Helsinki University of Technology
Publications in Telecommunications Software and Multimedia
Teknillisen korkeakoulun tietoliikenneohjelmistojen ja multimedian julkaisuja
Espoo 2005 TML-A11

EFFICIENT ALGORITHMS FOR OCCLUSION CULLING AND SHADOWS

Timo Aila

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology
Publications in Telecommunications Software and Multimedia
Teknillisen korkeakoulun tietoliikenneohjelmistojen ja multimedian julkaisuja
Espoo 2005 TML-A11

Efficient Algorithms for Occlusion Culling and Shadows

Timo Aila

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering, for public examination and debate in Auditorium AS1 at Helsinki University of Technology (Espoo, Finland) on the 25th of February, 2005, at 12 noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Telecommunications Software and Multimedia Laboratory

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietoliikenneohjelmistojen ja multimedian laboratorio

Distribution:
Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
P.O.Box 5400
FIN-02015 HUT
Finland
Tel. +358-9-451 2870
Fax. +358-9-451 5014
http://www.tml.hut.fi/

Available in PDF format at http://lib.hut.fi/Diss/2005/isbn9512274833/

©Timo Aila

ISBN 951-22-7480-9 (printed version)
ISSN 1456-7911
ISBN 951-22-7483-3 (electronic version)
ISSN 1455-9722

Otamedia Oy
Espoo 2005

ABSTRACT

Author Timo Aila
Title Efficient Algorithms for Occlusion Culling and Shadows

The goal of this research is to develop more efficient techniques for computing the visibility and shadows in real-time rendering of three-dimensional scenes. Visibility algorithms determine what is visible from a camera, whereas shadow algorithms solve the same problem from the viewpoint of a light source.

In rendering, a lot of computational resources are often spent on primitives that are not visible in the final image. One visibility algorithm for reducing the overhead is occlusion culling, which quickly discards the objects or primitives that are obstructed from the view by other primitives. A new method is presented for performing occlusion culling using silhouettes of meshes instead of triangles. Additionally, modifications are suggested to occlusion queries in order to reduce their computational overhead.

The performance of currently available graphics hardware depends on the ordering of input primitives. A new technique, called delay streams, is proposed as a generic solution to order-dependent problems. The technique significantly reduces the pixel processing requirements by improving the efficiency of occlusion culling inside graphics hardware. Additionally, the memory requirements of order-independent transparency algorithms are reduced.

A shadow map is a discretized representation of the scene geometry as seen by a light source. Typically the discretization causes difficult aliasing issues, such as jagged shadow boundaries and incorrect self-shadowing. A novel solution is presented for suppressing all types of aliasing artifacts by providing the correct sampling points for shadow maps, thus fully abandoning the previously used regular structures. Also, a simple technique is introduced for limiting the shadow map lookups to the pixels that get projected inside the shadow map.

The fillrate problem of hardware-accelerated shadow volumes is greatly reduced with a new hierarchical rendering technique. The algorithm performs per-pixel shadow computations only at visible shadow boundaries, and uses lower resolution shadows for the parts of the screen that are guaranteed to be either fully lit or fully in shadow.

The proposed techniques are expected to improve the rendering performance in most real-time applications that use 3D graphics, especially in computer games. More efficient algorithms for occlusion culling and shadows are important steps towards larger, more realistic virtual environments.

UDC 004.925, 004.383.5
Keywords computer graphics, shadow algorithms, occlusion culling, 3D graphics hardware, order-independent transparency

PREFACE

This research was primarily carried out at the Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, Espoo, during 2002–2004. The algorithms used in dPVS were developed at Hybrid Graphics 2000–2001, and the hardware simulations of delay streams were done at Bitboys in 2002.

I want to express my gratitude to Prof. Lauri Savioja, my thesis supervisor, for encouragement and for letting me choose the research topic freely.

I would like to thank my co-authors Tomas Akenine-Möller, Jukka Arvo, Samuli Laine, Petri Nordlund, and Ville Miettinen for fruitful collaboration, and my colleagues Jaakko Lehtinen, Janne Kontkanen, and Jussi Räsänen for insightful conversations, and for providing feedback on early versions of the publications.

Furthermore, I would like to thank the pre-examiners of this thesis, Prof. Michael McCool and Dr. Kari Pulli for positive feedback and constructive comments. Thanks to Timo Haanpää for proofreading the thesis.

I am grateful to the organizations that have supported this work financially: the Department of Computer Science at Helsinki University of Technology, ATI Technologies, Tekniikan edistämissäätiö (The Ministry of Trade and Industry), Tekes (The National Technology Agency), Bitboys, Hybrid Graphics, Nokia, and Remedy Entertainment.

Finally, I am deeply indebted to Tiina Rekola for love, support, and patience during my studies.

Otaniemi, Espoo, 20th January 2005

Timo Aila

TABLE OF CONTENTS

LIST OF PUBLICATIONS

This thesis summarizes the following articles and publications, referred to as [P1]–[P5]:

[P1] T. Aila, V. Miettinen, and P. Nordlund. Delay Streams for Graphics Hardware. *ACM Transactions on Graphics*, 22(3):792–800, 2003.

[P2] T. Aila and T. Akenine-Möller. A Hierarchical Shadow Volume Algorithm. In *Graphics Hardware 2004 (Eurographics Symposium Proceedings)*, pages 15–23. Eurographics Association, 2004.

[P3] T. Aila and V. Miettinen. dPVS: An Occlusion Culling System for Massive Dynamic Environments. *IEEE Computer Graphics and Applications*, 24(2):86–97, 2004.

[P4] J. Arvo and T. Aila. Optimized Shadow Mapping Using the Stencil Buffer. *Journal of Graphics Tools*, 8(3):23–32, 2003.

[P5] T. Aila and S. Laine. Alias-Free Shadow Maps. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering)*, pages 161–166. Eurographics Association, 2004.

LIST OF ABBREVIATIONS

3D	Three-dimensional
CPU	Central processing unit
GPU	Graphics processing unit
ID	Identifier
Z_{min}	Minimum depth value of an $N \times M$ pixel tile
Z_{max}	Maximum depth value of an $N \times M$ pixel tile

1 INTRODUCTION

This thesis relates to computer graphics and particularly to rendering, which is the process of synthesizing images from a description of a virtual scene. The description typically includes at least surfaces of objects, light sources, and material properties that describe how the surfaces reflect light. Modeling is the process of creating this description, and rendering generates the final image by simulating the behavior of light in the environment.

There are few, if any, scenes that cannot be satisfactorily rendered if an unbounded amount of processing time is available. A high-fidelity simulation of indirect illumination currently leads to rendering times of several hours or days per frame, which can be prohibitive even in movie production. Therefore, numerous approximations need to be made for real-time applications that require rendering of at least 25 frames per second. The difference in realism is easily observed by comparing almost photorealistic computer-generated feature films against significantly less convincing computer games. Major shortcomings in real-time applications include simplistic, plastic-like appearance of materials, the lack of optical effects such as motion blur and depth of field, and the commonly used lighting models that omit not only the indirect illumination but often also the shadows of direct illumination.

Currently most efforts in rendering research are concentrated in finding faster methods for producing high-fidelity images, the ultimate goal being photorealistic real-time rendering. Even if this goal approaches steadily due to technological advances, the progress can be greatly sped up by developing algorithms that use the available computational power as efficiently as possible. This thesis proposes new methods for improving the performance of the rendering process without compromising image quality.

1.1 Scope of This Thesis

The primary topic of this thesis is improving the performance of shadow generation from infinitesimal light sources that create perfectly sharp shadow boundaries. While such light sources do not exist in the real world, they provide a convenient intermediate step between having no shadows and computationally more demanding accurate soft shadows from area light sources. Furthermore, most algorithms for computing soft shadows are based on the techniques designed for infinitesimal light sources, and thus inherit their performance and quality characteristics.

The secondary topic of this thesis is visibility algorithms, and in particular occlusion culling, which quickly identifies the set of rendering primitives that are not obstructed from the view by other primitives. Visibility and shadow algorithms are semantically similar, as the first one determines what is visible from a camera while the latter solves the same problem from the point of view of a light source.

This thesis is based on five publications [P1–P5] that propose new, more efficient algorithms for occlusion culling and shadows. Figure 1.1

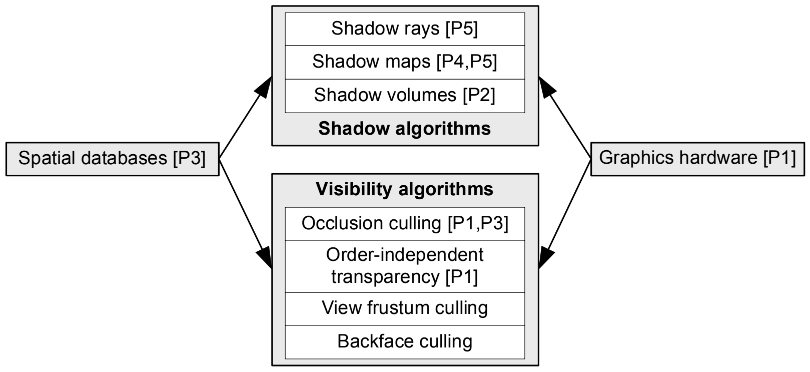


Figure 1.1: The publications [P1–P5] propose improved algorithms for shadows and visibility. In general, both visibility and shadow algorithms use a spatial database for hierarchical processing of the scene, and utilize graphics hardware for performing a part of the computations.

