Graphics Hardware (2004)
T. Akenine-Möller, M. McCool (Editors)

A Hierarchical Shadow Volume Algorithm

Timo Aila[*†]   Tomas Akenine-Möller[‡]

[*]Helsinki University of Technology/TML, [†]Hybrid Graphics, Ltd., [‡]Lund University

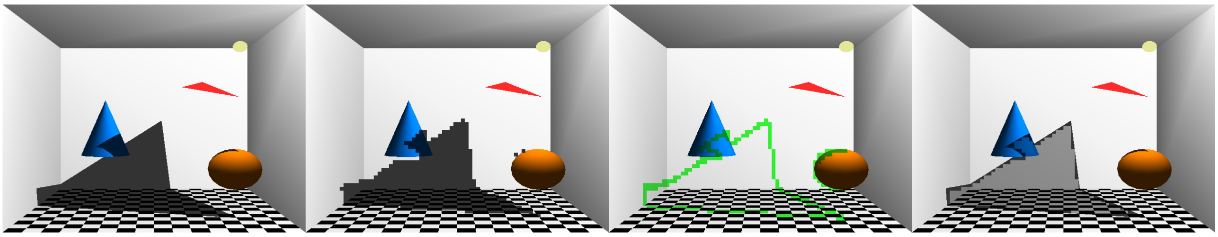

Figure 1: *From left to right: Correct shadows, low-resolution shadows (sample per* 8×8 *pixel tile), tiles that may contain a shadow boundary (green/light gray), and an image showing accurately processed boundary tiles (darker gray) and data copied from the low-resolution shadows (lighter gray). If a tile contains a shadow boundary (3rd image), the corresponding low-resolution shadow data is more or less random. This is corrected by applying per-pixel rasterization to boundary tiles.*

Abstract

The shadow volume algorithm is a popular technique for real-time shadow generation using graphics hardware. Its major disadvantage is that it is inherently fillrate-limited, as the performance is inversely proportional to the area of the projected shadow volumes. We present a new algorithm that reduces the shadow volume rasterization work significantly. With our algorithm, the amount of per-pixel processing becomes proportional to the screen-space length of the visible shadow boundary instead of the projected area. The first stage of the algorithm finds 8×8 *pixel tiles, whose 3D bounding boxes are either completely inside or outside the shadow volume. After that, the second stage performs per-pixel computations only for the potential shadow boundary tiles. We outline a two-pass implementation, and also describe an efficient single-pass hardware architecture, in which the two stages are separated using a delay stream. The only modification required in applications is a new pair of calls for marking the beginning and end of a shadow volume. In our test scenes, the algorithm processes up to* 11.5 *times fewer pixels compared to current state-of-the-art methods, while reducing the external video memory bandwidth by a factor of up to* 17.1.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shadowing I.3.1 [Computer Graphics]: Hardware Architecture—Graphics Processors

1. Introduction

Rendering of shadows is a key ingredient in computer-generated images. Shadows both increase the level of realism and provide information about spatial relationships of objects. For real-time rendering, variants of the shadow mapping algorithm [Wil78] and the shadow volume algorithm [Cro77] are among the most popular techniques. Shadow mapping is an image-based technique that often suffers from aliasing problems. The shadow volume algorithm does not have these issues, but has been criticized for its excessive use of fillrate. Shadow volumes are extruded from shadow casters to the attenuation range of a light source. Even relatively small objects, such as characters, often create shadow volumes that cover a significant portion of the screen.

The main motivation for our work is that when a shadow volume is rasterized, only a small portion of the processed pixels typically define the footprint of the shadow, and the resulting shadow often consists of large homogeneous areas. Clearly, there appears to be ample room for improvement in terms of performance.

We present a two-stage algorithm for hierarchical shadow volume rendering. The first stage finds 8×8 pixel tiles that are either completely outside or inside shadow volumes of individual objects, i.e., fully lit or fully in shadow. The results of this stage are valid after the entire shadow volume has been processed. Therefore the shadow volumes need to be either submitted twice, or the shadow volume triangles need to be delayed in the rasterization pipeline by using a delay stream [AMN03]. After the first stage is complete, the second stage needs to perform per-pixel shadow volume rasterization only for the boundary tiles.

Compared to existing acceleration techniques [NVI03], the results show that our algorithm processes 2.8–11.5 times fewer pixels, and reduces the related bandwidth usage by a factor of 2.4–17.1. We firmly believe that these results justify the proposed hardware modifications, because without our new algorithm, up to over 90% of the total rendering time was spent in shadow volume rasterization, as discussed in Section 6.

Although this paper focuses on hard shadows, the technique can be used in different contexts as well. The soft shadow volume algorithm [AAM03] can be accelerated, since it contains a hard shadow volume pass, and its culling optimization [ADMAM03] can benefit from our new method.

