The Dissertation Committee for Gregory Scott Johnson
certifies that this is the approved version of the following dissertation:

A Hybrid Real-Time Visible Surface Solution for Rays With a Common Origin and Arbitrary Directions

Committee:

William R. Mark, Supervisor

Okan Arikan

Donald S. Fussell

Stephen W. Keckler

David B. Kirk

A Hybrid Real-Time Visible Surface Solution for Rays With a Common Origin and Arbitrary Directions

by

Gregory Scott Johnson, B.S., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2008

Acknowledgments

Learn everything you can, anytime you can, from anyone you can - there will always come a time when you will be grateful you did.

– Sarah Caldwell

I have had the good fortune to know and to learn from several individuals who wittingly or otherwise contributed to this work. Foremost among these, my advisor Bill Mark. For persistently insisting on a high standard of research, writing, and clarity of thought, I am grateful. I am likewise grateful for five years of advice and encouragement. To the other members of my graduate committee, particularly Don Fussell, I am grateful for the wisdom and for making the proposal and defense among the most positive experiences of my student career. Beyond my committee, I am indebted to Jay Boisseau. He is the Ranger to my ring bearer. To Jay and to Kelly Gaither I am grateful for seven years of guidance and unfailing support.

Several others have directly participated in this research, none more so than Chris Burns. Chris will (rightly) attest that one of Dante's nine circles of Hell is reserved for systems researchers who dare to write a hardware simulator. Though this sin is mine the punishment was shared. You will see his name throughout the following text. This work is also better for the contributions of Warren Hunt, Allen Hux, Stephen Junkins, Jeff Boody, Juhyun Lee, Alex Joly, and Ikrima Elhassan.

Less direct, but no less important, are the contributions of my family and friends. In a myriad of ways large and small they have helped make this seven year journey bearable. My mother, father, and sister in particular have underwritten this work through their unwavering encouragement. I am likewise grateful to Theresa, for persistently reminding me that life exists beyond this work, for forgiving my absences, and for politely refraining from comment on my increasingly grey hair as the years mounted.

Finally, I am sincerely grateful to the NVIDIA Corporation and the Intel Corporation for their support of this work from its inception through completion, in the form of funding, hardware and software systems, and early access to new chip architectures.

Gregory Scott Johnson

The University of Texas at Austin
December 2008

A Hybrid Real-Time Visible Surface Solution for Rays With a Common Origin and Arbitrary Directions

Publication No. _____

Gregory Scott Johnson, Ph.D.
The University of Texas at Austin, 2008

Supervisor: William R. Mark

A fundamental operation in computer graphics is to determine for a given point and direction in a scene, which geometric surface is nearest this point from this direction and thus visible. Conceptually, the point and direction define a "ray". Z-buffer hardware can compute surface visibility for a set of rays with a common origin (i.e. eye point) and a regular pattern of directions in real-time. However, this hardware is much less efficient at performing other visibility computations such as those required to accurately render shadows. A more flexible solution to the visible surface problem is needed.

This work introduces the *irregular Z-buffer* algorithm, which efficiently solves the visible surface problem for rays with a common origin and *arbitrary* directions. In addition, we identify several changes to classical graphics architectures needed for hardware acceleration of this algorithm. Though these modifications are incremental in nature (i.e. no new functional units are introduced), we show that they enable significant new capability. In tandem with the irregular Z-buffer algorithm, a GPU with these changes has applications in: shadow rendering, indirect illumination, frameless rendering, adaptive anti-aliasing, adaptive textures, and jittered sampling. We explore the performance of hard and soft shadow rendering in particular, by way of a detailed hardware simulator.

Contents

List of Tables

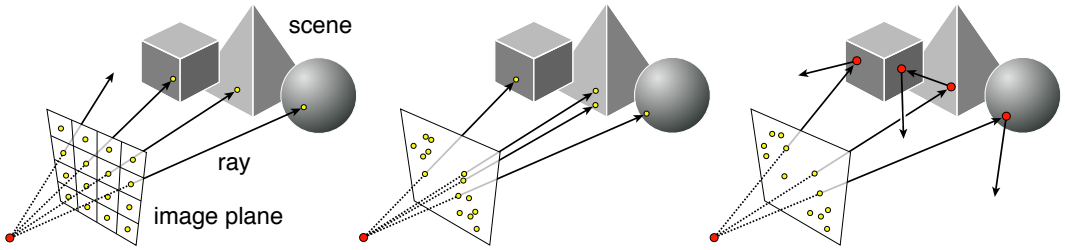
List of Figures

Chapter 1

Introduction

Shadow rendering is one example of a problem in real-time graphics for which robust solutions remain elusive in scenes with high geometric complexity and dynamism. Current methods such as shadow mapping and shadow volumes, yield unsatisfactory image quality or performance. One reason for this is the inflexibility inherent in the solution to the visible surface problem employed by commodity graphics hardware.

Conceptually, the visible surface problem can be expressed by the question: *for a point and direction or* "ray" *in a scene, which geometric surface is nearest this point from this direction and thus visible?* The visible surface problem is commonly solved with the Z-buffer algorithm [27] or ray tracing [138]. The Z-buffer algorithm is limited to rays emanating from a shared origin along a regular pattern of directions (Figure 1.1a), but is backed by specialized hardware that is widely available and achieves high performance. Ray tracing is applicable to the general case of rays with arbitrary origins and directions (Figure 1.1c), but is considered to be non-real-time for secondary ray effects like soft shadows, due to the lack of suitable acceleration hardware. We posit that important problems in real-time graphics such as shadow rendering, can be addressed with a visibility solution that is more flexible than the classical Z-buffer but less general than ray tracing, enabling hardware acceleration to be achieved through incremental changes to conventional GPU designs.

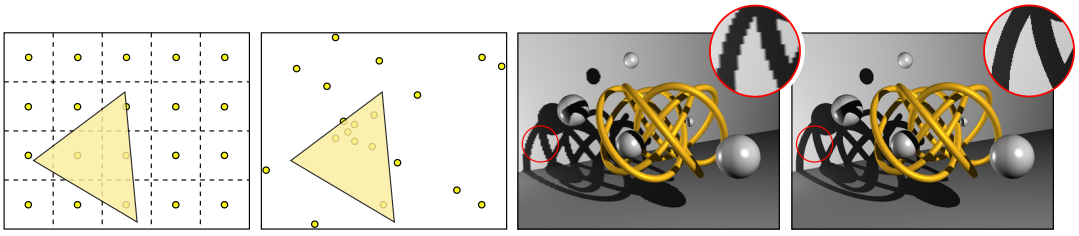


(a) Classical Z-buffer. (b) New: Irregular Z-buffer. (c) Ray tracing.

Figure 1.1: Three solutions to the visible surface problem in order of flexibility. Red dots denote ray origins. Yellow dots are points in the scene seen from the respective positions in the image plane. The classical Z-buffer (a) is suitable for rays with a common origin and a regular pattern of directions. The *irregular Z-buffer* (b) is suitable for the case shown in (a) as well as for rays with a common origin and arbitrary directions. Ray tracing (c) is the most general solution. It is suitable for cases (a) and (b) as well as for rays with arbitrary origins and directions.

1.1 Overview

We introduce the *irregular Z-buffer* algorithm. It solves the visible surface problem for rays with a common origin and *arbitrary* directions (Figure 1.1b). The efficacy of this approach can be seen in shadow rendering. Shadows provide key visual cues and improve the realism of rendered scenes. Current research largely focuses on soft shadows, but even the conceptually simpler case of hard shadows has not been solved. Existing solutions perform poorly intensive or are susceptible to visual artifacts. For example, shadow mapping [139] can produce hard shadows with high performance, but results in aliasing and self-shadowing artifacts (Figure 1.2c). The scene is rendered from the eye and light, and the two views are compared to ascertain if points in the scene visible from the eye are also visible from the light (i.e. not in shadow), or are occluded by intervening geometry. This comparison is error prone due to the lack of spatial correspondence between the eye and light sampling patterns (Figure 1.4a). *Irregular shadow mapping* is based on the irregular Z-buffer algorithm and avoids these artifacts by deriving the light-view sample positions from the points in the scene visible from the eye (Figure 1.4b).



(a) Classical Z-buffer. (b) Irregular Z-buffer. (c) Classical shadow mapping. (d) Irregular shadow mapping.

Figure 1.2: The classical Z-buffer (a) samples a scene at regularly-spaced points in the image plane. The irregular Z-buffer (b) samples a scene at arbitrary points in the image plane. This capability has applications throughout real-time graphics including shadow rendering, where it eliminates (d) the aliasing and self-shadowing artifacts common to classical shadow mapping (c).

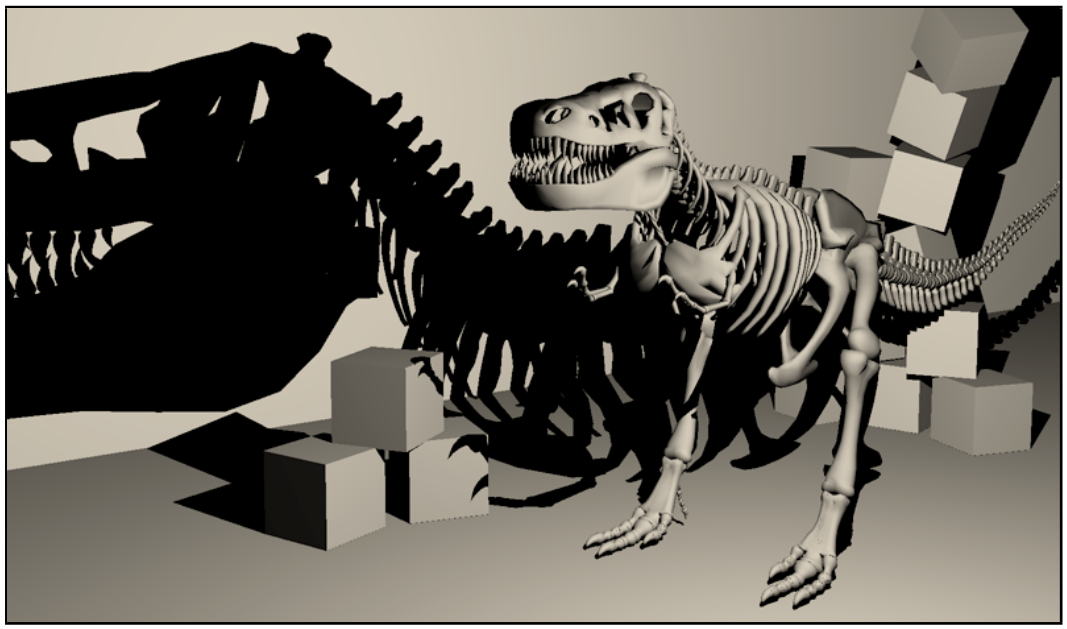
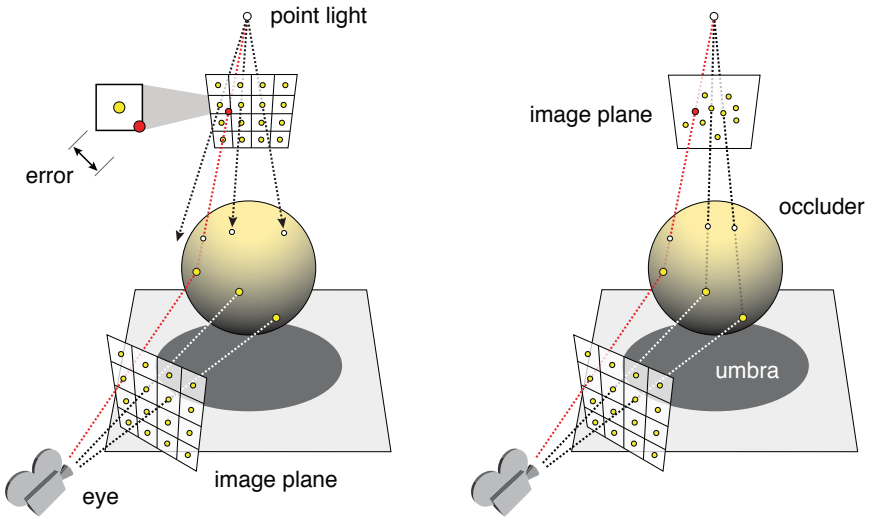


Figure 1.3: An example of hard shadows rendered using irregular shadow mapping.



(a) Classical shadow mapping. (b) Irregular shadow mapping.

Figure 1.4: The artifacts associated with classical shadow mapping (a) result from a mismatch in the eye and light-view sampling patterns. Irregular shadow mapping (b) avoids these artifacts by rendering the scene to positions in the light-view image plane computed from the points in the scene visible from the eye.

Note that the resulting light-view sample pattern is irregular (Figure 1.2b). This irregularity inhibits efficient implementation of the irregular Z-buffer algorithm on conventional GPUs. To understand why, consider the classical Z-buffer algorithm. Scene geometry is first projected into the image plane. It is then determined which samples from a regularly-spaced set lie inside each primitive (Figure 1.2a). Since the locations of these samples (i.e. pixels) are implicit, this determination can be made by testing the edges of the projected primitive against the sample grid. However, if the sample locations are irregularly-spaced and cannot be computed from a formula (Figure 1.2b), then the classical Z-buffer approach is invalid. Instead, the irregular Z-buffer algorithm stores the sample locations in a non-uniform spatial acceleration structure. Samples are organized by their relative spatial positions, enabling efficient range queries even on sets of arbitrarily-placed points.

GPUs circa 2005 provide the programmability and branching [23] required for hardware-accelerated *traversal* of irregular data structures, but are missing the architectural features necessary for their efficient *construction*. These features are crucial for rendering dynamic scenes in which the data structure is updated per frame, but are just beginning to appear in next-generation GPU designs. As part of this dissertation work, we identify the changes needed for hardware-accelerated construction of 2D and 3D non-uniform data structures. These modifications are incremental in nature and do not entail the addition of new functional units. Paired with the irregular Z-buffer algorithm, a modified GPU has potential applications in: shadow rendering, indirect illumination, frameless rendering, adaptive anti-aliasing, adaptive textures, and jittered sampling. Here, we focus on the performance of hard and soft shadow rendering using a detailed hardware simulator which implements these changes.

1.2 Thesis Statement

This dissertation work is based on two assertions. First: *new, efficient, and robust solutions to important but unresolved problems in real-time graphics are attainable given a method for computing surface visibility that is more flexible than the classical Z-buffer, but which need not be as general as ray tracing.* Second: *hardware support for this new visible surface algorithm is achievable through incremental changes to classical GPU designs, achieving high performance with reasonable cost.*

As an example, we will show that it is possible to efficiently render hard and soft shadows that are geometrically-equivalent to those produced by ray tracing (i.e. high quality), using a modestly more flexible solution to the visible surface problem than the classical Z-buffer algorithm. Further, such shadows can be rendered with high performance in dynamic and geometrically-complex scenes, with only minimal changes to conventional GPU designs.

1.3 Approach

In principle, there are several viable approaches to developing a real-time visibility solution with greater flexibility than the classical Z-buffer algorithm. These include: developing acceleration hardware specific to ray tracing, optimizing ray tracing for general-purpose hardware, or adapting the Z-buffer algorithm and its underlying architecture to support ray tracing like functionality. Work is already under way on a hardware architecture for ray tracing [143], and substantial recent effort has centered on optimizing ray tracing for GPUs [107] and single and multi-core CPUs [112, 125, 134, 133, 61]. This dissertation work focuses on the relatively unexplored middle ground between Z-buffer rendering and ray tracing. While our approach does not achieve the full flexibility of ray tracing, it does enable substantial improvements in functionality while keeping the system organization and performance advantages of commodity graphics hardware.

1.4 Contributions

The key contributions of this work form a system for real-time graphics as seen in Figure 1.5 and are discussed more in the next subsections. These contributions are:

1. an advanced Z-buffer algorithm (the *irregular Z-buffer*) for determining surface visibility at arbitrary points in the image plane,

2. a set of architectural enhancements to classical GPUs for hardware-accelerated creation of 2D and 3D irregular spatial data structures for storing these points,

3. new methods based on the irregular Z-buffer algorithm and its architecture, for rendering high-quality hard and soft shadows with high performance in dynamic and geometrically-complex scenes, and

4. evaluation of the irregular Z-buffer algorithm, architecture, and applications through the use of an execution-driven, hardware performance simulator.

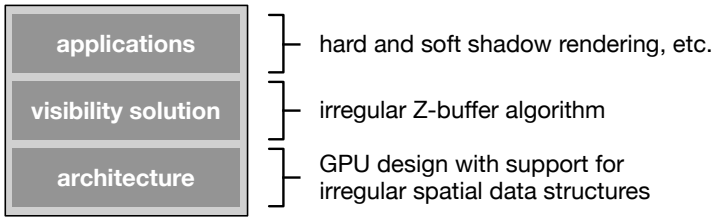


Figure 1.5: The major original contributions of this dissertation work relate to elements of a system for real-time graphics. The system is composed of a hardware architecture, a visibility algorithm implemented on top of this architecture, and applications of the combined algorithm and architecture.

1.5 Visibility Algorithm

To review, the irregular Z-buffer algorithm and its supporting architecture solve the visible surface problem for rays with a shared origin and arbitrary directions, as in Figure 1.1b. The algorithm proceeds in three steps: data structure construction, classical rasterization, and irregular rasterization. Ray directions are represented as points within a 2D image plane. During *data structure construction*, coordinates associated with these points are inserted into a 2D or 3D acceleration structure. In principle, any data structure which supports efficient range queries can be used, such as a tree or a grid. We employ a grid-based structure, enabling the use of *classical rasterization* to conservatively estimate the set of points which will be tested against a given primitive during *irregular rasterization*. During classical rasterization, scene primitives are individually projected into the image plane. For each pixel overlapped by a primitive, irregular rasterization performs a range query on the acceleration structure to retrieve points located within the pixel extents, and tests these points for occlusion against the primitive. For each covered point, a depth comparison is performed and the respective value is conditionally updated in Z-buffer memory.

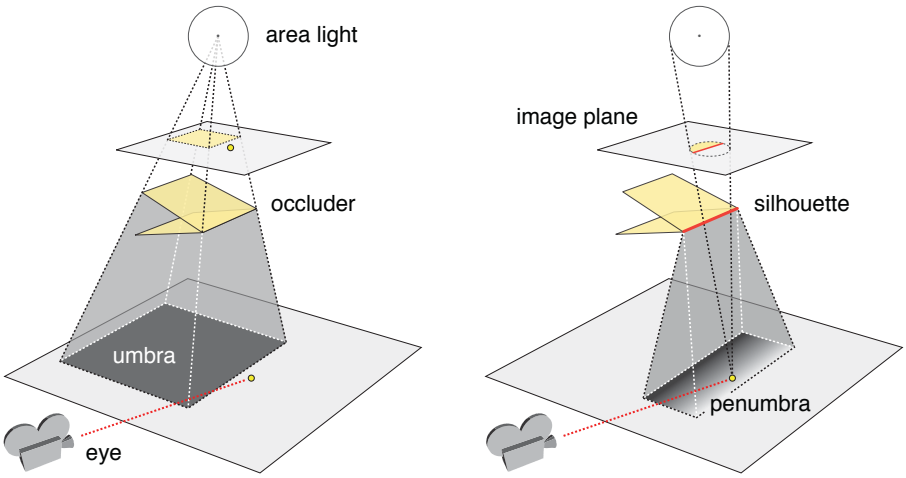
1.6 Architectural Support

The irregular Z-buffer algorithm cannot be implemented efficiently on classical hardware, due to the absence of support for construction and traversal of irregular data structures. CPUs lack the required memory bandwidth and parallelism common to GPUs. GPUs circa 2005 (when this work began) lack a small but important set of architectural features found in CPUs. For example, GPUs of this era are missing the ability to write to addresses in memory computed at run time (i.e. a scatter operation). This capability is key to the construction of irregular data structures, when the storage location of a sample point depends on the spatial relationship between the eye, light, and scene geometry, and cannot be known a priori. We show how a GPU design circa 2005 can be modified to support this and other features required for high performance construction and traversal of certain types of irregular data structures.

More recently, the feature set of commercial GPUs has reached parity with and in some cases now exceeds that of our design. The Larrabee architecture from Intel due in 2009 or 2010 is one example. We show how such a design can be used for high performance construction and traversal of a large class of memory bandwidth efficient 2D and 3D irregular data structures.

1.7 Applications

The irregular Z-buffer algorithm and architecture are motivated by applications in real-time graphics for which the optimal arrangement of sample points in the image plane is not a regular grid. Hard shadow rendering (Section 1.1) and soft shadow rendering (discussed next) are two of the most prominent examples.



(a) Computation of umbrae. (b) Computation of penumbrae.

Figure 1.6: Soft irregular shadow mapping is based on our hard shadow algorithm. Umbrae are computed as in hard irregular shadow mapping. Geometric primitives and points in the scene visible from the eye are projected into the light-view image plane and tested for overlap (a). Penumbrae are computed separately by projecting an area light and silhouette edges into the image plane and measuring the area of overlap (b). The total accumulated occlusion determines the degree of shadow at the eye-view pixel corresponding to each sample point.

1.7.1 Soft Shadows

Soft shadows more closely reproduce the qualitative properties of real-world light sources in comparison to hard shadows, further improving the realism of computer-generated images. While hard shadow *umbrae* result when a light source is fully occluded by geometry as seen from a point in the scene, soft shadow *penumbrae* occur when an area light is partially occluded as seen from the point. In this context, hard shadows are a special case of soft shadows in which the light has zero width. This observation leads to an extension of irregular shadow mapping for soft shadows. In addition to point-sampling the scene from the light (Figure 1.6a), *soft irregular shadow mapping* area-samples the scene near the silhouette edges of objects (Figure 1.6b). As in the hard shadow case, soft shadows are computed at

locations in the light-view image plane exactly as required by the position of the eye, and so *only where visible from the eye*. As we'll see, this property leads to an implementation capable of rendering soft shadows at frame rates equivalent to the fastest existing methods, with substantially higher image quality.

1.7.2 Additional Applications

The irregular Z-buffer algorithm and architecture have potential applications beyond shadow rendering. Other examples in real-time time graphics in which the desired sample pattern is similarly non-uniform include: frameless rendering, adaptive anti-aliasing, adaptive textures, and jittered sampling. More broadly, the architectural changes underlying the irregular Z-buffer algorithm have potential relevance outside graphics. For example, irregular data structures and scatter operations on memory support general-purpose computing on GPUs [24]. However, in this work we focus on applications in real-time graphics, particularly hard and soft shadow rendering.

Figure 1.7: An example of soft shadows rendered using irregular shadow mapping.

1.8 Dissertation Organization

The applicability of our overall approach is best illustrated by example. Therefore, in Chapter 2 we present a method for hard shadow rendering which utilizes the flexibility of the irregular Z-buffer algorithm to set a new balance between accuracy and efficiency, and contrast this approach with previous work. Similarly, in Chapter 3 we explore the application of the irregular Z-buffer algorithm to the problem of soft shadows, and place our strategy in the space of possible solutions. Both algorithms store points in the scene visible from the eye in a light-space spatial acceleration structure which is later queried during rasterization. We discuss the specific data structures used and associated design considerations in Chapter 4. Construction of such data data structures is inefficient on GPUs circa 2005. We examine the required architectural enhancements in Chapter 5, and analyze the performance of hard and soft shadow rendering in the presence of these enhancements in Chapter 6. Finally, we review key points and summarize the larger role of hybrid rendering algorithms such as ours in real-time graphics in Chapter 7.

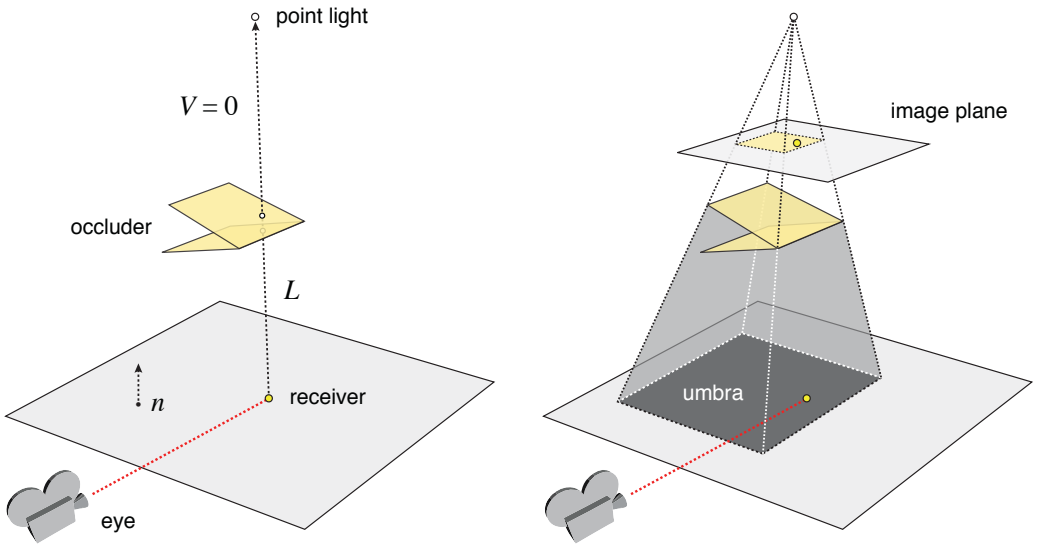
Chapter 2

Hard Shadows

The computation of hard shadows involves determining the irradiance from a diffuse point light which reaches each location in the scene visible from the eye (Figure 2.1). More formally, the irradiance E from a diffuse point light incident on a receiver with normal $\vec{n}$, is given by Equation 2.1. The term Φ is the intensity of the light, and the vector L extends from the receiver point to the light. For opaque surfaces the visibility term V is a binary value. It is 1 when the light is visible from the receiver point and 0 otherwise. Note that *occlusion* from the light (denoted $\tilde{V}$) is the dual of V and can be represented as simply $1 - V$.

$$E = (\hat{n} \cdot \hat{L}) \frac{\Phi}{|L|^2} V \tag{2.1}$$

Note that the dot product and the distance attenuation term $\Phi/|L|^2$ are simple to compute since $\vec{n}$ is readily available, L depends only on the position of the receiver and the light, and Φ is a constant. The visibility term is comparatively expensive as it requires a search over the scene geometry for occluders and must be recomputed for each receiver point.



(a)
A point light is not visible from a receiver point.

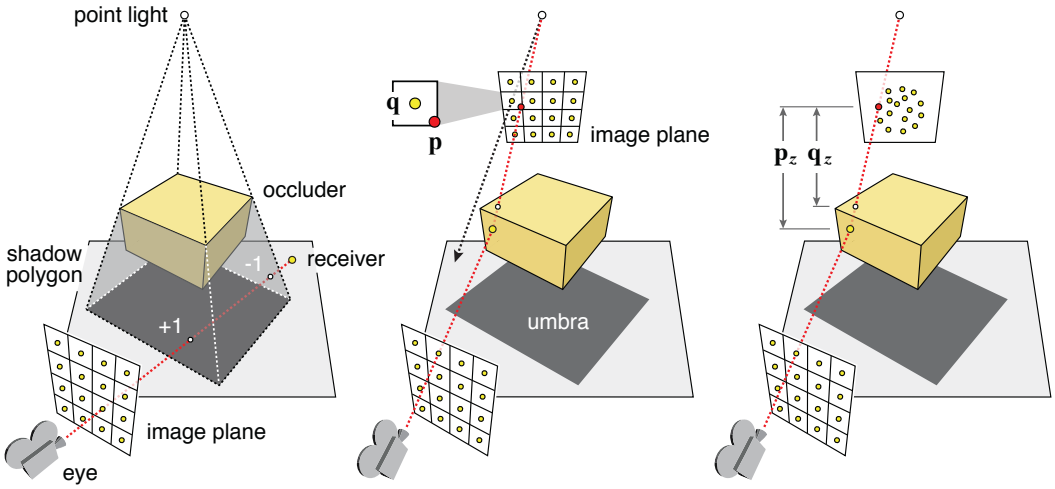
(b)
The shadow umbra cast by the occluder is shown.

Figure 2.1: The geometry of hard shadows. A point light source is not visible from a point in the scene seen from the eye ("receiver") due to occluding geometry (a). As a result, the point lies inside the umbra cast by the occluder (b). The determination of shadow at the receiver point can be found in 2D by projecting the occluder and the receiver into the light-view image plane, and testing the projected point against the footprint of the occluder.

2.1 Methodology

Conceptually, there are many possible strategies for computing V. However, not all of these methods are viable in an object-order (e.g. Z-buffer) system, where triangles are sequentially processed and discarded. In such a system the visibility of a given triangle is determined for all receiver points before moving onto the next triangle[1]. Doing so requires that the positions of the receiver points be known a priori.

[1] In contrast, image-order algorithms such as ray tracing typically determine the visibility of a triangle from a given point (for all triangles) before moving onto the next point.



(a) Shadow geometry. (b) Classical shadow mapping. (c) Irregular shadow mapping.

Figure 2.2: Three methods for rendering hard shadows are compared. Occlusion from the light can be determined in eye space using shadow geometry (a). This method yields accurate umbrae, but the number and extents of shadow polygons results in high depth complexity. Alternatively, occlusion can be rendered in light space to a shadow map which is later resampled in eye space (b). This method can achieve high performance, but is prone to artifacts due to a mismatch in the eye and light-view sample patterns. Finally, occlusion can be determined entirely in light space by storing receiver points in a light-view acceleration structure which is queried during the visibility computation (c). In principle, this method combines the image quality of (a) with the performance of (b).

Under this constraint, there are effectively three methods for finding V that vary by the coordinate system (eye, light, or both) in which visibility is determined. Occlusion from the light can be measured in eye space at the receiver points using "shadow geometry" as seen in Figure 2.2a. This method yields high-quality umbrae. However, the shadow polygons are often large when viewed from the eye, extending from an occluder to the far clipping plane of the light. This property leads to high average depth complexity from the eye and comparatively poor performance (Subsection 2.5.1). Alternatively, occlusion from the light can be rendered in light space to a "shadow map" which is later resampled in eye space at the receiver

point positions, as seen in Figure 2.2b. This method often performs well but yields lower quality umbrae due to a mismatch in the eye and light-view sample patterns (Subsection 2.5.2). Finally, occlusion from the light can be determined entirely in light space by transforming receiver points into the light view and storing them in a spatial acceleration structure which is queried during the visibility computation, as seen in Figure 2.2c. This method is a variation of shadow mapping in which the eye and light-view sample patterns are aligned, and in principle combines the umbral quality of shadow geometry with the performance of shadow mapping. Such algorithms have not been well explored previously since the architectural support necessary for building irregular data structures per frame in real-time (Chapter 5) has only recently become available.