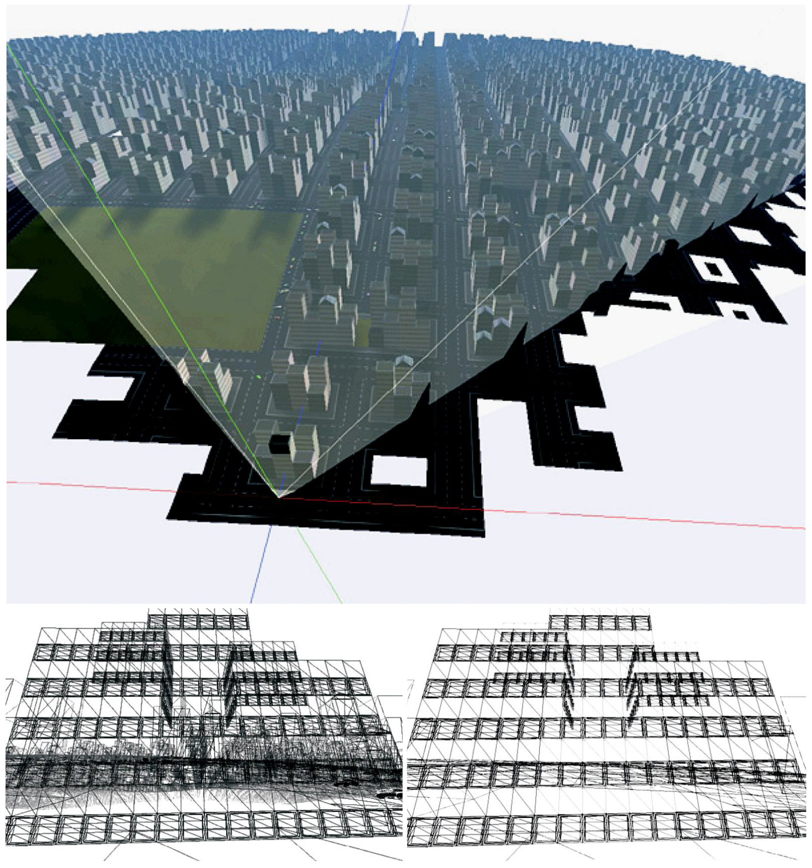
illustrates the positioning of the publications. The techniques target real-time applications with moving objects or otherwise time-varying scenes. In such applications, visibility and shadows need to be updated dynamically to correctly account for changes in the viewing parameters and the environment.

1.2 Visibility Algorithms

Virtually all real-time rendering systems determine the visible surfaces by using the Z-buffer algorithm [17]. Unfortunately, Z-buffering has two major weaknesses. First, it fails to blend semi-transparent surfaces in the correct order unless the application has pre-sorted them. Methods that generalize the Z-buffer to semi-transparent surfaces are referred to as order-independent transparency algorithms. The second problem is one of performance: all primitives of a scene are always processed, even if only a small part of the scene is visible at a time.

The processing of a scene can be optimized in several ways. View frustum culling discards primitives that are outside the view frustum. Hierarchical versions [24] avoid performing a separate test for each primitive. Often planar primitives are visible only from one side, and backface culling removes primitives that face away from the camera. Occlusion culling complements the other culling methods by quickly removing primitives or entire objects that are obstructed from the view by other primitives. Figure 1.2 illustrates the efficiency of occlusion culling in a large city environment.

For hierarchical variants of view frustum and occlusion culling the objects of the scene need to be organized into a hierarchy [77]. This thesis refers to such spatial data structures as spatial databases. The database is typically traversed in an approximate front-to-back order to maximize the efficiency of Z-buffer processing and occlusion culling.



a) b)

Figure 1.2: The goal of occlusion culling is to minimize the rendering time of a frame by finding a tight superset of the visible objects as quickly as possible. The top image shows a third person view of an urban environment intersected by a view frustum. The bottom row images are wireframe renderings from the marked viewpoint. a) Without occlusion culling. b) With occlusion culling enabled. In this case occlusion culling reduces the rendering workload approximately hundred-fold.

1.3 Shadow Algorithms

Until recently, almost all real-time applications have computed the contribution of a light source according to a grossly simplistic assumption that the line-of-sight between the point to be shaded and the light source is never blocked. If this assumption is made, the illumination computations are trivial, and depend only on the properties of the point and the light source. However, the presence of shadows is valuable because they reveal information about the spatial relationships of objects and increase the level of realism. Figure 1.3 illustrates one possible ambiguity if shadows are omitted from a picture.

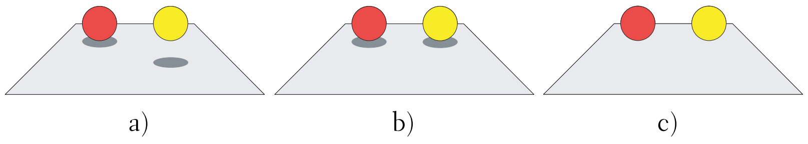


Figure 1.3: Shadows convey valuable information about spatial relationships of objects. Without shadows the two configurations a) and b) would look identical, c).

The geometric interpretation of shadow generation is simple, and was already known 500 years ago, as demonstrated in Figure 1.4. Unfortunately, determining if a light source is visible from a given point may involve a large number of other objects, and is therefore computationally demanding. The primary obstacle for including shadows into real-time applications has been the lack of sufficient computing power. Another, partially overlapping reason is that most shadow generation algorithms perform a lot of redundant computations.

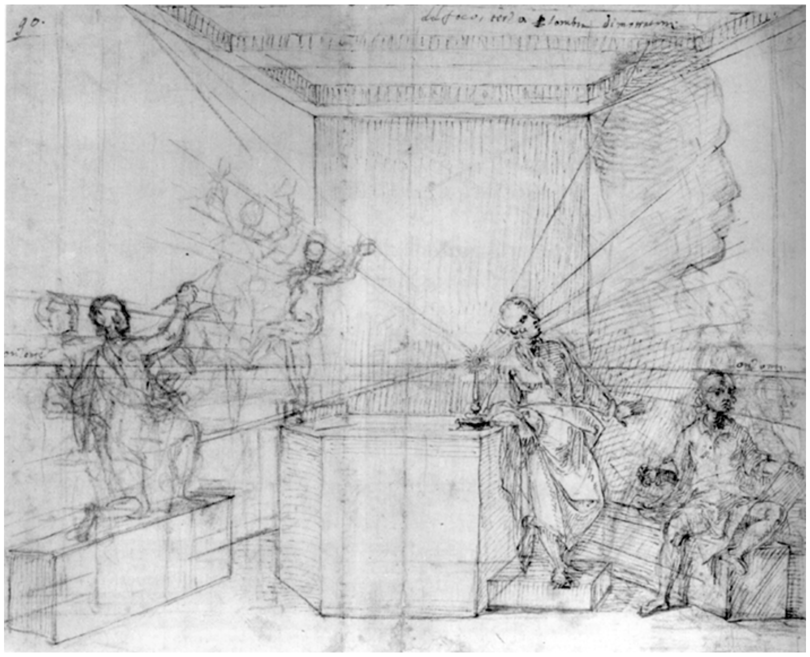


Figure 1.4: An early study of shadows from the school of Leonardo da Vinci (Manuscript *Codex Huygens*). If one end of a long pen is mounted at the light source (candle) and the silhouettes of the objects are traced with the pen, the far end of the pen draws the shadow boundaries to the walls.

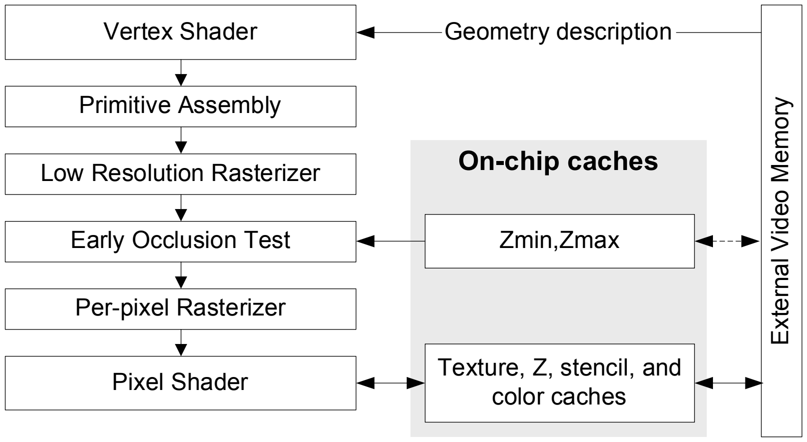


Figure 1.5: Flowchart of a modern graphics processing unit.

1.4 Relationship of Visibility and Shadow Algorithms

As visibility and shadow algorithms solve the same problem from different viewpoints, they are generally inter-exchangeable. In fact, some visibility algorithms have been directly applied to shadow generation [17, 91, 5, 90] and vice versa [28, 51]. However, shadows need to be computed only for the parts of surfaces that are visible from the camera. Thus shadow generation can be optimized by first solving the visibility from the camera, and then limiting the shadow tests to the visible samples.

Typically the visible samples of the output image are spaced irregularly when viewed from the light source. The light-space sampling points used in shadow computations should correspond exactly to these visible samples. If this requirement is not met, difficult aliasing problems arise because the results are computed at slightly incorrect positions. Nevertheless, algorithms that discretize the light space using a regular grid are widely used due to their simplicity [91].

1.5 Graphics Hardware

Virtually all real-time rendering is performed using specialized graphics processing units (GPUs). Figure 1.5 shows a simplified flowchart of a modern GPU. The input typically consists of vertex parameters and connectivity information. The vertex shader is a programmable unit that computes the transformed vertex positions, and optionally other attributes such as non-shadowed direct lighting. The primitive assembly creates triangles by connecting vertices. The low-resolution rasterizer finds the $N \times M$ pixel tiles that are at least partially covered by the triangle [42]. The early occlusion test [69] performs tile-specific optimizations. Typically the minimum (Z_{min}) [4] and maximum (Z_{max}) [69] depth values are maintained for each tile. If the part of the input triangle that overlaps the tile is behind Z_{max}, further processing inside the tile can be skipped, and if the triangle is in front of Z_{min}, per-pixel depth values do not need to be tested. These optimizations cannot be applied in certain special cases, e.g., when particular stencil buffer operations [73] are enabled or when the depth values of the input triangle are modified by the subsequent units. The per-pixel rasterizer enumerates the pixels that need to be processed. The pixel shader is a programmable unit that computes the color, and possibly other attributes for each pixel. Modern GPUs have a number of vertex and pixel shader units executing in parallel. In many applications the overall performance is limited by the number of processed pixels and the related communication to external video memory.