The rest of the paper is organized as follows. We begin by reviewing related work in Section 2, and proceed by describing our algorithm in Section 3. Section 4 outlines two implementations, and Section 5 presents the details regarding a hardware implementation, followed by our test results and discussion in Section 6. Finally, we offer conclusions and some thoughts on future work in Section 7.

2. Previous Work

In this section we focus on presenting previous work on real-time generation of hard shadows using graphics hardware, and briefly cover the use of delay streams. For a general survey on shadow generation, consult Woo et al. [WPF90]. A more recent survey covering real-time algorithms is also available [HM01].

Shadow mapping [Wil78] generates a depth buffer from the point of view of the 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 depth value, transformed into the light-space, is greater than the corresponding depth value in the shadow map. The discretization often causes aliasing artifacts such as jagged shadow boundaries and incorrect self-shadowing. Sen et al. [SCH03] improve the accuracy of shadow boundaries, and give references to other work regarding the aliasing issues.

A shadow volume consists of three parts. The *light cap* is the shadow casting object itself, the *dark cap* closes the far end of the shadow volume at infinity or at the attenuation range of the light source. The *side quads*, extruded from the silhouette edges of the shadow caster, connect the two caps into a closed volume.

In a hardware accelerated variant [Hei91] of the original shadow volume algorithm [Cro77], the scene is first rendered using ambient lighting. In the second pass, the shadow volumes are rendered into the stencil buffer so that visible pixels of the front-facing and back-facing triangles increment and decrement the stencil values. This creates a shadow mask in the stencil buffer, where values greater than zero indicate that a pixel is in shadow. In a third pass, the scene is rendered with full lighting, and the per-pixel shadow terms of the stencil buffer are used to mask out shadowed pixels. This method is commonly known as the *Z-pass* shadow volume algorithm. In contrast to shadow mapping, the shadow volume algorithm does not suffer from aliasing problems.

Problems occur when the camera is inside a shadow volume, and a more robust solution is used nowadays. The *Z-fail* algorithm has been independently described by Bilodeau & Songy and by Carmack, as discussed by Everitt and Kilgard [EK02]. The depth test is simply reversed when rendering the shadow volumes, so that only the triangles that are behind the contents of the Z-buffer affect the stencil buffer. Two solutions are presented for the cases where a shadow volume intersects the far plane of the view frustum.

Shadow Volume Optimizations Two-sided stencil test [EK02] halves the geometry processing requirements of shadow volumes. NVIDIA's UltraShadow [NVI03] allows the programmer to define the minimum and maximum depth values for a shadow volume. A pixel cannot be inside the shadow volume if the Z-buffer value of the pixel is outside the defined *depth bounds*. Therefore a significant portion of shadow volume rasterization work can potentially be avoided when the shadow receivers are outside the depth bounds. As an optimization, the bounds can be clamped to bounding volumes of local light sources or to scene geometry, e.g., walls of a room. 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.

Lengyel [Len02] uses the scissor test to limit the rasterization work to portions of the screen bounded by the influence region of the light source. McGuire et al. [MHE*03] present a framework that combines depth bounds and Lengyel's scissoring with new optimizations. They introduce an algorithm for creating the dark cap with a minimum number of triangles, and thus in terms of triangle count the light cap becomes the most complex part of a shadow volume. The Z-pass algorithm does not need the light cap, and thus it can be faster to use Z-fail only for the shadow volumes that may intersect the viewport [EK02]. Lloyd et al. [LWGM04] clamp shadow volumes so that the rasterization is limited to regions that contain shadow receivers. They also use occlusion queries to avoid casting shadows from objects that are entirely in shadow. All optimizations listed above can also be used in conjunction with our algorithm.

Hybrid Algorithms McCool [McC00] uses a shadow map and image-space edge detection to reconstruct the shadow volume of a set of objects. This shadow volume does not contain any overlapping volumes, which means that a lot of redundant rasterization work can be avoided. However, the creation of the shadow volume is rather complex and slow, so the technique is not suitable for dynamic scenes. Also, due to the discrete resolution of a depth map and the use of an edge detection algorithm, robustness issues arise.

Govindaraju et al. [GLY*03] use a combination of shadow maps and object-space shadows on a three-PC system to generate hard shadows in complex environments. Their optimizations for limiting the number of shadow casters and receivers could also be used for accelerated rendering of shadow volumes.

Chan and Durand [CD04] first determine the boundary pixels of shadow regions from 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.

Occlusion Culling Meissner et al. [MBGS01] perform two-pass visibility driven rasterization by first marking the tiles that are fully hidden and then skipping the subsequent rasterization to those tiles. Our algorithm bears some similarities to this technique.

The delay stream [AMN03] is a hardware mechanism that delays triangles in the rasterization pipeline. It can be used to generate Z-buffer information early in the pipeline, and after that the triangles are pushed into the delay stream. At a later stage, the triangles are popped, and at that time it is likely that enough Z-buffer information has been accumulated, so that many additional triangles can be culled.

