Miss Avg.
L1	16 KB	8	64 B	1	6	6
L2	256 KB	8	128 B	6	42	75 - 118*

Table 2: *The configuration of our memory simulator. All processor cores and on-chip busses run at 1.2GHz. Caches utilize LRU replacement policy and are write-allocate, write-back. L2 minimum miss latency assumes a DRAM page hit and no contention. DRAM subsystem: 256-bit wide, eight-channel configuration using eight 700 MHz DDR chips. *The L2 average miss latency varies by scene.*

We separately simulate two key phases of the ISM algorithm: construction of the shadow map data structure, which runs on the CPU-like processor; and the portion of irregular rasterization that runs on the fragment processors. The simulated code has been carefully optimized to reduce instruction count – for example, the construction phase processes pixels in pairs to maximize use of 4-wide vector operations.

In the construction phase, each thread processes a different set of pixels. Within a thread, pixels are processed one tile at a time. The input data (eye-view Z-buffer) resides in cacheable DRAM, as does the output data (accelerator data structure and stencil buffer marking non-empty physical-grid cells). Prior to the construction phase we must clear the physical grid in the accelerator data structure (one pointer per cell) as well as the stencil buffer, but we exclude these clearing operations from our measurements because their costs are already well understood. Table 3 shows the simulated performance of the construction phase for each light in the two different scenes for an image resolution of 1280 x 1024.

Scene	Total Cycles	Equiv. Frame Rate	Cycles / Point	Useful Cycles (%)
Doom light 0	26,107,800	45.96	19.92	98.17
Doom light 1	25,805,020	46.50	19.69	99.32
Knot light 0	26,025,554	46.11	19.86	98.48

Table 3: *Simulated performance of the construction of the spatial acceleration structure, for each light view of the respective scene. In all cases the accelerator contains 1,310,720 points. A single CPU-like processor is used. The processor is multithreaded (8-way). "Useful cycles" refers to the percent of processor cycles spent on useful work (not stalling).*

In the rasterization phase of computation, grid fragments generated by the conventional rasterizer are assigned to the first available fragment processor thread. Our performance simulation assumes that the stencil test described in section 5 rejects grid fragments for empty physical grid cells, so these fragments are not assigned to a fragment-processor thread. We do not model the stencil buffer memory traffic or the Z test/replace memory traffic in our performance simulation, but the additional bandwidth consumed by these operations can be computed from Table 1.

Table 4 summarizes the performance of the fragment-processor stage of irregular rasterization, for four processors each with eight threads. These results demonstrate that our algorithm achieves real-time rates on the proposed architecture.

Scene	Total Cycles	Equiv. Frame Rate	Cycles / Tri. Sample	Useful Cycles (%)
Doom light 0	19,401,660	61.85	4.45	86.07
Doom light 1	9,663,102	124.18	3.17	94.32
Knot light 0	20,424,874	58.75	5.81	95.79

Table 4: *Simulated performance of the irregular Z-buffer during rasterization. The scene is rasterized to each light view of the respective scene. Each light view contains 1,310,720 samples. Four multithreaded fragment processors are used, each with eight threads. "Useful cycles" refers to the percent of processor cycles spent on useful work (not stalling). "Cycles / Tri. Sample" denotes the cycles required to process one triangle fragment against one light-view sample.*

7 Discussion and Conclusion

We have presented several results in this paper:

- **The irregular Z-buffer:** We have extended the Z-buffer algorithm to support arbitrary sample locations in the image plane.

- **Irregular shadow mapping:** We have demonstrated that the irregular Z-buffer can be used to generate shadow mapped shadows, and that its use eliminates the artifacts traditionally associated with shadow mapping.

- **Hardware performance analysis:** We have proposed a set of architectural enhancements to support irregular rasterization, and demonstrated in simulation that these capabilities enable the technique to run at interactive frame rates.

It is likely that the irregular Z-buffer has additional applications elsewhere in real-time rendering, including reflection mapping, adaptive sampling, and frameless rendering.

A key characteristic of the irregular Z-buffer approach is that it builds and traverses an adaptive spatial data structure. This class of data structures is widely used in graphics and is likely to be particularly important in the future for real-time ray tracing of scenes containing deformable objects. We believe that some of the algorithmic and architectural ideas we have presented can be applied to this broader problem. We hope that our work will help to inspire CPU and GPU architects to design future processors that efficiently support the construction, update, and traversal of adaptive data structures.

8 Acknowledgements

Ikrima Elhassan wrote the initial version of our Doom 3 scene parser and implemented adaptive shadow mapping. Don Fussell gave us a variety of useful suggestions, and in particular was the first of several people to urge us to investigate a grid-of-lists data structure as an alternative to the kd-tree data structure that we started with. Kelly Gaither and Jay Boisseau of the Texas Advanced Computing Center strongly encouraged and supported our work; the work environment they created made this research possible. Doug Burger gave us access to his memory hierarchy simulator and advice on how to use it. J. Mike O'Connor, Karthikeyan Sankaralingam and Stephen Keckler gave us useful advice on architecture choices and simulation. This work was supported by the National Science Foundation, Microsoft, and Intel.