An important characteristic of a typical GPU is that the input triangles are not reordered during the pipeline. This greatly simplifies the design but also causes a number of drawbacks. For example, the efficiency of Z_{min} and Z_{max} optimizations depend on the ordering of the input primitives. Also, the semi-transparent primitives are rendered incorrectly unless they are submitted in a strict back-to-front order by the application.

1.6 Organization of the Thesis

This thesis is organized as follows. Chapter 2 gives an overview of the related research on occlusion culling and shadow algorithms. Chapters 3 and 4 present the new contributions to occlusion culling and shadow algorithms, respectively. The concept of delay streams is explained in Chapter 5. Finally, Chapter 6 provides a summary of the thesis and the author's contributions.

2 RELATED RESEARCH

This chapter reviews related work on occlusion culling in dynamic scenes, and on shadow algorithms that assume point-like light sources creating sharp shadow boundaries. Techniques for producing soft shadows are discussed only as extensions of hard shadow algorithms. Several surveys are available on visibility determination [33, 3, 25] and shadow computation [97, 44, 46].

2.1 Visibility and Occlusion

A number of occlusion culling systems have been proposed for the limited case of urban environments [94]. These systems reduce the visibility problem into a two-dimensional sub-problem by assuming that buildings are essentially boxes with different heights.

Airey et al. [1] and Teller and Séquin [87] propose subdividing the scene into *cells* connected by *portals*. The subdivision is particularly well suited for indoor scenes, where the cells correspond to rooms and the portals to doors and windows. Haumont et al. [47] and Lefebvre and Hornus [58] describe methods for constructing cells and portals automatically from a scene description.

Several object-space visibility algorithms have been described for static scenes [33, 25]. Extending these methods to dynamic scenes is challenging, and thus we concentrate on image-space techniques.

Image-space hierarchies and graphics hardware

Greene and Kass [43] reduce the number of per-pixel depth comparisons by organizing the depth buffer into a hierarchy. Due to the challenges in maintaining a full hierarchy in graphics hardware, several simplified designs have been proposed. Xie and Shantz [99] update the hierarchy only a few times per frame according to a heuristic. Morein [69] describes an architecture that incorporates a two-level depth pyramid and adds an *occlusion test* stage into the traditional rendering pipeline. In addition to memory bandwidth optimizations, parts of the incoming triangles can be culled before rasterization. The early depth rejection works best when the input primitives arrive in an approximate front-to-back order. Publication [P1] deals with the issue of order dependency.

An alternative technique for reducing pixel shading work is to render the scene twice: the first pass constructs the depth buffer and the second applies shading for the visible pixels [30]. Meißner et al. [67] perform two-pass visibility driven rasterization by first marking the tiles that are fully hidden and then skipping the subsequent rasterization to those tiles. PowerVR [75] captures the geometry of the entire frame, and uses tile-based rendering with internal buffers. This reduces the memory bandwidth requirements but the limited amount of video memory makes capturing large scenes impractical. Occlusion culling is implemented by using on-chip sorting, and thus the culling efficiency is not affected by the ordering of input primitives.

PixelFlow [68] and Talisman [88] composite the final image from multiple layers. Occlusion culling is difficult to address efficiently in these architectures. SaarCOR [79, 80] determines the visible surfaces using ray casting [5], and performs occlusion culling implicitly.

Occlusion queries

Greene and Kass [43] represent the scene as an octree [38] that provides an approximate front-to-back traversal. During the traversal, an occlusion query is performed for each octree node. If a node is hidden, neither it nor its children need further processing. Typically a front-to-back traversal of a spatial database requires hundreds or even thousands of queries every frame [57, 49]. This imposes unwanted synchronization between the CPU that traverses the database and the GPU that executes the occlusion queries.

Direct3D [32] and OpenGL [73] are the dominant application programming interfaces for real-time rendering. Bartz et al. [9] discuss extending OpenGL to handle occlusion queries, and the queries are now implemented in commodity graphics hardware. The present semantics in Direct3D [32] and OpenGL [27] require returning the number of visible pixels, and thus the implementations cannot early-exit when the first visible pixel is found. This results in unnecessarily large pixel processing requirements and decreased performance in applications that need only a binary result.

Bittner et al. [12] reduce the number of occlusion queries by always rendering the objects that were visible in the previous frame. Occlusion queries are issued only for the nodes where the previous database traversal terminated. Furthermore, the result of a query is waited for only if the node was hidden in the previous frame. Otherwise the result is used during the next frame. These techniques greatly reduce the number of pixels processed by the queries and the synchronization issues between the CPU and the GPU.

In occlusion culling, the word *conservative* refers to an approximation that never causes rendering artifacts. Zhang et al. [100] note that an occlusion query can be split into two sub-tests: one for coverage and one for depth. The result of an occlusion query is conservatively correct, as long as the coverage test is performed at full resolution. Their lower resolution depth test uses a *depth estimation buffer*, which stores a conservative Z_{max} for a tile of pixels.

Occlusion culling systems

Zhang et al. [100] classify a subset of objects as prominent occluders during a preprocessing stage. The occluders are then used for culling other objects at runtime. Although the results are convincing, the static occluder classification may fail to block some lines-of-sight, especially in dynamic scenes, and the performance degrades.

Klosowski and Silva [56] render an approximation of a static scene using a fixed budget of resources. They subdivide the scene into convex cells as a preprocess, and compute a *solidity estimate* for each cell. At runtime, the cells are rendered in the order of decreasing importance, so that the importance of a cell is estimated from the accumulated solidity values of the corresponding screen-projection area. A simple extension [57] finds the rest of the visible objects by using hardware occlusion queries.

Baxter et al. [10] describe a two-GPU occlusion culling system for interactive walkthroughs in complex environments. The first GPU generates a Z-buffer for occlusion culling purposes, and the second GPU performs the actual rendering once the visible objects have been determined. Govindaraju et al. [40] present a three-GPU culling system. The first GPU executes occlusion queries for the current frame, and passes the visible objects to the other two GPUs. The second GPU renders the visible objects, and the third GPU uses them to generate a Z-buffer for the *next* frame. In order to avoid transferring the Z-buffer, the first and third GPUs switch functions every frame. Impressive results are shown in complex environments. Both systems increase the latency of rendering a frame, and dynamic updates to the spatial database are not considered.

Wonka et al. [95] compute the visibility simultaneously from multiple viewpoints by shrinking the occluders sufficiently. Their results for large scenes are impressive but the approach does not lend itself easily to dynamic environments due to the required preprocessing. Publication [P3] describes a generic occlusion culling system for dynamic scenes.

Order-independent transparency

In order to generalize Z-buffering [17] to semi-transparent surfaces, the A-buffer algorithm [16, 78, 54] stores all visible surfaces for each pixel. The final color of a pixel is computed by first sorting the surfaces according to increasing depth and then blending them from back to front. Alternatively, Painter's algorithm [70] can be used for ordering the semi-transparent surfaces.

Mammen [62], Diefenbach [31], and more recently Everitt [34] describe multi-pass algorithms that *peel* semi-transparent layers one at a time, and blend them to the frame buffer. Two depth buffers are used concurrently for determining the farthest unprocessed semi-transparent surface for each pixel. The process is repeated until all semi-transparent surfaces have been blended. Order-independent transparency is computed correctly since the per-pixel sorting is performed implicitly using selection sort. Wittenbrink [93] proposes performing the peeling operation fully in hardware. All visible samples of semi-transparent surfaces are first stored into a separate ring buffer, and then peeled to the frame buffer. Publication [P1] reduces the memory requirements of Wittenbrink's approach.

2.2 Shadow Volumes

Shadow volumes [28] define regions of space that are in shadow with respect to an infinitesimal light source. A shadow volume consists of three parts. The *light cap* is the shadow casting object itself, and the *dark cap* closes the far end of the shadow volume beyond the attenuation range of the light source. The *side quads* that are extruded from the silhouette edges of the shadow caster connect the two caps into a closed volume. See Figure 2.1 for an illustration.

Several authors [20, 84, 23] have described object-space techniques for clipping the shadow receivers into lit and shadowed parts. Image-space

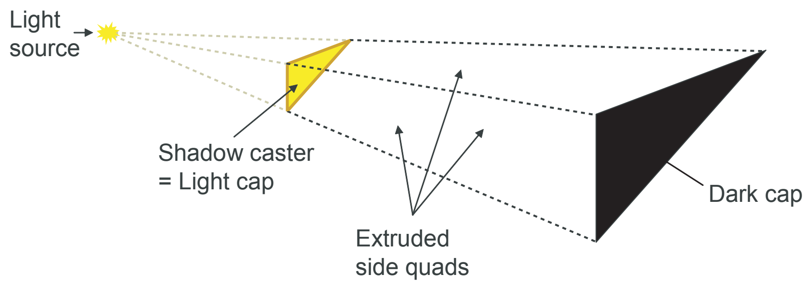


Figure 2.1: A shadow volume consists of a light cap, extruded side quads, and a dark cap. Together these parts bound a closed volume, which is in shadow with respect to the light source.

methods [37] avoid the clipping by computing the shadows directly to the frame buffer. Such algorithms need to determine if a three-dimensional point **p** is inside a shadow volume. Let **s** denote a reference point that is guaranteed to be outside the shadow volume, and then count how many times the line segment from **s** to **p** enters and exits the shadow volume. If the number of entry events is larger than the number of exit events, **p** is in shadow. Heidmann's *Z-pass* method places the reference point at the camera position [48], and is therefore limited to cases when the viewport is fully outside shadow volumes.