3. Hierarchical Shadow Volume Algorithm

The following explanation of our new algorithm assumes that the frame buffer has been divided into 8 × 8 pixel tiles. For each tile, the z_{min} [AMS03] and z_{max} [Mor00] of the Z-buffer are maintained. The vertical and horizontal bounds of a tile, along with its z_{min} and z_{max} define a three-dimensional axis-aligned box in screen space. After rasterization of the scene, the box contains all the visible geometry, and is therefore a 3D bounding box of the tile.

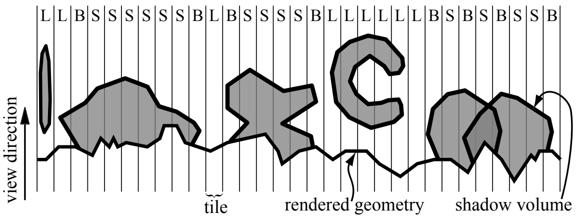
L L B S S S S S S B L B S S S S B L L L L L L B S B S B S S B
view direction
tile
rendered geometry
shadow volume

Figure 2: *This 2D illustration of our algorithm assumes the Z-fail algorithm, and thus only the parts of the shadow volumes that are behind the rendered geometry can affect the stencil buffer. The boundary tiles (B) contain intersections of rendered geometry and shadow volume triangles. A visible shadow boundary can only exist inside the B-tiles, and thus only these tiles need to be rasterized accurately. For all other tiles, it is sufficient to execute the shadow volume algorithm for a single point inside the tile, because the result is the same for the whole tile. The L-tiles are fully lit and the S-tiles are fully in shadow.*

3.1. Geometrical Explanation

Our algorithm builds on the fundamental observation that a visible shadow boundary can be inside a tile only if at least one shadow volume triangle intersects with the 3D bounding box of the tile. Two facts guarantee that the approach works. First, shadow volumes are closed by definition. Second, the triangles defining a shadow volume indicate transitions between light and shadow, i.e., if there are no intersections inside a tile, the tile cannot contain a shadow boundary. Hence, it is sufficient to execute the shadow volume algorithm for a single, arbitrarily chosen point inside a non-boundary tile in order to determine the shadow term for all pixels inside the tile. Figure 2 illustrates the idea in two dimensions.

Intuitively the shadow volume algorithm corresponds to shooting a ray through a point inside the tile, and counting the number of intersected frontfacing and backfacing triangles. A key to our method is that depending on which point is chosen, the number of intersections along the ray can change, but the net result is always the same. As an example, consider the leftmost shadow volume in Figure 2 that resides completely inside a single tile. Depending on which point (i.e., vertical ray in Figure 2) we choose inside the tile, the shadow volume may either be completely missed, or the test point will register one back-facing and one front-facing triangle, which cancel each other. The correct result is obtained in both cases: the shadow volume does not contribute to the visible shadow, and can be culled.

3.2. Algorithm Outline

Our algorithm executes in two stages. The process begins when a marker, `BSV` (Beginning of Shadow Volume), is encountered. First, the screen-space tiles are classified to be either fully lit, fully in shadow, or potentially containing a shadow boundary. This classification depends on the shadow volume as a whole, and remains incomplete until the corresponding `ESV` (End of Shadow Volume) marker is received. When the classification is complete, it can be exploited for efficient rasterization, as we will see in Section 4.

In general there are a number of shadow casters per light source, and in the following explanation each shadow casting object creates a separate shadow volume. Furthermore, the depth buffer has been initialized by rendering the scene geometry. The rest of this section explains the two stages of our algorithm in detail without being limited to any specific implementation.

3.2.1. Stage 1: Low-Resolution Shadows and Tile Classification

The tile classification is performed by using a per shadow volume (PSV) buffer, which stores for each tile an 8-bit `stencil` value and a Boolean `boundary`. Initially `boundary=false` and `stencil=`S_{clear}, where S_{clear} is the stencil buffer clear value. Without loss of generality, the rest of this paper assumes that stencil values greater than S_{clear} indicate shadow.

If a triangle intersects with the 3D bounding box of a tile, there is a potential shadow boundary inside the tile. Such boundary tiles are marked by setting their `boundary` to `true`. The intersection needs to be computed in a conservatively correct manner, i.e., at least all the actual intersections must be detected. Notice that any tile can be classified as a boundary tile without introducing visual artifacts.

If none of the shadow volume triangles intersect with the 3D bounding box of a tile, the entire tile is either fully lit or fully in shadow. Thus the classification can be carried out by executing the shadow volume algorithm for a single, arbitrarily chosen sampling point inside the tile. The shadow volume algorithm updates the `stencil` of the tile by using the defined stencil and depth modes.

After an entire shadow volume has been processed, the corresponding tile classifications are ready. If the `boundary` bit of a tile is set, per-pixel stencil rasterization is needed in order to determine the exact shadow boundary. Otherwise the rasterization can be skipped, because the entire tile is either in shadow (`stencil>`S_{clear}) or lit with respect to the current shadow volume.