2.2 Contribution

The comparative value of a new shadow rendering algorithm can be quantified by relating the image quality and performance to existing methods. Irregular shadow mapping achieves a new balance between these metrics. As we'll see in Chapter 6, the algorithm *yields shadow umbrae of the highest possible quality with performance superior to existing methods of comparable quality.* Our novel strategy of precisely aligning the eye and light sample patterns avoids the two most significant sources of artifacts common to existing shadow mapping methods. Irregular shadow mapping fully eliminates perspective aliasing (sawtooth patterns at umbral edges), as well as projection aliasing (incorrect self-shadowing). Further, this alignment results in computational efficiency. Shadows are computed only where visible from the eye rather than throughout the scene, yielding high performance.

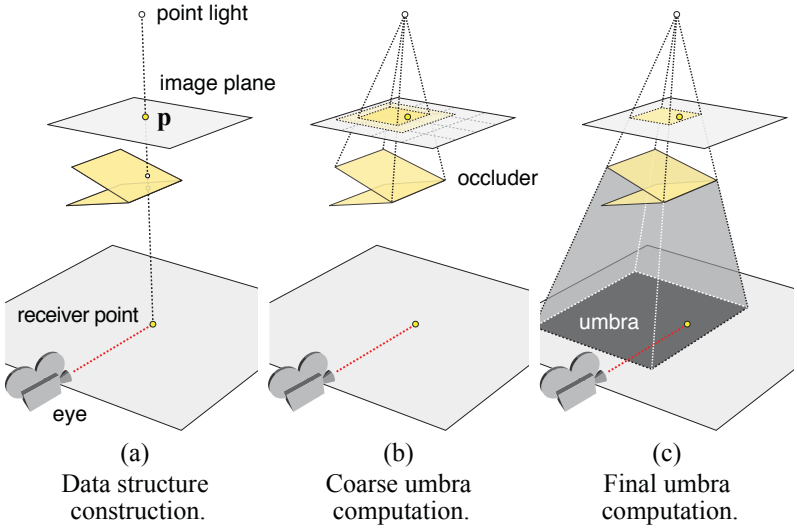


Figure 2.3: The steps of our hard shadow algorithm are illustrated geometrically. The scene is rendered from the eye, and the visible (i.e. "receiver") points are inserted into a light-space spatial acceleration structure (a). Umbrae are computed in two steps. The first rasterizes scene geometry from the light (b). For each point $\mathbf{p}$ located in a light-view pixel overlapped by a primitive, a more accurate visibility test is performed (c). The result of this test determines if the eye-view pixel corresponding to the receiver point is in shadow.

2.3 Algorithm

Hard irregular shadow mapping is illustrated geometrically in Figure 2.3. Receiver points are identified by rasterizing scene geometry from the eye (a). These points are transformed into light space and inserted into a spatial acceleration structure (Section 4.2). Umbral occlusion is computed by rasterizing scene geometry from the light to coordinates $\mathbf{p}$ in the image plane given by the receiver points. This process occurs in two steps. A coarse visibility test is used to estimate the set of points occluded by a given primitive (b). For each point $\mathbf{p}$ within an occluded pixel, a final visibility test is performed (c). The result determines if the eye-view pixel corresponding to the receiver point is in an umbra cast by the primitive.

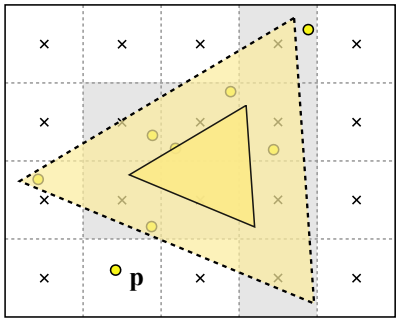


Figure 2.4: The spatial data structures introduced in Section 4.2 organize receiver points into light-view screen-space pixels. As a result, the set of points affected by a given primitive can be estimated using classical rasterization. However, rasterizers typically evaluate the visibility of a primitive at the pixel center. To ensure *every* pixel overlapped by a primitive is considered, the edges of the primitive are moved outward by half the width of a pixel (Equation 2.2) prior to rasterization. Umbral occlusion is computed using the original unexpanded primitive, for receiver points located in pixels (shown in gray) covered by the expanded primitive.

2.3.1 Primitive Expansion

Identifying the set of receiver points occluded by a given surface is the fundamental operation in any hard shadow algorithm, but testing each point against each surface is inefficient: $\mathrm{O}(N * P)$ for N primitives and P points. Note that identifying the set of points affected by a given primitive is an instance of the visible surface problem, and can in principle be addressed with any surface visibility solution. In practice, the hardware support for irregular shadow mapping derives from commodity GPUs (Chapter 5), and these architectures favor the classical Z-buffer algorithm. For this reason, we estimate the set of receiver points occluded by a given primitive using classical rasterization as described below. Umbral occlusion is then computed only at the receiver points in pixels overlapped by a primitive. Doing so raises the overall efficiency of our hard shadow algorithm to $max(\mathrm{O}(N_{pixel} * P_{pixel}), \mathrm{O}(N_{point} * P_{point}))$, where N_{pixel} and N_{point} are the average number of primitives occluding a pixel or point respectively, for P_{pixel} pixels and P_{point} points.

$$\Delta = \frac{\hat{n}_x + \hat{n}_y}{2} * \mathbf{W}_{pixel} \tag{2.2}$$

As we'll see in Chapter 4, the spatial acceleration structures used in irregular shadow mapping are grid based. These structures sort receiver points into pixels in the light-view image plane, enabling range queries to be performed via classical rasterization. However, rasterizers often evaluate the visibility of a primitive at the pixel center. A primitive which does not cover this point is not "visible" even when the primitive and pixel partially overlap. In such cases, receiver points in the pixel are not tested for occlusion against the primitive. For this reason, the edges of primitives are moved outward prior to rasterization, by half the width of a pixel weighted by the edge slope (Equation 2.2 where $\hat{n}$ is the edge normal). Umbral occlusion is computed using the *unexpanded* primitive, only at receiver points located in pixels covered by the *expanded* primitive as seen in Figure 2.4.

2.3.2 Umbral Occlusion

For a given primitive and receiver point, the computation of umbral occlusion is straightforward. This calculation is illustrated in Equation 2.3, and is geometrically equivalent to tracing a shadow ray with the receiver point as its origin. Unlike ray tracing, intersection testing is performed in 2D. The primitive and receiver point are projected into the light-view image plane and examined for overlap via the same 2D point-in-polygon operation used during classical rasterization. The distance from the light to the unprojected primitive is interpolated at the point of intersection ($\mathbf{q}$) [101]. If the result is less than the depth of the receiver point ($\mathbf{p}_z$), then the receiver point is occluded by the primitive.

$$V = 1 - \bigcup_{i=1}^{N} \tilde{V}'(object_i, \mathbf{p}) \ni \tilde{V}'(object_i, \mathbf{p}) = \begin{cases} 0, \text{ if } \mathbf{q}_z \geq \mathbf{p}_z \text{ for } \mathbf{q} \text{ on } object_i \\ 1, \text{ if } \mathbf{q}_z < \mathbf{p}_z \text{ for } \mathbf{q} \text{ on } object_i \end{cases} \tag{2.3}$$

2.4 Optimizations

Irregular shadow mapping is compatible with several existing optimizations from ray tracing and conventional shadow mapping. We summarize three here. Two of these optimizations reduce the work in scenes with opaque geometry, by terminating the computation of umbral occlusion at a given receiver point once it is known to be in shadow. The remaining optimization relates to image quality and addresses a potential source of aliasing not directly resolved by our algorithm: aliasing due to limited numerical precision.

2.4.1 Early Termination

In Equation 2.3, occlusion from multiple objects is combined via the union operator, and $\tilde{V}'$ is a binary function. In other words, the umbral occlusion at a receiver point given opaque geometry is independent of the number of occluding objects and is not specific to a particular occluder. Once the point is found to lie in shadow, no further computation need be performed. This observation leads to early termination of shadow rays in ray tracers, and is equally applicable here. In our algorithm, this optimization is implemented by removing the receiver point from the spatial acceleration structure after it is determined to be in shadow, or by marking the point with a value denoting its state.

2.4.2 Reducing the Number of Receiver Points

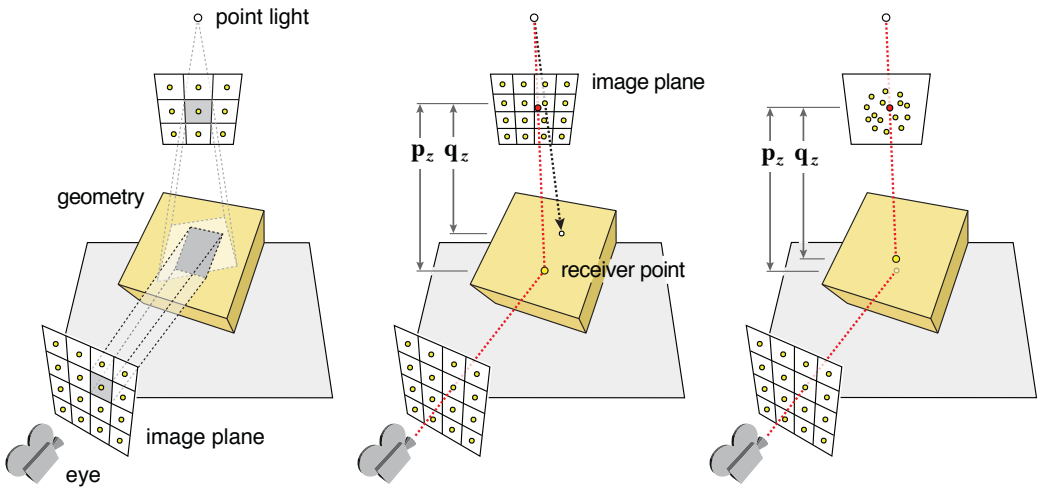
Similarly, Arvo has observed that umbral occlusion need not be computed at receiver points on opaque surfaces facing away from the light [11]. Such a point *must* lie inside the umbra cast by the surface, since the surface overlaps the point in the light-view image plane and is nearer the light (Subsection 2.3.2). In irregular shadow mapping, the surface normal, light position, and receiver points are known in advance of data structure construction. As a result, points sitting on surfaces facing away from the light can be discarded rather than inserted into the spatial acceleration structure.

2.4.3 Back Face Rendering and Depth Bias

The visual artifacts associated with conventional shadow mapping represent three distinct types of aliasing. Of these, irregular shadow mapping addresses perspective and projection aliasing [123], but not aliasing due to limited numerical precision. *Perspective* aliasing derives from the spatial relationship between the eye and light relative to a given surface. It occurs when the area of a shadow map texel projected onto the surface is greater than the projected area of an eye-view pixel (Figure 2.5a), and appears as a sawtooth pattern at umbral boundaries (Figure 1.2c). *Projection* aliasing is due to the orientation of a surface relative to the light. It occurs when a surface is nearly parallel to the light. In such cases, even a slight variance in the position of points $\mathbf{p}$ and $\mathbf{q}$ in the light-view image plane yields a large difference in the depth of the surface as seen from the light (Figure 2.5b), and can lead to self-shadowing (e.g. a front facing triangle casts a shadow onto itself). By precisely aligning the eye and light sample patterns (Figure 2.2c), irregular shadow mapping avoids both perspective and projection aliasing[2].

Perspective and projection aliasing result from the misalignment of points in the X and Y dimensions of light space. In contrast, aliasing in Z occurs primarily due to limited numerical precision in the underlying graphics hardware. As such, irregular shadow mapping cannot directly resolve this source of artifacts. Here, the light-view depth of a receiver point measured using two different methods produces two different values, leading to self-shadowing. Consider Figure 2.5c. The depth of a receiver point from the light as seen from the eye ($\mathbf{p}_z$) does not match the depth as seen from the light ($\mathbf{q}_z$). The former is computed using a transformation matrix while the latter is computed by interpolation. If the variance is such that $\mathbf{q}_z < \mathbf{p}_z$ then the receiver point is incorrectly found to be occluded by the surface on which

[2]In principle, limited numerical precision within the light-view image plane can still result in projection aliasing for surfaces parallel to the light. In practice, this problem is avoided by treating such surfaces as back facing.



(a) Perspective aliasing. (b) Projection aliasing. (c) Numerical precision aliasing.

Figure 2.5: The visual artifacts associated with classical shadow mapping represent three types of aliasing. *Perspective* aliasing causes a sawtooth pattern at the edges of umbrae and is due to undersampling from the light (a). *Projection* aliasing produces self-shadowing and is due to a discrepancy in depth between the receiver point ($\mathbf{p}$) and the nearest point in the shadow map ($\mathbf{q}$) even when both points lie on the same surface (b). Aliasing can also result from limited numerical precision in the host graphics hardware (c). This aliasing also produces self-shadowing, but is due to a slight variance in light-view depth between the receiver point as seen from the eye ($\mathbf{p}$) and from the light ($\mathbf{q}$). Irregular shadow mapping eliminates perspective and projection aliasing, but not aliasing due to limited numerical precision.

it sits. The incidence of these self-shadowing artifacts can be reduced by rendering back facing surfaces from the light rather than front facing [136], and discarding receiver points which lie on back facing surfaces as in Subsection 2.4.2. In tandem, these simple optimizations ensure that the computation of umbral occlusion is never performed between a receiver point and the surface on which it sits. Aliasing is still possible in the case of thin surfaces, but can be addressed using a small depth bias.

2.5 Further Related Work

Section 2.1 describes a taxonomy of methods for detecting occlusion from the light, based on the coordinate system in which the computation is performed. Existing object-order (i.e. Z-buffer) hard shadow algorithms can be classified according to this taxonomy, highlighting high-level differences in operation, image quality, and performance. To review, occlusion from the light can be measured in eye space at the receiver points using "shadow geometry". Alternatively, occlusion from the light can be rendered in light space to a "shadow map" which is later resampled in eye space at the receiver point positions. Finally, occlusion from the light can be determined entirely in light space by storing receiver points in a light-view spatial acceleration structure which is queried during the visibility computation. Existing algorithms occupy the first two categories (Subsection 2.5.1 and 2.5.2), while our algorithm (and a related method from Aila et al. [6] and Arvo [11]) forms the third.

2.5.1 Shadow Geometry

Shadow geometry delimits regions of the scene within the umbrae cast by occluding objects. This geometry typically takes the form of polygons which extend from the silhouette edges of an occluder to the far light-view clip plane, as seen in Figure 2.2a [34]. A receiver point that lies inside the volume of space defined by these polygons is occluded from the light. More specifically, occlusion can be determined by counting the number of entry and exit points along a ray from the eye to a receiver point, as it passes through one or more shadow volumes. If the counter is initialized to 0, and is incremented or decremented per entry or exit respectively, the final value will be > 0 for all points in the scene in shadow. On a modern GPU, this computation occurs by rasterizing shadow geometry from the eye into a stencil buffer [42]. The stencil value for a given pixel is modified according to the orientation of each polygon seen from the pixel. It is incremented for polygons facing the eye and decremented otherwise. A final value > 0 indicates that the corresponding pixel is in shadow.

Representing umbral bounds as geometric primitives reduces the problem of identifying regions of the scene in shadow to a straightforward visible surface computation. This calculation has low error[3] on modern GPUs and so results in high quality hard shadows. However, shadow polygons are frequently large as seen from the eye (Figure 2.2a), particularly those from occluding objects near the light. Further, the number of shadow polygons can be large, and is equal to the number of silhouette edges for all objects visible from the light. Together, these properties lead to high average depth complexity from the eye. As a result, shadow geometry algorithms often require rasterization hardware capable of high fill rates.

Efforts to minimize this fill rate pressure focus on two strategies. The first reduces the number and eye-view extents of shadow primitives through aggressive clipping and culling [89] as well as careful selection of the light-view depth bounds [95]. The second rasterizes shadow geometry only to eye-view pixels corresponding to receiver points known to be near a shadow silhouette [5, 29]. Even with these optimizations and recent hardware specializations [99] the geometric scene complexity must be held below what would be possible in the absence of any shadow primitives.

2.5.2 Resampling / Filtering Approximate Visibility

Alternatively, occlusion from the light can be sampled in light space and stored into a "shadow map" that is later resampled in eye space from the receiver points. Classical shadow mapping [139] is illustrated in Figure 2.2b. Scene geometry is first rendered from the light (giving $\mathbf{q}_z$) and then from the eye. The distance between each receiver point and the light-view image plane is computed (giving $\mathbf{p}_z$). Point $\mathbf{p}$ is determined to be in shadow according to the inequality seen in Equation 2.3. The value of $\mathbf{q}_z$ used in this comparison is estimated from the values for the light-view samples ($\mathbf{q}$) nearest $\mathbf{p}$ in the image plane.

[3]The visible surface calculation is accurate to the numerical precision of the depth buffer, at the point within a pixel where visibility is determined. Sub-pixel accuracy is possible with supersample anti-aliasing.

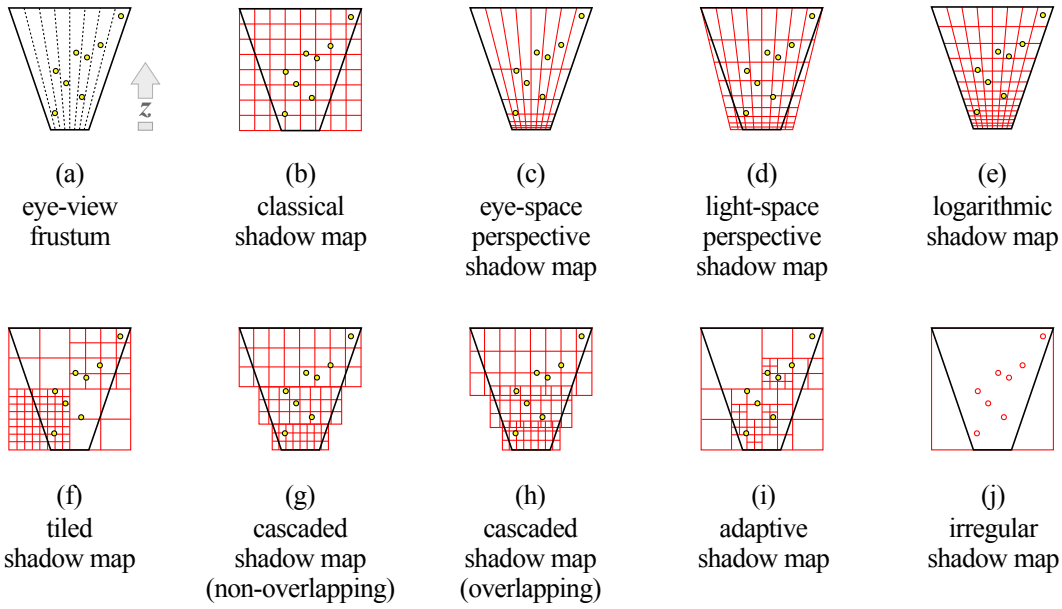


Figure 2.6: A figure extended from Lloyd et al. [86] illustrating several shadow map based algorithms. The trapezoid indicates the bounds of the eye-view frustum as seen from above by a directional light. Dots denote points in the scene visible from the eye (receiver points). Observe that the eye-view sampling rate decreases with depth (a). A classical shadow map undersamples the scene near the eye (b) yielding a large estimation error. Warping methods reduce this estimation error by fitting the shadow map to the eye-view bounds using perspective (c - d) and / or logarithmic (e) parameterizations. Partitioning methods (f - i) reduce the estimation error by replacing the single shadow map with multiple sub-maps sampled at different rates, each determined by the local eye-view sampling rate. Irregular shadow mapping (our method) samples the scene *exactly* at the light-view locations given by the receiver points (j). This avoids estimation errors and oversampling.

Shadow mapping is generally considered capable of higher performance than shadow geometry methods. The former finds umbral occlusion by rasterizing *scene geometry* from the light, while the latter rasterizes *shadow geometry* from the eye. Rasterization performance is inversely proportional to the average depth complexity of the geometry seen from a view point, and this measure is commonly higher for shadow polygons visible from the eye (Subsection 2.5.1) than for scene primitives visible from the light. Further, shadow mapping avoids the need to create, clip, and

cull shadow polygons. However, error from the estimation of $\mathbf{q}_z$ leads to aliasing and self-shadowing artifacts. The magnitude of this error corresponds to the distance from $\mathbf{q}$ to $\mathbf{p}$, and is generally unbounded. As a result, the number and severity of artifacts can be high. *Even a small error can yield a large bias in intensity as seen in* Figure 2.2b, *where the indicated pixel is incorrectly found to be lit.*

Methods for reducing this estimation error (and so minimize the number and severity of artifacts) typically follow one of three approaches. The first utilizes silhouette information from occluding objects to more accurately identify umbral boundaries in a conventional shadow map [119]. To do so, a separate "silhouette map" encodes a piece-wise linear representation of the occluder silhouette. Each receiver point ($\mathbf{p}$) is projected into the silhouette map and into the shadow map. The position of $\mathbf{p}$ relative to the silhouette determines which of the four nearest shadow map samples ($\mathbf{q}$) will be used in the inequality in Equation 2.3. This strategy reduces the incidence of estimation artifacts, but high frequency umbral details may be truncated or lost due to limited precision in the silhouette contours.

The second approach warps the shadow map to better align the light-view sample pattern (points $\mathbf{q}$) with the distribution of receiver points (points $\mathbf{p}$). These methods reduce, but do not eliminate estimation error and the associated artifacts. In effect, warping shifts shadow map resolution from oversampled regions of the scene to undersampled areas. It has been shown [141, 90, 87] that the parameterization which produces the smallest average estimation error in a grid-based shadow map is logarithmic (Figure 2.6e). However, current graphics hardware lacks support for fast evaluation of log-based transforms. Instead, warping methods for existing GPUs [123, 93, 141, 90] utilize parameterizations based on perspective transforms (Figure 2.6c - d). In general, as the shadow map resolution approaches infinity, the image quality of warping methods converges to that of a ray tracer, but for practical resolutions some estimation error remains [87]. Further, these parameterizations are view dependent. Camera motion relative to the light changes the estimation error. Without correction [87], these changes lead to temporal artifacts (e.g. flickering).

The third approach divides the light-view shadow map into regions sampled at different rates based on the position of the camera or scene geometry relative to the light. Partitioning can be achieved by splitting the shadow map into tiles [10], using multiple adjacent [41] or overlapping [127] shadow maps, or by constructing an adaptive hierarchy of shadow maps of increasing resolution [44, 79, 80]. These methods are illustrated in Figure 2.6 (f - i). Warping-partitioning combinations have also been proposed [31, 73, 90] and are examined in detail by Lloyd et al. [88]. As with warping, the image quality of partitioning algorithms converges to that of a ray tracer in the limit, but at the cost of substantially more light-view samples and proportionately greater computation and memory use.

2.5.3 Closely Related Work

Concurrently with this work, Aila and Laine [6] and Arvo [11] have proposed similar algorithms, specifically for the purpose of rendering hard shadows on conventional hardware, but neither is real-time. Like our algorithm, these methods store receiver points in an explicit spatial acceleration structure (k-d tree [6] and grid [11]), which is later queried during the occlusion computation. Our work is distinct from these efforts in four ways. First, we consider the irregular Z-buffer algorithm as a general solution to the visible surface problem for which shadow rendering is but one application. Second, we show how the irregular Z-buffer algorithm can be used to render high-quality hard *and* soft shadows. Third, we identify architectural enhancements to existing GPUs to permit efficient sampling at arbitrary points in the image plane. Fourth, through detailed simulation and analysis, we show how these changes lead to high performance when rendering hard and soft shadows in scenes from modern computer games.

2.6 Summary

Real-time hard shadow rendering has been well explored. Even so, a solution which yields high image quality *and* good performance in dynamic scenes from modern games, has proven elusive. Shadow geometry algorithms can render umbrae comparable to those from a ray tracer, and so have been used[4] in mainstream games like Doom 3 [62]. However, the performance of these methods is inhibited by an intrinsic inefficiency. Some (potentially many) receiver points will not be in shadow even though the corresponding pixels are covered by shadow polygons. This case is illustrated in Figure 2.2a and is common where the scene geometry onto which a shadow is cast is nearly parallel to the eye look direction. In contrast, conventional shadow mapping algorithms can achieve high performance, but are prone to artifacts arising from misalignment of the eye and light view sample patterns (Figure 2.2b). This mismatch is fundamental and in practice cannot be fully resolved by warping and / or partitioning the shadow map (Figure 2.6b - i).

Irregular shadow mapping (our algorithm) precisely aligns the eye and light sampling patterns as seen in Figure 2.2c. Receiver points are stored explicitly in a light-view spatial acceleration structure. This permits occlusion to be computed entirely in light space with high accuracy, and in principle combines the umbral quality of shadow geometry with the performance of classical shadow mapping. This method does incur some overhead from the per-frame construction of a spatial acceleration structure. We examine this overhead and the overall performance and image quality of the algorithm in Chapter 6.

[4]The use of shadow geometry in games is often restricted to occluders which are highly dynamic (e.g. deformable characters). Shadows from other sources are precomputed. Further, only a subset of the lights in a given scene typically cast shadows.

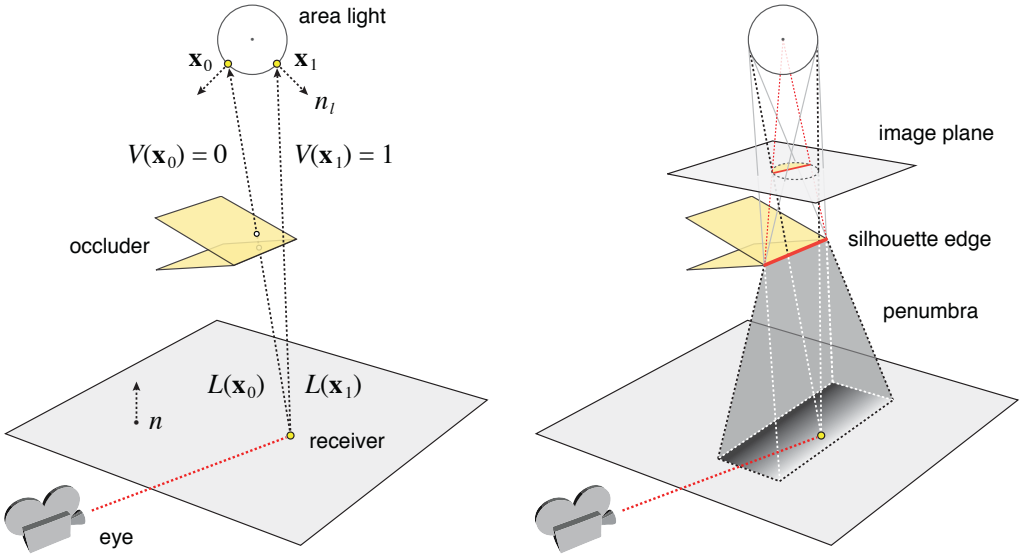
Chapter 3

Soft Shadows

Soft shadows are a generalization of the light transport problem for hard shadows in which the light source is of arbitrary size and shape. More formally, the irradiance E from a diffuse area light incident on a receiver point with normal $\vec{n}$ (Figure 3.1a) can be expressed as an integral over the area A of the light. In Equation 3.1, the term $\Phi(\mathbf{x})$ is the intensity of the light at point $\mathbf{x}$. The shape of the light determines both $\hat{n}_l(\mathbf{x})$ and the domain of integration A. Vector $L(\mathbf{x})$ extends from the receiver point to $\mathbf{x}$. For opaque surfaces the visibility term $V(\mathbf{x})$ is a binary value. It is 1 when $\mathbf{x}$ is visible from the receiver point and 0 otherwise.

$$E = \int_{\mathbf{x} \in A} (\hat{n} \cdot \hat{L}(\mathbf{x}))(\hat{n}_l(\mathbf{x}) \cdot - \hat{L}(\mathbf{x})) \frac{\Phi(\mathbf{x})}{|L(\mathbf{x})|^2} V(\mathbf{x}) d\mathbf{x} \tag{3.1}$$

[†]The first two paragraphs of Chapter 3, and the text of Section 3.5 and 3.6 are edited from material written by Chris Burns and used with permission. Table 3.1 and the taxonomy around which Section 3.6 is organized, were co-developed by Chris Burns and Greg Johnson.



(a)
A light is partially visible from a receiver point.

(b)
The shadow penumbra cast by a silhouette edge of the occluder is shown.

Figure 3.1: The geometry of soft shadows. A spherical light source is only partially visible from a point in the scene seen from the eye ("receiver") due to occluding geometry (a). As a result, the point lies inside a penumbra cast by the occluder (b). The shadow umbra is not shown. Given a light source of uniform intensity and no distance attenuation, the degree of shadow at the receiver point can be computed in 2D by projecting the light and the occluder into the light-view image plane, and measuring the area of the light footprint covered by the occluder.

Many common light sources have nearly constant values for $\hat{n}_l$ as well as Φ. Further, if the light is small (also common) then L is nearly constant, and both of the dot products as well as the distance term $1/|L(\mathbf{x})|^2$ can be moved outside the integral (Equation 3.2), leaving only the visibility term $V(\mathbf{x})$. This term is largely responsible for the visual quality of shadow penumbrae, but is also the most difficult to compute, requiring a search over the scene geometry to identify occluders.

$$E \approx (\hat{n} \cdot \hat{L})(\hat{n}_l \cdot -\hat{L}) \frac{\Phi}{|L|^2} \int_{\mathbf{x} \in A} V(\mathbf{x}) d\mathbf{x} \tag{3.2}$$

3.1 Methodology

Observe the similarity in the irradiance equations for diffuse point (Equation 2.1) and area (Equation 3.2) lights. The latter evaluates the integral of the visibility function V over the surface of the light, *but the semantics of V itself are the same in both equations.* As a result, the design of our soft shadow algorithm follows the same methodology outlined in Section 2.1. Receiver points are stored in a light-view spatial acceleration structure, enabling penumbral occlusion to be computed entirely in light space with comparative accuracy and efficiency.

3.2 Contribution

Like hard irregular shadow mapping, our soft shadow algorithm occupies a unique position in the space of possible solutions defined by image quality and performance. Specifically, soft irregular shadow mapping *achieves frame rates comparable to the best performing existing methods, but produces substantially higher image quality* (Chapter 6). The image quality of this algorithm is commonly indistinguishable from that of physically-accurate (but much lower performance) methods such as beam tracing. As in the hard shadow case, this novel combination of quality and performance results from the precise alignment of eye and light sample patterns (Figure 2.2c). But this alignment also has a third benefit: *robust image quality.* Unlike existing algorithms based on shadow mapping, ours requires no per-frame parameter tuning to achieve high quality. Like other soft shadow algorithms in the real-time performance regime, ours is approximate. Two geometric approximations are used to simplify the computation of penumbral occlusion, but neither introduces high frequency spatial or temporal artifacts.