Figure 2.2 illustrates a robust variant (*Z-fail*), which sets **s** to be infinitely far behind the pixel as seen from the camera [35]. Infinity is a practical choice because it is never inside a shadow volume. All events along the line segment need to be counted, and thus the shadow volumes cannot be clipped to the far plane of the view frustum. Everitt and Kilgard [35] present two solutions for avoiding the clipping.

In practice, Z-fail shadow volumes are rendered into the stencil buffer [73] so that hidden pixels of the frontfacing and backfacing triangles decrement and increment the stencil values, respectively [35]. This creates a shadow mask in the stencil buffer, where values greater than zero indicate that a pixel is in shadow. Finally, the scene is rendered with full lighting, and the per-pixel shadow terms are fetched from the stencil buffer.

Shadow volumes are quite simple to implement and map well to rasterization architectures. Also, the algorithm does not suffer from aliasing artifacts due to its geometric nature. However, even relatively small objects often create shadow volumes that cover a large area on the screen. As a result, the number of processed pixels and the related bandwidth to video memory becomes a serious performance bottleneck. This fillrate problem is addressed in publication [P2].

Shadow volume optimizations

Nvidia's extension [72] allows the application to define the minimum and maximum depth values for a shadow volume. The entire rasterized shadow volume has to be inside the defined *depth bounds*. Then, the rasterization of the shadow volume can be limited to the pixels whose Z-buffer values are

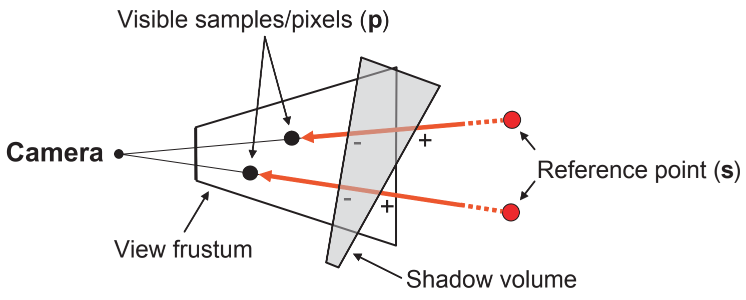


Figure 2.2: A robust hardware accelerated version of the shadow volume algorithm places the non-shadowed reference point to infinitely far behind the pixel as seen from camera. The entry (+) and exit (−) events are then counted along the line segment from the reference point to the pixel. All events must be considered, and thus far clipping of shadow volumes needs to be disabled.

inside the bounds, as demonstrated in Figure 2.3. Depth bounds testing is effective when the shadow volume is approximately perpendicular to the viewing direction. However, with other orientations the bounds may cover a major part of the scene, and the efficiency degrades. Also, the testing does not accelerate the rendering of shadowed regions. In some applications, the bounds can be made tighter by clamping them to the scene geometry, e.g., walls of a room.

Lengyel [59] defines a scissor rectangle so that rasterization work is limited to the parts of the screen that are inside the attenuation range of a light source. Lloyd et al. [60] rasterize shadow volumes only to the regions of space that contain shadow receivers. They also use occlusion queries to avoid casting shadows from objects that are entirely in shadow. McGuire et al. [66] present an algorithm for creating the dark cap with a minimum number of triangles. The Z-pass algorithm does not need capping unless

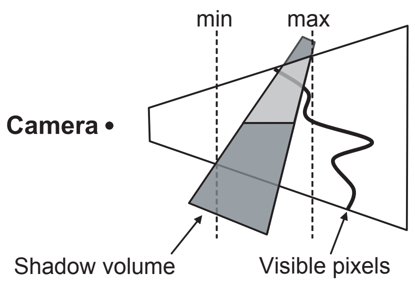


Figure 2.3: Depth bounds allow an application to define the minimum and maximum depth values for a shadow volume. The rasterized shadow volume can affect only the shadow receivers that are inside the depth bounds, and thus the shadow volume rasterization can be limited to pixels whose depth values are inside the bounds (marked with light gray).

the shadow volume intersects the viewport [35], and thus it can be faster to use Z-fail only for the shadow volumes that may intersect the viewport. A two-sided stencil test [35] halves the geometry processing requirements of shadow volumes.

Generalizations of shadow volumes

Bergeron [11] extends shadow volumes to non-closed meshes. Brotman and Badler [15] compute soft shadows by representing an area light source using a number of point lights. Nishita and Nakamae [71] and Chin and Feiner [21] construct separate volumes for the umbra and penumbra, i.e., full shadow and partial shadow. Soft shadows are then computed analytically for pixels inside the penumbra region. Tanaka and Takahashi [86] present efficient algorithms for culling the objects that do not affect the shadow term of a given point.

A penumbra wedge [2] is the bounding volume of the penumbra region defined by a silhouette edge of an occluder. Thus the shadow term of a point can be affected by an edge only if the point is inside the corresponding wedge. Assarsson and Akenine-Möller [8] use penumbra wedges, and generate approximate soft shadows in real-time.

2.3 Shadow Maps

Shadow mapping [91, 81] generates a depth buffer from the point of view of a light source. This image, called a shadow map, is a discretized representation of the scene geometry as seen by the light source. A pixel is in shadow if its projection to the image plane of the light source is behind the corresponding depth value in the shadow map, Figure 2.4. The fundamental caveat is that a regular shadow map does not, in general, contain information that corresponds to the visible samples of the output image. This results in aliasing artifacts, such as jagged shadow boundaries and incorrect self-shadowing. An obvious solution is to increase the shadow map resolution until the artifacts disappear, but often this results in excessive memory consumption and high computational requirements. Publication [P5] presents a novel solution to this problem.

A single shadow map can cover a limited field of view, and multiple shadow maps are required for omni-directional light sources that are inside or near the view frustum. Typically the shadow map lookups are performed for each pixel of the rendered image, even if only a fraction of the pixels gets projected inside a particular shadow map. For example, the floor of a room is often represented as a single object; yet only a part of it is generally affected by a single shadow map. Publication [P4] presents a solution to this performance issue.

Resolution mismatch with shadow maps

The resolution mismatch between an output image and a shadow map has been studied by several authors. Brabec and Seidel [13] shrink the frustum of the light source to tightly enclose the visible part of the view frustum. Fernando et al. [36] and Arvo [6] divide the shadow map area adaptively. The resolution of the shadow map is increased in the vicinity of

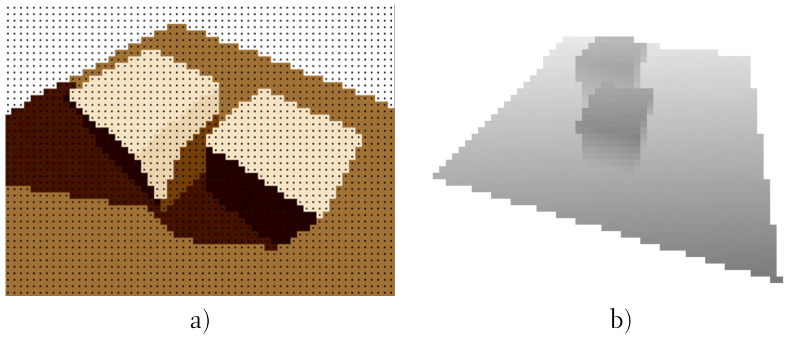


Figure 2.4: a) A simple scene rendered with shadows. The black dots are the sampling points. b) The corresponding shadow map is color-coded so that darker colors are closer to the light source. A sampling point in (a) is in shadow if its projection to the image plane of the light source is behind the corresponding shadow map depth value.

depth discontinuities, and in the areas where the resolution of the output image exceeds the resolution of the shadow map. Sen et al. [82] augment shadow maps with silhouettes of shadow casting objects. The proposed implementation is limited to scenes that have low tessellation because only one silhouette vertex is stored per pixel. In more complex scenes, the approximation leads to artifacts.

Stamminger and Drettakis [85] generate shadow maps in post-perspective space, i.e., after the projection to the view frustum. This perspective shadow mapping improves the shadow quality in many scenes. Special treatment is needed to ensure that all shadow casting objects are taken into account. Wimmer et al. [92] further improve the quality by performing a perspective transformation in light space. Their method also avoids singularities and special cases of perspective shadow maps. Martin and Tan [63] present a rather similar method, which approximates the view frustum using a trapezoid, and then warps the trapezoid into a shadow map. Chong and Gortler [22] show how to compute an accurate shadow map for an arbitrary plane. In addition to a regular shadow map, separate maps are allocated for a few of the most important planes. A pixel shader then selects the most appropriate shadow map for each pixel. Concurrently with publication [P5], Johnson et al. [53] provide the correct sampling points for shadow maps, and thus solve the resolution issues.

False self-shadowing with shadow maps

False self-shadowing occurs when an illuminated surface appears to reside behind its own shadow map footprint. The problem is caused primarily by the limited precision of a discrete shadow map. Williams [91] proposes adding a constant bias value to the transformed depth value in order to weaken the coincidence test. Unfortunately, a constant bias factor cannot handle all situations satisfactorily. The artifacts can be reduced by replacing the depth of the closest surface either with the depth of the second closest surface [89] or with the average depth of the two closest surfaces [96, 41].

Grant [41] suggests storing the plane equations in addition to the depth values. Hourcade and Nicolas [50] assign each object or primitive a unique ID, and augment the depth values with the corresponding IDs. Incorrect self-shadowing is reduced by comparing the IDs instead of depth values. These techniques help in many cases, but fail when more than one object or primitive should be represented inside a single shadow map pixel.