The classification allows alternating between Z-pass and Z-fail algorithms, and does not pose limitations on the shadow volume topology or on the ordering of the triangles.

3.2.2. Stage 2: Stencil Buffer Updates

When the `ESV` marker has emerged from Stage 1, the shadow volume triangles are rasterized according to the tile classifications in the corresponding PSV-buffer. The following three cases are possible:

1. Tile is fully inside the shadow volume (S-tiles in Figure 2). All pixels in the tile are set to be in shadow.
2. Tile is fully outside the shadow volume (L-tiles). No updates needed.
3. The shadow volume intersects the 3D bounding box of the tile (B-tiles). Per-pixel stencil operations are performed, as requested by the application.

4. Implementing the Algorithm

Our algorithm can be implemented in various ways. We first introduce a hierarchical variant of the stencil buffer in order to make the stencil buffer updates in Stage 2 more efficient. We then briefly outline a two-pass implementation, and finally explain a more efficient single-pass implementation using a delay stream. Both implementations use the hierarchical stencil buffer.

4.1. Hierarchical Stencil Buffer

We propose a two-level version of the stencil buffer. For each 8×8 pixel tile, the minimum and maximum stencil buffer values, S_{min} and S_{max}, are maintained in a low-resolution stencil buffer. Furthermore, we propose that the stencil test is carried out as follows. If the result of the stencil test can be determined from $[S_{min}, S_{max}]$, the per-pixel stencil buffer must not be accessed. For example, if $S_{min} = S_{max} = 1$, the per-pixel values are not needed for any of the stencil comparison modes. This opens the possibility of not updating or accessing the per-pixel stencil values unless absolutely necessary.

4.1.1. Optimized Stage 2: Hierarchical Updates

The stencil buffer updates in Section 3.2.2 can be optimized by using the hierarchical stencil buffer:

1. Tile is fully inside the shadow volume. The corresponding entry in the low-resolution stencil buffer is updated. Per-pixel operations to the tile are skipped.
2. Tile is fully outside the shadow volume. No updates needed.
3. The shadow volume intersects the 3D bounding box of the tile. Per-pixel stencil operations are performed, as requested by the application. S_{min} and S_{max} are updated according to the per-pixel stencil buffer rasterization.

Per-pixel rasterization and the related video memory bandwidth are limited to the boundary tiles. In general, this results in significant reduction of fillrate requirements, as illustrated in Figure 1. It is worth pointing out that the hierarchical stencil buffer alone would provide only small performance improvements; the two-stage shadow volume algorithm is needed for efficiently exploiting the hierarchical updates.

4.1.2. Final Lighting Pass

After all shadow casters have been processed, the full lighting contribution of the light source is accumulated into the frame buffer so that the shadow terms are read from the hierarchical stencil buffer. Per-pixel stencil values are fetched only for the boundary tiles of the combined shadow area of all shadow volumes. This provides an additional reduction to the memory bandwidth requirements.

4.2. Two-Pass Implementation

Perhaps the simplest implementation of our algorithm requires the application or the device driver to submit each shadow volume twice. The two passes then correspond to Stage 1 and optimized Stage 2. However, this alternative would need a second geometry pass and further modifications to the applications in order to facilitate efficient pipelined execution.

4.3. Single-Pass Implementation Using a Delay Stream

This single-pass implementation uses the optimized Stage 2, whereas Stage 1 remains identical to Section 3.2.1. Between the two stages, the triangles defining a shadow volume are temporarily stored into a delay stream, which is a ring buffer in external video memory [AMN03]. Unlike delayed occlusion culling, our algorithm uses the delay stream only for shadow volumes. Note that the same stream could be used by both algorithms. The `ESV` marker is passed directly from Stage 1 to Stage 2 without going through the delay stream. The rest of this section explains our architecture in detail, and a general overview is given in Figure 3.

4.3.1. Delay Stream

The delay stream should be large enough to hold all the triangles of a single shadow volume in order to delay the stencil buffer rasterization up to the point when the classification is finished. Typically the geometry defining a shadow volume consumes only a small amount of memory for two reasons. First, the shadow triangles usually contain only positional information, i.e., no colors or texture coordinates per vertex, and this leads to good compression of the delay stream. Second, for the majority of shadow volumes, only the side quads are needed [MHE*03], and for a mesh of n triangles, the number of silhouette edges can be as low as $O(\sqrt{n})$ [SHSG01].

In certain pathological cases, e.g., when all objects of a scene are presented as a single triangle soup, the allocated delay stream may not be able to store even one entire shadow volume. If this happens, the per-pixel stencil buffer rasterization has to start before the tile classification is complete, and the performance degrades gracefully towards the traditional shadow volume algorithm. Visual artifacts are avoided by treating all tiles as boundary tiles until the classification finishes. After that, the remaining per-pixel rasterization is skipped for tiles that were classified to be fully in shadow. The upper bound of per-pixel stencil rasterization work is equal to the traditional shadow volume algorithm.