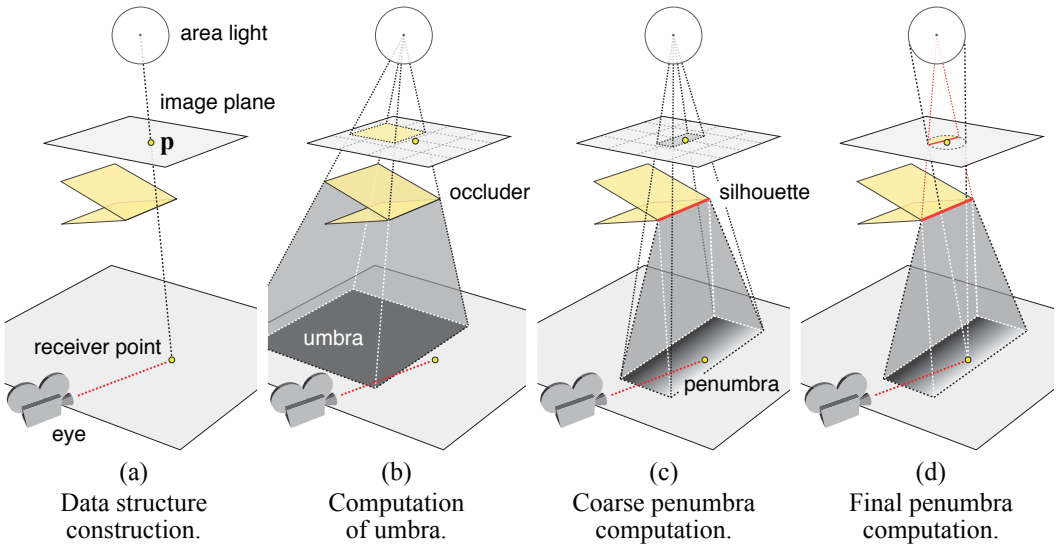


Figure 3.2: The four steps of our soft shadow algorithm are illustrated geometrically. First, the scene is rendered from the eye. Visible points are transformed into light space (a) and inserted into an irregular spatial acceleration structure. Umbrae are determined by rasterizing scene primitives into the light-view (b) and computing a point-in-triangle test at points **p**. Penumbrae are computed in two steps. The first rasterizes a set of quads representing the estimated light-view screen-space bounds of the penumbra cast by each silhouette edge (c). For each point overlapped by a quad, a more accurate visibility test is performed. Here, the silhouette edge and adjacent surface are clipped to a circle centered on the point (d) and the normalized, fractional area of occlusion is measured. Finally, the umbral and penumbral occlusion is combined and used to determine the degree of shadow at the eye-view pixel corresponding to each receiver point.

3.3 Algorithm

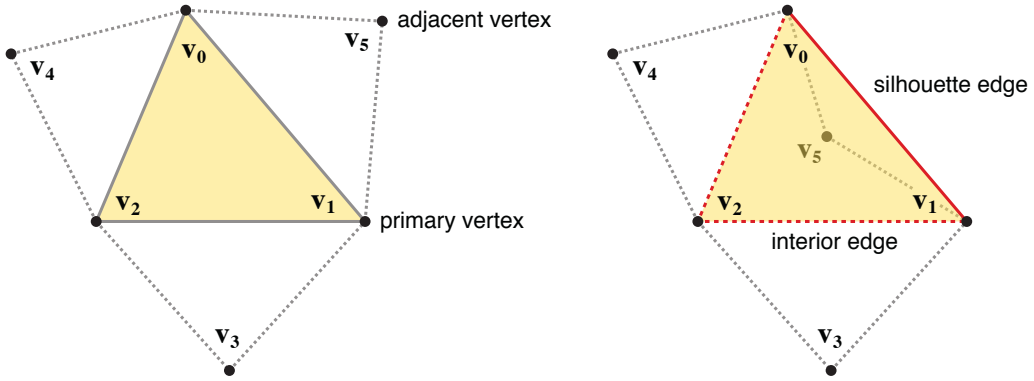
Conceptually, the computation of penumbral occlusion consists of determining the area of the light occluded by geometry as seen from a receiver point. For a light of uniform intensity, the degree of occlusion can be found in 2D by projecting the light and the occluder into the light-view image plane and measuring the area of overlap. Our algorithm does exactly this, and combines the result with umbral occlusion computed via hard irregular shadow mapping.

The full algorithm is seen in Figure 3.2. For clarity, it is shown as a sequence of four steps, but shadow umbrae and penumbrae can be computed concurrently. Receiver points are identified by rasterizing scene geometry from the eye (a). These points are transformed into light space and inserted into a spatial data structure, as in Section 4.2. Umbral occlusion is computed (b) by rasterizing scene geometry from the light to coordinates $\mathbf{p}$ in the image plane given by the receiver points, as in Section 2.3. Penumbrae are comparatively difficult to compute. As a result, a coarse visibility test is performed to estimate the set of receiver points affected by a given silhouette edge (c). This test consists of rasterizing shadow geometry from the light, where each primitive represents the expected screen-space extents of the penumbra cast by a silhouette edge (Subsection 3.3.1 and 3.3.2). The computation of penumbral occlusion is performed only at points $\mathbf{p}$ covered by a shadow primitive (d). Here, the surface adjacent to the silhouette edge is clipped against the footprint of the light in 2D within the light-view image plane (Subsection 3.3.3). Finally, the umbral and penumbral occlusion accumulated at a receiver point (Subsection 3.3.4), is used to modulate the intensity of the corresponding eye-view pixel.

3.3.1 Silhouette Edge Detection

An important property of physically-correct soft shadows is that penumbrae are cast *only at the silhouette edges of occluder geometry.* The size and shape of a penumbra is defined by the position and orientation of the silhouette edge relative to the light and receiving geometry, not by primitives in the interior of the occluder. This property can be exploited to improve the performance of soft shadow algorithms by localizing computation of the visibility integral (Equation 3.2) to receiver points known to be near silhouette edges.

For manifold (i.e. closed) geometry, a silhouette edge is defined as an edge shared by two faces of an occluder such that one face is oriented toward the light and one away from the light. For non-manifold geometry, an edge adjacent to a single



(a) Primary versus adjacent vertices.

(b) A silhouette edge joins two opposing faces.

Figure 3.3: Shadow penumbrae are cast only at the silhouette edges of an object. A silhouette edge is commonly defined as an edge shared by two faces of an occluder such that one face is oriented toward the light and one away. Vertex adjacency information (a) is accessible through modern graphics APIs like DirectX 10 [19]. This information can be used to determine object silhouettes by examining the light-view winding order of the primary and adjacent vertices (b).

face is also considered a silhouette. The identification of silhouette edges in either case is straightforward on modern graphics hardware[1]. For example, the DirectX 10 API [19] exposes vertex adjacency information (Figure 3.3a). This information can be used within a geometry shader to determine object silhouettes in light space by examining the winding order of the vertices composing the primary and adjacent faces (Figure 3.3b), or in eye space by similarly inexpensive means. Note that the identification of silhouettes improves the performance of our algorithm but *is not necessary for correctness*. In the absence of adjacency information all edges are assumed to be silhouettes. The composition of umbral and penumbral occlusion described in Subsection 3.3.4 ensures correctness is preserved even in the case of interior edges (Figure 3.4).

[1]The given definition is incomplete as it does not guarantee that a silhouette edge is visible to the light. Edges identified as silhouettes but which are not visible to the light do not cast penumbrae. Such edges result in unnecessary computation in many soft shadow algorithms (ours included). However, the visibility of a given edge cannot be accurately ascertained with high performance due to parallax (Subsection 3.5.1).

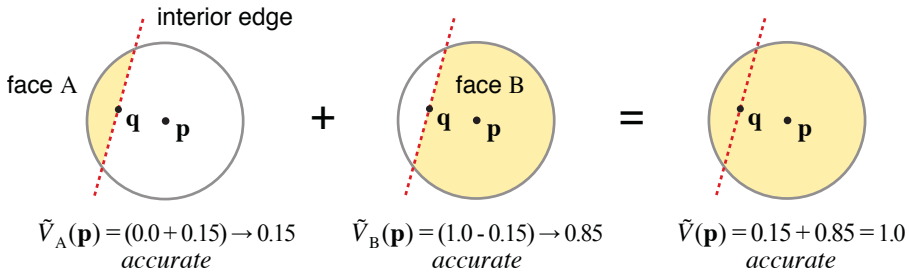


Figure 3.4: Our algorithm area-samples geometry only near silhouette edges. This strategy is necessary for performance but not correctness. Area-sampling interior geometry introduces no error as seen here. The radii of the samples to which faces A and B are clipped are equal, since this value is derived from the light-view depth at the point on the shared edge ($\mathbf{q}$) nearest $\mathbf{p}$. This computation is explained further in Subsection 3.3.3.

3.3.2 Silhouette Edge Geometry

Once identified, the silhouette edges are used in a two-part penumbral computation. As in other image-space soft shadow algorithms, a coarse visibility solution is used to conservatively estimate the set of silhouette edges affecting a given receiver point. A final solution is then computed for the point from this edge set. Here, coarse visibility is determined from a set of geometric primitives representing the extents of penumbrae as seen from the light (Figure 3.2c), while final visibility is computed directly from the original scene geometry (Figure 3.2d).

Observe that the penumbra cast by a silhouette edge forms a wedge (Figure 3.1b), and that the projection of this wedge into the light-view image plane can be represented as a rectangle[2]. As a result, the shadow primitives used during the coarse visibility test in our algorithm are rectangular and coplanar to the light-view image plane. The width of a primitive (Figure 3.5) depends on the light width,

[2]This assumes a single point of projection (Subsection 3.5.1). Further, for a spherical light source the projection of this wedge is a rectangle with hemicircular end caps, but can be conservatively approximated with a larger rectangle.

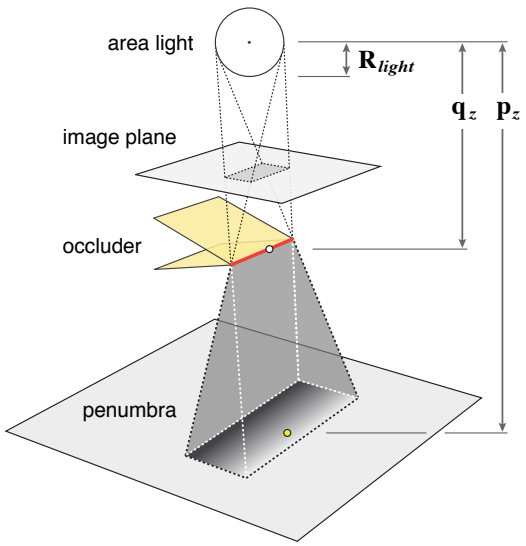


Figure 3.5: The width of a given shadow primitive in the light-view image plane is proportional to the light width ($\mathbf{R}_{light}$), the depth of the silhouette edge ($\mathbf{q}_z$), and the maximum depth of any receiver point ($\mathbf{p}_z$) occluded by the surface adjacent to the edge, as described in Equation 3.3 [105].

the minimum depth of the silhouette edge, and the maximum depth of any receiver point occluded by the surface adjacent to the edge, as described by Equation 3.3. Unfortunately, the maximum receiver depth cannot be accurately determined prior to computing a complete visibility solution. Therefore, this value is initially set to the maximum light-view depth of any receiver point in the scene and later refined (Subsection 3.4.1 and 3.4.2).

In general, the performance of soft shadow algorithms that employ shadow geometry is often limited by the number and size of the primitives (Subsection 3.6.3). However, the screen-space extents of a penumbra wedge as seen from the eye are commonly much greater than the extents as seen from the light (as in our algorithm). Consider a case in which the eye and light are perpendicular and equidistant from the wedge. The extents of the wedge are proportional to the solid angle subtended, which itself depends on the slope of the wedge faces. From Equation 3.3, this

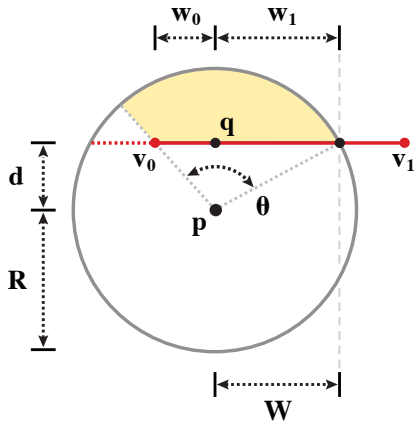


Figure 3.6: The geometry of an area sample is illustrated. A silhouette edge (red) and its adjacent surface (yellow) are clipped to the sample bounds. The radius $\mathbf{R}$ is determined from the light-view depth to the point ($\mathbf{q}$) nearest $\mathbf{p}$ on the silhouette edge segment (Equation 3.3). The signed and normalized area of the minor circular segment defined by the silhouette edge is computed with Equation 3.4. This method is accurate even when one or both edge vertices lie inside the sample bounds.

slope is the ratio of the light-occluder distance ($\mathbf{q}_z$) to the light radius ($\mathbf{R}_{light}$). Therefore, the light-view extents of the wedge will be greater than the eye-view extents only when $\mathbf{R}_{light} >> \mathbf{q}_z$. This situation occurs when the light source is very large (uncommon) or when an occluder is very near the light (also uncommon).

3.3.3 Penumbral Occlusion

To review, occlusion from a spherical light of uniform intensity can be determined by projecting the light and silhouette edge into the light-view image plane, and measuring the area of the light footprint clipped by the edge. This clipping operation (Figure 3.6) is straightforward and avoids penumbral aliasing often found in methods which measure occlusion at discreet points on the light surface. In practice, the image plane projection of a spherical light forms an ellipse as seen from a receiver point, when the point is not directly beneath the light. We approximate this ellipse with a circle. Its radius is given by Equation 3.3, where $\mathbf{q}$ is a point on the silhouette

edge segment nearest $\mathbf{p}$, and $\mathbf{R}_{light}$ is the radius of the light [105]. A negative value indicates the silhouette geometry is further from the light than the receiver point and no occlusion is possible. For positive values of $\mathbf{R}$, the normalized area of the minor circular segment clipped by the silhouette edge (shaded area in Figure 3.6) is computed according to Equation 3.4. As we'll see in Section 3.3.4, the sign of the dot product of the edge and $\mathbf{p}$ determines if the result is added or subtracted (i.e. $\mathbf{p}$ is outside or inside the edge) from the accumulated occlusion at the receiver point.

$$\mathbf{R} = \left(\frac{\mathbf{p}_z - \mathbf{q}_z}{\mathbf{q}_z}\right) * \mathbf{R}_{light} \tag{3.3}$$

$$\mathbf{w}_0 = -\left(\frac{(\mathbf{v}_0 - \mathbf{p}) \cdot (\mathbf{v}_1 - \mathbf{v}_0)}{\mathbf{R}}\right)_{[-\mathbf{W},\mathbf{W}]}$$

$$\mathbf{w}_1 = +\left(\frac{(\mathbf{v}_1 - \mathbf{p}) \cdot (\mathbf{v}_1 - \mathbf{v}_0)}{\mathbf{R}}\right)_{[-\mathbf{W},\mathbf{W}]}$$

$$\theta = cos^{-1}\left(\frac{\mathbf{d}^2 - \mathbf{w}_0 * \mathbf{w}_1}{\sqrt{\left(\mathbf{d}^2 + \mathbf{w}_0^2\right)\left(\mathbf{d}^2 + \mathbf{w}_1^2\right)}}\right)$$

$$\tilde{V}'' = \frac{\pm\theta - \mathbf{d} * (\mathbf{w}_0 + \mathbf{w}_1)}{2\pi} \tag{3.4}$$

This strategy of determining penumbral occlusion by measuring the area of overlap between a light and silhouette geometry is not unique. For example, Assarsson et al. [14] describe an occlusion kernel for spherical lights similar to ours, though the method of computation is different. In our algorithm, an inverse cosine function call is used in place of texture lookups into a table of inverse tangent values. The latency of the function call can be higher than that for the texture lookups if the table is in cache, but the increased accuracy inhibits penumbral banding. Further, in our algorithm the clipping operation is performed in 2D and so avoids evaluating the quadratic equation used in intersecting a line with a cone in 3D.

The computation of penumbral occlusion using a spherical light is motivated by the relative simplicity of the calculation which results from the radial symmetry of this shape. However, the overall algorithm is *not restricted to spherical lights*. For example, Assarsson et al. describe an occlusion kernel for rectangular lights [14], which can be used with our algorithm. In general, any function which returns the degree of occlusion given a silhouette edge and receiver point can be used, though the performance of the overall algorithm is dominated by this kernel.

3.3.4 Composition of Umbral and Penumbral Occlusion

The visibility function at a given receiver point cannot be accurately reconstructed from penumbral occlusion alone [76]. The computation of $\tilde{V}''$ is only performed for silhouette edges that pass within a distance of $\mathbf{R}$ (Equation 3.3) of the point. As a result, the two cases seen in Figure 3.7 cannot be disambiguated. Occlusion due to surfaces which fully cover the light as seen from the receiver point is not accounted for. Since this second type of occlusion is constant for all points on the light (i.e. the light is fully occluded or fully unoccluded) it can be measured from a single point anywhere on the light surface using a hard shadow algorithm. Combined with the penumbral occlusion, the result forms a complete solution to the visibility term of Equation 3.2 [14, 76, 45].

$$\int_{\mathbf{x}\in A} V(\mathbf{x})d\mathbf{x} \approx 1 - \left(\underbrace{\sum_{i=1}^{N} \tilde{V}'(object_i, \mathbf{p})}_{\text{umbra}} + \underbrace{\sum_{j=1}^{E} \tilde{V}''(edge_j, \mathbf{p})}_{\text{penumbra}} \right)_{[0,1]} \tag{3.5}$$

The composition of the two sources of occlusion can be expressed in terms of depth complexity (Equation 3.5). For a point $\mathbf{x}$ on the light, *depth complexity* is the number of intervening surfaces between $\mathbf{x}$ and a receiver point. In this context, silhouette edges denote changes in depth complexity across the surface of the light

(a) The light is partially occluded. (b) The light is fully occluded.

Figure 3.7: The visibility function at a receiver point cannot be reconstructed from penumbral occlusion alone. The absence of occlusion from surfaces which fully cover the light leads to ambiguity. For example, a case in which the light is partially occluded by a single surface (a) cannot be distinguished from a case in which the light is partially occluded by one surface and fully occluded by a second surface (b).

relative to a constant factor measured at a point on the light [76]. In our algorithm, this constant factor $\tilde{V}'$ is computed from the center of the light source as seen from a receiver point ($\mathbf{p}$) via hard irregular shadow mapping (Section 2.3). The result is subsequently adjusted by relative changes in depth complexity ($\tilde{V}''$) computed via Equation 3.4, and clamped to the range $[0, 1]$.

This composition of umbral and penumbral occlusion is illustrated in Figure 3.8 for several occluder configurations. Consider case (e). The umbral computation yields a value of 0.0, while the penumbral computation yields +0.15 and +0.2 for the two silhouette edges shown. The total accumulated occlusion is 0.35. Similarly, in case (g) the umbral and penumbral values are 2.0, -0.15, and -0.2 respectively, for a total occlusion value (after clamping) of 1.0. This method is fast, robust, requires no precomputation, and is accurate in many common cases. It is approximate where occluders overlap within the bounds of the area sample but do not fully cover the sample, as in (h). However, no high-frequency artifacts are introduced. Rather, the resulting penumbrae are continuous but overly narrow. The accuracy of this method and the visual impact of the approximation are discussed further in Subsection 3.5.2.

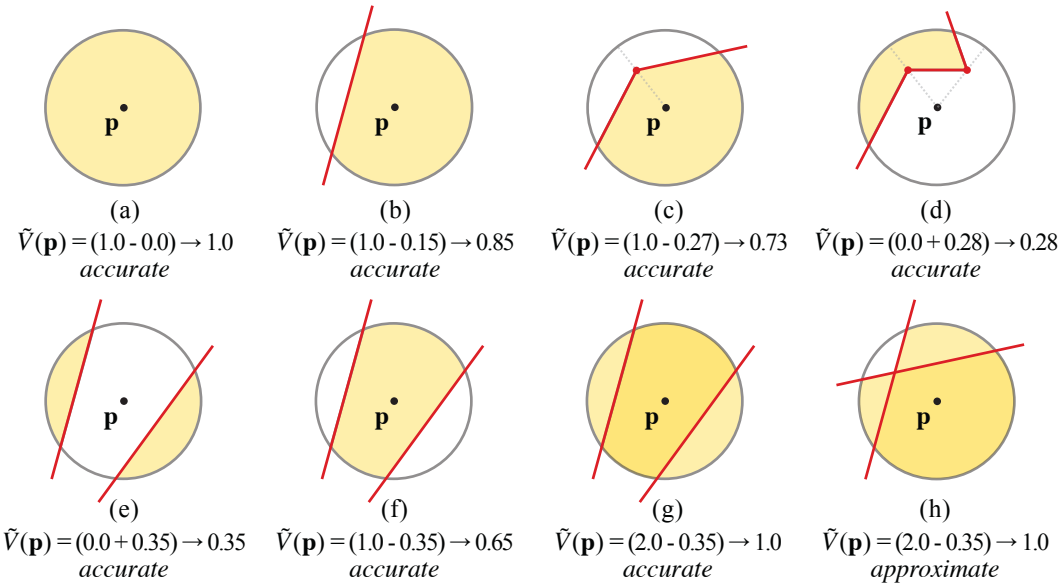


Figure 3.8: The evaluation of Equation 3.5 is illustrated geometrically for several occluder configurations. The umbral calculation is performed by point-sampling the scene at location **p** in the image plane. The penumbral calculation is performed by area-sampling about **p**, with the overlap between any two occluders assumed to be 0. The umbral and penumbral occlusion is combined to determine the total visibility at **p** and thereby the degree of shadow for the eye-view pixel corresponding to **p**. This method is accurate in many common cases (a - g). It is approximate only where two or more occluding surfaces overlap within the bounds of the area sample but do not fully occlude the sample (h). The resulting penumbrae remain continuous and visually pleasing, but the intensity falloff is sharper.

3.4 Optimizations

A key source of inefficiency in nearly all soft shadow algorithms is overdraw. Here, *overdraw* refers to the unnecessary computation of penumbral occlusion at a receiver point, and occurs when the edge is tested for occlusion against the point but is found not to occlude the point. Our algorithm addresses overdraw in part by estimating the set of receiver points affected by a given silhouette edge, prior to performing the actual penumbral occlusion calculation. This estimation step employs a set of shadow primitives representing the expected light-view screen-space bounds of the

penumbra cast by each silhouette edge (Figure 3.5). Each primitive must be wide enough to occlude all receiver points potentially affected by the respective edge, but overestimation of the width can itself result in significant overdraw. Unfortunately, this width cannot be accurately known until *after* the occlusion computation, since it is proportional to the maximum depth of any object occluded by the silhouette edge (Equation 3.3). This width is bounded by the maximum light-view depth of any receiver point, but this bound is insufficient to avoid substantial overdraw. We address this problem with two simple optimizations. One reduces overdraw in the X and Y dimensions in light space, while the second reduces overdraw in Z.

3.4.1 Reducing Overdraw in X and Y

The cyclic dependence described above can be addressed via iterative refinement. In this case, the width of a shadow primitive in the $n+1$ iteration can be recomputed from the maximum depth of any receiver point occluded during iteration n. Iterative refinement adds overhead, but much of this overhead is subsumed in Z-buffer systems in which rasterization itself is performed iteratively. For example, *tiled rasterizers* presort geometry into screen-space bins and scan-convert primitives in bin order [47, 118]. This process may be thought of as 2-level hierarchical rasterization, and can be used to reduce the screen-space extents of shadow primitives as follows. The width of a shadow primitive is initially computed from the maximum depth of any receiver point. This width is then recomputed during sorting based on the maximum depth of any receiver point located in each bin (determined during data structure construction) overlapped by the primitive. If the resized shadow primitive no longer overlaps the bin it is not included in the geometry list for that bin.

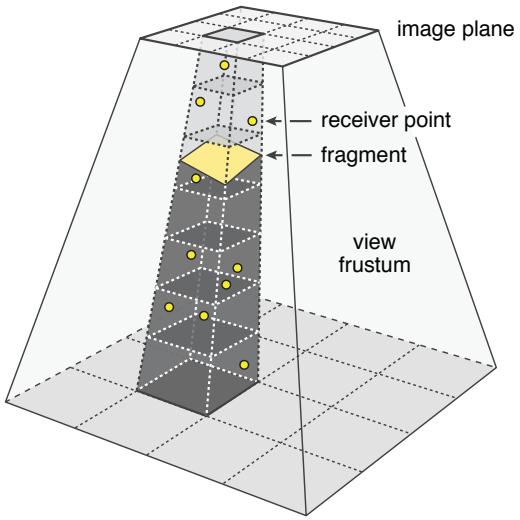


Figure 3.9: A simple spatial acceleration structure for storing the coordinates of receiver points. This structure is composed of a 3D perspective grid, such that the face nearest the light lies within the light-view image plane.

3.4.2 Reducing Overdraw in Z

To review, the penumbra cast by a silhouette edge forms a wedge, and only receiver points inside the wedge are affected by the edge. This property can be exploited to reduce overdraw in Z by intersecting the wedge and the spatial acceleration structure storing the receiver points. Penumbral occlusion is then computed only at points within the area of overlap. For example, consider an acceleration structure composed of a simple 3D grid such that the face nearest the light lies in the light-view image plane (Figure 3.9). For each light-view pixel occluded by a shadow primitive, the intersection of the respective penumbra wedge and the column of grid cells beneath the pixel is determined (Figure 3.10). The intersection point can be estimated from the point on the silhouette edge ($\mathbf{q}$) nearest the pixel center and the slope of the wedge (Figure 3.5). From Equation 3.3, this slope is the ratio of the light-occluder distance ($\mathbf{q}_z$) to the light radius ($\mathbf{R}_{light}$). The computation of penumbral occlusion from Equation 3.4 is only performed for receiver points at the intersection point and deeper in the grid.

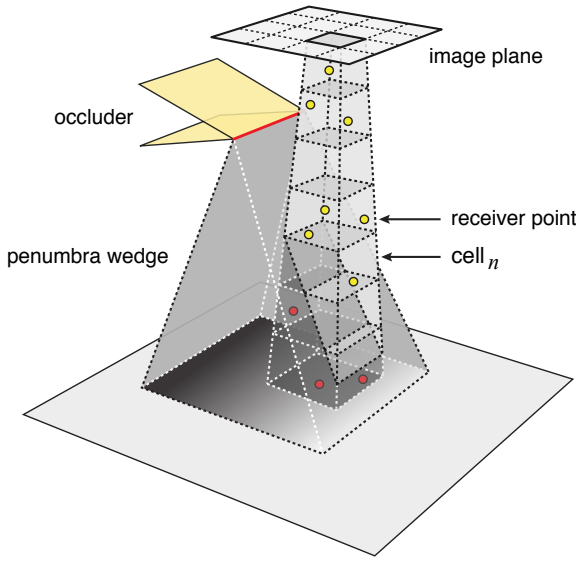
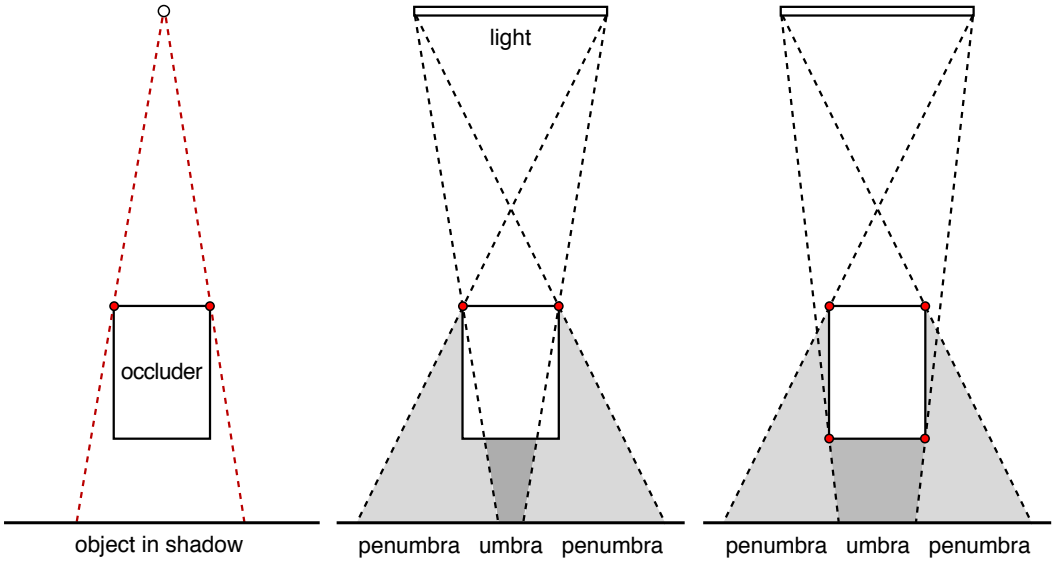


Figure 3.10: The penumbra cast by a silhouette edge forms a wedge. The number of receiver points tested for occlusion against the edge can be reduced by considering only the points inside the wedge. We do this by intersecting the wedge with a spatial acceleration structure, in this case a 3D grid (only a single column of cells is shown). Here, the wedge intercepts the column of grid cells at cell $_n$. Only the points located in this cell and deeper in the grid (denoted in red) need be tested for occlusion.