Publication [P5] and Johnson et al. [53] significantly reduce self-shadowing artifacts by using irregular shadow maps.

Generalizations of shadow maps

Deep shadow maps by Lokovic and Veach [61] generalize shadow maps to semi-transparent surfaces and volumetric effects while also reducing the aliasing artifacts in off-line rendering. Dachsbacher and Stamminger [29] augment shadow maps with irradiance values and normal vectors in order to approximate subsurface scattering.

Reeves et al. [76] and Brabec and Seidel [14] create approximate soft shadows by blurring the shadow boundaries. Several authors have proposed shadow map-based methods for approximating the penumbra regions more accurately [18, 98, 7, 52].

2.4 Hybrid Shadow Algorithms

McCool [64] reconstructs a shadow volume from a shadow map by using an edge detection algorithm. The resulting shadow volume does not contain any overlapping volumes, and thus a lot of redundant rasterization work can be avoided. Due to the discrete resolution of shadow maps and the use of a heuristic edge detection, robustness issues arise. Govindaraju et al. [39] use a combination of shadow maps and object-space shadows on a cluster of PCs to generate hard shadows in complex environments.

Chan and Durand [19] find the boundary pixels of shadow regions by using a low resolution shadow map. The boundary pixels are then processed accurately using shadow volumes, while the rest of the pixels are handled with the shadow map. Classification errors are possible due to the limited precision of the shadow map. This work was carried out concurrently with publication [P2], and both are based on the same observation that only the shadow boundaries need accurate processing. However, the algorithms are quite different, and in particular, our method does not introduce artifacts.

2.5 Shadow Rays

In ray tracing [90], shadows from infinitesimal light sources are computed by tracing a shadow ray from the point of interest to the light source. If the shadow ray is blocked, the point is in shadow. Haines and Greenberg [45] speed up the intersection tests by exploiting coherence among shadow rays.

Parker et al. [74] render approximate soft shadows by using "soft-edged" objects and a single shadow ray per pixel. Stochastic ray tracing algorithms [26] compute soft shadows by sampling an area light source using a large number of shadow rays.

3 VISIBILITY ALGORITHMS

Publication [P3] describes a general purpose visibility software library for dynamic scenes. New techniques are proposed for reducing the cost of occlusion queries, and for performing efficient occlusion culling using silhouettes of meshes.

3.1 Visible Point Tracking and Conditional Occlusion Queries

An object is visible if there is at least one non-blocked line-of-sight between the camera and the object. Publication [P3] exploits this observation by caching the object-space coordinates of an arbitrary visible point between frames, and by testing the visibility of that point before proceeding with more involved occlusion queries. Figure 3.1 illustrates the idea of visible point tracking.

The framework receives the 3D coordinates of some, randomly selected visible point as feedback from the occlusion culling system. Adding hardware support for this feedback should be worthwhile because in our tests 80–100% of the visible objects were found using visible point tracking, i.e., with single-pixel occlusion queries. This greatly reduced the pixel processing requirements of the queries.

Publication [P3] also proposes conditional occlusion queries. Multiple dependent occlusion queries are issued simultaneously, and the latter queries are executed only if the previous ones pass. This approach reduces the amount of synchronization between the spatial database and the occlusion culling system, thus improving performance.

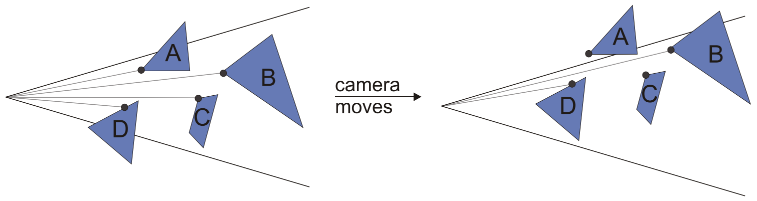


Figure 3.1: In visible point tracking, an object-space point (black dot) is cached for each object. After the camera is moved, the visibility of objects **B** and **D** can be proven with a single-pixel occlusion query to their cached points, whereas objects **A** and **C** need further tests.

3.2 Occlusion Culling Using Silhouettes of Meshes

Publication [P3] introduces a fast software-only system that makes occlusion culling usable in computer games on the heterogeneous PC platform. According to the game developers using the described system, an important reason for employing occlusion culling is to make games playable on older and slower machines, thus widening the potential audience. Hardware-assisted occlusion queries have become available only recently.

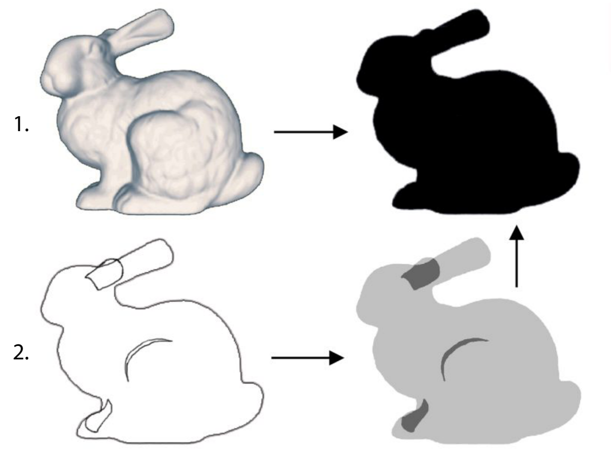


Figure 3.2: This figure illustrates two ways of determining the pixels that are covered by a mesh. The first rasterizes the mesh using triangles. The second reconstructs the depth complexity function from the silhouette edges of the mesh, and finally converts the depth complexity function to the coverage mask.

Most occlusion culling methods rasterize the occluding primitives into a Z-buffer. Publication [P3] exploits the observation of Zhang et al. [100], and replaces the Z-buffer with a per-pixel coverage mask and a low-resolution depth estimation buffer. This allows us to develop a specialized coverage rasterizer, and quickly estimate the depth values of occluders from their bounding boxes.

Figure 3.2 illustrates a new method for generating the coverage mask. A polygonal silhouette is first extracted in object space. The silhouette edges always form closed loops, and mark all potential changes in visibility, i.e., *visibility events*. Visibility events are the first (generalized) derivative of the depth complexity function, which indicates how many times a pixel is covered, and thus the function can be reconstructed by integrating the visibility events. In practice, the reconstruction is implemented by rasterizing the projected edges using the value $+1$ for the left edges and -1 for the right edges. The per-pixel depth complexity values are then accumulated from left to right for up to 64 pixels in parallel. The resulting peak fillrate is comparable to the latest GPUs.

The method of reconstructing the coverage mask using silhouettes is appealing because the average number of silhouette edges in a mesh of n triangles is typically much lower than n; estimates vary between $n^{0.5}$ [55] and $n^{0.8}$ [65]. Publication [P3] reduces the cost of silhouette extraction by employing a two-pass method. The first pass caches an intermediate presentation from the current camera position. Then, the silhouette can be efficiently extracted from any point inside the bounding radius of the intermediate presentation. This method allows caching the intermediate representation between frames, thus amortizing the cost of silhouette extraction over several frames.

4 SHADOW ALGORITHMS

This chapter outlines the new contributions to the performance and quality of shadow algorithms. Publication [P5] provides the correct sampling points for shadow maps, and greatly diminishes aliasing artifacts. Another new technique [P4] reduces the number of redundant shadow map lookups. The performance of shadow volumes is improved by using a new hierarchical rendering technique [P2].

4.1 Alias-Free Shadow Maps

Publication [P5] presents a novel solution to the aliasing problems of shadow maps. After the visible surfaces have been determined from the point of view of the camera, the visible samples $P(x, y, z)$ are transformed to the image plane of the light source, producing sampling points (x', y') and the corresponding light-space depth values z'. Figure 4.1 visualizes the samples in screen space and on the image plane of the light source. The (x', y') are the optimal sampling points for the shadow map, and exactly correspond to the intersections of shadow rays and the image plane of the light source. Thus the method can also be seen as an optimization technique for shadow rays. Finally, the irregularly spaced points (x', y') are used as sampling points when the scene is rasterized from the light source.

The large empty areas in Figure 4.1b are not visible from the camera, and thus need no shadow information. The irregular shadow maps never suffer from jagged shadow boundaries or other resolution-related aliasing artifacts. Self-shadowing artifacts are also greatly reduced as the only remaining source of inaccuracy is the transformation between the coordinate systems. Unlike earlier methods [91], our bias value does not depend on the scene or the viewing parameters.

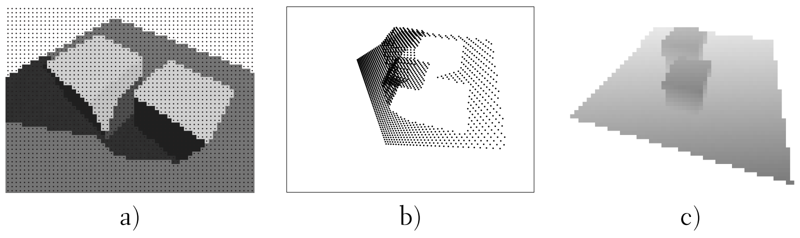


Figure 4.1: a) A simple scene rendered with shadows. The pixel centers are marked using black dots. b) The visible samples at pixel centers of (a) transformed to the image plane of the light source. The irregularly spaced dots are used as sampling points when the scene is rasterized to the shadow map. c) The corresponding traditional shadow map is shown for comparison purposes only. In a traditional shadow map algorithm, the regularly sampled map (c) would be tested exactly at the sampling points shown in (b). Clearly, the regular structure of (c) is not suitable for accurately answering the queries.