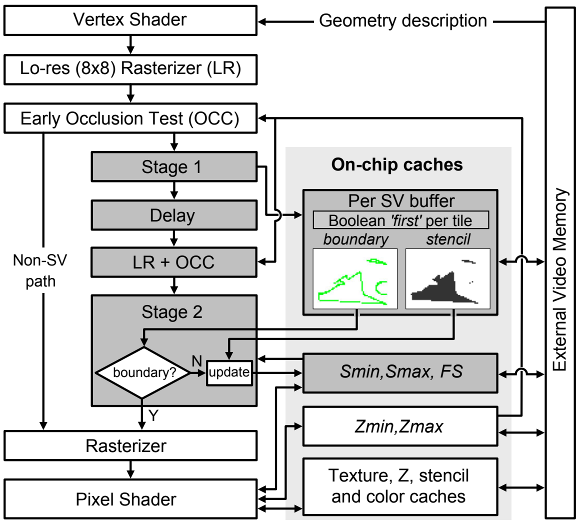

Figure 3: *The positioning and connections of our algorithm inside a modern programmable graphics pipeline. The new units and on-chip caches are marked with gray.*

4.3.2. Further Optimizations

Usually there are multiple objects casting shadows from a light source. When the contribution of a shadow volume is added to the stencil buffer, the overall area covered by the shadow grows monotonically. For example, a tile that has been previously classified as being fully in shadow cannot be downgraded to a boundary tile. This is especially useful for scenes that have a high depth complexity, such as a city scene.

We perform this check by using a full shadow (FS) buffer, which stores one bit per tile. This optimization needs a separate FS-buffer because the contents of the stencil buffer are modified during the rasterization of a shadow volume, and are thus in a meaningful state only between two shadow volumes. Therefore we must update the FS-buffer exactly once per shadow volume. For this purpose, we add a Boolean `first` to each tile in the PSV-buffer, and initialize them to `true` immediately after receiving the BSV in Stage 1. If the `first` bit of a tile is set, the FS bit is set to `true` if $S_{min} > S_{clear}$ and to `false` otherwise. The `first` is then cleared in order to avoid examining the stencil buffer during the processing of a shadow volume. Subsequent rasterization to a tile is skipped if the FS bit is set to `true`.

Appendix A contains pseudo code for Stages 1 and 2. The use of FS-buffer makes the performance of our algorithm slightly dependent on the processing order of shadow volumes. However, we have not optimized our test applications to take this into account.

5. Hardware Implementation of Single-Pass Algorithm

The pipelined processing of shadow volumes deserves special attention. The tile classifications are made individually for each shadow volume, and thus Stages 1 and 2 are generally processing different shadow volumes. Therefore multiple per shadow volume (PSV) buffers are needed. We handle this by allocating in the driver a small number of buffers, according to the screen resolution. The buffers are stored in external video memory and accessed through an on-chip cache.

A PSV-buffer is locked for a particular shadow volume in Stage 1 when `BSV` is encountered. If no buffers are available, the unit stalls. The buffer is released in Stage 2 after `ESV` is received. Only a part of each buffer is generally accessed by a shadow volume, and thus a clear bit [Mor00] per 32×32 pixels provides a fast way of clearing the necessary parts of the PSV-buffers in Stage 1.

5.1. New Hardware Resources

The new hardware resources consist of computational logic in Stages 1 and 2, a delay stream, additional instances of the low-resolution rasterizer and early occlusion test units, and on-chip storage. The amount of new logic is fairly low as can be seen from the pseudo code in Appendix A. The only nontrivial operations are finding the intersection of a shadow volume triangle with the 3D bounding box of a tile, and determining if the triangle is behind the bounding box. However, both of these questions need to be answered already in the preceding “Early Occlusion Test” unit (Figure 3) in order to perform z_{min} and z_{max} culling. Such a unit already exists in most modern graphics chips, and thus the implementation details are omitted from this paper.

The delay stream is a ring buffer in external memory, and the new hardware units are limited to a simple history-based compression [AMN03], which eliminates duplicate vertices and stores only the active attributes of vertices, i.e., the position. The render state changes are also stored, although there are usually very few of them during the rasterization of shadow volumes.

On-chip storage consists of the PSV-buffer cache, FS-buffer and $S_{min,max}$ buffers. Since the tile classification (Section 3.2.1) allows arbitrary positioning of the sample point within the tile, one can let four tiles, placed in a 2×2 configuration, share the same sample point, and thus also the same `stencil` value in the PSV-buffer. In this approach, the sample point is located at the shared corner of the four tiles. The storage for the PSV-buffer is then $2 + 8/4 = 4$ bits per tile. A 2kB cache was used in our tests. The FS-buffer uses one bit per tile, which corresponds to 2.5kB in 1280×1024 resolution.