3.5 Approximations

The visibility integral of Equation 3.2 can be solved analytically [103], but not yet at real-time frame rates. To achieve real-time performance in dynamic scenes it is generally understood that one or more approximations are required. The challenge is to choose approximations which defray the most computational cost while sacrificing the least visual quality. Note that the space of possible approximations can be loosely bisected: those that may produce objectionable artifacts (e.g. aliasing, light leaks), and those that yield plausible but inaccurate results. Though approximations of the former type often enable an arbitrary reduction in computation (typically through one or more tunable parameters), the error is similarly unbounded. For this reason, we use approximations of the second type. These generally result in more



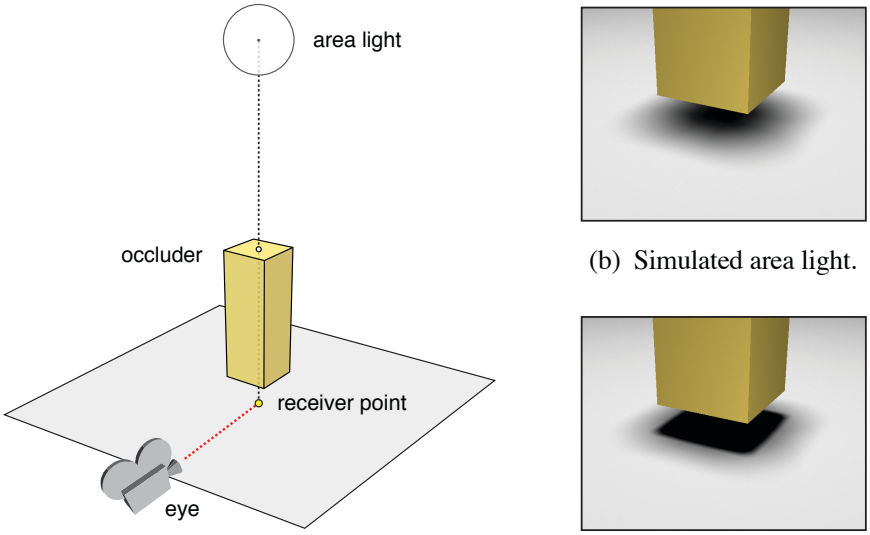
(a) Single point of visibility. (b) Simulated area light. (c) Physically correct area light.

Figure 3.11: The visibility solution for an area light can be approximated using a single point of projection (a) in place of multiple points on the light surface. The width of penumbrae cast by the simulated area light can then be estimated. However, the resulting umbrae are undersampled (b) in comparison to those from a physically correct solution (c). This undersampling is due to a difference in the set of silhouette corners (red dots) visible to the light in each of the two cases.

modest performance gains, but require no tuning and offer better error bounds. Our algorithm uses two such approximations: single point of projection and independent evaluation of occluders.

3.5.1 Single Point of Projection

The visibility term of Equation 3.2 can be estimated by determining occlusion from a single point (rather than from multiple points) on the surface of an area light. Note that shadows cast by a point light have penumbrae of zero width, and every geometric object has a fixed silhouette from the view of the light. In contrast, the view of an object from an area light varies from point to point across the light



(a) A simple scene.

(b) Simulated area light.

(c) Physically correct area light.

Figure 3.12: A figure inspired by Assarsson and Akenine-Möller [13] illustrating the visual impact of estimating area light visibility using a single point of projection (Figure 3.11). The simple scene (a) contains a tall occluder oriented toward the light such that only the four edges nearest the light are considered silhouettes as seen from the single point of projection. This orientation maximizes the "single silhouette" error. The rendered images show the estimated result (b) and the correct result (c).

surface. This is the fundamental cause of penumbrae. Expressed in the context of parallax, an observer moving along a 1D light sees a nearby triangle move with respect to a more distant surface. This motion traces out a penumbra on that distant surface. For a 2D light the principle is the same. Unfortunately, computing an accurate visibility solution for P points on an area light requires P times the expense of that for a single point on the light.

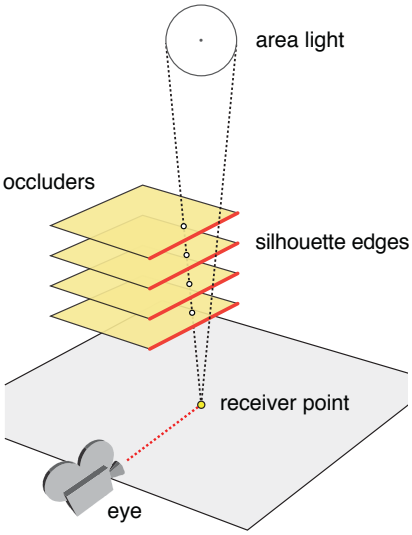
As an alternative, the magnitude of the parallax effect can be estimated heuristically, as in our algorithm. This value is proportional to the width of the penumbra cast by one object onto another, which is itself the ratio of the distances from the light to the occluder and the light to the object in shadow (Figure 3.5). Expressed in Equation 3.3, the result is accurate for planar occluders and receivers

parallel to the light [105]. This estimate is widely used in lieu of computing visibility from multiple points on the light [43, 14, 105, 54, 144, 21, 28], since the error is not significant, biased, or objectionable even when the light is large relative to occluders.

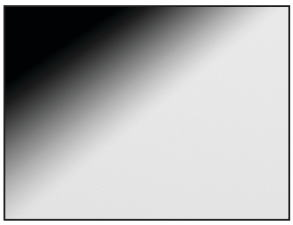
However, there is a second effect due to parallax in which previously hidden silhouette edges become visible [14]. The contribution of these edges can impact the shape of the penumbrae and the apparent size of the umbrae cast by the object (Figure 3.11 and 3.12), but cannot be estimated effectively with only a single view point on the light. The resulting error is expressed primarily as umbrae which are smaller than expected, and correspondingly larger penumbrae. No high-frequency or other objectionable artifacts are introduced. The magnitude of this error derives from the light-view depth bounds of the occluder, the distance from the occluder to the receiver point, and the size of the light. More specifically, the error is proportional to the ratio of $\mathbf{R}_{near}$ to $\mathbf{R}_{far}$, where $\mathbf{R}$ is given by Equation 3.3 for points ($\mathbf{q}$) on the near and far silhouette edges of the occluder. The incidence of this error can be reduced by incorporating visibility information from multiple points on the light surface. For example, Assarsson et al. represent a single large area light as a collection of smaller lights [14]. This strategy is likewise compatible with the soft shadow algorithm described in Section 3.3.

3.5.2 Independent Evaluation of Occluders

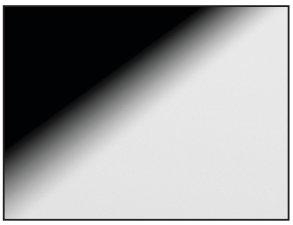
Several soft shadow algorithms (including ours) evaluate the visibility term from Equation 3.2 per-occluder, but these partial visibility terms are not independent. Therefore, information on the degree of overlap between occluders is lost, and the total accumulated occlusion is overestimated by an unknown amount C as illustrated in Equation 3.6 and Figure 3.8h. Note that $\tilde{V}'(object_i, \mathbf{x})$ is 1 when a point on the light $\mathbf{x}$ is *occluded* by object i and 0 otherwise, and $\tilde{V}'(object_i, \mathbf{x}) = 1 - V'(object_i, \mathbf{x})$.



(a) A simple scene.



(b) Eye view: 1 occluder.



(c) Eye view: 2 occluders.

(d) Eye view: 3 occluders.

(e) Eye view: 4 occluders.

Figure 3.13: The visual impact of estimating the overlap from multiple occluding surfaces (Figure 3.8h) is illustrated in a simple scene (a). The scene contains up to four occluders aligned so the silhouette edges are nearly colinear as seen from the light. This alignment maximizes the overlap and thus the estimation error. The rendered images show the correct result (b), and the increasingly sharp falloff of the penumbra as more occluders are introduced (c - e).

$$\int_{\mathbf{x}\in A} V(\mathbf{x})d\mathbf{x} = \int_{\mathbf{x}\in A} \bigcup_{i=1}^{N} V'(object_i, \mathbf{x})d\mathbf{x} = 1 - \left[\left(\sum_{i=1}^{N} \int_{\mathbf{x}\in A} \tilde{V}'(object_i, \mathbf{x})d\mathbf{x}\right) - C\right] \quad (3.6)$$

Accurately determining C is difficult, as it requires clipping an incoming occluder to an arbitrarily-shaped silhouette representing the composition of previous occluders, and storing this silhouette per sample point. Bitwise coverage masks can be used to simplify clipping and reduce the storage requirements, but at the cost of introducing aliasing into the penumbrae [116]. In practice, algorithms following Equation 3.6 typically approximate C. This value is bounded by 0 (no occluders

overlap) and the sum of the area of all but the largest occluder (occluders maximally overlap). Some algorithms compute an average over the occluders [14] while others use the maximum [54, 144, 28]. Of particular interest is the approximation used by Soler and Sillion, given by $\frac{1}{2}(min(V_1, V_2) + max(0, V_1 + V_2))$. Here, V_1 and V_2 are integrated visibility terms for a pair of occluders. The error in this case is bounded by $\frac{1}{4}$ [122].

In our algorithm, the occlusion from an object is determined independently of that of other objects. Implicit in our approach is the assumption that $C = 0$. This approximation is fast to compute and accurate in a surprising number of common cases (Figure 3.8a - g). Further, where it is approximate (Figure 3.8h) no objectionable artifacts are introduced. Estimated penumbrae are continuous and visually pleasing, but are biased towards darker values (Figure 3.13). In the limit, a penumbra of zero width can result. However, this extreme case requires a large number of silhouette edges to be (nearly) colinear after projection into the light-view image plane, and is therefore infrequent.

3.6 Further Related Work

The approximations discussed in Section 3.5 are part of a larger space of solutions for solving the visibility integral of Equation 3.2. This space can be structured according to methods for approximating the domain of integration, and methods for computing the integrand V. The former includes: single point of projection, independent evaluation of occluders, restricted light geometry, proxy scene geometry, and precomputation / band-limiting (Subsection 3.5.1, 3.5.2, 3.6.1, 3.6.2, 3.6.4). The latter includes: shadow geometry, and resampling / filtering approximate visibility (3.6.3, 3.6.5). Interactive soft shadow algorithms employ a combination of methods, and differences in operation and performance, and image quality (Table 3.1) stem from the specific set used. As in the hard shadow case, our soft shadow algorithm

	Plausible But Inaccurate			Noticeably Implausible		
Algorithm	**Inner Penumbra**	**Outer Penumbra**	**Accurate Overlap**	**No Aliasing**	**No Light Leaks**	**All Frequency**
[Agrawala et al. 2000]	✓	✓	✓		✓	✓
[Annen et al. 2008]	✓	✓		✓	✓	⋆
[Assarsson et al. 2003]	✓	✓		✓	✓	✓
[Bavoil et al. 2006]	✓	✓	✓			✓
[Brabec and Seidel 2002]	✓	✓			✓	✓
[Chan and Durand 2003]		✓		⋆⋆		✓
[Eisemann and Décoret 2006]	✓	✓		✓		⋆
[Fernando 2005]	✓	✓	✓		✓	✓
[Forest et al. 2008]	✓	✓	✓		✓	✓
[Guennebaud et al. 2007]	✓	✓			✓	✓
[Haines 2001]		✓		✓	✓	✓
[Johnson et al. 2008]	✓	✓		✓	✓	✓
[Ren et al. 2006]	✓	✓	✓	✓	✓	
[Schwarz and Stamminger 2007]	✓	✓	✓		✓	✓
[Sintorn et al. 2008]	✓	✓	✓		✓	✓
[Soler and Sillion 1998]	✓	✓	✓			✓
[Wyman and Hansen 2003]		✓			✓	✓
[Zhou et al. 2005]	✓	✓	✓	✓	✓	

Table 3.1: The visual qualities of several soft shadow algorithms compared. Ours is marked in gray. Accuracy is loosely proportional to the number of check marks. The absence of a check in a *plausible but inaccurate* column indicates the algorithm yields plausible-looking but potentially incorrect penumbrae, while the absence of a check in an *noticeably implausible* column denotes the potential for objectionable artifacts. Penumbrae are divided into *inner* and *outer* regions by the hard silhouette of an object. The lack of either yields overly narrow and misaligned penumbrae. *Accurate Overlap* denotes accurate computation of the area of overlap between occluders. Inaccuracies here commonly result in a bias towards overly dark penumbrae. *Alias-free* algorithms produce no sawtooth or banding patterns, or flickering in penumbrae during object-light motion. *Light leaks* appear as intensity discontinuities in otherwise unbroken regions of shadow. *All-frequency* algorithms do not impose artificial bounds on the frequency content of illumination via undersampling in frequency space, or prefiltering of undersampled depth (i.e. shadow) maps. Many algorithms reduce the incidence of these artifacts using various means, but do not resolve the underlying issue(s). The accuracy of our algorithm is similar to that of Assarsson et al., which employs eye-space shadow geometry. ★ Frequency content is lost by prefiltering the shadow map. ★★ Texture mapping can re-introduce aliasing.

is distinguished primarily in the method used to determine V. The algorithm does not determine occlusion from the light in eye space using shadow geometry[3], or a discretized representation of the scene geometry (i.e. shadow map). Rather, receiver points are stored in a light-view spatial acceleration structure, and occlusion is computed at these locations in light space directly from the scene geometry.

3.6.1 Restricted Light Geometry

Observe that many sources of light in the real world (e.g. sun, incandescent bulbs, fluorescent fixtures) are simple shapes with symmetry, and the visual impact of light shape on penumbrae is subtle. As a result, area light sources are frequently defined as planar rectangles or discs. For rectangular lights, a simple linear parameterization can be used to obtain a uniform distribution of samples across the light surface [75, 74, 2]. Soft shadow algorithms that use bilinear [39] or percentage closer filtering [43] to estimate penumbrae from umbral silhouettes, are implicitly assuming a square or rectangular area light due to their use of square or rectangular filter kernels. Disc-shaped lights mimic omnidirectional spheres of constant intensity [21, 54], and produce a sinusoidal intensity falloff which can be approximated with a Bernstein cubic interpolation function [105, 144, 28]. The simplicity of the disc also enables fast analytic integration of occlusion resulting in high-quality penumbrae (Equation 3.4).

3.6.2 Proxy Scene Geometry

Just as arbitrarily-shaped light sources complicate the visibility integral, arbitrary scene geometry can be similarly difficult to support. One alternative is to compute visibility from simplified "proxy" geometry rather than the original scene geometry. Doing so can reduce the number of silhouette edges and reduce the complexity of

[3]Light-space shadow geometry is used to determine a coarse solution for V. However, the final visibility computation employs no shadow primitives.

the visibility computation per edge. For example, Ren et al. replace the occluding geometry with a hierarchy of spheres [111]. The size of each sphere approximates the scale of the local geometry replaced by that sphere. This strategy simplifies the spherical harmonic rotations and exponentiations required to compute the degree of occlusion at a point in the scene visible from the eye. However, finer shadow details are lost in regions with greater geometric complexity than that expressed by the proxy geometry.

A more common approach is to represent the scene geometry (explicitly or implicitly) as thin planar occluders parallel to the light source. Soler and Sillion explicitly decompose occluding geometry into planar elements and convolve these elements with the light to produce soft shadow textures [122]. Similarly, Eisemann and Décoret create planar proxy geometry by uniformly subdividing scene geometry by distance from the light, and projecting the contents of each partition into a plane parallel to the light [39]. These methods can achieve relatively high performance, but are susceptible to light leaks between planar elements.

Alternatively, several methods use a conventional shadow map as a discrete representation of occluder geometry [116, 51, 17]. Shadow map texels are back-projected onto the light, and the projected area is compared with the area of the light to produce a visibility estimate. This approach can achieve interactive [116] and even real-time performance [51] in simple scenes, but high-frequency geometric features (and thus finer shadow details) are lost due to the discretization. Moreover, a straightforward implementation of this technique can introduce holes in occluders resulting in light leaks.

3.6.3 Shadow Geometry

While proxy geometry is used as a stand-in for scene geometry, *shadow geometry* delimits regions of the scene inside the umbra or penumbra of an occluder. These methods are similar to their hard shadow counterparts (Subsection 2.5.1) with the added complexity of defining the boundaries between three types of space (umbra, penumbra, fully-lit) rather than two (umbra, fully-lit).

Observe that the penumbra cast by a silhouette edge forms a wedge as shown in Figure 3.1b. The faces of this wedge can be represented with shadow polygons. Assarsson et al. and Forest et al. compute penumbral occlusion (Subsection 3.3.3) at each receiver point corresponding to a pixel covered by these shadow primitives [14, 45]. The result is combined with the occlusion computed from a second set of shadow primitives which delimit the bounds of umbrae (Subsection 2.5.1). Haines utilizes shadow geometry composed of sheets and cones forming the faces and corners of penumbra wedges, shaded with a gradient mimicking the transition from light to shadow [54]. This geometry is rendered to a texture which is then projected back onto the scene. Chan and Durand generate a set of polygons representing the extents of penumbrae as seen from the light [28]. This geometry is rasterized into a light-view "smoothie buffer". Penumbral occlusion is computed by sampling this buffer, while umbral occlusion is computed by sampling a classical shadow map. In all three cases, penumbral regions are defined geometrically and can be rendered directly, yielding plausible looking shadows without aliasing or band limiting. However, the shape and structure of the shadow geometry often derives from assumptions about the light geometry, resulting in approximations particularly at the joints between neighboring shadow polygons [28].

Interestingly, Laine et al. use shadow geometry to accelerate visibility queries in several offline soft shadow algorithms built into a ray tracer [75, 76, 74]. Here the role of the shadow primitives is more closely related to that of the split planes in a

spatial acceleration structure. The percentage of an area light visible to a receiver point is determined by testing the point against a hierarchy of shadow volumes. Each volume is defined by a different subregion of the light and the silhouette of an occluding object.

3.6.4 Precomputation and Band-Limiting

Part or all of the visibility computation can be moved offline and the results stored in a form readily accessible at runtime. However, precomputing visibility or illumination requires making assumptions about the spatial relationship between elements of the scene. These assumptions lead to restrictions on the motion of the camera or light [2] and / or on rigid body motion or deformation of scene geometry [148].

Ren et al. precompute low-frequency visibility information using spherical harmonic exponentiation [111]. Since spherical harmonic coefficients represent the illumination in frequency space, band-limiting bounds the storage requirements of the precomputed data. This method yields interactive frame rates and handles the difficult case of self-shadowing in deformable models. It also works well for large area lights and environment maps, but shadows from small local light sources are unconvincing.

3.6.5 Resampling / Filtering Approximate Visibility

Alternatively, the visibility calculation can be minimized by resampling and / or filtering an approximate solution to achieve often plausible looking (but potentially inaccurate) penumbrae. For example, the hard shadow silhouettes found via classical shadow mapping, can be blurred using bilinear filtering. The performance of these methods benefits from hardware-accelerated filtering and from the relative inexpense of the approximate visibility function.

Soler and Sillion use an explicit convolution kernel to produce penumbrae from an image of the occluders as seen from the light. The resulting soft shadow textures are then re-projected back onto scene geometry [122]. The constant-width filter used is accurate only for planar scene geometry parallel to the light. Fernando uses a variable-width kernel with percentage closer filtering [109] to blur hard shadow boundaries in a classical shadow map on a per-sample basis [43]. The size of the filter is proportional to the penumbra width and is estimated as the ratio of the distances from the light to the occluder and the light to the shadowed geometry (Equation 3.3). The algorithm is simple to implement but is bandwidth intensive since the number of shadow map texels retrieved from memory grows as the square of the kernel width. Further, the resulting penumbrae can exhibit aliasing artifacts since the shadow map is a discretized representation of the occluding geometry. Mipmapping and summed area tables [77] can be used to prefilter shadow maps to avoid this aliasing, but doing so results in the loss of high frequency shadow information [8, 39].

3.6.6 Closely Related Work

Concurrent with our work, Sintorn et al. have developed a similar algorithm [120]. As in our approach, receiver points are stored in a light-view spatial acceleration structure. This structure is a 2D form of the 3D perspective-correct grid described in Subsection 4.2.2. The method for determining umbral occlusion (Section 2.3), and the use of shadow geometry to estimate the set of receiver points affected by a silhouette (Subsection 3.3.2), are also similar. The two algorithms differ primarily in the computation of penumbral occlusion and in the optimizations used. Sintorn et al. estimate the integral of the visibility function in Equation 3.2, by evaluating V at several discreet points on the light surface. In our algorithm, we evaluate this integral analytically, avoiding a potential source of penumbral aliasing. Further, our

algorithm is distinct in its use of optimizations for reducing overdraw associated with the shadow primitives (Section 3.4). The similarity in the two algorithms, though striking, is not unexpected. The grid-based data structure and use of light-space shadow geometry in particular are logical extensions of ideas introduced in hard irregular shadow mapping [65].

3.7 Summary

Though extensively studied, there remains a gap in the solution space for real-time soft shadow rendering. Specifically, no existing algorithm has been demonstrated to achieve both high image quality *and* high performance in dynamic scenes with the geometric complexity of modern games. Existing approaches vary in the visibility function V and in the method used to estimate the integral of V over the light surface, but image quality and performance are primarily influenced by the former. As with hard shadows, most soft shadow algorithms determine V from shadow geometry or a shadow map. Methods using *eye-space* shadow geometry can produce penumbrae comparable to a beam tracer, but the performance is constrained by overdraw (Subsection 2.6). Alternatively, algorithms based on shadow mapping can achieve high performance, but are subject to aliasing, loss of frequency content, and light leaks. The incidence of these artifacts can be reduced by prefiltering, resampling, and oversampling, but such solutions do not address the root causes: misalignment of the eye and light-view sample patterns (Figure 2.2b) and loss of geometric detail due to the discretization.

Soft irregular shadow mapping (our algorithm) occupies a unique point in the solution space defined by image quality and performance. As we'll see in Chapter 6, this algorithm achieves frame rates comparable to the highest performing existing methods, but yields substantially higher image quality. More specifically, the image quality produced is typically indistinguishable from that of physically-accurate (but

lower performance) approaches like beam tracing. This novel combination of image quality and performance is achieved through four key properties of our algorithm. First, the points in the scene visible from the eye (i.e. receiver points) are stored in an explicit spatial acceleration structure in light space, and the computation of umbral and penumbral occlusion is performed only at these points rather than for all points in the scene, resulting in high efficiency. Second, the computation of occlusion is performed *exactly* at these points, avoiding aliasing and self-shadowing artifacts endemic to classical shadow mapping methods. Third, the umbral occlusion computation is separated from the computation of penumbral occlusion, enabling the relatively expensive penumbral component to be restricted to receiver points near silhouette edges. Fourth, the computation of penumbral occlusion is performed analytically per silhouette edge using shadow primitives. Performance loss due to overdraw is minimized by rasterizing these primitives from the light rather than from the eye. Note that the composition of occlusion from silhouette edges from different occluding surfaces is approximate. However, this approximation does not introduce objectionable artifacts. We explore the overall image quality and performance of this algorithm in detail in Chapter 6.

Chapter 4

Spatial Acceleration Structures

A distinguishing characteristic of the irregular Z-buffer algorithm is the use of non-uniform spatial acceleration structures for storing the coordinates of sample points in an image plane. The primary purpose of this structure is to organize the sample points spatially such that those within a specific region of interest can be quickly loaded from memory. In the classical Z-buffer algorithm, an explicit acceleration structure is unnecessary. Here, the samples form a grid and the coordinates of any point can be determined from a simple formula given the indices of the point. In the irregular Z-buffer algorithm, the placement of samples in the image plane is typically scene dependent and as such cannot be readily computed from a formula. As a result, the sample coordinates must be stored as they are determined, for later use during irregular rasterization.

4.1 Design Considerations

Spatial acceleration structures may take any of several forms including variants of BSP trees (e.g. quadtrees, k-d trees), grids, or combinations of the two, and the best choice often depends on the intended application. For example, in cases where the distribution of samples changes significantly from frame to frame, the cost of construction can be as important as query cost, and both may vary asymptotically between structure types. In the case of irregular shadow mapping, a primary design consideration is memory bandwidth utilization. Recall that this algorithm consists of two phases: construction of the spatial acceleration structure, and computation of umbral and penumbral occlusion at the receiver points stored in this structure. As we'll see in Chapter 6, performance is dominated by the occlusion computation and this phase of the algorithm is memory rather than compute bound.

4.1.1 Grids Versus Trees

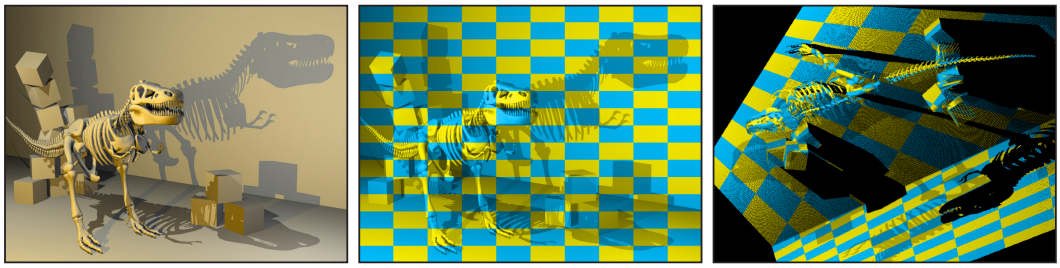
The relative advantage of a grid or tree-based spatial acceleration structure depends on a wide range of variables, and a given structure is unlikely to outperform another across all cases. In principle, trees offer several benefits over grids, including faster range queries since fewer spatial partitions are typically visited, and more efficient queries since spatial partitions vary in size according to their contents. Conversely, grids are easier to construct and traverse, and construction is easier to parallelize. In practice, the performance of a given structure is dominated by properties of the scene (e.g. receiver point distribution, image plane extents of geometric primitives), the functionality present in current graphics hardware, and the values of structure-specific parameters (e.g. maximum tree depth, grid resolution). A complete analysis of the relative merits of grid and tree-based acceleration structures is beyond the scope of this dissertation. Here, we focus on the spatial efficiency of range queries, as this is a primary factor influencing the memory bandwidth used during irregular shadow mapping.

Conceptually, the spatial efficiency of a range query can be defined as the useful data loaded from memory divided by the total data loaded from memory. In the case of shadow rendering, this value is defined as the total depth complexity over all receiver points[1], divided by the total number of receiver points accessed. A *perfect* spatial acceleration structure is one in which the value of this efficiency measure is 1.0. In a practical structure, this value is sensitive to the average size of a geometric primitive relative to a spatial partition, and the average point density in partitions with high depth complexity. For a given number of spatial partitions, a k-d tree will typically achieve higher efficiency than a grid, since the partition size is adaptive based on the local point density. However, this efficiency comes with substantially greater construction and traversal costs and implementation complexity. Moreover, we have found that a simple grid can achieve an efficiency value of 0.7 with little tuning of the grid resolution in a game-like scene, while enabling range queries to be performed via classical rasterization (Subsection 2.3.1). For these reasons, the two data structures discussed in Section 4.2 are both grid-based.

4.1.2 Storage Order in Memory

The storage order of receiver points in the spatial acceleration structure is key to maximizing spatial reuse, and minimizing memory bandwidth consumption. Recall that receiver points correspond to locations in the scene visible from the *eye*, but occlusion is computed at these points in *light space*. As a result, the receiver points can be organized in memory according to the order of the respective pixels in the eye view, or by their relative spatial locations in the light view. Eye-view indexing exposes some locality and does not require sorting receiver points frame-to-frame. Light-view indexing exposes maximal locality but requires sorting points per frame.

[1]The depth complexity of a receiver point is the number of surfaces hit by a ray from the receiver point to the light source.



(a) Eye view. (b) Eye view (tiled). (c) Light view.

Figure 4.1: A scene with two light sources is seen in (a). This view is overlaid with colored tiles in (b). Points in the scene visible from the eye are transformed into the image plane of the overhead light (c). Each point retains the color of the respective eye-view pixel. Note that spatial locality is in part preserved between the eye and light view tiles. Two points from neighboring pixels in the eye view tend to remain nearby after projection into the light-view image plane.

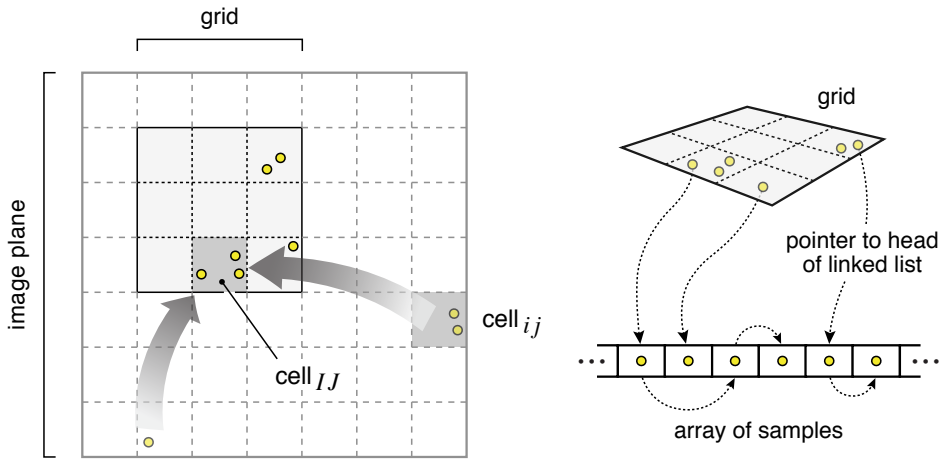
With eye-view indexing, the goal of achieving high spatial reuse might at first seem irreconcilable with the unpredictable arrangement of receiver points in the light view. However, though the relationship between the positions of pixels in the eye-view raster and the corresponding receiver points in the light-view image plane is scene dependent it is *not necessarily incoherent*. In fact, pixels that are nearby in the eye view tend to remain nearby after transformation into the light view. This property is illustrated in Figure 4.1. Observe that tiles in the eye-view remain largely intact following transformation and projection into the light-view image plane. The degree to which this property holds depends on the amount of local variation in the per-pixel depth of the geometry seen from the eye. The greater the variation, the more the corresponding receiver points will be dispersed in the light-view image plane. Light-view indexing improves spatial reuse by eliminating the impact of this dispersal on locality. Receiver points are sorted based on their relative spatial locations in the light-view image plane ensuring that neighboring points are also nearby in memory.

4.2 Practical Data Structures for Shadow Rendering

Here, we introduce two spatial acceleration structures for use with irregular shadow mapping. Both structures are based on grids (Section 4.1.1), but are irregular. This irregularity results from the property that the number of receiver points varies from cell to cell based on the spatial relationship of the geometry, light, and eye, and is bounded only by the number of eye-view pixels. The two data structures differ primarily in the organization of the receiver points in memory, and subsequently in the hardware features needed for efficient construction and traversal.

4.2.1 A Spatial Acceleration Structure with Eye-View Indexing

A simple spatial acceleration structure for shadow rendering is seen in Figure 4.2. It is composed of a grid with linked lists at each cell which store the receiver points located in the cell. This structure is similar to the grid of variable-lengthed arrays used by Purcell et al. in GPU-based ray tracing [107] and photon mapping [108]. It is distinct in the use of a large logical grid spanning the image plane, in tandem with a smaller stored grid. Cells in the image plane are wrapped into cells in the grid, reducing the memory footprint of this portion of the data structure [110]. The mapping function is described by Equation 4.1, and has several key properties. It avoids folding a region of high sample density in the image plane back onto itself in the grid, inhibiting further concentration of samples in hot spots. This mapping also preserves a high degree of spatial locality. Points nearby in the image plane tend to remain nearby in the grid. This locality is exploited during irregular rasterization by tiling the grid in memory and processing fragments in tile order [94]. Note however that the mapping function can place receiver points from multiple independent regions of the image plane into the same linked list.