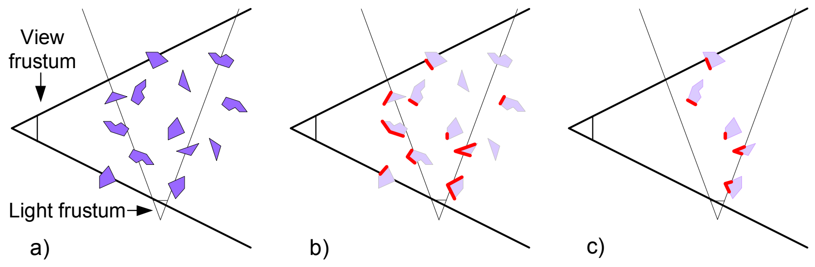


Figure 4.2: a) A view frustum, a light frustum, and several objects. b) The regions that are visible from the viewpoint are marked with bold lines. A brute force algorithm would perform shadow map lookups for these regions. c) The visible regions that are inside the light frustum, and would thus get projected inside the corresponding shadow map. Our method processes exactly these regions.

4.2 Limiting the Number of Shadow Map Lookups

Publication [P4] describes a simple method for limiting the number of shadow map lookups to the pixels that get projected inside the shadow map. This is accomplished by using the frustum of the light source as a shadow volume [28]. In contrast to the original shadow volumes, this volume contains the parts of the scene that can be lit by the light source. The light volume is rasterized into the stencil buffer similarly to the stencil buffer-based shadow volumes [48]. Then, a pixel shader program that performs the shadow map lookups and per-pixel lighting computations is executed only for the pixels that are inside the light frustum. Figure 4.2 illustrates the reduction in pixel processing requirements. The technique is useful for local light sources, i.e., when the light frustum covers only a part of the view frustum. Compared to an optimized implementation, 2.3–5.7 times fewer pixels were processed in our test scenes, and the overall performance improved by a factor of up to 2.2.

Omni-directional light sources and cube maps
In Publication [P4] we used six shadow maps to model an omni-directional light source, and performed the lookups separately for each shadow map. Alternatively, the shadow maps could be represented as a cube map. This would reduce the number of redundant shadow map lookups in the comparison method and improve its performance. However, this would affect the results in only one of the tested scenes. Our new method could be adapted to support cube maps by replacing the light frustum with a bounding volume that encloses the regions affected by the light source.

4.3 Hierarchical Rendering of Shadow Volumes

Publication [P2] introduces a hierarchical rasterization technique for shadow volumes. The method reduces the number of processed pixels by exploiting coherence in shadowed and lit regions, as demonstrated in

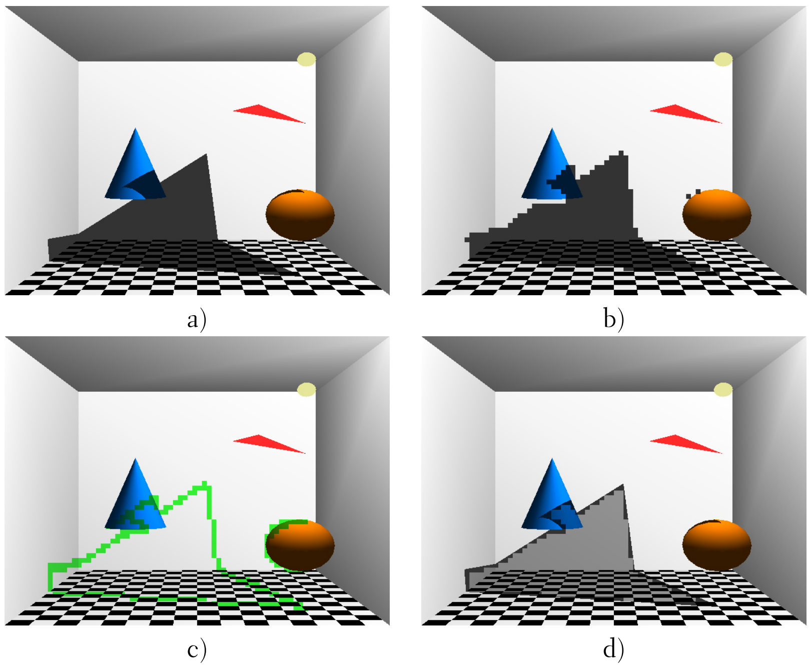


Figure 4.3: a) Shadows in a simple scene that has one shadow casting triangle. b) The result of executing the shadow volume algorithm for a single sampling point per 8×8 pixel tile, and then using the result for all pixels inside the tile. Despite the blocky appearance of shadows, large areas are correctly classified. c) The tiles that may contain a shadow boundary are marked with light green. d) The result of executing the per-pixel shadow volume algorithm for potential boundary tiles (dark gray), and copying the low-resolution shadows (b) to the rest of the tiles.

Figure 4.3. In order to implement the technique efficiently, hardware modifications are required. The step that warrants detailed explanation is the classification to potential boundary tiles.

Shadow volumes are closed by definition, and the triangles defining a shadow volume indicate all potential transitions between lit and shadowed regions. Now, consider an arbitrary bounding volume inside a scene. If the bounding volume is not intersected by any of the shadow volume triangles, there cannot be shadow boundaries inside the bounding volume, and the entire bounding volume is either fully lit or fully in shadow. Thus the entire bounding volume can be classified by executing the shadow volume algorithm for a single, arbitrarily chosen point inside the bounding volume. Publication [P2] constructs for each $N \times M$ pixel tile an axis-aligned bounding box in 3D screen space. The box is defined by the area of $N \times M$ pixels along with the minimum and maximum depth values inside the tile. This is a practical choice because the Z_{min} and Z_{max} values are already maintained by the latest GPUs.

A tile may contain a shadow boundary if at least one shadow volume triangle intersects with the bounding box of the tile. This classification is complete after the entire shadow volume has been processed, and thus the application needs to mark the beginning and end of each shadow volume. The low-resolution shadows are computed simultaneously with the tile classification. Once the classification is finished, the second stage of the algorithm performs per-pixel shadow volume computations for the boundary tiles, and copies the low-resolution shadows to the rest of the tiles. In order to reduce the amount of data written to the stencil buffer, a two-level stencil buffer is proposed. An alternative implementation could employ some kind of tile-based compression of the stencil buffer.

In our test scenes, 2.8–11.5 times fewer pixels were processed compared to a state-of-the-art hardware method [72]. The related accesses to external memory were reduced by a factor of 2.4–17.1. As a result, the fillrate problem of shadow volumes was significantly reduced.

5 GRAPHICS HARDWARE: DELAY STREAMS

The delay stream architecture [P1] extends traditional stream processing methods, as illustrated in Figure 5.1. The execution units **A** and **B** are connected using two data paths: one immediate and one delayed. The immediate *analysis* is executed before a data element is pushed to the delay unit, which is typically implemented as a ring buffer in external memory. By the time the element arrives to unit **B**, the analysis has been performed for a large number of subsequent elements. Therefore unit **B** can use information about data elements that were submitted *after* the element it currently processes. This ability to break causal relationships is the fundamental property of delay streams.

5.1 Delayed Occlusion Culling

The delay stream architecture can be used for occlusion culling, as discussed in publication [P1]. The units in Figure 5.1 are placed so that the early occlusion test in Figure 1.5 is embedded into unit **A**. Unit **A** builds conservative low-resolution Z_{min} and Z_{max} buffers without consulting external memory. Unit **B** performs another occlusion test by using the low-resolution buffers that have been augmented by a substantial number of triangles that were submitted after the current triangle.

In commonly used benchmark scenes, the resulting depth complexity was reduced to within 30% of the theoretical optimum by using a 2MB ring buffer. This confirms that the ordering of the input triangles affects the efficiency of delayed occlusion culling only weakly. Compared to the commonly used early occlusion test [69], 1.8–4.0 times fewer pixels were rendered using delayed occlusion culling. The communication to external video memory was reduced almost proportionally. In many applications the overall performance is limited by pixel processing resources, and thus delayed occlusion culling is expected to improve the frame rate significantly.

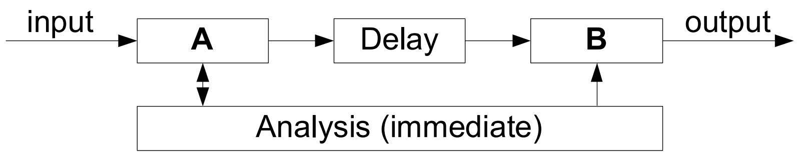


Figure 5.1: The delay stream architecture contains two paths from execution unit **A** to unit **B**, one goes through a delay and the other one is immediate. Once a data element emerges from the delay, the analysis path has already gathered a substantial amount of data about elements that were submitted *after* the current element.

Utilization of vertex and pixel shaders

Publication [P1] also mentions that a delay stream between vertex and pixel shaders should improve their utilization. Unfortunately, we were not able to measure this effect because it would have required full chip-level simulations, and not all components were available. Recently Sheaffer et al. [83] introduced a simulation framework for graphics architectures, and as a test case they provided statistics about vertex and pixel shader utilization of a hypothetical GPU in the game *Return to Castle Wolfenstein: Enemy Territory*. Their results show that with modest buffering the vertex shaders are idle when pixel shaders are busy and vice versa. The drastically increased buffering capacity provided by a delay stream should improve the utilization considerably.

An alternative solution would be to use the same physical units for vertex and pixel shading, and perform load balancing dynamically. This would provide high shader utilization with a smaller amount of buffering. Such an approach is a viable alternative, as the instruction sets of the shaders are likely to become identical.

5.2 Other Applications of Delay Streams

Publication [P1] explains two other uses of delay streams: order-independent transparency and discontinuity edge detection for adaptive antialiasing. The latter is outside the scope of this thesis, and is thus not reviewed here. A hardware implementation [93] of depth peeling [62, 34] is an efficient method for computing order-independent transparency. Its primary problems are the required amount of external memory and the related data transfers. The memory consumption was reduced by a factor of 7.0–16.3 when a delay stream was used.