$S_{min,max}$ can be useful for generic computations using the stencil buffer. However, if they are only used for the purposes of shadow volume rendering, the stencil value can mean only one of two things, namely the pixel being lit or in shadow. With such encoding "1" indicates shadow and "0" means lit, and partial shadow is indicated by $S_{min} = 0$ and $S_{max} = 1$. While this 1-bit representation is adequate for shadows, it imposes two limitations. First, the function deciding what stencil values correspond to shadow should not be changed between constructing and using the stencil buffer. This does not appear to be a serious limitation in practice. Second, the $S_{min,max}$ buffers contain meaningful data for shadow computations, but should be disabled for other computations that perform stencil tests. One way of implementing this is to prefix the stencil test with a new shadow test, which uses $S_{min,max}$. The traditional stencil test is executed only if the shadow test cannot determine the results, e.g., a tile is partially in shadow or the shadow test is disabled. As a result, the size of $S_{min,max}$ can be made as small as 5kB in 1280×1024 resolution. For comparison, the existing $z_{min,max}$ buffers consume 16 times more memory, assuming 16-bit depth values.

6. Results and Discussion

The performance of our single-pass algorithm was benchmarked using walkthroughs in four scenes (Table 1). The complexity of the scenes is rather accurately illustrated by the average number of rendered shadow volume triangles per frame. Only eight were rendered in the Simple scene, and at the other end of the scale, 572k in the Powerplant. The two intermediate scenes, Knights (18k) and Amphitheater (6.6k), include various animated characters. All measurements were done using a single light source, and for Simple scene, Knights, and Powerplant the light source was moving.

Three different variants of the shadow volume algorithm were implemented inside MESA (`www.mesa3d.org`), which is a software renderer with an OpenGL-like interface. The first variant is a brute force approach, the second is based on depth bounds [NVI03], and the third is our hierarchical algorithm. All three utilize z_{min}, z_{max}, and lossless compression for the Z-buffer [Mor00]. The compression is assumed to reduce Z-buffer bandwidth usage by 50%. The depth bounds were computed separately for each shadow casting object. The bounds were initialized with the bounding box of the shadow caster, and then extended to include the intersection of the view frustum and the shadow volume of the bounding box. In Amphitheater shadow volumes are extruded from the local light source to infinity instead of the attenuation range, and thus the tightness of depth bounds could have been further improved by clamping the bounds to the bounding volume of the light source.

The images were rendered at 1280×1024 resolution, and all accesses to $z_{min,max}$, $S_{min,max}$ and FS-buffer were on-chip.

		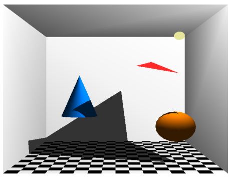						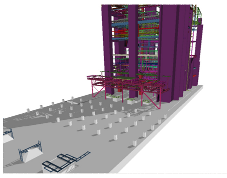				
		Simple scene			**Knights**			**Powerplant (Section 16)**			**Amphitheater**	
Algorithm	*BF*	*DB*	*H*	*BF*	*DB*	*H*	*BF*	*DB*	*H*	*BF*	*DB*	*H*
Stencil BW in shadow pass	0.54	0.48	0.03	48.96	18.65	5.90	4329	1798	123	9.59	9.05	1.66
Z-buffer BW in shadow pass	0.05	0.05	0.05	3.00	3.00	2.93	160.3	160.3	115.1	0.97	0.97	0.84
Stencil BW in lighting pass	1.24	1.24	0.02	0.82	0.82	0.11	1.16	1.16	0.01	0.81	0.81	0.07
PSV BW from 2kB cache	-	-	0.005	-	-	0.035	-	-	29.2	-	-	0.19
Delay stream BW	-	-	0.001	-	-	0.43	-	-	15.1	-	-	0.16
Total bandwidth	1.82	1.77	0.10	52.78	22.46	9.40	4490	1959	283	11.37	10.82	2.91
#Pixels in shadow pass (M)	0.28	0.25	0.02	25.67	9.78	3.48	2269	943	99	5.03	4.74	0.94
Ratio in bandwidth		1.03:1	**17.6:1**		2.35:1	**5.61:1**		2.3:1	**16.0:1**		1.05:1	**3.91:1**
Ratio in #processed pixels		1.1:1	**12.7:1**		2.63:1	**7.38:1**		2.4:1	**22.9:1**		1.06:1	**5.35:1**

Table 1: *Average per-frame statistics for brute force (*BF*), depth bounds (*DB*) and our hierarchical algorithm (*H*) in* 1280 × 1024 *resolution with 32-bit depth buffer and 8-bit stencil buffer. All bandwidth (BW) measurements are shown in megabytes (MB). Ratios are relative to the brute force algorithm.*

The measured total bandwidth in Table 1 refers to the cost of constructing the shadow mask into the stencil buffer, and then fetching the shadow terms in the subsequent illumination pass. Z-fail was used in the Simple Scene, Knights and Powerplant due to its robustness. Adaptive switching between Z-pass and Z-fail could have reduced the number of shadow volume triangles, but we did not implement it because the fillrate measurements are independent of the triangle count, and there is no proof that either Z-pass or Z-fail has better bandwidth usage. However, the Amphitheater is publicly available as a part of an application that readily implements the switching. The size of the delay stream was set to 1MB, and Stage 2 started to process a shadow volume as soon as Stage 1 finished classifying it.