(a)
Light-view acceleration structure.
(2D Wrapped Grid)

(b)
Sample storage in memory.

Figure 4.2: A spatial acceleration structure used for shadow rendering, in which the storage order of samples in memory is fixed. It consists of a 2D wrapped grid (a), where each cell stores the memory address of the head of a linked list containing the samples located in the cell. Note that a given list may contain samples from multiple independent regions of the image plane due to the wrapping function, and the order of the samples in a given list is undefined. Linked list nodes are stored contiguously in memory in a single 1D array indexed by the positions of the respective pixels in the eye-view raster (b). Each node contains the image plane coordinates of the sample (32-bit), its depth (32-bit), and a tail pointer to the next node in the linked list (32-bit). This acceleration structure is fast and simple to build and traverse on GPUs circa 2005 with the incremental modifications discussed in Section 5.4.

$$\begin{aligned} \text{cell}_I &= (|\,\text{cell}_i| + \left\lfloor \frac{\text{cell}_j}{\mathbf{w}_{grid}} \right\rfloor * \mathbf{w}_{tile}) \;\%\; \mathbf{w}_{grid} \\ \text{cell}_J &= (|\,\text{cell}_j| + \left\lfloor \frac{\text{cell}_i}{\mathbf{w}_{grid}} \right\rfloor * \mathbf{h}_{tile}) \;\%\; \mathbf{w}_{grid} \end{aligned} \tag{4.1}$$

Linked list nodes are stored together in a single contiguous 1D array as seen in Figure 4.2b. The storage order matches that of the corresponding pixels in the eye-view raster (Subsection 4.1.2). Since the pixel storage order is set at compile time, the storage order of points in the 1D array is similarly static and need not change frame-to-frame, simplifying data structure construction[2]. Further, as we'll see in Section 5.4, the absence of per-frame sorting reduces the hardware features necessary to support high performance construction. We discuss an implementation of this data structure for the irregular Z-buffer architecture in Section 5.5 and evaluate the performance in Section 6.4.

4.2.2 A Spatial Acceleration Structure with Light-View Indexing

To review, eye-view indexing exposes some spatial reuse. However, maximizing reuse requires sorting the receiver points in memory based on their relative positions in the light view, since the data structure is traversed from the point of view of the light rather than from the eye. One example of such a spatial acceleration structure is seen in Figure 4.3. Conceptually, it is a 3D perspective grid where each cell stores the receiver points located within its bounds. In practice, these points are stored contiguously in a separate 1D array indexed by the grid. The order of the points in this array changes per frame in dynamic scenes in response to changes in the position of the eye, light, and / or geometry. Points from the same 3D grid cell are stored (unordered) in consecutive memory locations, followed by points from grid cells in the same column, followed by points from grid cells in neighboring columns. Note that a fragment from an occluding object can only shadow points in the grid with a depth greater than that of the fragment. Importantly, the set of points potentially in shadow are those in consecutive memory locations in the 1D array beginning at the

[2]The *links* between nodes do change per frame based on the relative positions of the respective points in the light view.

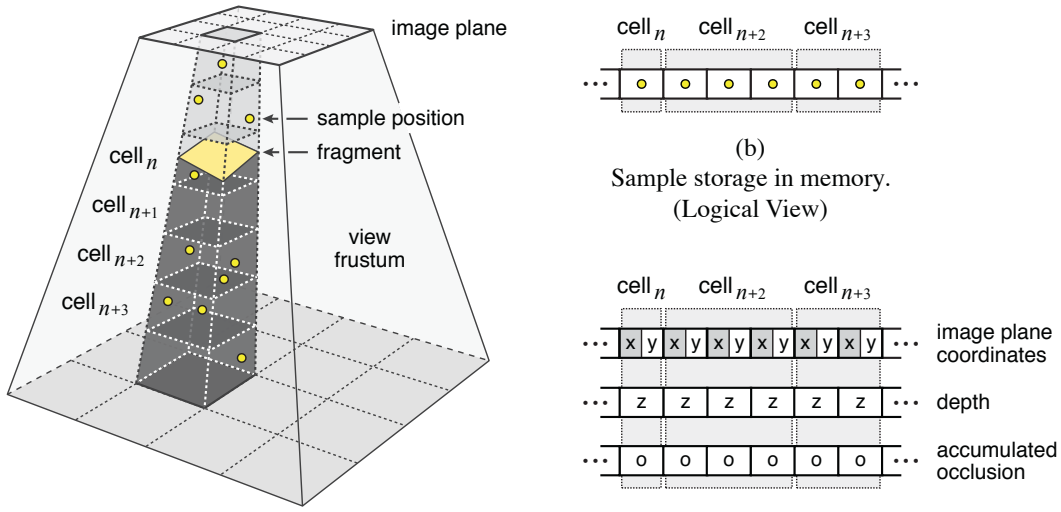


Figure 4.3: A spatial acceleration structure used for shadow rendering, in which the storage order of samples in memory changes frame-to-frame. It consists of a 3D perspective grid (a), where each cell stores the memory address of the first sample in the cell. Samples are arranged contiguously in memory in a single 1D array (b). Sample image plane coordinates (16-bit), depth (32-bit), and accumulated occlusion (32-bit) are stored in separate planes of this array (c). Samples from grid cells in the same column are stored in consecutive memory locations. For example, the samples from cell $_{n+2}$ follow those from cell $_n$ in memory (cell $_{n+1}$ contains no samples). This acceleration structure is fast and simple to build, requiring only a sort and a scan. It is also fast to traverse. For example, samples potentially occluded by the fragment shown are those in the 1D array in consecutive memory locations beginning at the address given by cell $_n$. This arrangement exposes spatial locality on cache lines and enables simple and efficient vectorization.

index given by the cell occupied by the fragment (cell$_n$). This storage order exposes spatial reuse of points on the same cache line, and maximizes the efficiency of vector memory operations during computation of occlusion, but requires hardware support for high performance sorting (Subsection 5.6.5).

The use of a 3D grid is motivated by the observation that depth information associated with a fragment can be employed to reduce the computation required by our shadow rendering algorithm. For example, the penumbra cast by a silhouette edge forms a wedge (Figure 3.1). Samples outside of this wedge are not in the penumbra and can be skipped. Implicit in this design choice is the assumption that samples are not constant in depth as seen from the light. This property commonly holds in game scenes (Figure 6.2b and c). We discuss an implementation of this data structure for the Larrabee architecture in Section 5.8 and evaluate the performance in Section 6.5.

4.3 Summary

The irregular Z-buffer algorithm is unusual among visible surface solutions for GPUs in its use of an explicit spatial acceleration structure for storing the coordinates of sample points. In this chapter, we have discussed design considerations related to these data structures as used in memory bandwidth intensive applications like shadow rendering. We have also presented a pair of practical acceleration structures for use with irregular shadow mapping. Other work has explored the use of tree-based structures [79, 80, 78] and grid-based structures similar to ours [107, 108] on GPUs. However, a key insight from our work is that hardware specialized to the task of rasterizing scene geometry to a uniform grid can be used to perform range queries on non-uniform grid-based spatial acceleration structures (Subsection 2.3.1).

Chapter 5

Architectural Support

The algorithms used in real-time graphics have traditionally been tightly coupled to the architectures that support them. Classical GPUs are single-chip parallel processors customized to the task of rendering arbitrary collections of geometric primitives via the standard Z-buffer algorithm (Figure 1.1a). High performance and low cost are achieved through parallelism, pipelining, and specialization. As such, these architectures contain a mix of fixed-function and SIMD programmable units, with multiple copies of each unit [70]. Contemporary trends in GPU design have favored increased programmability and replication of functional units, but the overall structure has changed surprisingly little over the past 20 years. As a result, the classical Z-buffer algorithm continues to prevail despite work indicating that more flexible solutions to the visible surface problem yield higher-quality images [32, 126, 125] and would be adopted if they could be implemented with adequate performance.

5.1 Methodology

To review, the classical Z-buffer algorithm samples scene geometry from regularly-spaced points in an image plane (Figure 1.2a). The irregular Z-buffer algorithm goes further by enabling samples to be computed at arbitrary locations in the image plane (Figure 1.2b). This flexibility is useful in several important applications throughout real-time graphics, but the algorithm is not real-time on existing CPUs [6] or classical GPUs [11, 120]. CPUs lack the necessary parallelism and memory bandwidth, and neither shortcoming can be addressed without significant architectural changes. In contrast, GPUs circa 2005 (when this work began) lack a comparatively small set of features related to memory access and computational flexibility. For this reason, our efforts to achieve real-time performance with the irregular Z-buffer algorithm focus on improving the architectural support in classical GPUs, and evaluating the performance of our algorithm on next-generation commercial designs.

5.2 Contribution

An important contribution of this work is an analysis of the architectural features needed for hardware-accelerated construction and query of irregular data structures, and (consequently) hardware-acceleration of the irregular Z-buffer algorithm. This analysis has two parts: *the design and evaluation of a new GPU based on incremental changes to graphics processors circa 2005, and evaluation of a next-generation commercial architecture expected by 2010.* GPUs circa 2005 lack efficient support for scatter operations and for most forms of read-modify-write memory operations. We show that the memory access flexibility enabled by these operations (along with other enhancements) enables high-speed construction and traversal of certain types of irregular data structures. Further, we show how additional architectural features expected to fully appear in GPUs circa 2010 enable high-performance manipulation of a large class of memory bandwidth efficient 2D and 3D irregular data structures.

5.3 GPU Architecture Circa 2005

Classical GPUs are specialized stream processors. This specialization yields high speed as well as architectural simplifications enabling rapid advances in performance and task-specific functionality, but limits the applicability of the GPU beyond the classical Z-buffer algorithm and similar types of computation. In this section, we briefly review the classical Z-buffer algorithm, the architecture of GPUs circa 2005, and limitations arising from the tight integration between the two.

5.3.1 Classical Z-Buffer Software Pipeline

Stream processing is a restricted form of parallel computing in which a program is expressed as a sequence (typically pipelined) of computational kernels which iterate over the elements in a data stream. This abstraction simplifies parallelization of the application, since synchronization, interprocessor communication, and resource allocation are implicitly managed by the architecture rather than explicitly stated in the program code.

The classical Z-buffer algorithm can be expressed as a pipelined sequence of computational kernels which iterate over a stream of data. This pipeline [19] is illustrated in Figure 5.1. The input to this pipeline is a geometric scene specification typically consisting of vertex data, connectivity information for the vertices, texture data, and light and camera descriptions. The output of this pipeline is commonly a colored and shaded image stored in framebuffer memory. The stages of this pipeline operate as follows. Data for a given vertex is brought together from multiple 1D channels of the input stream (e.g. colors, normals) in the *vertex assembly* stage. Vertices are subsequently transformed from the local object coordinate system into eye space, repositioned, lit, and / or shaded during *vertex shading*. The vertices for a given geometric primitive are further processed in the *geometry shading* stage. Here, per-primitive transformations and surface attributes (e.g. edge equations, Z

interpolation coefficients) are computed, before being passed onto the *rasterization* stage. In this step, perspective and other screen-space transformations are applied. Primitives are clipped to the viewport bounds and conditionally discarded via hierarchical early Z-culling. The remaining primitives are scan-converted into per-pixel fragments. Scissor or stencil testing may be performed on the resulting fragments. Per-pixel lighting and shading are then computed, often based on surface attributes interpolated at the point visible through the pixel covered by the fragment. One or more color values (along with a single depth value) are produced and passed onto the *raster operations* stage. Here, framebuffer memory associated with the current pixel is conditionally updated according to the outcome of a depth test.

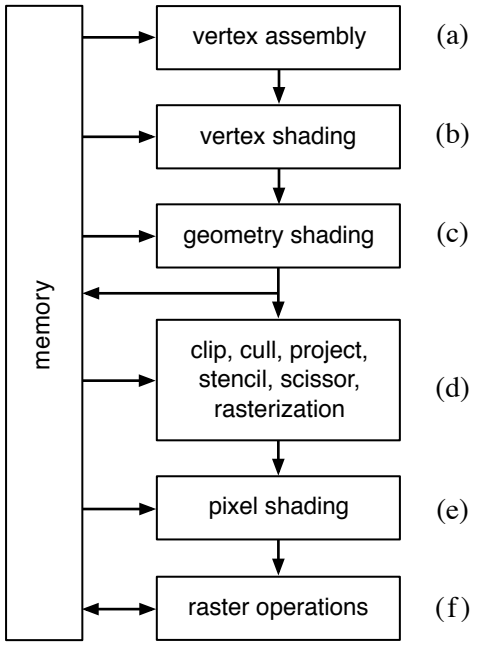


Figure 5.1: The classical Z-buffer algorithm can be expressed as a software pipeline. In commodity graphics hardware circa 2005, stages (a), (d), and (f) are performed via fixed-function units, while stages (b), (c), and (e) are implemented as user-defined programs which execute on multithreaded processors. The irregular Z-buffer algorithm modifies the behavior of stages (e) and (f).

5.3.2 Hardware Acceleration

The stream abstraction also reduces architectural complexity. Chip multiprocessors based on this model (e.g. GPUs circa 2005) require smaller caches, less control logic, and fewer connections between functional units than general-purpose multi-core CPUs. A key design goal is the exploitation of abundant thread and data parallelism rather than high-performance serial execution. GPUs in particular achieve this goal through replication of functional units at multiple scales, with tens (in some cases hundreds) of instances of each unit. These units can be subdivided into: fixed-function blocks for generic tasks like rasterization (Figure 5.1 a, d, and f), and programmable units for application-specific operations such as shading (Figure 5.1 b, c, and e). The programmable units are multithreaded and vectorized, share a program counter (i.e. are SIMD not MIMD), and have no direct write access to memory [70]. A typical instruction set [23] includes simple and compound arithmetic operations, graphics-specific functions (e.g. texture sampling), and rudimentary flow control with limited branching, but no synchronization or explicit atomic operations.

5.3.3 Limitations

GPUs circa 2005 perform well on computations with abundant parallelism and few data dependencies, as in the case of the classical Z-buffer algorithm, but are much less efficient at other types of computation. For example, the construction of an irregular data structure can require data dependent branching, explicit emission of outputs, atomic operations, and / or writes to addresses in memory determined at run time. The performance of these operations is inhibited by limitations of the architecture including: minimal connectivity between functional units, restricted access to memory, SIMD rather than MIMD programmable cores, and small local caches with no coherence mechanism. Note that these limitations result from the same architectural specializations that yield high efficiency in the case of the classical Z-buffer algorithm!

5.4 Irregular Z-Buffer Architecture

The irregular Z-buffer architecture is derived from GPUs circa 2005, and relaxes some but not all of the aforementioned restrictions through incremental design changes (i.e. no new functional units are added). In aggregate, the goal of these changes is to enable hardware-accelerated construction and traversal of irregular data structures such as the linked list grid from Figure 4.2. At the algorithm level, our design supports the ability to write to memory addresses computed at run time (i.e. a scatter operation), and multiple output records per input record. The scatter operation is key to spatial data structure construction, while multiple outputs per input is needed for data structure traversal during rasterization. At the hardware level, these changes are expressed as a more flexible routed interconnect between sets of functional units, MIMD rather than SIMD programmable units, and the addition of an instruction for explicit emission of outputs.

5.4.1 Primary Functional Units

Figure 5.2 illustrates the irregular Z-buffer architecture at a high level. The design provides efficient support for construction and traversal of irregular spatial data structures. The basic architecture is similar to that of the GeForce 6800 GPU [70] and Eldridge et al.'s sort-everywhere design [40], and includes 16 multithreaded processors with support for 4-wide vector operations, 16 raster operation units, and a 512-bit wide memory interface. The memory system includes four controllers each with two memory chips. The rasterizer is fed by a *geometry processor* (not shown), which can access data for all vertices of a given triangle.

Figure 5.3a illustrates the design of the primary computational unit. This unit is composed of a programmable core and an L1 texture cache, and is similar in design to that of shader processors circa 2005. The core is multithreaded (16-way) with a zero-cycle context switch, round-robin scheduling, and a single execution

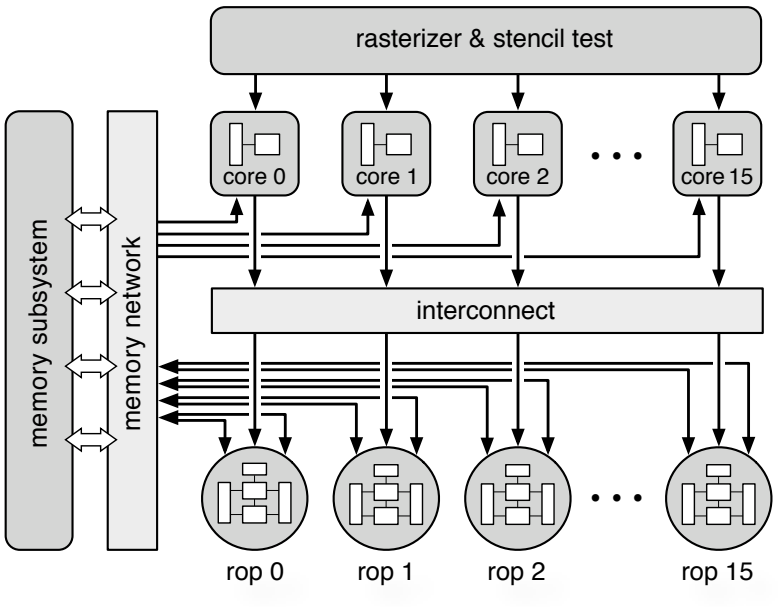
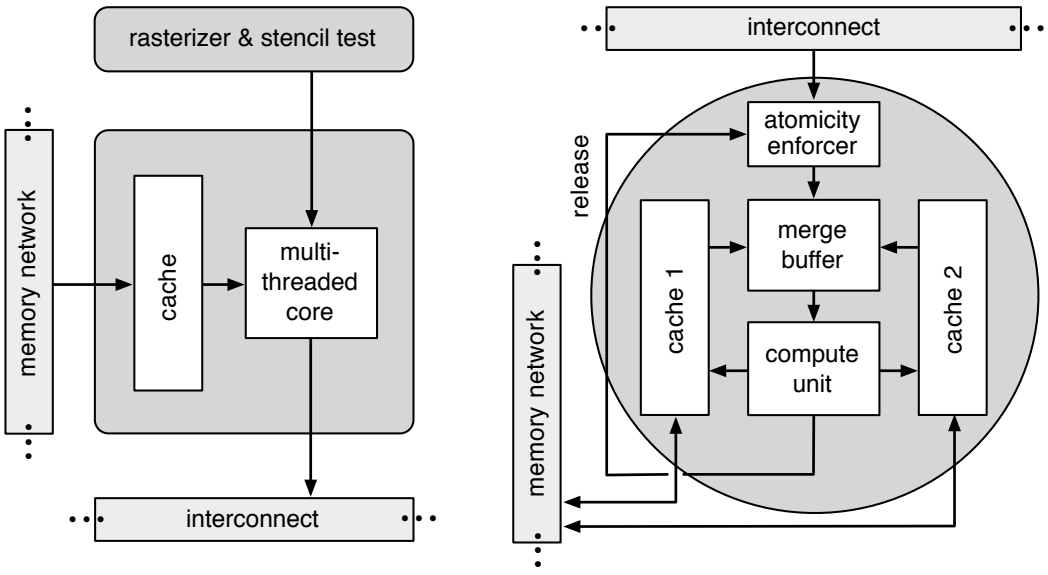


Figure 5.2: A high-level illustration of the irregular Z-buffer architecture. This design is composed of 16 programmable cores with support for multithreading and 4-wide vectorization, 16 raster operation units, and 4 memory controllers each with 2 DRAM chips. The data paths between the system components shown are all 16 bytes wide.

pipeline. Both scalar and 4-wide SIMD operations are supported. The instruction set largely follows the syntax and semantics of `NV_fragment_program2` [23], with arithmetic operations, logical shifts, conditional execution, and limited control flow including a branch instruction.

Figure 5.3b illustrates the design of our raster operation (ROP) unit. As in classical GPUs, it is a fixed-function unit which supports a limited set of atomic read-modify-write operations on memory, such as those used in alpha blending and depth buffer updates. Similarly, our ROP also supports multiple simultaneous in-flight memory transactions. For this reason, the central challenge in its design is to preserve the atomic semantics of the read-modify-write operation. A key insight used in other GPUs is that atomicity need only be enforced per pixel (i.e. per



(a) Multithreaded processor. (b) Raster operation unit.

Figure 5.3: A block-level diagram of the primary computational units in the irregular Z-buffer architecture. The multithreaded main processor (a) is fully programmable and is equipped with a 16 KB, 8-way set associative, read-only cache. The raster operation unit (b) is a configurable fixed-function unit specialized for performing atomic read-modify-write operations on memory. It is equipped with dual read-write caches (16 KB, 8-way set associative).

memory address). Transactions to independent pixels may be arbitrarily intermingled. The *atomicity enforcer* insures the ROP has no more than a single transaction in progress for any particular pixel. This enforcer is similar to that described by VanDyke et al. [131], and maintains a table indexed by a hash of the pixel address, indicating whether a fragment is in-flight for a given hash. If a fragment is in-flight, further fragments for that pixel are stalled (in order) in a queue until the offending fragment clears, while fragments for other pixels continue to pass through the atomicity enforcer. A *release* signal notifies the atomicity enforcer when all memory transactions related to a given pixel have completed, at which point it is safe to begin processing another fragment for this pixel.

5.4.2 Features Not Found In GPUs Circa 2005

The irregular Z-buffer architecture incorporates several features not found in GPUs circa 2005. Two of the most important are the ability to write to addresses in memory computed on the fly, and generate an arbitrary number of output records per input to the multithreaded processor.

Multithreaded Processor

The programmable cores in GPU designs circa 2005 have several restrictions. These cores share a program counter, can emit no more than a single output record per input, and do not support global reduction operations. Further, the destination memory address for a given output record is defined upstream by the rasterizer and is immutable to the programmable core. The design of the multithreaded processor used in the irregular Z-buffer architecture relaxes these restrictions. It is a true MIMD design rather than a SIMD implementation of a MIMD instruction set [70]. Further, it includes the ability to generate an arbitrary number of output records per input, the ability to specify the destination memory address for each output record (subject to certain constraints discussed in Section 5.6), and support for minimum and maximum global associative reduction operations. Additionally, this processor has access to per-triangle data computed by the geometry processor, including the homogeneous equations describing the edges and Z interpolation coefficients [101].

Raster Operation Unit

Our raster operation unit design is more flexible than that in GPUs circa 2005. In particular, classical ROP units cannot issue writes to memory addresses other than that specified by the input record. Ours loosens this restriction and is able to write to addresses in memory computed within the ROP on-the-fly (i.e. a scatter operation). For example, this unit can write to locations in memory computed from

data read from the address associated with the input record. For this reason, a pixel cache architecture [48] rather than a coalesce buffer [128] is used. In principle, this capability leads to the potential for non-determinism as discussed in Section 5.6. In practice, the operation of the irregular Z-buffer algorithm as it is used for shadow rendering avoids non-determinism.

Interconnect

To improve both spatial and temporal reuse of cached data, memory addresses are statically mapped to ROP units. A fragment is routed to the ROP that "owns" the memory address of the data corresponding to the input record. The interconnect performs this task. It is an $m \times n$ network with internal buffering capable of accepting input from up to m processors and routing output to up to n ROP units each cycle. An interconnect with this degree of flexibility is generally unnecessary in classical GPU designs, where routing is straightforward. Addresses are statically partitioned in memory and in the rasterizer such that processors and ROPs can be connected one-to-one. ROPs and memory controllers are similarly connected. In the irregular Z-buffer algorithm, this routing scheme is unworkable. Consider construction of a grid-based acceleration structure for shadow rendering. The spatial relationship between pixels in the eye-view raster and the corresponding samples in the light-view image plane (Figure 4.1) is scene dependent and cannot be known a priori. Specifically, the light-view grid cell into which a given sample falls, is unknown until the sample is transformed into light space. As a result, a static partitioning of grid cell addresses relative to eye-view pixel addresses cannot be set in the rasterizer until *after* the construction computation is performed.

5.4.3 Cost of Additional Features

Estimating the incremental cost of the novel features of this architecture is difficult. We have not modeled our design at the RTL level, and there is insufficient published information regarding classical GPUs to permit an accurate comparison. However, we can identify several likely sources of increased cost. Most significantly, our MIMD processor requires a small per-core instruction cache and decode unit unnecessary in SIMD designs. Moreover, our $m \times n$ interconnect is more complex than the fast-path routing found in classical designs, and thus requires greater die area. Lastly, the dual-port pixel cache design of our ROP (see Figure 6.4 for configuration details) necessitates additional cache wiring and control logic.

5.5 Algorithm Mapping I

Recall that the irregular Z-buffer algorithm occurs in two phases. In the first phase, the image plane coordinates of the desired sample points are entered into a spatial acceleration structure. In the second, scene geometry is rasterized to the positions encoded in this structure. Here, we describe the role of our multithreaded processor and raster operation unit in data structure construction and irregular rasterization. This discussion occurs in the context of shadow rendering, but the operation of these units is likely to be similar in other applications of the irregular Z-buffer algorithm.

5.5.1 Data Structure Construction

Data structure construction is a two part process. First, the light-space coordinates of points in the scene seen from the eye (i.e. receiver points) are computed by the multithreaded processor. Second, these points are inserted into a spatial acceleration structure by the raster operation unit. The data structure used here is the 2D grid of linked lists seen in Figure 4.2, in which the storage order of receiver points in memory is determined by the order of the corresponding eye-view pixels (Subsection 4.2.1).

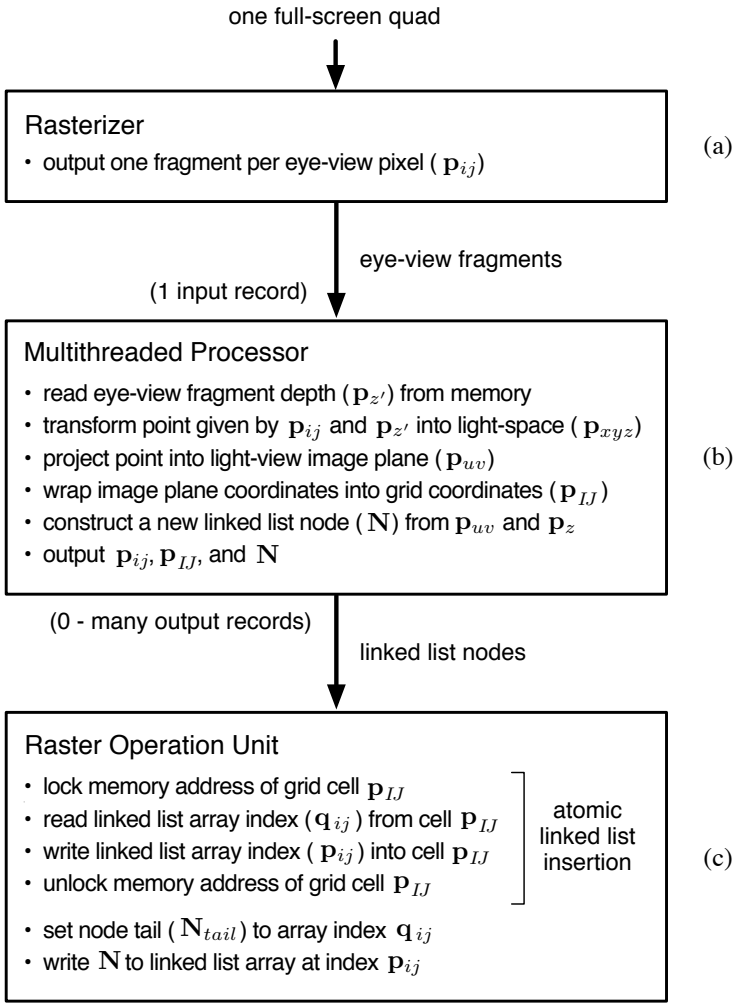
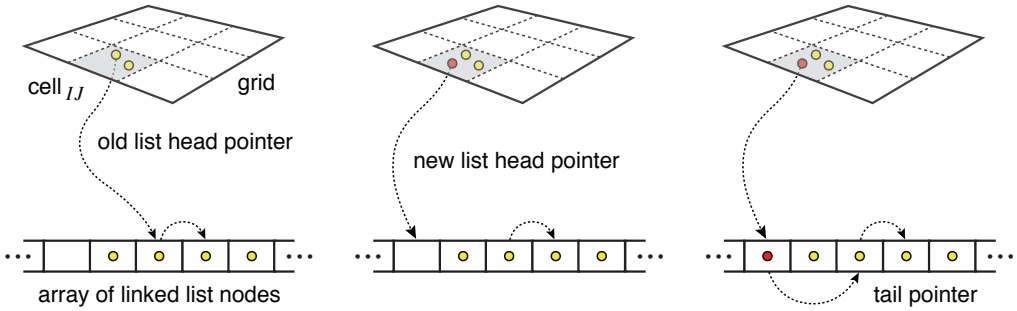


Figure 5.4: The construction phase of the irregular Z-buffer algorithm as it is used for shadow rendering on our architecture. Samples are inserted into a grid of linked lists for later use during rasterization. Linked list nodes are stored together in a 1D array indexed by the positions ($\mathbf{p}_{ij}$) of the respective pixels in the eye-view raster.



(a) Data structure before insertion.

(b) Atomic update of linked list head.

(c) Sample written and tail pointer set.

Figure 5.5: The raster operation unit in the irregular Z-buffer architecture can be used to prepend a sample point onto a linked list (a). Insertion consists of an atomic update of the pointer to the head of the linked list (b), and storage of the node data and tail pointer (c). All linked list nodes are stored contiguously in a single 1D array. Here, only links between nodes in the list affected by this insertion are shown.

In the case of the multithreaded processor, a simple program determines the coordinates of a receiver point ($\mathbf{p}$) from the indices of the corresponding pixel in the eye-view raster and its depth value (Figure 5.4b). This point is transformed into light space, and the grid cell into which it falls is determined. A new linked list node ($\mathbf{N}$) is constructed from the light-space coordinates of the point, and sent on to a ROP unit, along with a destination address in memory. This address is the memory location of the respective grid cell. The `min()` and `max()` reduction operations described in Section 5.4.2 are used to track the bounding box of points in the light-view image plane. These bounds are used during rasterization for viewport clipping and as the extents of a stencil mask for early rejection of unnecessary fragments.