The hierarchical shadow volume algorithm (Section 4.3) can be implemented in a single rendering pass by employing a delay stream [P2]. This variant has two advantages over the two-pass implementation: only a single geometry pass is required and pixel shaders can be kept busy by pipelining the processing of multiple shadow volumes.

6 MAIN RESULTS OF THE THESIS AND CONTRIBUTIONS OF THE AUTHOR

None of the publications [P1–P5] have previously formed a part of another thesis. The main results of this thesis and the author's contributions can be summarized as follows.

Publication [P1]
Delay streams are introduced as a general hardware solution to order-dependent problems. The idea is applied to occlusion culling, order-independent transparency, and discontinuity edge detection. Significant performance improvements are shown in all three application areas.

The author and Mr. Petri Nordlund formulated the initial concept of delay streams for the purposes of occlusion culling. The author carried out the design, testing, and documentation of the delayed occlusion culling (Section 3) and order-independent transparency (Section 4). Mr. Ville Miettinen offered insightful feedback on the two sections, and designed and documented the algorithm for detecting discontinuity edges (Section 5). The rest of the paper was written by the author and Mr. Miettinen in equal proportions.

Publication [P2]
A hierarchical rendering technique is presented as a solution to the well-known fillrate problem of shadow volumes. The algorithm executes in two stages. First, low-resolution shadows are computed and the corresponding 8×8 pixel tiles are classified to be either fully lit, fully in shadow, or intersected by a shadow boundary. In the second pass, per-pixel rasterization is performed only to the boundary tiles. As a result, the number of processed pixels was reduced by up to over 90%. Two implementations are outlined along with new optimizations, such as a hierarchical stencil buffer. The most efficient variant uses a delay stream [P1], which allows single-pass execution.

The author and Dr. Tomas Akenine-Möller designed and implemented the algorithm in tight collaboration, and thus the new scientific contribution is divided equally. The author wrote 60% of the text.

Publication [P3]
Both existing and new algorithms are combined into a framework that performs efficient occlusion culling in dynamic environments without preprocessing. Novel algorithms are presented for performing occlusion culling using silhouettes of meshes, silhouette extraction, adaptive occluder selection, and for reducing the cost of occlusion queries.

The author and Mr. Ville Miettinen designed and implemented the whole system together, and it is therefore difficult to single out individual contributions. However, Mr. Miettinen had primary responsibility of the dynamic spatial database, and the author was primarily responsible for the visibility solver and silhouette rasterization. The author wrote half of the text.

Publication [P4]
This article introduces a method for reducing the number of shadow map lookups by utilizing techniques related to shadow volumes. As a result, almost an optimal amount of per-pixel shadow computations is performed, and the resulting frame rate increased up to 2.2 times in the used test scenes.

Mr. Jukka Arvo made the observation that shadow map lookups can be implemented as a part of deferred shading on currently available graphics hardware. The author utilized this observation for bounding the parts of the scene that can be affected by a shadow map by using a light frustum and the stencil buffer, and also wrote 90% of the text. Mr. Arvo implemented the described algorithm.

Publication [P5]
This article abandons the regular structure of shadow maps. The visible samples are first transformed from screen space to the image plane of a light source. The transformed points are then used as sampling points when the geometry is rasterized into the shadow map. This provides optimal sampling points for shadow maps, and matches the result of tracing a shadow ray from each visible sample to the light source. As a result, all resolution issues of shadow maps are eliminated. Incorrect self-shadowing is also greatly reduced, and semi-transparent shadow casters and receivers can be supported.

The author invented the original idea and wrote the paper. The hierarchical rasterization algorithm was designed by Mr. Samuli Laine, who was also responsible for the programming work.

BIBLIOGRAPHY

[1] John Airey, John Rohlf, and Fredrick P. Brooks. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):141–150, 1990.

[2] Tomas Akenine-Möller and Ulf Assarsson. Approximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges. In *Proceedings of the 13th Eurographics Workshop on Rendering*, pages 297–305. Eurographics Association, 2002.

[3] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering, 2nd edition*. A.K. Peters Ltd., 2002.

[4] Tomas Akenine-Möller and Jacob Ström. Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones. *ACM Transactions on Graphics*, 22(3):801–808, 2003.

[5] Arthur Appel. Some Techniques for Shading Machine Renderings of Solids. In *AFIPS Conference Proceedings*, volume 32, pages 37–45, 1968.

[6] Jukka Arvo. Tiled Shadow Maps. In *Proceedings of Computer Graphics International*, pages 240–247. IEEE Computer Society, 2004.

[7] Jukka Arvo, Mika Hirvikorpi, and Joonas Tyystjärvi. Approximate Soft Shadows with an Image-Space Flood-Fill Algorithm. *Computer Graphics Forum*, 23(3):271–279, 2004.

[8] 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.

[9] Dirk Bartz, Michael Meißner, and Tobias Hüttner. Extending Graphics Hardware for Occlusion Queries in OpenGL. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 97–104. ACM Press, 1998.

[10] William Baxter, Avneesh Sud, Naga Govindaraju, and Dinesh Manocha. GigaWalk: Interactive Walkthrough of Complex Environments. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 203–214. Eurographics Association, 2002.

[11] Philippe Bergeron. A General Version of Crow's Shadow Volumes. *IEEE Computer Graphics and Applications*, 6(9):17–28, 1986.

[12] Jirí Bittner, Michael Wimmer, Harald Piringer, and Werner Purgathofer. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum*, 23(3):615–624, 2004.

[13] Stefan Brabec and Hans-Peter Seidel. Practical Shadow Mapping. *Journal of Graphics Tools*, 7(4):9–18, 2002.

[14] Stefan Brabec and Hans-Peter Seidel. Single Sample Soft Shadows using Depth Maps. In *Graphics Interface 2002*, pages 219–228, 2002.

[15] Lynne Brotman and Norman Badler. Generating Soft Shadows with a Depth Buffer Algorithm. *IEEE Computer Graphics and Applications*, 4(10):5–12, 1984.

[16] Loren Carpenter. The A -buffer, An Antialiased Hidden Surface Method. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, pages 103–108. ACM, 1984.

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

[18] Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. In *Proceedings of the Eurographics Symposium on Rendering*, pages 208–218. Eurographics Association, 2003.

[19] Eric Chan and Frédo Durand. An Efficient Hybrid Shadow Rendering Algorithm. In *Proceedings of the Eurographics Symposium on Rendering*, pages 185–195. Eurographics Association, 2004.

[20] Norman Chin and Steven Feiner. Near real-time shadow generation using BSP trees. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, pages 99–106. ACM, 1989.

[21] Norman Chin and Steven Feiner. Fast Object-Precision Shadow Generation for Area Light Source using BSP Trees. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 21–30. ACM Press, 1992.

[22] Hamilton Chong and Steven Gortler. A Lixel for every Pixel. In *Proceedings of the Eurographics Symposium on Rendering*, pages 167–172. Eurographics Association, 2004.

[23] Yiorgos Chrysanthou and Mel Slater. Shadow volume BSP trees for computation of shadows in dynamic scenes. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 45–50. ACM Press, 1995.

[24] James Clark. Hierarchical Geometric Model for Visible Surface Algorithms. *Communications of ACM*, 19(10):547–554, 1976.

[25] Daniel Cohen-Or, Yiorgos Chrysanthou, Claudio Silva, and Frédo Durand. A Survey of Visibility for Walkthrough Applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, 2003.

[26] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. In *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, pages 137–145. ACM, 1984.

[27] Matt Craighead. NV_occlusion_query specification. http://www.nvidia.com/dev_content/nvopenglspecs/-GL_NV_occlusion_query.txt, 2002.

[28] Frank Crow. Shadow Algorithms for Computer Graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 77)*, pages 242–248. ACM, 1977.

[29] Carsten Dachsbacher and Marc Stamminger. Translucent Shadow Maps. In *Proceedings of Eurographics Symposium on Rendering*, pages 197–201. Eurographics Association, 2003.

[30] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, pages 21–30. ACM, 1988.

[31] Paul Diefenbach. *Pipeline Rendering: Interaction and Realism Through Hardware-based Multi-Pass Rendering.* PhD thesis, University of Pennsylvania, 1996.

[32] DirectX. Microsoft DirectX SDK Documentation. http://www.microsoft.com/directx, 2004.

[33] Frédo Durand. *3D Visibility: Analytical Study and Applications*. PhD thesis, Université Grenoble I - Joseph Fourier Sciences et Géographie, 1999.

[34] Cass Everitt. Interactive Order-Independent Transparency. http://developer.nvidia.com, 2001.

[35] Cass Everitt and Mark Kilgard. Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. http://developer.nvidia.com, 2002.

[36] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive Shadow Maps. In *Proceedings of ACM SIGGRAPH 2001*, pages 387–390. ACM Press, 2001.

[37] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, John G. Eyles, and John Poulton. Fast spheres, shadows, textures, transparencies, and image enhancements in pixel-planes. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, pages 111–120. ACM, 1985.

[38] Andrew Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, 1984.

[39] Naga K. Govindaraju, Brandon Lloyd, Sung-Eui Yoon, Avneesh Sud, and Dinesh Manocha. Interactive Shadow Generation in Complex Environments. *ACM Transactions on Graphics*, 22(3):501–510, 2003.

[40] Naga K. Govindaraju, Avneesh Sud, Sung-Eui Yoon, and Dinesh Manocha. Interactive Visibility Culling in Complex Environments using Occlusion-Switches. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 103–112. ACM Press, 2003.

[41] Charles Grant. *Visibility Algorithms in Image Synthesis*. PhD thesis, University of California, 1992.

[42] Ned Greene and Pat Hanrahan. Method and apparatus for occlusion culling in graphics systems. US patent 6480205, 2002.

[43] Ned Greene and Michael Kass. Hierarchical Z-Buffer Visibility. In *Proceedings of ACM SIGGRAPH 93*, pages 231–240. ACM Press, 1993.