Our algorithm reduced the total bandwidth by a factor of 3.9–17.6 compared to the brute force variant, and by a factor of 2.4–17.1 compared to the depth bounds. In the shadow volume pass, our algorithm processed 5.3–22.9 times fewer pixels than brute force, and 2.8–11.5 times fewer than depth bounds.

We argue that it is the number of processed pixels that ultimately limits the performance. The bandwidth to external memory is a smaller problem, because less than three bytes were transferred per pixel for Knights, Powerplant, and Amphitheater. Also, it might be possible to reduce the bandwidth further by employing compression techniques to the stencil buffer. For Powerplant, our largest scene, as many as 2.2 billion pixels were processed every frame by the brute force algorithm. The depth bounds reduced the number of pixels to 943 million, and our algorithm provided a further reduction to 99 million. In an additional test, the brute force variant ran at less than one frame per second on a 3GHz Pentium 4 with an ATI Radeon 9800XT graphics card. When the shadow volume rasterization was omitted, the frame rate jumped to over 50. Thus the overall performance of the application was clearly bounded by the processing of shadow volumes. Assuming a 20-fold speedup in the shadow volume rasterization, a hardware implementation of our algorithm should boost the overall frame rate from 1 to around 15.

It may seem surprising that the Z-buffer bandwidth is also reduced by our algorithm. This results from skipping the processing of tiles that are already in shadow due to previous shadow volumes. In the Powerplant scene, a further reduction of bandwidth usage of 15MB (5%) was obtained when the cache for the PSV-buffers was doubled from 2kB to 4kB.

It is worth emphasizing that the performance ratio between our algorithm and depth bounds gets even larger with higher resolutions, because the rasterization work done by our algorithm is primarily affected by the length of the shadow boundary. In contrast, with depth bounds at least the whole shadow area needs to be rasterized accurately. When resolution increases, the length of the shadow boundary typically grows much slower than the covered area. At 2048 × 1536 resolution, our algorithm performed an additional 20–30% better than depth bounds.

Discussion The performance of our algorithm could be further improved by combining it with depth bounds testing, which offers a complementary and inexpensive way to discard tiles prior to Stage 1. In applications that always extrude the shadow volumes to infinity instead of the attenuation range the performance of depth bounds testing could be further improved by limiting the bounds with bounding volumes of light sources and with scene geometry. Under such circumstances, accurately computed depth bounds would allow skipping boundary tiles that are not affected by the light source, and thus also optimize the per-pixel operations in Stage 2.

The performance improvements are a result of hierarchical processing of non-boundary tiles. The worst case input for our algorithm would be a very complex object, such as a tree that could at least theoretically cast shadows that have a boundary inside every tile. In such cases, hierarchical processing cannot be exploited due to the complete lack of coherence in shadowed regions.

Highly tessellated shadow casters, such as the characters in the Knights scene, result in long and narrow shadow volume triangles that may not cover even a single tile entirely. That does not reduce the performance of our algorithm because we consider the shadow volume as a whole, and the size of individual triangles is not important.

7. Conclusions and Future Work

We have presented a new algorithm for accelerated rendering of shadow volumes. For a single-pass implementation using a delay stream, the benchmarks indicate that our shadow volume culling algorithm processes 2.8–11.5 times fewer pixels, and performs between 2.4 and 17.1 times fewer accesses to external memory than previous methods. Thus, we argue that the main critique against shadow volumes is eliminated by our method.

In future work, we would like to make an accurate comparison of computational and bandwidth requirements of our hierarchical shadow volume algorithm and the shadow mapping algorithm. A performance evaluation with different tile sizes is also an interesting line of future work. Furthermore, it might be fruitful from a performance viewpoint to study in which order the shadow volumes should be submitted.

Acknowledgments

Simple scene courtesy of Tom McReynolds, SGI. Powerplant courtesy of the Walkthrough Group at the University of North Carolina at Chapel Hill. Amphitheater courtesy of Sam Howell, used with permission of Morgan McGuire. The Tick model used in Amphitheater was created by Carl Schell (`carl@cschell.com`). The Perelith knight model courtesy of James Green (`www.perilith.com`, `james@perilith.com`). Thanks to Jonas Svensson and Ulf Borgenstam for providing the Knights demo, Karl Schultz for help with MESA, and Ville Miettinen, Jacob Ström, Eric Haines, Ulf Assarsson, Lauri Savioja and the 3DR group at Helsinki University of Technology for helpful comments. Timo Aila was partially supported by the National Technology Agency of Finland, Bitboys, Hybrid Graphics, Nokia and Remedy Entertainment.