The raster operation unit inserts nodes emitted by a multithreaded processor into the spatial acceleration structure (Figure 5.4c). Each node is prepended onto the linked list of the specified grid cell. This process is seen in Figure 5.5, and occurs in two steps. First, the pointer to the head of the linked list is atomically updated to the destination memory address of the new node. Next, the tail pointer of the new

node is set to the previous head of the linked list and the node contents are written to memory. Here, the ability to generate writes to memory addresses determined on-the-fly is crucial. Observe that this second write is to an address included in the data sent by the multithreaded processor *but is not the destination memory address specified by the processor* (i.e. address of the grid cell).

5.5.2 Irregular Rasterization

Irregular rasterization is similarly a two part process. First, the points stored in a given linked list are tested against the bounds of a geometric primitive. If the primitive occludes a point, a depth value is computed. Second, this computed depth value is compared against the respective value in memory, and the stored value is conditionally updated. These two steps occur on the multithreaded processor and raster operation unit respectively.

In the case of the multithreaded processor, a simple program traverses the linked list of a grid cell associated with a fragment produced by the rasterization unit (Figure 5.6c). For each list node, the coordinates of the point stored in the node are tested against the edge equations [106, 101] of the geometric primitive undergoing rasterization. If a point is occluded by the primitive, the Z interpolation equation for that primitive is evaluated at the coordinates given by the point. The resulting depth value, along with the address of the eye-view pixel corresponding to the point, are passed onto the ROP. It is here that the ability to generate multiple output records per input (since each node may result in one output), and specify the destination address of each output are needed.

The raster operation unit compares the depth values emitted by the multithreaded processor to the corresponding values in memory (Figure 5.6d). The stored value is updated if the incoming value is closer to the light. The novel ability of our ROP unit to write to arbitrary addresses in memory is unnecessary here. Rather, its operation in this phase of the algorithm is very similar to that in classical GPUs.

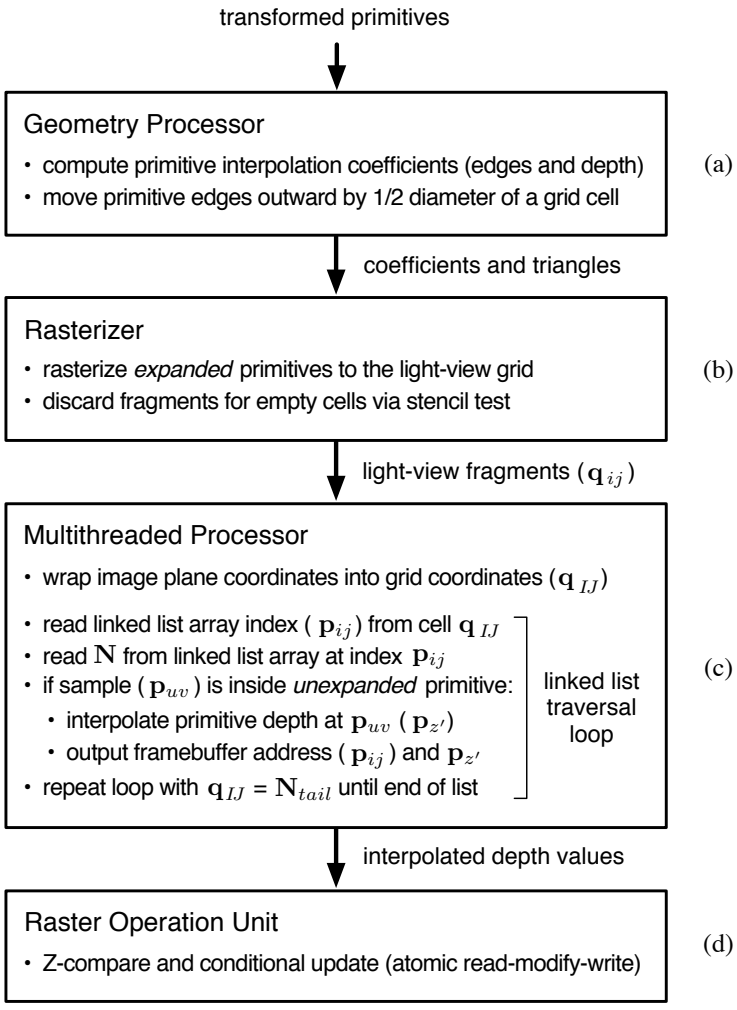


Figure 5.6: The rasterization phase of the irregular Z-buffer algorithm as it is used for shadow rendering on our architecture. Geometric primitives are rasterized from the light. For each fragment produced by the rasterizer, the multithreaded processor queries the spatial data structure to determine which samples are occluded by the primitive. Primitive expansion (a) is explained in Figure 2.4.

5.6 Irregular Z-Buffer Architecture Discussion

In principle, the architectural changes expressed in the irregular Z-buffer GPU can be used in a manner which leads to race conditions and non-determinism. We discuss this property here as it relates to atomicity and the preservation of fragment order. Further, we discuss the impact of these architectural changes on memory allocation and on the applicability of optimizations commonly found in classical GPU designs, as well as limitations arising from the incremental nature of these changes.

5.6.1 Atomicity

Recall that a key feature of our ROP unit is its ability to write to addresses in memory computed from input data (typically associated with a pixel or a grid cell in the spatial acceleration structure). These indirect writes occur after the pixel or cell address has cleared the atomicity enforcer. As a result the addresses of the writes cannot be known (and thus protected) by the enforcer, potentially leading to race conditions and non-determinism. It is the responsibility of the software driver or programmer to avoid such cases.

An example of such an indirect write occurs in the course of data structure construction in the irregular Z-buffer algorithm. During assembly of the the linked list grid as seen in Figure 5.5, the ROP issues two writes. The first updates the value stored at the incoming memory address (the address of the node at the head of the linked list). The second writes the contents of the new linked list head node. The address of this node is computed from data provided by the multithreaded processor, and is unknown to the atomicity enforcer. However, a property of this phase of the algorithm is that each node is written only once (i.e. each sample is inserted into the data structure once). Moreover, our ROP caches support selective writes. Only dirty sub-regions of a block are written out to memory. As a result, though a given linked list node may exist in multiple cache lines on the same ROP

(or in different ROPs), each node is guaranteed to be written to memory only once. Although avoided in this example, non-determinism cannot be guaranteed in all cases. By exposing the irregular Z-buffer algorithm through a high-level OpenGL / DirectX API instead of as a set of low-level ROP capabilities, the driver can insure that the machine is never used in an unsafe manner.

5.6.2 Fragment Order

Real-time graphics APIs specify precise ordering semantics for non-commutative operations such as alpha blending and depth-buffered color writes. In some cases these semantics are directly useful to application programmers, but they are also important to guard against non-determinism frame-to-frame or from one hardware generation to another. Fortunately, *order does not matter* when using the irregular Z-buffer algorithm to generate shadow or other depth-only maps. For example, the order of the samples in each grid cell of the spatial acceleration structure (Figure 4.2) is unimportant and opaque to the user if the data structure is hidden behind a high-level API as suggested in Subsection 5.6.1. Similarly, fragment order is invisible to the user during irregular rasterization since the Z-compare and update operation is commutative (assuming no auxiliary information is carried with it such as color). However, the irregular Z-buffer can be used in applications where color *is* carried with Z (e.g. reflection map generation). For these applications, the preservation of fragment order during rasterization matters.

Preserving order in a parallel architecture can be difficult. With current API definitions, architectures can maintain order by processing fragments in SIMD lockstep. In our architecture, this solution is impractical since our multithreaded processors can compute the destination memory address of output fragments on the fly, and produce a variable number of outputs per input. In principle, preserving fragment order would require an expensive, global order enforcement mechanism.

However, the irregular Z-buffer algorithm does not require the full generality of such functionality. For example, during irregular rasterization we can guarantee that the set of memory addresses written to as the result of a given fragment, are unique to the light-view grid cell corresponding to that fragment. Thus, by routing fragments to processors by grid cell indices (Subsection 5.4.2), the global ordering problem is simplified to one of enforcing order locally within the execution pipeline of each processor. Here, a small table of locks hashed by memory address suffices (since order is similarly preserved in the ROP units). Again, to guarantee that programmers cannot implement non-deterministic code, some of the basic hardware functionality must be hidden behind higher-level APIs.

5.6.3 Memory Allocation

The irregular Z-buffer algorithm, as applied to shadow rendering, has the unusual property that no explicit memory allocation is performed even though a dynamic data structure is used. Explicit allocation is unnecessary as storage is implicitly reserved for each light-view sample. The reserved memory is associated with the eye-view pixel corresponding to the sample. This strategy is feasible due to the one-to-one correspondence between eye-view pixels and light-view samples. The advantages of avoiding explicit memory allocation include freedom from potential serialization bottlenecks in the memory allocator, and reduced chance of unbalanced allocation across memory partitions. However, there is an important disadvantage. Since the storage location of a sample (in the acceleration structure from Figure 4.2) is associated with the position of the respective pixel in the eye-view, it is *not* associated with the position of the sample in the light-view. As a result the any-to-any routing capability of the interconnect is required for light-view irregular rasterization during shadow rendering (Subsection 5.4.2). Alternatively, memory can be explicitly allocated for each sample. Here, the sample is placed in the memory

partition associated with the region of the light-view containing the sample point. This design eliminates the need for an any-to-any interconnect during irregular rasterization, but such a network is then required during construction of the spatial acceleration structure.

5.6.4 Optimizations

Our architecture does not include some of the more sophisticated optimizations used by classical GPUs such as hierarchical Z-culling [49] and framebuffer compression. These optimizations are not well documented in the open literature and they are challenging to implement effectively even for large industrial architecture teams. Though we have not implemented these optimizations in our hardware simulator, we have given them some consideration. In particular, we believe that hierarchical Z-culling is fully compatible with the irregular Z-buffer algorithm.

Hierarchical Z-culling reduces unnecessary pixel shading and Z-compare and update operations by discarding fragments or entire primitives known to be hidden from the view point by other geometric objects. This optimization works by tracking the depth bounds of samples within a subregion of the image plane and discarding fragments with a minimum depth greater than the maximum depth of any sample in the subregion. Hierarchical Z-culling can be implemented in the irregular Z-buffer algorithm down to the level of a grid cell in our spatial acceleration structure. During irregular rasterization, if a fragment is generated for a grid cell and the respective triangle fully covers the cell, the depth bounds of the cell are recomputed. The new value is the minimum of the maximum depth of all samples contained in the cell and the maximum triangle depth at the cell corners. This result is passed on to the hierarchical Z subsystem.

5.6.5 Limitations

At the algorithm level, the primary limitation of the irregular Z-buffer architecture is the unavailability of support for global atomic operations on arbitrary memory addresses. Though atomicity is guaranteed locally within a given ROP, it is not guaranteed across ROP units, nor between multithreaded processors and ROP units. At the hardware level, this shortcoming results from the presence of small local caches throughout the design and the absence of a cache coherence mechanism.

The lack of global atomicity limits the types of irregular data structures that can be built efficiently. Specifically, *it is impractical to construct data structures in which the storage order of member elements in memory is determined at run time.* Recall that the storage order of samples in the spatial acceleration structure used above is fixed. Specifically, the storage order of samples in the array of linked list nodes seen in Figure 4.2b matches the storage order of pixels in the eye-view raster. This order is fixed at compile time and so cannot vary frame-to-frame. In contrast, consider the construction of a spatial acceleration structure for a set of samples, in which the storage order is defined at run time by a radix sort of the sample coordinates. The appropriate insertion point for a given sample can be computed by a multithreaded processor, but the actual insertion must occur in a ROP since the processor has no write access to memory (Figure 5.2). Therefore, the state of the data structure as seen by the processor may be stale due to in-flight memory transactions produced by a ROP, potentially leading to non-determinism.

5.7 GPU Architecture Circa 2010

As of this writing, the feature set of modern GPUs has reached parity with and in some cases exceeded that of the irregular Z-buffer architecture. For example, the Xenos GPU from AMD supports a scatter operation [16], and global atomic

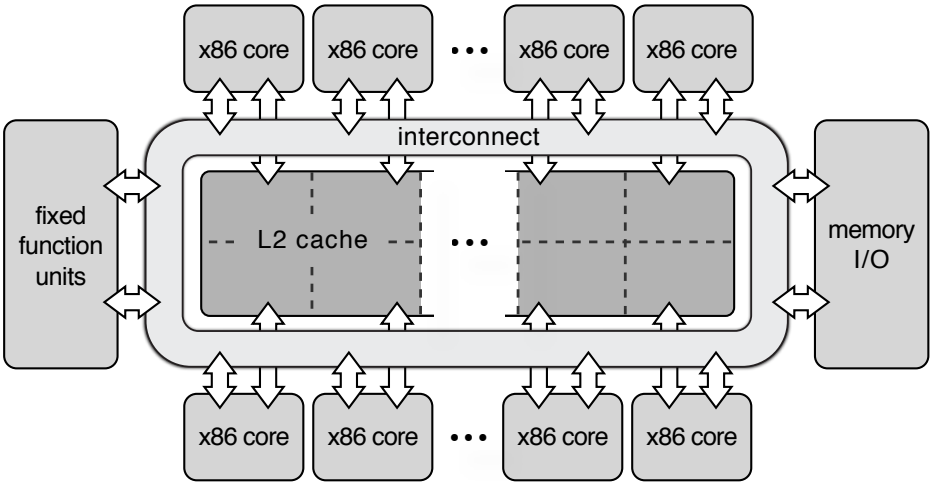


Figure 5.7: The Larrabee architecture is a scalable, multi-core GPU design from Intel [118]. Its major components include: programmable x86-based cores with support for multithreading and 16-wide vectorization, large L2 cache, memory controller(s), ring-based interconnect, and fixed-function units for specialized high-performance memory operations such as texture filtering.

read-modify-write operations on memory are implemented in the GeForce GTX 280 from NVIDIA [30]. This trend away from the hardware specializations of the classical graphics pipeline and towards support for general-purpose parallel computation continues in next-generation GPUs. For example, the Larrabee architecture from Intel expected in 2009 or 2010 [64] features high-performance gather / scatter, global atomic operations, and global synchronization, with the addition of a large L2 cache and cache coherence [118]. We describe this architecture in brief and show how it can be used in the construction and traversal of irregular data structures in which data elements are reordered in memory to expose maximal spatial locality.

5.7.1 Larrabee Architecture

The major functional elements of the Larrabee architecture can be seen in Figure 5.7. The design incorporates a scalable number of programmable cores, on-chip L2 cache

and memory controllers, and fixed-function units for specialized high-performance operations on memory (e.g. texture filtering), joined by a bi-directional, ring-based, routed interconnect. The primary computational block is a general-purpose multi-threaded core derived from the Intel Pentium® line of CPUs. As such, it supports 32 and 64-bit integer and floating point scalar arithmetic and the Intel Pentium x86 instruction set, but is distinct in its short in-order instruction pipeline. Die area normally occupied by out-of-order control logic is instead devoted to 16-wide SIMD units. These units support 32-bit integer and 32 and 64-bit floating point arithmetic, conditional execution through vector element masking, and vector load / store and gather / scatter memory operations. The gather / scatter operations can load or store up to 16 data values from one source vector operand to non-contiguous addresses in memory specified by a second source vector operand. The memory hierarchy itself is fully cache coherent and is composed of L1 and a large partitioned L2 cache, and off-chip memory accessible through controllers distributed around the interconnect. L1 is shared among threads of the same core, while data sharing between cores is enabled by hardware-assisted communication across L2 partitions. All vector memory instructions operate through cache. As a result, the performance of gather / scatter is limited only by the speed with which the cache subsystem can fulfill the requests.

5.8 Algorithm Mapping II

To first order, implementation of the irregular Z-buffer algorithm on the Larrabee architecture is similar to that on our architecture. As before, the algorithm occurs in two phases: coordinates of the desired sample points are inserted into a spatial acceleration structure which is later queried during rasterization. However, the Larrabee implementation is distinct in two important respects: the storage order of samples contained in the acceleration structure is determined at run time, and

parallelism is managed explicitly in the implementation rather than implicitly by the architecture. The data structure used here is a 3D perspective grid with indices into a separate 1D array of samples (Figure 4.3). The storage order of samples in the 1D array is designed to maximize spatial reuse during rasterization, by placing sample points which are nearby in space nearby in memory (Subsection 4.2.2). This strategy replaces bandwidth inefficient operations such as gather / scatter, with simple and efficient vector loads and stores in the memory-intensive rasterization phase. The algorithm itself is implemented in C with compiler intrinsics for explicit creation of threads, synchronization, vectorization, and atomic operations. Our implementation can be invoked as a library routine from within the Larrabee graphics pipeline (Subsection 6.5.2), which is likewise implemented fully in software. Load balancing, assignment of threads to cores, and related tasks are handled by a conventional system-level thread scheduler.

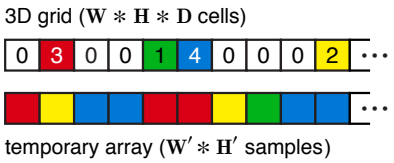
5.8.1 Data Structure Construction

The Larrabee implementation of data structure construction is more sophisticated than that on our architecture. Rather than a simple linked list insertion, sample points are ordered in memory based on their relative spatial positions. Specifically, a partial radix sort is performed on the sample coordinates as seen in the pseudocode in Figure 5.8. This code constructs a light-space acceleration structure for use in shadow rendering, but the procedure is likely to be similar in other applications. Construction occurs in three steps. First, the light-space coordinates of points in the scene visible from the eye are computed, and the number of samples which fall within each 3D grid cell is determined. Second, a linear sum converts these counts into offsets into the 1D array of samples. Third, the sample points are reordered in

[†]The data structure construction recipe described here was developed in collaboration with Warren Hunt and is used with permission.

foreach eye-view pixel ($\mathbf{p}_{ij} : i = \{0 .. \mathbf{W}'\text{-}1\}, j = \{0 .. \mathbf{H}'\text{-}1\}$)
- read eye-view depth ($\mathbf{p}_{z'}$) from memory
- transform point given by $\mathbf{p}_{ij}$ and $\mathbf{p}_{z'}$ into light space ($\mathbf{p}_{xyz}$)
- project point into light-view image plane ($\mathbf{p}_{uv}$)
- transform $\mathbf{p}_{uvz}$ into grid coordinates ($\mathbf{p}_{IJK}$)
- increment counter ($\mathbf{p}_N$) stored in cell $\mathbf{p}_{IJK}$
- write point $\mathbf{p}_{uvz}$ to temporary array at index $\mathbf{p}_{ij}$

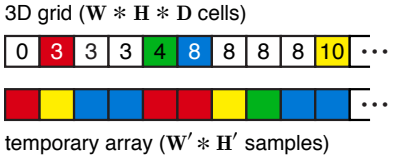
end foreach



(a) Transform visible points into light space and count per grid cell.

foreach grid cell ($\mathbf{q}_{IJK} : I = \{0 .. \mathbf{W}\text{-}1\}, J = \{0 .. \mathbf{H}\text{-}1\}, K = \{0 .. \mathbf{D}\text{-}1\}$)
- read counter ($\mathbf{q}_N$) from cell $\mathbf{q}_{IJK}$
- read counter ($\mathbf{q}_{N'}$) from cell $\mathbf{q}_{IJK\text{-}1}$
- write offset ($\mathbf{q}_N + \mathbf{q}_{N'}$) to cell $\mathbf{q}_{IJK}$

end foreach

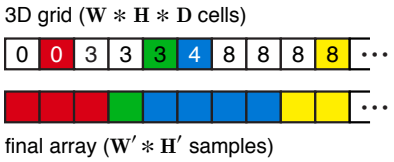


(b) Transform point counts into offsets into the array of samples via linear sum.

foreach light-view sample ($\mathbf{p}_{ij} : i = \{0 .. \mathbf{W}'\text{-}1\}, j = \{0 .. \mathbf{H}'\text{-}1\}$)
- read point $\mathbf{p}_{uvz}$ from temporary array at index $\mathbf{p}_{ij}$
- transform $\mathbf{p}_{uvz}$ into grid coordinates ($\mathbf{p}_{IJK}$)
- read offset ($\mathbf{p}_N$) from grid cell $\mathbf{p}_{IJK}$
- write point $\mathbf{p}_{uvz}$ to final array at index $\mathbf{p}_N$
- increment $\mathbf{p}_N$ and write to cell $\mathbf{p}_{IJK}$

] atomic update

end foreach



(c) Reorder sample points in memory and shift offsets in grid 1 cell to the right (not shown).

Figure 5.8: Pseudocode for the data structure construction phase of the irregular Z-buffer algorithm, as it is used for shadow rendering on the Larrabee architecture. Points in the scene visible from the eye are transformed into light space and inserted into the 3D grid-based acceleration structure seen in Figure 4.3. A partial radix sort on the point coordinates results in a storage order in memory such that spatial reuse is maximized during rasterization.

memory according to their positions in the 3D grid. Note that parallelization and synchronization directives are not shown. However, this procedure utilizes several operations that do not exist or are inefficient on GPUs circa 2005 and / or on the irregular Z-buffer architecture including: atomic increment, gather / scatter memory operations, and parallel prefix sum.

foreach fragment ($\mathbf{q}_{IJ}$) from *expanded* primitive
- determine grid cell ($\mathbf{q}_{IJK}$) intersected by fragment
- read offset ($\mathbf{q}_{N}$) from grid cell $\mathbf{q}_{IJK}$
- read offset ($\mathbf{q}_{N'}$) from grid cell $\mathbf{q}_{IJD}$

foreach sample array index ($\mathbf{p}_N : N = \{\mathbf{q}_N .. \mathbf{q}_{N'} - 1\}$)
- read sample $\mathbf{p}_{uvz}$ from array at index $\mathbf{p}_N$
- if sample ($\mathbf{p}_{uv}$) is inside *unexpanded* primitive:
  - interpolate primitive depth at $\mathbf{p}_{uv}$ ($\mathbf{p}_{z'}$)
  - if $\mathbf{p}_{z'} < \mathbf{p}_z$ sample $\mathbf{p}$ is occluded by primitive

end foreach
end foreach

Figure 5.9: Pseudocode for the irregular rasterization phase of the irregular Z-buffer algorithm, as it is used for shadow rendering on the Larrabee architecture. Scene geometry is scan-converted to the front face of the 3D grid acceleration structure seen in Figure 4.3, via classical rasterization. The list of samples in the column of grid cells beneath a given fragment are tested for occlusion against the geometric primitive which produced the fragment.

5.8.2 Irregular Rasterization

During irregular rasterization, scene geometry is evaluated for occlusion against the sample points encoded in the spatial data structure. The Larrabee implementation of this process is similar to that on our architecture. The main difference is that linked list traversal is no longer needed due to the storage order of the samples in memory. Rather, the set of samples potentially occluded by a fragment are located at consecutive addresses, leading to simple and efficient vectorization. Pseudocode for the Larrabee implementation is seen in Figure 5.9. As before, the input to this process are fragments produced by a traditional rasterizer (in this case the Larrabee software rasterizer), by scan-converting primitives to the front face of the 3D grid. Depth information associated with a fragment is used to determine the intersection between it and the column of grid cells beneath the "pixel" for which the fragment was generated (Figure 4.3a). Only samples at this point and deeper in the grid are tested for occlusion, using a computation similar to that described in Subsection 5.5.2.

The partial radix sort performed during data structure construction orders samples in the 1D array first in Z and then in X and Y. Thus, samples evaluated for occlusion against a fragment during rasterization, occupy consecutive array locations beginning at the index stored in the 3D grid cell intersected by the fragment. As a result, the occlusion computation is trivially vectorizable across samples under the same fragment. Only simple vector loads and stores (versus gather / scatter), and vector arithmetic operations are needed. Further, spatial reuse of samples on the same cache line is maximized. Thread-level parallelism is also straightforward: fragments for different pixels are assigned to different threads. This parallelization strategy is not specific to Larrabee. For example, the same algorithm could be implemented in a high-level API like CUDA [100] for NVIDIA GPUs, though the relative performance is unclear due to significant architectural differences.

5.9 Summary

In this chapter we have shown how support for arbitrary sampling patterns can be implemented on two distinct architectures. In the case of the irregular Z-buffer GPU, our algorithm is implemented in part in hardware. Other work has examined similarly specialized modifications to classical GPUs for non-uniform sampling. For example, the SAGE architecture [36] supports irregular sampling patterns within a pixel for multisample antialiasing, and the per-pixel sample count is configurable. However, the sample count and sample pattern are constant from pixel to pixel. Other work has proposed changes to fixed-function rasterization units to enable reparameterization of the sampling space. Logarithmic [87] and perspective [123, 93, 141, 90] parameterizations have been proposed for improving the quality of hard shadows rendered with shadow mapping, but the sample count and pattern remain constant from pixel to pixel. In contrast, the irregular Z-buffer algorithm on our architecture supports an arbitrary number of samples per pixel with no restriction on the sampling pattern. We have shown how this property results in increased accuracy and efficiency when used for shadow rendering (Subsection 2.5.2).

In the case of the Larrabee architecture, the data structure construction and traversal phases of the irregular Z-buffer algorithm are implemented completely in software. Other work has explored the construction and traversal of semi-regular and irregular data structures on classical GPUs, including grid [107, 108] and tree-based [79, 80, 78] structures. However, the performance of these solutions is constrained by the capabilities of the GPUs of the day. As we'll see in the next chapter, the irregular Z-buffer algorithm on the Larrabee architecture achieves real-time frame rates during data structure construction, and real-time to near real-time frame rates during traversal in the case of hard and soft shadow rendering.

Chapter 6

Evaluation

Any new technique targeted at real-time graphics applications must run fast as well as produce images of the desired quality. At a minimum, a new method should achieve superior image quality with no loss of performance over existing methods, or achieve superior performance with no loss of image quality. However, assessing the image quality and performance of the irregular Z-buffer algorithm is challenging since it relies on hardware features not found in shipping GPUs as of this evaluation.

6.1 Methodology

The performance of an algorithm is typically measured in one of two ways: at a high level by counting arithmetic operations over a particular data set, or at a low level by implementing the algorithm on current hardware and timing its execution. Here, neither approach is adequate due to the absence of physical hardware. Memory hierarchy characteristics such as cache size and miss latency strongly influence the overall performance of memory intensive methods like irregular shadow mapping. In such cases, operation counting is insufficient to accurately estimate performance. Similarly, timing an implementation of the irregular Z-buffer algorithm on classical

GPU architectures would be uninformative due to a lack of support for features crucial to its performance (Section 5.7). For this reason, we evaluate the image quality and run-time characteristics of hard and soft irregular shadow mapping via detailed hardware simulation.

6.2 Contribution

Our evaluation is in two parts. First, we characterize the performance of hard and soft irregular shadow mapping at a high level (e.g. overhead, behavior in relation to scene-specific properties). Second, we evaluate the image quality and performance of the algorithm in particular scenes through simulation. The simulators used model the irregular Z-buffer architecture described in Section 5.4 and the Intel Larrabee architecture discussed in Subsection 5.7.1. The former provides low-level insight into the performance characteristics of irregular shadow mapping, while the latter supports a coarse comparative evaluation of image quality and performance against several existing state-of-the-art algorithms.

6.3 Performance Characterization

The simulation data in Section 6.4 and 6.5 illustrates the performance of irregular shadow mapping in specific scenes on specific architectures, but does not yield high-level guidance on the behavior of this algorithm in relation to other shadow mapping methods or in response to scene-specific properties. We provide this guidance here. First, we examine the overhead of irregular shadow mapping due to data structure construction, in comparison to shadow mapping methods which do not require an explicit spatial acceleration structure. Second, we characterize the performance of the algorithm in the context of several scene related parameters, such as the number of samples and the width of an area light.

6.3.1 Overhead

As we've shown in Chapter 2 and 3, the irregular Z-buffer algorithm can increase the efficiency of applications in which the desired sampling pattern is non-uniform. However, this efficiency comes at the cost of increased overhead. In particular, the construction of the spatial acceleration structure used in our algorithm introduces overhead not found in the classical Z-buffer algorithm. Recall that the latter does not require an explicit acceleration structure since the position of a pixel / sample in the raster can be computed from a simple formula. Here, we consider the overhead associated with data structure construction against the improvement in efficiency in the case of shadow rendering via shadow mapping.

To review, shadow mapping works by rendering the scene from the eye and light and comparing the two sets of sample points to identify regions of the scene in shadow. Shadow mapping based on the classical Z-buffer algorithm is prone to artifacts due to a mismatch between the eye and light sample patterns (Figure 2.2b). Variations of classical shadow mapping reduce these artifacts by approximating the desired sampling pattern through oversampling (i.e. increasing the resolution of the shadow map) in tandem with other techniques.

The number of shadow map samples necessary to substantially reduce the incidence of artifacts can be large. Lloyd et al. give partial bounds on the number required per eye-view pixel [86]. These bounds are accurate when the eye and light views are orthogonal, and are sufficient to avoid *perspective* aliasing (Figure 2.5), but not *projection* aliasing. These bounds are $O((f/n)^2)$ for classical shadow mapping, and $O(\log(f/n))$ for logarithmic perspective shadow mapping (Figure 2.6b, d, e), where n and f are the eye-view distances to the near and far clip planes. In irregular shadow mapping the number of shadow map samples per eye-view pixel required to avoid both perspective *and* projection aliasing is 1, regardless of the orientation of the eye and light views. Further, the added cost of data structure construction is small in comparison to the cost of the occlusion computation. The former averages 15% the cost of the latter for the cases discussed in Subsection 6.4.3.

n	Data Structure Construction	Hard Shadows	Soft Shadows
image resolution	O(n)	O(n)	O(n)
scene geometry	na	O(n)	na
shadow geometry	na	na	O(n)
area light width	na	na	O(n^2)

Table 6.1: The asymptotic performance of irregular shadow mapping is shown in relation to several scene-specific properties. The data structure construction and hard and soft shadow rendering phases of the algorithm are considered individually. Recall that the number of light-view samples is equal to the number of eye-view pixels, and the scene and shadow primitives are used only in the computation of umbral and penumbral occlusion respectively.

6.3.2 Performance Sensitivity

The performance of a given rendering algorithm typically fluctuates in response to changes in properties of a specific scene such as the number of geometric primitives or image resolution. Irregular shadow mapping is no different. Table 6.1 illustrates the sensitivity of the algorithm to eye-view image resolution, number and coverage of scene primitives, number of silhouette edges, and area light width.