References

ADILETTA, M., ROSENBLUTH, M., BERNSTEIN, D., WOLRICH, G., AND WILKINSON, H. 2002. The next generation of Intel IXP network processors. *Intel Technology Journal 6*, 3 (August).

ASSARSSON, U., AND AKENINE-MÖLLER, T. 2003. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics (TOG) 22*, 3, 511–520.

BERETTA, B., BROWN, P., CRAIGHEAD, M., AND EVERITT, C. 2003. GL_ARB_fragment_program extension specification.

BROWN, P., AND KILGARD, M. 2003. GL_NV_vertex_program2 extension specification.

BURGER, D., KÄGI, A., AND HRISHIKESH, M. S. 1999. Memory hierarchy extensions to the simplescalar tool set. *UT-Austin Computer Sciences Technical Report TR-99-25* (September).

CAŞCAVAL, C., NOS, J. G. C., CEZE, L., DENNEAU, M., GUPTA, M., LIEBER, D., MOREIRA, J. E., STRAUSS, K., AND JR, H. S. W. 2002. Evaluation of a multithreaded architecture for cellular computing. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA'02)*, IEEE Computer Society, 311–322.

CATMULL, E. 1974. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD dissertation.

CHAN, E., AND DURAND, F. 2003. Rendering fake soft shadows with smoothies. In *Proceedings of the Eurographics Symposium on Rendering*, Eurographics Association, 208–218.

CROW, F. C. 1977. Shadow algorithms for computer graphics. *Computer Graphics (SIGGRAPH '77 Proceedings) 11*, 2 (July).

EVERITT, C., AND KILGARD, M. 2002. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. *SIGGRAPH 2002 Course Notes course #31*.

FERNANDO, R., FERNANDEZ, S., BALA, K., AND GREENBERG, D. P. 2001. Adaptive shadow maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH 2001)*, ACM Press, 387–390.

HOURCADE, J. C., AND NICOLAS, A. 1985. Algorithms for antialiased cast shadows. *Computers & Graphics 9*, 3, 259–265.

KAPASI, U. J., DALLY, W. J., RIXNER, S., OWENS, J. D., AND KHAILANY, B. 2002. The Imagine stream processor. In *Proceedings of the IEEE Conference on Computer Design*, 295–302.

KOLB, C. 1997. Rayshade homepage.

LEVY, M. 2002. Tying up a MIPS32 processor with threads: Lexra evolves the LX4380 pipeline into multithreading processor. *Microprocessor Report* (November).

MCCORMACK, J., MCNAMARA, R., GIANOS, C., SEILER, L., JOUPPI, N. P., AND CORRELL, K. 1998. Neon: a single-chip 3d workstation graphics accelerator. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, ACM Press, 123–132.

MCGUIRE, M., HUGHES, J. F., EGAN, K., KILGARD, M., AND EVERITT, C. 2003. Fast, practical and robust shadows. *Brown University Technical Report CS03-19* (October).

MICRON TECHNOLOGY, INC. 2003. Micron Technology 256 Mb (x32) GDDR3 SDRAM datasheet.

MICROSOFT CORPORATION. 2003. DirectX 9.0 SDK.

NVIDIA CORP. 2003. NVIDIA GeForce FX 5900, 5700 and Go5700 GPUs: Ultra Shadow technology.

OLANO, M., AND GREER, T. 1997. Triangle scan conversion using 2D homogeneous coordinates. In *Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Workshop on Graphics Hardware*, ACM Press, 89–95.

OPENGL ARCHITECTURAL REVIEW BOARD. 2003. OpenGL 1.5 specification.

PINEDA, J. 1988. A parallel algorithm for polygon rasterization. *Computer Graphics (SIGGRAPH '88 Proceedings) 22*, 4 (August), 17–20.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering antialiased shadows with depth maps. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, 283–291.

REINHARD, E., SMITS, B., AND HANSEN, C. 2000. Dynamic acceleration structures for interactive ray tracing. In *Proceedings of the 11th Eurographics Workshop on Rendering*, Eurographics Association, 299–306.

SAMET, H. 1990. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Longman Publishing Co., Inc.

SEN, P., CAMMARANO, M., AND HANRAHAN, P. 2003. Shadow silhouette maps. *ACM Transactions on Graphics (TOG) 22*, 3, 521–526.

STAMMINGER, M., AND DRETTAKIS, G. 2002. Perspective shadow maps. In *Proceedings of ACM SIGGRAPH 2002*, ACM Press / ACM SIGGRAPH, J. Hughes, Ed., 557–563.

WALD, I., PURCELL, T. J., SCHMITTLER, J., BENTHIN, C., AND SLUSALLEK, P. 2003. Realtime ray tracing and its use for interactive global illumination. In *State of the Art Reports, EUROGRAPHICS 2003*.

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6 (June), 343–349.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings) 12*, 3 (August), 270–274.