[44] Eric Haines and Tomas Möller. Real-Time Shadows. In *Proceeding of Game Developers Conference*, pages 335–352, 2001.

[45] Eric A. Haines and Donald P. Greenberg. The Light Buffer: A Ray Tracer Shadow Testing Accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, 1986.

[46] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A Survey of Real-Time Soft Shadows Algorithms. *Computer Graphics Forum*, 22(4), 2003.

[47] Denis Haumont, Olivier Debeir, and François Sillion. Volumetric Cell-and-Portal Generation. *Computer Graphics Forum*, 22(3), 2003.

[48] Tim Heidmann. Real Shadows, Real Time. *Iris Universe*, 18:28–31, 1991.

[49] Karl Hillesland, Brian Salomon, Anselmo Lastra, and Dinesh Manocha. Fast and Simple Occlusion Culling using Hardware-Based Depth Queries. Technical Report TR02-039, UNC Chapel Hill, 2002.

[50] J. C. Hourcade and A. Nicolas. Algorithms for Antialiased Cast Shadows. *Computer Graphics*, 9(3):259–265, 1985.

[51] Thomas Hudson, Dinesh Manocha, Jonathan Cohen, Ming Lin, Kenneth Hoff, and Hansong Zhang. Accelerated Occlusion Culling Using Shadow Frusta. In *Proceesings of ACM Symposium on Computational Geometry*, pages 1–10, 1997.

[52] Bjarke Jakobsen, Niels Christensen, Bent Larsen, and Kim Petersen. Boundary Correct Real-Time Soft Shadows. In *Proceedings of Computer Graphics International*, pages 232–239. IEEE Computer Society, 2004.

[53] Gregory S. Johnson, William R. Mark, and Christopher A. Burns. The Irregular Z-Buffer and its Application to Shadow Mapping. Technical report, The University of Texas at Austin, Department of Computer Sciences, April 2004.

[54] Norman Jouppi and Chun-Fa Chang. Z3: An Economical Hardware Technique for High-Quality Antialiasing and Transparency. In *Proceedings of the 1999 Eurographics/SIGGRAPH workshop on Graphics hardware*, pages 85–93. ACM Press, 1999.

[55] Lutz Kettner and Emo Welzl. Contour Edge Analysis for Polyhedron Projections. In *Geometric Modeling: Theory and Practice*, pages 379–394. Springer, 1997.

[56] James T. Klosowski and Claudio T. Silva. The Priorized-Layered Projection Algorithm for Visible Set Estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, 2000.

[57] James T. Klosowski and Claudio T. Silva. Efficient Conservative Visibility Culling Using the Priorized-Layered Projection Algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):365–379, 2001.

[58] Sylvain Lefebvre and Samuel Hornus. Automatic Cell-and-portal Decomposition. Technical report, INRIA Rhône-Alpes, July 2003.

[59] Eric Lengyel. The Mechanics of Robust Stencil Shadows. *http://www.gamasutra.com*, October 2002.

[60] Brandon Lloyd, Jeremy Wendt, Naga Govindaraju, and Dinesh Manocha. CC Shadow Volumes. In *Proceedings of the Eurographics Symposium on Rendering*, pages 197–205. Eurographics Association, 2004.

[61] Tom Lokovic and Eric Veach. Deep Shadow Maps. In *Proceedings of ACM SIGGRAPH 2000*, pages 385–392. ACM Press, 2000.

[62] Abraham Mammen. Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique. *IEEE Computer Graphics and Applications*, 9(4):43–55, 1989.

[63] Tobias Martin and Tiow-Seng Tan. Anti-aliasing and Continuity with Trapezoidal Shadow Maps. In *Proceedings of the Eurographics Symposium on Rendering*, pages 153–160. Eurographics Association, 2004.

[64] Michael D. McCool. Shadow Volume Reconstruction from Depth Maps. *ACM Transactions on Graphics*, 19(1):1–26, 2000.

[65] Morgan McGuire. Observations on Silhouette Sizes. *Journal of Graphics Tools*, 9(1):1–12, 2004.

[66] Morgan McGuire, John F. Hugues, Kevin Egan, Mark Kilgard, and Cass Everitt. Fast, Practical and Robust Shadows. Technical Report CS03-19, Brown University, October 2003.

[67] Michael Meißner, Dirk Bartz, Richard Günther, and Wolfgang Straßer. Visibility Driven Rasterization. *Computer Graphics Forum*, 20(4):283–294, 2001.

[68] Steven Molnar, John Eyles, and John Poulton. PixelFlow: High-Speed Rendering Using Image Composition. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, pages 231–240. ACM, 1992.

[69] Steve Morein. ATI Radeon HyperZ Technology. In *Workshop on Graphics Hardware, Hot3D Proceedings*. ACM SIGGRAPH/Eurographics, ACM Press, 2000.

[70] M. Newell, R. Newell, and T. Sancha. A Solution to the Hidden Surface Problem. In *Proceedings of the ACM 1972 National Conference*, pages 443–450, 1972.

[71] Tomoyuki Nishita and Eihachiro Nakamae. Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources. In *IEEE Computer Software and Application Conference*, pages 237–242. IEEE, 1983.

[72] NVIDIA. NVIDIA GeForceFX 5900 GPUs: UltraShadow Technology. http://www.nvidia.com, 2003.

[73] Dave Schreiner OpenGL Architecture Review Board. *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.2.* Addison-Wesley, 1999.

[74] Steven Parker, Peter Shirley, and Brian Smits. Single Sample Soft Shadows. Technical report, University of Utah, UUCS-98-019, October 1998.

[75] PowerVR. PowerVR White Paper: 3D Graphical Processing. http://www.powervr.com, 2000.

[76] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering Antialiased Shadows with Depth Maps. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, pages 283–291. ACM, 1987.

[77] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[78] Andreas Schilling and Wolfgang Straßer. EXACT: Algorithm and Hardware Architecture for an Improved A-buffer. In *Proceedings of ACM SIGGRAPH 93*, pages 85–92. ACM Press, 1993.

[79] Jörg Schmittler, Ingo Wald, and Philipp Slusallek. SaarCOR: A Hardware Achitecture for Ray Tracing. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware*, pages 27–36. ACM Press, 2002.

[80] Jörg Schmittler, Sven Woop, Daniel Wagner, Wolfgang Paul, and Philipp Slusallek. Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip. In *Proceedings of Graphics Hardware*. ACM Press, 2004.

[81] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast Shadows and Lighting Effects Using Texture Mapping. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, pages 249–252. ACM, 1992.

[82] Pradeep Sen, Make Cammarano, and Pat Hanrahan. Shadow Silhouette Maps. *ACM Transactions on Graphics*, 22(3):521–526, 2003.

[83] Jeremy Sheaffer, David Luebke, and Kevin Skadron. A Flexible Simulation Framework for Graphics Architectures. In *Proceedings of Graphics Hardware*, pages 85–94. Eurographics Association, 2004.

[84] Mel Slater. A Comparison of Three Shadow Volume Algorithms. *The Visual Computer*, 9(1):25–38, 1992.

[85] Marc Stamminger and George Drettakis. Perspective Shadow Maps. *ACM Transactions on Graphics*, 21(3):557–562, 2002.

[86] Toshimitsu Tanaka and Tokiichiro Takahashi. Fast Analytic Shading and Shadowing for Area Light Sources. *Computer Graphics Forum*, 16(3):231–240, 1997.

[87] Seth Teller and Carlo Séquin. Visibility Preprocessing for Interactive Walkthroughs. In Thomas W. Sederberg, editor, *Computer Graphics (Proceedings of SIGGRAPH 91)*, pages 61–69. ACM, 1991.

[88] Jay Torborg and James T. Kajiya. Talisman: Commodity Realtime 3D Graphics for the PC. In *Proceedings of ACM SIGGRAPH 96*, pages 353–363. ACM Press, 1996.

[89] Yulan Wang and Steven Molnar. Second-Depth Shadow Mapping. Technical report, The University of North Carolina at Chapel Hill, 1994.

[90] Turner Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23(6):343–349, 1980.

[91] Lance Williams. Casting Curved Shadows on Curved Surfaces. In *Computer Graphics (Proceedings of ACM SIGGRAPH 78)*, pages 270–274. ACM, 1978.

[92] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light Space Perspective Shadow Maps. In *Proceedings of the Eurographics Symposium on Rendering*, pages 143–151. Eurographics Association, 2004.

[93] Craig M. Wittenbrink. R-buffer: A Pointerless A-buffer Hardware Architecture. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 73–80. ACM Press, 2001.

[94] Peter Wonka. *Occlusion Culling for Real-Time Rendering of Urban Environments*. PhD thesis, Institute of Computer Graphics, Vienna University of Technology, 2001.

[95] Peter Wonka, Michael Wimmer, and François Sillion. Instant Visibility. *Computer Graphics Forum*, 20(3), 2001.

[96] Andrew Woo. The Shadow Depth Map Revisited. *Graphics Gems III*, pages 338–342, 1992.

[97] Andrew Woo, Pierre Poulin, and Alain Fournier. A Survey of Shadow Algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, 1990.

[98] Chris Wyman and Charles Hansen. Penumbra maps: Approximate soft shadows in real-time. In *Proceedings of the Eurographics Symposium on Rendering*, pages 202–207. Eurographics Association, 2003.

[99] Feng Xie and Michael Shantz. Adaptive Hierarchical Visibility in a Tiled Architecture. In *Proceedings of the 1999 Eurographics/SIGGRAPH workshop on Graphics hardware*, pages 75–84. ACM Press, 1999.

[100] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff. Visibility Culling Using Hierarchical Occlusion Maps. In *Proceedings of ACM SIGGRAPH 97*, pages 77–88. ACM Press, 1997.