Image Resolution

Recall that there exists a single light-view sample per eye-view pixel. Irregular shadow mapping is linear in the number of eye-view pixels and hence in the number of light-view samples. In principle, construction of a spatial acceleration structure requires sorting the coordinates of the elements to be stored (in this case samples). In practice, our data structure consists of a grid in which the samples within a given cell are unordered. This partial sort can be performed in O(n) time rather than the O($n \log n$) time required for a full sort. Similarly, the hard and soft shadow kernels consist of point sampling or area sampling geometric primitives. This operation is constant per sample per primitive, and thus occurs in linear time overall.

Scene Geometry

The performance of irregular shadow mapping depends in part on the number and image plane extents of primitives composing the scene. This geometry is used only in the computation of umbral occlusion, and does not play a role in data structure construction or in the calculation of penumbral occlusion. During the computation of umbral occlusion, the geometry is rasterized to the sample coordinates stored in the data structure. As in classical rasterization, this function consists of a point-in-primitive test, and is linear in the average depth complexity of the scene.

Silhouette Edges

Soft irregular shadow mapping generates a shadow primitive per silhouette edge, denoting the expected light-view screen-space bounds of the penumbra cast by the edge. The computation of penumbral occlusion consists of rasterizing this shadow geometry to the sample coordinates stored in the data structure. An area sample is computed per point (Subsection 3.3.3). This operation is constant per sample per primitive, and thus occurs in linear time overall.

Area Light Width

Though the performance of the penumbral occlusion computation is linear in the number of shadow primitives, the light-view screen-space extents of these primitives are quadratic in the width of the area light. A light source that is twice the width of another, produces shadow geometry that occupies four times the area in the light-view image plane. This quadratic can be problematic in scenes with very large light sources and is an issue common to soft shadow algorithms based on shadow geometry or shadow mapping. However, the size of a penumbra (and thus extents of the shadow primitive) also depends on the distance of the occluding object from the light. Many common light sources are either large and distant (e.g. sun) or small and comparatively near occluding geometry (e.g. light fixture in a room).

6.4 Irregular Z-Buffer Architecture Evaluation

Though the above characterization provides high-level insight into the sensitivity of irregular shadow mapping to changes in scene-specific properties, it does not yield low-level performance data for specific scenes on a specific architecture. To this end, we have developed a performance simulator for the irregular Z-buffer architecture as described in Section 5.4. It is similar in spirit to the C-model performance simulators used in industry, although it is not as detailed or as broad in scope. We focus on the key performance aspects of the architecture, specifically memory hierarchy and parallelism effects. The behavior of the memory system is modeled in greatest detail, the processor performance at a medium level of detail, and the fixed-function units such as the rasterizer at a functionality level only. Using this simulator, we find that hard irregular shadow mapping can achieve near real-time frame rates in game-like scenes on hardware which is conservative (e.g. in clock rate, memory bandwidth, core count) by 2005 standards.

6.4.1 Simulation Infrastructure

Our simulation environment for the irregular Z-buffer architecture has three parts: a supervisory program which models the functionality of an end-to-end classical Z-buffer system, an implementation of hard irregular shadow mapping, and a hardware simulator.

Supervisory Program

The supervisory program incorporates code for rendering colored and shaded images from a scene specification using the classical Z-buffer algorithm. This code is used to perform classical rasterization as needed during the course of irregular shadow mapping. The supervisory program builds a memory image for the simulated GPU memory system and invokes the simulator with the appropriate parameters for each

phase of our algorithm. Following simulation termination, the supervisory program retrieves the shadow map from the simulated framebuffer and uses the values in the computation of a final image. This handoff is necessary since the hardware simulator is not configured to perform classical Z-buffer rendering.

Irregular Shadow Mapping Kernels

Hard irregular shadow mapping is implemented here as a pair of kernels written in assembly. The two perform data structure construction and irregular rasterization as described in Section 5.5. This code is interpreted by the hardware simulator and "executes" on the programmable cores. The target instruction set follows the syntax and semantics of `NV_fragment_program2`, with several additional instructions for logical shifts and for selecting the mode (e.g. linked list insertion, Z-compare) of the ROP units. As in shader programs for circa-2005 GPUs, these kernels contain no explicit parallelization or synchronization instructions. Thread creation, scheduling, and synchronization is performed in hardware.

Hardware Simulator

Our hardware simulator is built on top of the Liberty Simulation Environment (LSE) [130]. LSE simplifies the development of modular, event-driven, cycle-accurate hardware simulators, and is composed of a structural specification language and compiler, and a behavioral specification language based on C. The former is used to specify the number, type, and interconnectivity of the machine components, and the latter the functionality of each of these components.

At a high level, our LSE machine specification matches the design shown in Figure 5.2. There is a one-to-one correspondence between the custom LSE modules in the simulator, and the functional units composing the programmable processor, ROP, interconnect, and memory network. At the functional level, the programmable

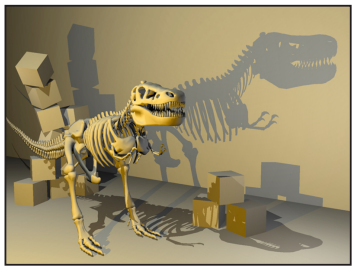
processor can issue a scalar or 4-wide vector operation each cycle, and all instructions except memory loads complete in one cycle. No effort is made to model stalls due to arithmetic unit latency. The ROP unit is pipelined. Each cycle it can issue up to two loads from the merge buffer and two stores from the compute unit, and retire a single fragment. Multiple fragments can be in-flight at the same time, subject to the atomicity restrictions discussed in Subsection 5.6.1. The memory system is based on Spinach (an LSE simulator for network interface controllers) [140], and the DSIM DRAM library [113]. We use the GDDR3 DRAM module from this library. The model honors all bank and channel timing restrictions from the manufacturer's data sheet [96].

We have made a reasonable effort to tune our simulator through application of several well-understood optimizations which reduce pressure on key resources, particularly memory bandwidth. The role of memory tiling and locality, as well as additional optimizations which may further improve the performance of our system are described in Subsection 4.1.2 and 5.6.4 respectively.

6.4.2 Workload

Our test suite consists of the scenes from Figure 6.1. Several characteristics of these scenes are summarized in Table 6.2. The *t-rex* scene is contrived, and is potentially challenging for our algorithm due to the large number of fine triangles composing the skeleton. The *Doom3* scene is from a game circa 2005, and light 0 is a particularly challenging case. Here, the light-view samples are primarily concentrated within a localized region of the image plane. This light sits near one wall which is visible over a large fraction of the eye-view viewport. With the light-view normal nearly parallel to the wall, the points on this wall visible from the eye project to a narrow band in the light view. This high concentration of points results in reduced utilization of the grid and correspondingly longer linked lists.

(a) Doom3 [id Software 2005].



(b) T-Rex.

Figure 6.1: Two scenes informing our analysis on the irregular Z-buffer architecture. The run time performance of hard irregular shadow mapping is examined in detail for two light sources in each scene. The Doom3 scene is from an actual game, and is challenging due to the sample distribution for the light source on the back wall. The t-rex scene is contrived but is challenging due to the large number of fine triangles.

6.4.3 Performance Results

The performance of irregular shadow mapping (hard shadows) on the irregular Z-buffer architecture for the scenes from Figure 6.1 is summarized in Table 6.2. These results do not include the cost of rasterization from the eye. The eye-view pass does not require or exercise our proposed architectural additions, our simulator does not support classical rasterization, and the performance of the classical rasterization computation is already well understood. Individual results for the data structure construction and irregular rasterization phases of the algorithm are seen in Table 6.3, for the machine configuration in Table 6.4. During data structure construction, utilization of the programmable cores is reduced due to transient load imbalances rather than memory-induced stalls. During irregular rasterization, a stencil test (Figure 5.6b) rejects fragments for grid cells in the spatial acceleration structure which contain no sample points. While we do not model this stencil buffer memory traffic, the additional bandwidth consumed can be computed from the columns in Table 6.5 marked *Rasterization*. The utilization of programmable cores during irregular rasterization is reduced by memory-induced stalls and indicates that the performance of the memory system bounds this phase of the algorithm.

Performance Summary

Scene	Triangle Count	Light Sources	Image Size	Total Samples	Sample Depth Complexity	Total Cycles	Hard Shadows
Doom3	7,581	2	1280 x 1024	2.6 M	4.1 / 4.3	45 M	11.20 fps
T-Rex	69,840	2	1280 x 1024	2.6 M	2.5 / 3.0	44 M	11.39 fps

Table 6.2: A summary of the simulation results for hard irregular shadow mapping on the irregular Z-buffer architecture given the scenes from Figure 6.1. The fifth column reports the average depth complexity per light-view sample per light source. The frame rates reported in the seventh column include the cost of data structure construction and irregular rasterization for the two light sources in each scene. These frame rates assume a clock frequency of 500 MHz. Note that the cost of the eye-view pass is not included. Simulation details are shown in Figure 6.3.

Construction

		Average Utilization / Stalls			Hit Rate			
Scene : Light	Time (Cycles)	Programmable Cores	ROP Units	Cycles / Sample	Core Cache	ROP Cache 1	ROP Cache 2	Bandwidth Utilization
Doom3 : 0	3.0 M	35.8% / 2.4%	2.7% / < 0.1%	2.4	93.8%	96.7%	45.1%	27.8%
Doom3 : 1	2.9 M	38.1% / 0.9%	2.9% / < 0.1%	2.2	93.8%	99.9%	48.7%	26.1%
T-Rex : 0	3.0 M	36.1% / 1.1%	2.7% / < 0.1%	2.3	93.8%	99.0%	45.9%	26.1%
T-Rex : 1	2.8 M	39.6% / 1.7%	3.0% / < 0.1%	2.1	93.8%	99.2%	46.6%	28.3%

Rasterization

		Average Utilization / Stalls			Hit Rate			
Scene : Light	Time (Cycles)	Programmable Cores	ROP Units	Cycles / Fragment	Core Cache	ROP Cache 1	ROP Cache 2	Bandwidth Utilization
Doom3 : 0	22.9 M	37.0% / 54.4%	1.4% / < 0.1%	4.3	59.7%	56.7%	57.6%	40.3%
Doom3 : 1	15.8 M	36.5% / 42.7%	2.2% / < 0.1%	2.8	63.5%	56.2%	56.6%	38.7%
T-Rex : 0	20.8 M	54.5% / 36.9%	1.0% / < 0.1%	6.3	73.7%	66.9%	64.9%	38.4%
T-Rex : 1	17.3 M	55.1% / 32.3%	1.1% / < 0.1%	5.5	74.0%	64.0%	64.7%	37.9%

Table 6.3: A detailed view of the performance summary in Table 6.2. Simulation results are shown for the data structure construction (top) and irregular rasterization (bottom) phases of our hard shadow algorithm on the irregular Z-buffer architecture. Columns 3 and 4 express the average time spent by the programmable cores and ROP units on useful work and average idle time due to memory latency induced stalls. Columns 6 through 8 report the hit rates for the three sets of caches in our system. The last column reports the DRAM bandwidth utilization as a fraction of theoretical maximum bandwidth.

Machine Configuration					Cache Configuration				
Clock Rate	Programmable Cores	ROP Units	Memory Controllers	DRAM Chips	Size	Line Size	Block Size	Assoc.	Hit Latency
500 MHz	16	16	4	8	16 KB	16 B	64 B	8	1 cycle

Table 6.4: The irregular Z-buffer architecture machine configuration simulated. All units including the memory interfaces are clocked at the rate shown, and all caches share the indicated configuration.

	Construction			Rasterization		
Scene : Light	Grid Dimensions	Non-Empty Cells	Samples / Cell (Min / Avg / Max)	Stencil Buffer Size	Fragments Pre / Post Stencil	Fragments Pre / Post Z-Compare
Doom3 : 0	512 x 512	21.3%	1 / 23 / 2195	553 x 501	1.5 M / 0.5 M	5.3 M / 1.0 M
Doom3 : 1	512 x 512	8.4%	1 / 59 / 1002	175 x 202	0.2 M / 0.2 M	5.7 M / 1.2 M
T-Rex : 0	512 x 512	25.3%	1 / 20 / 821	1346 x 407	0.8 M / 0.3 M	3.3 M / 1.3 M
T-Rex : 1	512 x 512	27.3%	1 / 18 / 1051	937 x 320	0.3 M / 0.3 M	3.2 M / 0.9 M

Table 6.5: Simulation details for irregular shadow mapping on the irregular Z-buffer architecture. The third column reports the percentage of non-empty grid cells as a fraction of the total cell count. The fourth column shows the (non-empty) minimum, average, and maximum linked list lengths. The stencil buffer sizes reported in the fifth column are computed via the `min()` and `max()` reduction operations across the programmable processors.

Irregular rasterization dominates the execution time of hard irregular shadow mapping. For the scenes tested, rasterization requires four to seven times the number of cycles as data structure construction. Although the cost of irregular rasterization depends strongly on the geometric and depth complexity of the scene, our timings indicate that the cost of data structure construction is secondary. This result is interesting since construction is more architecturally challenging to support due to the need for atomic read-modify-write operations during linked list insertion.

6.5 Larrabee Architecture Evaluation

The analysis in Section 6.4 yields low-level insight into the performance of irregular shadow mapping in specific scenes on the irregular Z-buffer architecture, but it does not provide guidance on the performance of our algorithm relative to other state-of-the-art approaches. Such a comparative analysis is impractical on this architecture due to the absence of support for classical rasterization and the primitive nature of the simulation environment (e.g. no higher-level graphics APIs). Moreover, GPU architectures and workloads have advanced since the development of our hardware simulator. For example, the Larrabee architecture expected in 2009 or 2010 includes high-performance support for construction and traversal of irregular data structures (Subsection 5.7.1). Though not yet in silicon, this GPU is modeled in a production level simulation environment with support for classical rasterization and high-level programming APIs. We have developed an implementation of hard and soft irregular shadow mapping for this architecture and use it to compare the image quality and performance of our algorithm against several existing methods. Using this simulator, we find that hard irregular shadow mapping can achieve real-time frame rates and outperform the fastest existing methods with similar image quality. Further, we find that soft irregular shadow mapping achieves performance comparable to the fastest existing methods while providing superior image quality.

6.5.1 Comparison Approach

Comparing the performance of irregular shadow mapping on Larrabee with that of existing methods on other architectures is challenging. Porting other algorithms to Larrabee (as of this writing) is impractical due to the prototypical nature of the software tool chain used. Alternatively, the published results for those algorithms could be normalized in relation to Larrabee based on FLOP counts or another metric of the respective architectures. However, such scaling is questionable as Larrabee

is a radically different GPU in the design and performance of its fixed-function units, programmable cores, ISA, interconnect, and memory hierarchy. Instead, we report competing results as published. As guidance on interpreting the results in Subsection 6.5.4 we note that the raw FLOP count of the Larrabee configuration simulated (24 cores at 1 GHz) is within a factor of 2 of recent GPUs. Moreover, this bias in favor of Larrabee is substantially reduced by our use of a prototype compiler and software rasterization pipeline. Neither is fully optimized.

6.5.2 Simulation Infrastructure

The Larrabee simulation environment has three elements: a software rasterization pipeline which models the functionality of the classical Z-buffer algorithm, a C/C++ reference implementation of hard and soft irregular shadow mapping, and a hardware simulator.

Software Rasterization Pipeline

The Larrabee software rasterization pipeline is a performance-oriented (but prototype) code for rendering colored and shaded images from a scene specification, using the classical Z-buffer algorithm. Its key features include: multithreading with minimal locking, vectorization, and a "sort-middle" pipeline structure [118]. When executed on the Larrabee hardware simulator, this code provides insight into the performance of conventional rasterization from the light, as it is used for coarse visibility testing in our algorithm. This code remains under development at Intel. As such, no effort has been made to integrate it with the reference implementation of irregular shadow mapping.

Irregular Shadow Mapping Reference Implementation

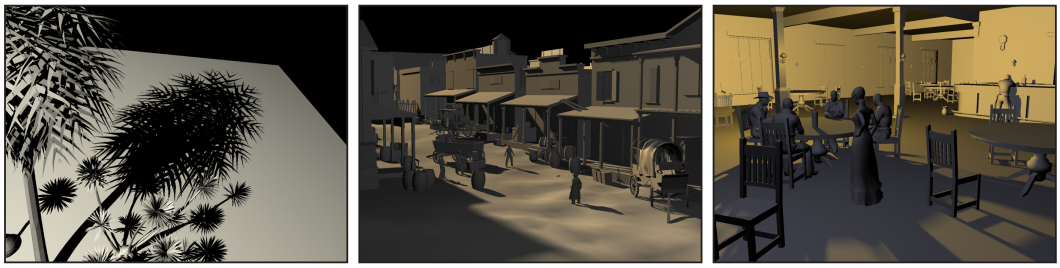
Our reference implementation of hard and soft irregular shadow mapping matches the description in Section 2.3 and 3.3. In addition to the routines specific to our shadow algorithm, the code contains a functionally complete (though unoptimized) Z-buffer based graphics pipeline capable of rendering a colored and shaded image from a scene specification. This code supports rapid evaluation of image quality and algorithmic correctness, and coarse performance assessment (e.g. operation counting and raw memory bandwidth estimation). Further, the data structure construction and hard and soft shadow kernels from this code have been hand vectorized and multithreaded using Larrabee intrinsics. The frame rates seen in Subsection 6.5.4 result from the execution of this code on the Larrabee hardware simulator.

Hardware Simulator

The Larrabee hardware simulator is a derivative of validated, cycle-accurate simulators used in the design of Intel multi-core CPUs. Different chip configurations can be modeled. The number of cores, threads per core, and clock rate are fully adjustable, as are properties of the memory hierarchy. The core count and the clock frequency of the actual Larrabee hardware have not yet been announced. Therefore, we simulate a conservative chip configuration (i.e. known to be within the Larrabee design envelope) with 24 cores and a 1 GHz clock [118].

6.5.3 Workload

Our test suite consists of the scenes from Figure 6.2 and 6.3. The *palm* and *fern* scenes provide a basis for image quality and performance comparisons with several existing hard [80] and soft [8, 43, 50, 116] shadow algorithms. These algorithms are considered to be state-of-the-art in terms image quality and performance, and are widely cited in the literature. Belying the substantial body of existing work



(a) Palm [Lefohn et al. 2007]. (b) Street (Call of Juarez). (c) Saloon (Call of Juarez).

Figure 6.2: The four scenes informing our performance analysis on the Larrabee architecture include the three shown here and the fern from Figure 6.3. The image quality and run time of irregular shadow mapping is compared against existing state-of-the-art approaches, using the palm and fern scenes. The street and saloon scenes reveal the performance of our algorithm in environments from a modern game. These two scenes are from Call of Juarez by Techland and are used with the permission of the developer.

in shadow rendering is the dearth of published performance results for scenes with geometric complexity comparable to that of modern computer games rendered at resolutions matching that of modern computer displays. For this reason, our test suite includes scenes from a modern game rendered at 1600×1200 pixel resolution. These scenes represent two environments with substantially different light-geometry relationships. The *street* scene is an exterior environment lit with a single light source positioned far from the geometry composing the scene, and is challenging for our algorithm due to the number of silhouette edges (Subsection 6.3.2). The *saloon* scene is an interior environment lit with a single light that sits within the eye-view frustum and is comparatively near the geometry composing the scene. This scene is a challenging case due to the size of the light relative to its average distance from the geometry, resulting in wide penumbrae as seen from the eye.

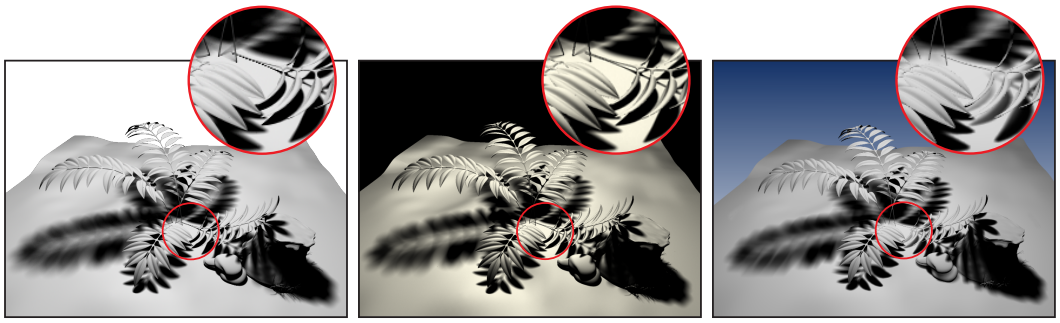
Scene	Triangle Count	Silhouette Edges	Image Size	Algorithm	Data Structure Construction	Hard Shadows	Soft Shadows
Palm	44K	N/A	1024x1024	[Lefohn et al. 2007]		40 fps	na
				[Johnson et al. 2008]	88 fps	60 fps	na
Fern	212K	13K	800x600	[Annen et al. 2008]		na	23 fps
				Fernando. 2005]		na	18 fps
				[Guennebaud et al. 2006]		na	41 fps
				[Schwarz et al. 2007]		na	19 fps
				[Johnson et al. 2008]	157 fps	72 fps	15 fps
Street	155K	29K	1600x1200	[Johnson et al. 2008]	70 fps	38 fps	27 fps
Saloon	60K	8K	1600x1200	[Johnson et al. 2008]	84 fps	61 fps	33 fps

Table 6.6: Simulated frame rates for our algorithm (highlighted in gray) on the Larrabee architecture equipped with 24 cores at 1 GHz, for the scenes from Figure 6.2 and 6.3. For comparison, results are reported for five other recent hard and soft shadow algorithms measured on a NVIDIA 8800 GTX part [8]. The frame rates for our algorithm include the cost of data structure construction (grid resolution of $800 \times 600 \times 8$) but not the computation of eye-view color, depth, or shading. These results were collected using a prototype compiler, thread scheduler, and software rasterization pipeline. Even so, they are within a factor of 2 of all but one other method. Further, our algorithm yields noticeably higher image quality than any other method (Figure 6.3) and with no parameter tuning necessary.

6.5.4 Performance Results

The performance of irregular shadow mapping on the Larrabee architecture for the scenes from Figure 6.2 and 6.3 is seen in Table 6.6. The hard and soft shadow frame rates include the cost of data structure construction. Similarly, the soft shadow frame rates include the cost of hard shadow rendering (used to compute umbrae). For the scenes tested, hard irregular shadow mapping substantially outperforms resolution-matched shadow mapping [80], while soft irregular shadow mapping is comparable in performance to all but one other method, yet produces substantially higher image quality than any other method (Subsection 6.5.5).

Note that these results do not include the computation of eye-view color, depth, or shading. These costs are well understood and are expected to be small. Further, the cost of two geometry shaders and screen-space binning [118] used during



(a) Ray traced shadows. (b) Irregular shadow mapping. (c) [Annen et al. 2008].

Figure 6.3: The image quality produced by ray tracing (a), our algorithm (b), and Annen et al. (c) compared. The inset highlights a region that is challenging for many algorithms to render accurately. A key feature of irregular shadow mapping is the high quality of the umbrae *and* penumbrae and the seamless transition between the two. The resulting images compare favorably with those produced by a ray tracer, simultaneously capturing high-frequency shadow details and smooth, low-frequency penumbrae. The image in (c) is generously provided by Thomas Annen and is used with permission.

light-view rasterization are not included due to a dependence on features not present in the prototype software graphics pipeline used. The geometry shaders identify silhouette edges and generate a shadow polygon for each such edge (Subsection 3.3.2). The cost of both is modest. For example, silhouette edge detection requires at most one cross product and one dot product per edge using adjacency information provided by DirectX 10 [19], and is only performed for triangles which pass face culling and frustum clipping.

6.5.5 Image Quality Results

In tandem with high performance, the visual quality of the umbrae and penumbrae produced by irregular shadow mapping distinguishes this algorithm from existing work. Specifically, the umbrae are geometrically-equivalent to those produced by ray tracing, and the penumbrae (though approximate) are both plausible and in many

cases indistinguishable from those of a ray tracer. These properties are illustrated in Figure 6.3[1]. As seen here and in Figure 7b - f from Annen et al. 2008 [8], the image quality of irregular shadow mapping is significantly higher than that of other algorithms in the same performance regime, including percentage closer soft shadows [43], backprojection soft shadows [50], bitmask soft shadows [116], and convolution-based methods [8].

Our algorithm computes penumbrae directly from silhouette geometry at exactly the points in the scene visible from the eye. The resulting images are free from artifacts due to undersampling a discretized representation of the scene (i.e. shadow map). For example, convolution-based methods can omit shadow umbrae in cases when an occluder is near a receiver [8]. This is due to the use of an average occluder depth at each receiver point, and the loss of frequency content resulting from the use of a low-precision, pre-filtered shadow map. In contrast, our algorithm simultaneously captures high-frequency shadow details and smooth, low-frequency penumbrae. Our method is approximate (Section 3.5), but the errors introduced are not widespread and are less visually apparent.

6.6 Discussion

Though hard irregular shadow mapping achieves real-time frame rates in the game scenes tested, more work will be required to achieve similar performance with soft irregular shadow mapping. In particular, there remain two important sources of inefficiency: overdraw from overly conservative estimation of penumbral bounds, and unnecessary computation of penumbral occlusion from silhouette edges which are themselves occluded. Both inefficiencies are common to algorithms in which

[1]Due to differences in camera models, the position of the light in (b) was hand tuned to match that in (a) and (c), leading to a slight variance. This variance produces minor differences in shading where the tree stump faces away from the light.

silhouette edges are represented geometrically (for increased accuracy) rather than stored in discretized form in a shadow map, and both are worthy of further study.

Recall that penumbrae are computed in two steps. The set of receiver points occluded by a silhouette edge is estimated first. The final occlusion computation is then performed on this point set. For correctness, the estimation step is conservative, but unnecessary work (i.e. overdraw) results from an overly conservative estimate. The optimizations described in Section 3.4 substantially reduce overdraw, but do not eliminate it. For example, in the saloon scene from Call of Juarez (Figure 6.2c) an average of 81% of the receiver points tested for occlusion against a given silhouette edge, are unoccluded by the edge. Conceptually, the estimation operation can be thought of as computing the intersection of two spatial acceleration structures, one which stores the receiver points, and one which encodes the extents of penumbrae. In our algorithm, the second acceleration structure is implicit, since penumbral bounds are estimated using shadow geometry on the fly. Other work has explored performing the estimation operation via intersection of two explicit acceleration structures [75] but the result is not real-time.

The second inefficiency occurs when the penumbra cast by a silhouette edge lies within the umbra cast by another occluder, as seen from a given receiver point. Conceptually, this problem can be thought of as pruning a spatial data structure encoding penumbral extents, according to another which encodes umbral extents. Soft shadow algorithms which compute occlusion from the light using a discretized representation of the scene (i.e. a classical shadow map), implicitly perform this operation. However, to our knowledge no work has considered this problem in the context of CSG operations on a pair of acceleration structures.

6.7 Summary

In this chapter, we have characterized the behavior of irregular shadow mapping at a high level in response to changes in scene-specific properties, and assessed the image quality and performance of the algorithm in several scenes via low-level simulation. Our approach is quadratic in the width of an area light (in the case of soft shadows), but linear in image resolution, scene geometry, and number of silhouette edges. The algorithm is memory bandwidth rather than compute bound on a GPU based on circa 2005 technology. On a GPU expected by 2010, hard irregular shadow mapping can achieve real-time frame rates and outperform the fastest existing method with similar image quality, while soft irregular shadow mapping provides superior image quality with performance comparable to the fastest existing methods.

Chapter 7

Conclusions

Traditionally, game developers have been confronted with a tradeoff between image quality and performance due to limitations of the visible surface solution employed by commodity graphics hardware. This hardware provides efficient support for the classical Z-buffer algorithm, resulting in high performance. However, the classical Z-buffer algorithm is not readily adaptable to the simulation of physically-accurate light transport. Consequently, the resulting images are of low quality in comparison to those produced by ray tracing. Unfortunately, ray tracing is not well-supported in existing CPUs or GPUs, resulting in low performance.

Achieving high image quality *and* high performance in dynamic scenes with high geometric complexity is an open area of research addressed in part by this work. We do so by adapting the classical Z-buffer algorithm and its underlying architecture to support ray tracing like functionality. Though our approach does not achieve the full flexibility of ray tracing, we show that it does enable substantial improvements in image quality while preserving the system organization and performance advantages of commodity graphics hardware.

7.1 Summary

The major contribution of this dissertation is as follows. First, we introduce a new solution to the visible surface problem that is more flexible than the classical Z-buffer algorithm, but less general than ray tracing. Second, we identify the architectural features needed for hardware acceleration of this algorithm. We subsequently show how the combined algorithm and architecture enable simple, efficient, and robust solutions to important but unresolved problems in real-time graphics like hard and soft shadow rendering. We demonstrate how shadows which are visually indistinguishable from those produced by ray or beam tracing, can be rendered with high performance with only minimal changes to GPU designs circa 2005.

7.1.1 Visibility Algorithm

This dissertation introduces the irregular Z-buffer algorithm, a solution to the visible surface problem for rays with a common origin and arbitrary directions. In the space of possible solutions, ours sits between the classical Z-buffer algorithm (suited to rays with a common origin and a regular pattern of directions) and ray tracing (suited to rays with arbitrary origins and directions). This middle-ground approach combines the system organization and performance advantages of the former, with some of the sampling flexibility of the latter. Like the classical Z-buffer ours is an object-order algorithm, and as such is compatible with existing game engines and commodity graphics hardware (with modification). Like ray tracing, our approach supports the computation of surface visibility along arbitrary ray directions represented as points in a 2D image plane. These points are defined by the application at run time and are stored in a spatial acceleration structure which is queried during rasterization. This conceptually-simple idea has potentially broad applicability throughout real-time graphics including in: shadow rendering, indirect illumination, frameless rendering, adaptive anti-aliasing, adaptive textures, and jittered sampling.

7.1.2 Architectural Support

The irregular Z-buffer algorithm cannot be implemented efficiently on CPUs or on GPUs circa 2005. CPUs have insufficient parallelism and memory bandwidth, while classical GPUs lack the memory access flexibility for gather / scatter operations, and hardware synchronization for efficient global atomic operations.

This dissertation introduces a new GPU design for hardware acceleration of the irregular Z-buffer algorithm. Our architecture is based on GPUs circa-2005, but includes several refinements. Key features of this design are: an any-to-any routed interconnect, MIMD rather than SIMD cores, an instruction for explicit emission of output, and scatter-type memory operations. In simulation we show that this GPU provides efficient support for the creation and traversal of irregular data structures in which the storage order of member elements is fixed at *compile time*. Separately, we show how inclusion of global atomic operations in this feature set in a GPU expected in 2009 - 2010 leads to acceleration of irregular data structures in which the storage order is determined at *run time*. Irregular data structures of both types are widely used in graphics, and the addition of hardware support significantly widens the class of algorithms which are able to run efficiently on GPUs.