Appendix A: Pseudo Code

Stage 1 (Z-fail)

```
for each tile t overlapped by the triangle:

    if (triangle intersects 3D bounding box of t)
        t.PSV.boundary = true;

    else if (triangle behind 3D bounding box of t &&
          t's sample point inside triangle)
        if (triangle front-facing)
            t.PSV.stencil = t.PSV.stencil - 1;
        else
            t.PSV.stencil = t.PSV.stencil + 1;
```

Stage 2 (Optimized Architecture)

```
for each tile t overlapped by the triangle:

    if (t.PSV.first == true)
        t.PSV.first = false;
        if (t.Smin > Sclear)    // in shadow before this SV
            t.FS = true;

    if (t.FS == true)           // already in shadow
        continue;

    if (t.PSV.boundary == true)
        rasterizeToStencilBuffer(); // updates Smin/Smax too

    else if (t.PSV.stencil > Sclear)  // full shadow
        t.Smin = t.Smax = t.PSV.stencil;
        t.FS = true;
```

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware. *ACM Transactions on Graphics, 22*, 3 (2003), 511–520. 2

[ADMAM03] ASSARSSON U., DOUGHERTY M., MOUNIER M., AKENINE-MÖLLER T.: An Optimized Soft Shadow Volume Algorithm with Real-Time Performance. In *Graphics Hardware* (July 2003), ACM SIGGRAPH/Eurographics, pp. 33–40. 2

[AMN03] AILA T., MIETTINEN V., NORDLUND P.: Delay Streams for Graphics Hardware. *ACM Transactions on Graphics, 22*, 3 (2003), 792–800. 2, 3, 5, 6

[AMS03] AKENINE-MÖLLER T., STRÖM J.: Graphics for the Masses: A Hardware Rasterization Architecture for Mobile Phones. *ACM Transactions on Graphics, 22*, 3 (2003), 801–808. 3

[CD04] CHAN E., DURAND F.: An Efficient Hybrid Shadow Rendering Algorithm. In *Proceedings of the*

Eurographics Symposium on Rendering (2004), Eurographics Association. 3

[Cro77] CROW F.: Shadow Algorithms for Computer Graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 77)* (July 1977), ACM, pp. 242–248. 1, 2

[EK02] EVERITT C., KILGARD M.: Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. *http://developer.nvidia.com/* (2002). 2, 3

[GLY*03] GOVINDARAJU N. K., LLOYD B., YOON S.-E., SUD A., MANOCHA D.: Interactive Shadow Generation in Complex Environments. *ACM Transactions on Graphics, 22*, 3 (2003), 501–510. 3

[Hei91] HEIDMANN T.: Real Shadows, Real Time. *Iris Universe*, 18 (November 1991), 28–31. 2

[HM01] HAINES E., MÖLLER T.: Real-Time Shadows. In *Proceeding of Game Developers Conference* (March 2001), pp. 335–352. 2

[Len02] LENGYEL E.: The Mechanics of Robust Stencil Shadows. *http://www.gamasutra.com/* (October 2002). 2

[LWGM04] LLOYD B., WENDT J., GOVINDARAJU N., MANOCHA D.: CC Shadow Volumes. In *Proceedings of the Eurographics Symposium on Rendering* (2004), Eurographics Association. 3

[MBGS01] MEISSNER M., BARTZ D., GÜNTHER R., STRASSER W.: Visibility Driven Rasterization. *Computer Graphics Forum, 20*, 4 (2001), 283–294. 3

[McC00] MCCOOL M. D.: Shadow Volume Reconstruction from Depth Maps. *ACM Transactions on Graphics, 19*, 1 (2000), 1–26. 3

[MHE*03] MCGUIRE M., HUGUES J. F., EGAN K. T., KILGARD M., EVERITT C.: *Fast, Practical and Robust Shadows*. Tech. Rep. CS03-19, Brown University, October 2003. 2, 5

[Mor00] MOREIN S.: ATI Radeon HyperZ Technology. In *Workshop on Graphics Hardware, Hot3D Proceedings* (August 2000), ACM SIGGRAPH/Eurographics. 3, 6

[NVI03] NVIDIA: *NVIDIA GeForceFX 5900 GPUs: UltraShadow Technology*. Tech. rep., http://www.nvidia.com, 2003. 2, 6

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow Silhouette Maps. *ACM Transactions on Graphics, 22*, 3 (2003), 521–526. 2

[SHSG01] SANDER P. V., HOPPE H., SNYDER J., GORTLER S. J.: Discontinuity Edge Overdraw. In *Symposium on Interactive 3D Graphics* (March 2001), ACM SIGGRAPH, pp. 167–174. 5

[Wil78] WILLIAMS L.: Casting Curved Shadows on Curved Surfaces. In *Computer Graphics (Proceedings of ACM SIGGRAPH 78)* (August 1978), ACM, pp. 270–274. 1, 2

[WPF90] WOO A., POULIN P., FOURNIER A.: A Survey of Shadow Algorithms. *IEEE Computer Graphics and Applications 10*, 6 (November 1990), 13–32. 2