7.1.3 Shadow Rendering

The irregular Z-buffer algorithm is motivated by applications in which the desired sampling pattern is non-uniform. Shadow rendering is one example. This problem consists of identifying points in the scene (i.e. receiver points) occluded from the light by intervening geometry. Such a determination can be made by rasterizing the scene from the light, but the optimal sampling pattern is highly irregular, requiring that the receiver point coordinates be stored explicitly in a spatial acceleration structure. Existing algorithms based on the classical Z-buffer avoid the need to store receiver points by computing occlusion from the light in eye space using shadow geometry

or a discretized representation of the scene (i.e. shadow map) as seen from the light. Shadow geometry methods do not perform well due to the number and extents of shadow polygons even when umbrae and penumbrae are small, while shadow mapping methods perform well but result in self-shadowing and aliasing artifacts.

This dissertation introduces irregular shadow mapping, an implementation of the irregular Z-buffer algorithm for computing hard and soft shadows in dynamic and geometrically-complex scenes. Points in the scene visible from the eye are stored in a light-space spatial acceleration structure. Geometry is then rendered from the light to these points. For hard shadows, a simple point-in-triangle test is performed. For soft shadows, primitives are clipped to a circular region centered on each point. The width of the region is proportional to the light diameter as seen from the point. Hard irregular shadow mapping mimics the shadow computation in a ray tracer, while the soft algorithm is similar to beam tracing. Though approximate, our soft shadow algorithm does not produce objectionable artifacts.

Irregular shadow mapping is distinct from existing Z-buffer shadow methods in three important respects. First, the shadow computation is performed *only* at points in the scene directly visible from the eye, resulting in high efficiency. Second, the shadow computation is performed *exactly* at these points, yielding high accuracy and directly addressing the root source of visual artifacts common to classical shadow mapping. Third, the high accuracy of the algorithm in turn eliminates the need for per-frame parameter tuning for image quality, resulting in a robust solution.

7.1.4 Results

Real-time rendering algorithms are evaluated on image quality and performance. In comparison to existing methods on a given scene, a new algorithm should achieve superior image quality with no loss of performance or superior performance with no loss of quality. We assess irregular shadow mapping as follows. Existing algorithms

on existing hardware (NVIDIA 8800 GTX) are compared with ours under simulation on the Larrabee architecture. Exemplars include: resolution-matched (hard) shadow maps [80], and convolution [8], backprojection [50], bitmask [116], and percentage closer [43] soft shadows. For the scenes tested, hard irregular shadow mapping is 50% faster than the existing method, with image quality equivalent to a ray tracer (i.e. the highest quality possible). Soft irregular shadow mapping is comparable in performance to all but one other approach, with substantially higher image quality than any other method in the real-time performance regime.

7.2 Potential Impact

As of this writing, Intel has devoted substantive resources to the implementation and support of the irregular Z-buffer algorithm on the Larrabee architecture. In addition, GPUs introduced since 2005 have begun to include many of the features proposed in the irregular Z-buffer architecture, though the specific implementation varies[1]. Beyond these near-term consequences of our work, we anticipate that elements of the irregular Z-buffer algorithm and its architecture have the potential to influence the design of future chip-multiprocessors and rendering algorithms.

7.2.1 Architecture

We believe that commodity graphics architectures are on a convergent course with general-purpose chip-multiprocessors (CMPs). As a result, features of the irregular Z-buffer architecture could be used to enhance the performance of future general-purpose CMPs. In particular, atomic read-modify-write operations can be handled by a dedicated unit near memory. In general-purpose parallel architectures, these

[1]To be clear, we do not take credit for the appearance of these features in modern architectures. That said, as one of several groups advocating such features, we have helped to make the case for their implementation in commercial designs.

operations are typically implemented near the processor, in L1 or L2 cache equipped with a cache coherence mechanism. Moving such operations to specialized units near memory seems broadly promising in instances where the result of the update is not immediately required by a processor. This idea has been explored in the context of streaming processors [4], but should be considered more fully by the CPU community in future CMP designs.

7.2.2 Rendering Algorithms

The architectural convergence mentioned above will result in hardware suitable for high performance ray tracing. However, we posit that future rendering algorithms will be neither strictly Whitted ray tracers [138] nor classical Z-buffer rasterizers [27], but will share properties of both. As such, insights expressed in the irregular Z-buffer algorithm have the potential to influence the design of these renderers. Evidence of this can be seen in current work. For example, irregular shadow mapping has been used to accelerate the computation of hard shadows in an offline ray tracer [18]. Less directly, a recent ray tracer described by Hunt and Mark focusing on hard and soft shadows [61] shares several elements with irregular shadow mapping including a similar acceleration structure, explicit representation of ray directions as arbitrary points in a 2D image plane, and ray-primitive intersection testing performed in 2D.

7.3 Final Thoughts

We posit that the gap in image quality and performance between classical Z-buffer rendering (comparatively low image quality but high performance) and ray tracing (high image quality but comparatively low performance) can be reduced through simultaneous advances in rendering algorithms and architectures. In particular, we have shown how some of the image quality benefits of ray tracing can be achieved with modifications to the classical Z-buffer algorithm. We have also shown how the

modified algorithm can achieve high performance on hardware developed through incremental changes to GPUs circa 2005. The resulting system is capable of rendering hard shadows equivalent to those of a ray tracer, at real-time frame rates in scenes from a modern game. The system is also capable of rendering soft shadows commonly indistinguishable from those of a beam tracer, at near real-time frame rates in game scenes. The two shadow algorithms strike a balance between image quality and performance unmatched in earlier work, validating our hypothesis.

Bibliography

[1] Matthew Adiletta, Mark Rosenbluth, Debra Bernstein, Gilbert Wolrich, and Hugh Wilkinson. The next generation of Intel IXP network processors. *Intel Technology Journal*, 6(3), August 2002.

[2] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 375–384, New York, NY, USA, 2000. ACM Press / Addison-Wesley Publishing Co.

[3] Jung Ho Ahn, William J. Dally, Brucek Khailany, Ujval J. Kapasi, and Abhishek Das. Evaluating the imagine stream architecture. In *ISCA '04: Proceedings of the 31st Annual International Symposium on Computer Architecture*, page 14. IEEE Computer Society, 2004.

[4] Jung Ho Ahn, Mattan Erez, and William J. Dally. Scatter-add in data parallel architectures. In *Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture (HPCA-11 2005)*, pages 132–142, February 2005.

[5] Timo Aila and Tomas Akenine-Möller. A hierarchical shadow volume algorithm. In *Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Confer-*

ence on Graphics Hardware 2004, pages 15–23, New York, NY, USA, 2004. ACM Press.

[6] Timo Aila and Samuli Laine. Alias-free shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2004*, pages 161–166. Eurographics Association, 2004.

[7] Tomas Akenine-Möller and Ulf Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *EGRW '02: Proceedings of the 13th Eurographics Workshop on Rendering*, pages 297–306, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[8] Thomas Annen, Zhao Dong, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Real-time, all-frequency shadows in dynamic scenes. In *SIGGRAPH '08: ACM SIGGRAPH 2008 Papers*, New York, NY, USA, 2008. ACM Press.

[9] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the AFIPS Spring Joint Computer Conference*, volume 32, pages 37–45, 1968.

[10] Jukka Arvo. Tiled shadow maps. *Computer Graphics International*, 0:240–247, 2004.

[11] Jukka Arvo. Alias-free shadow maps using graphics hardware. *Journal of Graphics Tools*, 12(1):47–59, 2007.

[12] Jukka Arvo, Mika Hirvikorpi, and Joonas Tyystjärvi. Approximate soft shadows using image-space flood-fill algorithm. *Computer Graphics Forum*, 23(3):271–280, 2004.

[13] Ulf Assarsson and Tomas Akenine-Möller. A geometry-based soft shadow

volume algorithm using graphics hardware. *ACM Transactions on Graphics*, 22(3):511–520, 2003.

[14] Ulf Assarsson, Michael Dougherty, Michael Mounier, and Tomas Akenine-Möller. An optimized soft shadow volume algorithm with real-time performance. In *HWWS '03: Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Conference on Graphics Hardware*, pages 33–40, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[15] Lionel Atty, Nicolas Holzschuch, Marc Lapierre, Jean-Marc Hasenfratz, Chuck Hansen, and François Sillion. Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum*, 25(4):725–741, December 2006.

[16] Dave Baumann. ATI xenos: XBOX 360 graphics demystified, June 2005. http://www.beyond3d.com/articles/xenos.

[17] Louis Bavoil, Steven P. Callahan, and Claudio T. Silva. Robust soft shadow mapping with depth peeling. SCI Institute Technical Report UUSCI-2006-028, University of Utah, 2006.

[18] Blender Foundation. Blender 2.43: Irregular shadow buffer, February 2007. http://www.blender.org/development/release-logs/blender-243/irregular-shadow-buffer/.

[19] David Blythe. The Direct3D 10 system. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 724–734, New York, NY, USA, 2006. ACM Press.

[20] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Practical shadow mapping. *Journal of Graphics Tools, Special Issue on Hardware-Accelerated Rendering Techniques*, 1st quarter 2003.

[21] Stefan Brabec and Hans-Peter Seidel. Single sample soft shadows using depth maps. In *Graphics Interface (GI 2002 Proceedings)*, pages 219–228, May 2002.

[22] Lynne S. Brotman and Norman I. Badler. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications*, 4:5–12, October 1984.

[23] Pat Brown and Eric Werness. GL_NV_fragment_program2 extension specification. May 2004.

[24] Ian Buck. Taking the plunge into GPU computing. *GPU Gems 2: Programming Techniques for High Performance Graphics and General-Purpose Computation*, chapter 32, pages 509–519. Addison-Wesley Professional, 2005. Matt Pharr and Randima Fernando (Eds.).

[25] Doug Burger, Alain Kägi, and M. S. Hrishikesh. Memory hierarchy extensions to the simplescalar tool set. Technical Report TR-99-25, Department of Computer Sciences, The University of Texas at Austin, September 1999.

[26] Călin Caşcaval, José G. Casta nos, Luis Ceze, Monty Denneau, Manish Gupta, Derek Lieber, José E. Moreira, Karin Strauss, and Henry S. Warren Jr. Evaluation of a multithreaded architecture for cellular computing. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA '02)*, pages 311–322. IEEE Computer Society, 2002.

[27] Edwin Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Department of Computer Science, University of Utah, 1974.

[28] Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. In *EGRW '03: Proceedings of the 14th Eurographics Workshop on Rendering*, pages 208–218, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[29] Eric Chan and Frédo Durand. An efficient hybrid shadow rendering algorithm. In *Proceedings of the 15th Eurographics Workshop on Rendering Techniques (Rendering Techniques '04)*, pages 185–196. Eurographics Association, 2004.

[30] Marco Chiappetta. NVIDIA GeForce GTX 280 and GTX 260 unleashed, June 2008. http://hothardware.com/printarticle.aspx?articleid=1167.

[31] Hamilton Y. Chong and Steven J. Gortler. A lixel for every pixel. In *Proceedings of the Eurographics Symposium on Rendering*, pages 167–172. Eurographics Association, 2004.

[32] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The reyes image rendering architecture. In *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 95–102, New York, NY, USA, 1987. ACM Press.

[33] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *SIGGRAPH '84: Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, pages 137–145, New York, NY, USA, 1984. ACM Press.

[34] Franklin C. Crow. Shadow algorithms for computer graphics. *SIGGRAPH '77: Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*, 11(2):242–248, July 1977.

[35] Willem H. de Boer. Smooth penumbra transitions with shadow maps. *Journal of Graphics Tools*, 11(2):59–71, 2006.

[36] Michael Deering and David Naegle. The SAGE graphics architecture. In *SIGGRAPH 2002: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 683–692, New York, NY, USA, 2002. ACM Press.

[37] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *I3D '06: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, pages 161–165, New York, NY, USA, 2006. ACM Press.

[38] George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. In *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 223–230, New York, NY, USA, 1994. ACM Press.

[39] Elmar Eisemann and Xavier Décoret. Plausible image based soft shadows using occlusion textures. In *Proceedings of the Brazilian Symposium on Computer Graphics and Image Processing, 19 (SIBGRAPI)*, pages 155–162. IEEE Computer Society, 2006. Oliveira Neto, Manuel Menezes deCarceroni and Rodrigo Lima (Eds.).

[40] Matthew Eldridge, Homan Igehy, and Pat Hanrahan. Pomegranate: a fully scalable graphics architecture. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 443–454, New York, NY, USA, 2000. ACM Press / Addison-Wesley Publishing Co.

[41] Wolfgang Engel. Cascaded shadow maps. *Shader X* ^{5} *Advanced Rendering Techniques*, chapter 4.1, pages 197–206. Charles River Media, December 2006. Wolfgang Engel (Ed.).

[42] Cass Everitt and Mark Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. *SIGGRAPH 2002 Course Notes*, Course #31, 2002.

[43] Randima Fernando. Percentage-closer soft shadows. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 35, New York, NY, USA, 2005. ACM Press.

[44] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 387–390, New York, NY, USA, August 2001. ACM Press.

[45] Vincent Forest, Loïc Barthe, and Mathias Paulin. Accurate shadows by depth complexity sampling. In *Proceedings of Eurographics 2008*, pages 663–674. Eurographics Association, 2008. Computer Graphics Forum Volume 27, Number 2.

[46] Gordon C. Fossum and Donald S. Fussell. Generating soft shadows efficiently. Technical Report TR-87-22, Department of Computer Sciences, The University of Texas at Austin, 1987.

[47] Henry Fuchs, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebbs, and Laura Israel. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. In *SIGGRAPH '89: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, pages 79–88, New York, NY, USA, 1989. ACM Press.

[48] Andy Goris, Bob Fredrickson, and Jr. Harold L. Baeverstad. A configurable pixel cache for fast image generation. *IEEE Computer Graphics and Applications*, 7(3):24–32, March 1987.

[49] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical z-buffer visibility. In *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 231–238, New York, NY, USA, 1993. ACM Press.

[50] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Real-time soft shadow

mapping by backprojection. In *Eurographics Symposium on Rendering*, pages 227–234. Eurographics, June 2006.

[51] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. High-quality adaptive soft shadow mapping. *Computer Graphics Forum*, 26(3):525–533, 2007.

[52] Toshiya Hachisuka. High-quality global illumination rendering using rasterization. *GPU Gems 2: Programming Techniques for High Performance Graphics and General-Purpose Computation*, chapter 38, pages 615–633. Addison-Wesley Professional, 2005. Matt Pharr and Randima Fernando (Eds.).

[53] Paul Haeberli and Kurt Akeley. The accumulation buffer: Hardware support for high-quality rendering. In *SIGGRAPH '90: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, pages 309–318, New York, NY, USA, 1990. ACM Press.

[54] Eric Haines. Soft planar shadows using plateaus. *Journal of Graphics Tools*, 6(1):19–27, 2001.

[55] Eric A. Haines and Donald P. Greenberg. The light buffer: A shadow-testing accelerator. *IEEE Computer Graphics and Applications*, 6:6–16, September 1986.

[56] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, December 2003. State-of-the-Art Report.

[57] Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, School of Computer Science, Carnegie Mellon University, January 1997.

[58] Wolfgang Heidrich, Stefan Brabec, and Hans-Peter Seidel. Soft shadow maps

for linear lights. In *Proceedings of Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 269–280, June 2000.

[59] Michael Herf. Efficient generation of soft shadow textures. Technical Report CMU-CS-97-138, School of Computer Science, Carnegie Mellon University, May 1997.

[60] J. C. Hourcade and A. Nicolas. Algorithms for antialiased cast shadows. *Computers & Graphics*, 9(3):259–265, 1985.

[61] Warren A. Hunt and William R. Mark. Ray-specialized acceleration structures for ray tracing: The missing link between the z-buffer and ray tracing. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing 2008*, pages 3–10, September 2008. Submitted.

[62] id Software Incorporated. Doom 3, 2004. http://www.doom3.com.

[63] Yeon-Ho Im and Chang-Young Han. A method to generate soft shadows using a layered depth image and warping. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):265–272, 2005.

[64] Intel Corporation. First details on a future intel design codenamed "Larrabee", August 2008. http://www.intel.com/pressroom/archive/releases/20080804fact.htm.

[65] Gregory S. Johnson, Juhyun Lee, Christopher A. Burns, and William R. Mark. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics*, 24(4):1462–1482, 2005.

[66] Gregory S. Johnson, William R. Mark, and Christopher A. Burns. The irregular z-buffer and its application to shadow mapping. Technical Report TR-04-09, Department of Computer Sciences, The University of Texas at Austin, April 2004.

[67] Ujval J. Kapasi, William J. Dally, Scott Rixner, Peter R. Mattson, John D. Owens, and Brucek Khailany. Efficient conditional operations for data-parallel architectures. In *MICRO 33: Proceedings of the 33rd Annual ACM / IEEE International Symposium on Microarchitecture*, pages 159–170, New York, NY, USA, 2000. ACM Press.

[68] Ujval J. Kapasi, William J. Dally, Scott Rixner, John D. Owens, and Brucek Khailany. The Imagine stream processor. In *Proceedings of the IEEE Conference on Computer Design*, pages 295–302, September 2002.

[69] Brett Keating and Nelson Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Rendering Techniques '99 (Proceedings of the Eurographics Workshop on Rendering)*, pages 197–212, June 1999.

[70] Emmett Kilgariff and Randima Fernando. The GeForce 6 series GPU architecture. *GPU Gems 2: Programming Techniques for High Performance Graphics and General-Purpose Computation*, chapter 30, pages 471–491. Addison-Wesley Professional, 2005. Matt Pharr and Randima Fernando (Eds.).

[71] Florian Kirsch and Jurgen Dollner. Real-time soft shadows using a single light sample. *Journal of WSCG (Winter School on Computer Graphics)*, 11(1), February 2003.

[72] Craig Kolb. Rayshade homepage. July 1997.

[73] Simon Kozlov. Perspective shadow maps: Care and feeding. *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, chapter 14, pages 217–244. Addison-Wesley Professional, 2004. Randima Fernando (Ed.).

[74] Samuli Laine. An incremental shaft subdivision algorithm for computing shad-

ows and visibility. Master's thesis, Helsinki University of Technology, March 2006.

[75] Samuli Laine and Timo Aila. Hierarchical penumbra casting. *Computer Graphics Forum*, 24(3):313–322, 2005.

[76] Samuli Laine, Timo Aila, Ulf Assarsson, Jaakko Lehtinen, and Tomas Akenine-Möller. Soft shadow volumes for ray tracing. *ACM Transactions on Graphics*, 24(3):1156–1165, 2005.

[77] Andrew Lauritzen. Summed-area variance shadow maps. *GPU Gems 3*, chapter 8, pages 157–181. Addison-Wesley, July 2007.

[78] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. All-purpose texture sprites. Technical Report 5209, INRIA, May 2004.

[79] Aaron E. Lefohn, Shubhabrata Sengupta, Joe Kniss, Robert Strzodka, and John D. Owens. Glift: Generic, efficient, random-access GPU data structures. *ACM Transactions on Graphics*, 25(1):60–99, January 2006.

[80] Aaron E. Lefohn, Shubhabrata Sengupta, and John D. Owens. Resolution-matched shadow maps. *ACM Transactions on Graphics*, 26(4):20–42, 2007.

[81] Jaakko Lehtinen, Samuli Laine, and Timo Aila. An improved physically-based soft shadow volume algorithm. *Computer Graphics Forum*, 25(3):303–312, 2006.

[82] Jed Lengyel and Bruce Walter. The path-buffer. Technical Report PCG-95-4, Program of Computer Graphics, Cornell University, 1995.

[83] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 31–42, New York, NY, USA, 1996. ACM Press.

[84] Markus Levy. Tying up a MIPS32 processor with threads: Lexra evolves the LX4380 pipeline into multithreading processor. *Microprocessor Report*, November 2002.

[85] Erik Lindholm, Mark Kilgard, and Henry Moreton. A user-programmable vertex engine. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (ACM SIGGRAPH 2001)*, pages 149–158, New York, NY, USA, August 2001. ACM Press.

[86] Brandon Lloyd, Naga Govindaraju, Steven Molnar, and Dinesh Manocha. Practical logarithmic rasterization for low-error shadow maps. In *GH '07: Proceedings of the 22nd ACM SIGGRAPH / EUROGRAPHICS Symposium on Graphics Hardware*, pages 17–24, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[87] Brandon Lloyd, Naga Govindaraju, Cory Quammen, Steve Molnar, and Dinesh Manocha. Logarithmic perspective shadow maps. Technical Report TR07-005, Department of Computer Science, University of North Carolina at Chapel Hill, June 2007.

[88] Brandon Lloyd, David Tuft, Sung-Eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2006*, pages 215–226. Eurographics Association, 2006.

[89] Brandon Lloyd, Jeremy Wend, Naga K. Govindaraju, and Dinesh Manocha. CC shadow volumes. In *Proceedings of the 15th Eurographics Workshop on Rendering Techniques (Rendering Techniques '04)*, pages 197–206. Eurographics Association, 2004.

[90] Brandon Lloyd, Sung-Eui Yoon, David Tuft, and Dinesh Manocha. Subdivided

shadow maps. Technical Report TR05-024, Department of Computer Science, University of North Carolina at Chapel Hill, 2004.

[91] Tom Lokovic and Eric Veach. Deep shadow maps. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 385–392, New York, NY, USA, 2000. ACM Press / Addison-Wesley Publishing Co.

[92] Céline Loscos and George Drettakis. Interactive high-quality soft shadows in scenes with moving objects. *Computer Graphics Forum*, 16(3), 1997.

[93] Tobias Martin and Tiow-Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2004*, pages 153–160. Eurographics Association, 2004.

[94] Joel McCormack, Robert McNamara, Christopher Gianos, Larry Seiler, Norman P. Jouppi, and Ken Correll. Neon: a single-chip 3d workstation graphics accelerator. In *Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Workshop on Graphics Hardware*, pages 123–132. ACM Press, 1998.

[95] Morgan McGuire, John F. Hughes, Kevin Egan, Mark Kilgard, and Cass Everitt. Fast, practical and robust shadows. *Brown University Technical Report CS03-19*, October 2003.

[96] Micron Technology, Inc. Micron Technology 256 Mb (x32) GDDR3 SDRAM datasheet. June 2003.

[97] Microsoft Corporation. DirectX 9.0 SDK. 2003.

[98] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 376–381, New York, NY, USA, 2003. ACM Press.

[99] NVIDIA Corporation. NVIDIA GeForce FX 5900, 5700 and Go5700 GPUs: Ultra Shadow technology. October 2003.

[100] NVIDIA Corporation. NVIDIA CUDA compute unified device architecture programming guide 1.1. November 2007.

[101] Marc Olano and Trey Greer. Triangle scan conversion using 2D homogeneous coordinates. In *Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Workshop on Graphics Hardware*, pages 89–95, New York, NY, USA, August 1997. ACM Press.

[102] OpenGL Architectural Review Board. OpenGL 1.5 specification. October 2003.

[103] Ryan Overbeck, Ravi Ramamoorthi, and William R. Mark. A real-time beam tracer with application to exact soft shadows. In *Proceedings of the Eurographics Symposium on Rendering 2007*. Eurographics Association, June 2007.

[104] Woo-Chan Park, Kil-Whan Lee, Il-San Kim, Tack-Don Han, and Sung-Bong Yang. An effective pixel rasterization pipeline architecture for 3d rendering processors. *IEEE Transactions on Computers*, 52(11):1501–1508, November 2003.

[105] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical Report UCS-98-019, Computer Science Department, University of Utah, October 1998.

[106] Juan Pineda. A parallel algorithm for polygon rasterization. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):17–20, August 1988.

[107] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. In *SIGGRAPH '02: Proceed-*

ings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, pages 703–712, New York, NY, USA, 2002. ACM Press.

[108] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Conference on Graphics Hardware*, pages 41–50. Eurographics Association, 2003.

[109] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 283–291, New York, NY, USA, 1987. ACM Press.

[110] Erik Reinhard, Brian Smits, and Charles Hansen. Dynamic acceleration structures for interactive ray tracing. In *Proceedings of the 11th Eurographics Workshop on Rendering*, pages 299–306. Eurographics Association, June 2000.

[111] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Transactions on Graphics*, 25(3):977–986, 2006.

[112] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Transactions on Graphics*, 24(3):1176–1185, 2005.

[113] Scott Rixner. Memory controller optimizations for web servers. In *Proceedings of the 37th Annual International Symposium on Microarchitecture*, pages 355–366, December 2004.

[114] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[115] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 327–334, New York, NY, USA, 2000. ACM Press / Addison-Wesley Publishing Co.

[116] Michael Schwarz and Marc Stamminger. Bitmask soft shadows. *Computer Graphics Forum*, 26(3):515–524, September 2007.

[117] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):249–252, July 1992.

[118] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Pat Hanrahan, Stephen Junkins, Adam Lake, and Jeremy Sugerman. Larrabee: A many-core x86 architecture for visual computing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 Papers*, New York, NY, USA, 2008. ACM Press.

[119] Pradeep Sen, Mike Cammarano, and Pat Hanrahan. Shadow silhouette maps. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 521–526, New York, NY, USA, 2003. ACM Press.

[120] Erik Sintorn, Elmar Eisemann, and Ulf Assarsson. Sample based visibility for soft shadows using alias-free shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2008*, pages 1285–1292. Eurographics Association, 2008. Computer Graphics Forum Volume 27, Number 4.

[121] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer*

Graphics and Interactive Techniques, pages 527–536, New York, NY, USA, 2002. ACM Press.

[122] Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 321–332, New York, NY, USA, 1998. ACM Press.

[123] Marc Stamminger and George Drettakis. Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 557–562, New York, NY, USA, July 2002. ACM Press.

[124] A. James Stewart and Sherif Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 231–238, New York, NY, USA, 1994. ACM Press.

[125] Gordon Stoll, William R. Mark, Peter Djeu, Rui Wang, and Ikrima Elhassan. Razor: An architecture for dynamic multiresolution ray tracing. Technical Report TR-06-21, Department of Computer Sciences, The University of Texas at Austin, April 2006.

[126] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer generated films. *ACM Transactions on Graphics*, 23(3):469–476, 2004.

[127] Katsumi Tadamura, Xueying Qin, Guofang Jiao, and Eihachiro Nakamae. Rendering optimal solar shadows using plural sunlight depth buffers. In *CGI '99: Proceedings of the International Conference on Computer Graphics*, pages 166–174, Washington, DC, USA, 1999. IEEE Computer Society.

[128] Nick Triantos. Windowing system on a 3d pipeline. White paper, NVIDIA Corporation, February 2005.

[129] Yury Uralsky and Anis Ahmad. NVIDIA SDK white paper: Soft shadows. July 2004.

[130] Manish Vachharajani, Neil Vachharajani, David A. Penry, Jason Blome, and David I. August. The liberty simulation environment, version 1.0. *Performance Evaluation Review: Special Issue on Tools for Architecture Research*, 31(4), March 2004.

[131] James M. Van Dyke, Douglas A. Voorhies, III James E. Margeson, and John Montrym. System, method and article of manufacture for an interlock module in a computer graphics processing pipeline. May 2004. Patent #6734861.

[132] Channing P. Verbeck and Donald P. Greenberg. A comprehensive light source description for computer graphics. *IEEE Computer Graphics and Applications*, 4(7):66–75, July 1984.

[133] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics*, 26(1):6–33, 2007.

[134] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. Ray tracing animated scenes using coherent grid traversal. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 485–493, New York, NY, USA, 2006. ACM Press.

[135] Ingo Wald, Timothy J. Purcell, Jorg Schmittler, Carsten Benthin, and Philipp Slusallek. Realtime ray tracing and its use for interactive global illumination. In *State of the Art Reports, EUROGRAPHICS 2003*, 2003.

[136] Yulan Wang and Steven Molnar. Second-depth shadow mapping. Technical Report TR94-019, Department of Computer Science, University of North Carolina at Chapel Hill, 1994.

[137] Leonard R. Wanger, James A. Ferwerda, and Donald P. Greenberg. Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications*, 12(3):44–58, May 1992.

[138] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[139] Lance Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, pages 270–274, New York, NY, USA, August 1978. ACM Press.

[140] Paul Willmann, Michael Brogioli, and Vijay S. Pai. Spinach: a Liberty-based simulator for programmable network interface architectures. In *LCTES '04: Proceedings of the 2004 ACM SIGPLAN / SIGBED Conference on Languages, Compilers, and Tools*, pages 20–29, New York, NY, USA, 2004. ACM Press.

[141] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2004*, pages 143–151. Eurographics Association, June 2004.

[142] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics Applications*, 10(6):13–32, 1990.

[143] Sven Woop, Jörg Schmittler, and Philipp Slusallek. RPU: A programmable ray processing unit for realtime ray tracing. *ACM Transactions on Graphics*, 24(3):434–444, 2005.

[144] Chris Wyman and Charles Hansen. Penumbra maps: Approximate soft shadows in real-time. In *EGRW '03: Proceedings of the 14th Eurographics Workshop on Rendering*, pages 202–207. Eurographics Association, 2003.

[145] Feng Xie, Eric Tabellion, and Andrew Pearce. Soft shadows by ray tracing multilayer transparent shadow maps. In *Proceedings of the Eurographics Symposium on Rendering 2007*, pages 265–276. Eurographics Association, 2007.

[146] Zhengming Ying, Min Tang, and Jinxiang Dong. Soft shadow maps for area light by area approximation. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, pages 442–443, Washington, DC, USA, 2002. IEEE Computer Society.

[147] Fan Zhang, Hanqiu Sun, and Oskari Nyman. Parallel-split shadow maps on programmable GPUs. *GPU Gems 3*, chapter 10, pages 203–237. Addison-Wesley, July 2007.

[148] Kun Zhou, Yaohua Hu, Stephen Lin, Baining Guo, and Heung-Yeung Shum. Precomputed shadow fields for dynamic scenes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1196–1201, New York, NY, USA, 2005. ACM Press.

Vita

Gregory Scott Johnson began studying the interplay between light and shadow early one sunny morning in August of 1971 in Denver Colorado. He is the proud son of Garland Martin Johnson Jr. and Carol Ann Stevens. Gregory completed a Bachelor of Science degree in Computer Science (with honors) at the University of Alaska Fairbanks in May of 1993, followed by a Master of Science in Computer Science in May of 1995. Gregory was admitted to the Ph.D. program in Computer Sciences at the University of Texas at Austin in August of 2001.

Permanent Address: 9913 Nocturne Cove
Austin, Texas 78750

This dissertation was typeset with LaTeX 2_ε[2] by the author.

[2] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.