Shadow Silhouette Maps

Pradeep Sen Mike Cammarano Pat Hanrahan[*]
Stanford University

Abstract

The most popular techniques for interactive rendering of hard shadows are *shadow maps* and *shadow volumes*. Shadow maps work well in regions that are completely in light or in shadow but result in objectionable artifacts near shadow boundaries. In contrast, shadow volumes generate precise shadow boundaries but require high fill rates. In this paper, we propose the method of *silhouette maps*, in which a shadow depth map is augmented by storing the location of points on the geometric silhouette. This allows the shader to construct a piecewise linear approximation to the true shadow silhouette, improving the visual quality over the piecewise constant approximation of conventional shadow maps. We demonstrate an implementation of our approach running on programmable graphics hardware in real-time.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shading, Shadowing

Keywords: Rendering, Shadow Algorithms, Graphics Hardware

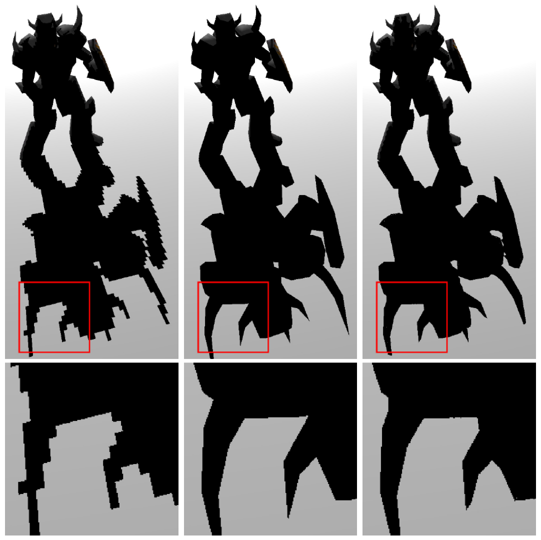
Figure 1: (Left) Standard shadow map. (Center) Shadow volumes. (Right) Silhouette map, at same resolution as shadow map in (Left).

1 Introduction

Although shadow generation has been studied for over two decades, developers seeking a real-time shadowing solution still face compromises in their choice of algorithm. Currently, the two main techniques for real-time shadows are shadow maps and shadow volumes. In this paper we propose a hybrid representation that combines the strengths of both methods.

For background, interested readers may consult Woo et al. [1990] for a thorough overview of early shadow generation algorithms. Here we shall focus only on work directly relevant to our approach. One of the earliest methods for shadow determination was shadow volumes, developed by Crow [1977] and implemented in hardware for the UNC Pixel-Planes4 project [Fuchs et al. 1985]. Shadow volumes are polyhedral regions in the shadow of an object. They are formed by extruding the silhouette edges away from the position of the light source. This auxiliary shadow volume geometry is then rendered along with the original scene geometry. By counting the number of front-facing and back-facing shadow volume faces in front of each rendered point, it is possible to determine whether or not it is in shadow. Shadow volume methods have the advantage of generating precise shadow boundaries. Unfortunately, implementations are not always robust, failing in special cases of near and far plane clipping of the shadow volumes. Recent work appears to have resolved these stability problems [Everitt and Kilgard 2002]. Nevertheless, the bandwidth requirements of rendering shadow volumes remain substantial. Rasterizing the overlapping faces of the shadow volumes can lead to a great deal of overdraw, making shadow volume methods heavy consumers of fill rate.

An alternative method, shadow maps, was introduced by Williams [1978]. Shadow maps require an extra rendering pass to generate a depth image from the point of view of the light. Subsequently, shadowing can be determined for any point in space by performing a constant-time lookup in this shadow map. To shade a point on a surface, it is projected into light space and tested against the corresponding shadow map depth sample. The point is in shadow if it is farther from the light than the stored depth value. Unfortunately, like any discrete image-based technique, shadow maps are prone to aliasing due to the limited resolution of the representation.

A number of approaches have sought to minimize the impact of aliasing near the shadow edges of shadow maps. Percentage closer filtering [Reeves et al. 1987] antialiases edges by taking into account the results of multiple depth comparisons. Adaptive shadow maps [Fernando et al. 2001] and perspective shadow maps [Stamminger and Drettakis 2002] attempt to minimize visible aliasing by better matching the resolution represented in the shadow map to that of the final image. Of these two, only the latter seems practical to implement on current graphics hardware. This technique involves a transformation of the scene geometry to increase the relative size of objects and match the perspective of the final image. By increasing the number of samples for objects near the eye, better shadows are generated. However, perspective shadow maps do not work in all situations. Consider a scene in which the viewer and the light are opposite each other. An object which is near the light and

[*]e-mail: {psen | mcammara | hanrahan}@graphics.stanford.edu

far from the viewer could appear small but still cast a large shadow. Since the perspective shadow map would dedicate a small number of samples to the distant object, the shadow would contain artifacts.

We propose a method that is a hybrid between shadow map and shadow volume techniques. As in the shadow volume methods, we first determine silhouettes from the scene geometry. Instead of extruding the silhouettes into quadrilaterals as is done in shadow volumes, we draw the silhouette lines into a texture in light space. This texture, a *silhouette map*, stores the coordinates of points on the silhouettes. Using a standard depth map and our silhouette map, we can then apply a technique similar to dual-contouring to approximate the shadowed regions.

2 Silhouette Map Algorithm

Our algorithm is based on the observation that shadow maps perform well in most areas of the image but suffer from objectionable aliasing in the regions near shadow boundaries. We augment standard shadow maps by storing additional information about the position of the shadow edge in a silhouette map, which is then used to improve the quality of the shadow near the boundaries.

Like the standard shadow map implementation, our algorithm has two stages: In the first stage, the scene is rendered from the point of view of the light to generate a depth map and a silhouette map. In the second stage, the scene is rendered from the viewer's perspective and shadow determination is made. Since the depth map is the same as in traditional shadow map implementations, we will only discuss the generation of the silhouette map and its use in shadow determination.

The purpose of the silhouette map is to provide information on the location of the shadow boundary so that it can be faithfully reconstructed. In deciding on a representation for storing this information, we are guided by two main criteria. First, the representation must guarantee a continuous shadow boundary. Second, the information has to be easy to store in a texture.

The shadow boundary can be approximated by a series of line segments. We use an approach inspired by the dual contouring algorithm (see, e.g., Ju et al. [2002]) and store points on the silhouette in a texture. We then reconstruct the piecewise linear contour during the second stage of the algorithm by connecting adjacent points. Thus, a silhouette map is a texture whose texels represent the (x,y) coordinates of points that lie on the silhouettes of objects, as seen from the light. Texels which do not have any silhouettes going through them are marked as empty. There can only be one silhouette point in each texel; if a texel has more than one silhouette passing through it, only the last point written to it will be stored.

To generate the silhouette map, the silhouette edges of the geometry must first be identified using any of the techniques for shadow volumes. These edges are then rasterized as line segments; the rasterization algorithm must generate a fragment for every texel intersected by the line segment. The resulting set of fragments are 4-connected, a condition needed to connect adjacent points in the silhouette map. The fragment program must pick a point to store in the silhouette map which is on the line segment and inside the current texel. Since it is important that only visible silhouettes be stored in the map, we compare the silhouette depth against the depth map to discard silhouettes that are hidden from the view of the light. In our algorithm, the depth map is shifted from the silhouette map by half a pixel in each direction so that the depth map represents the depth at the corners of each texel in the silhouette map. After the first stage is complete, the depth and silhouette maps can be used to reconstruct the shadows from the viewer's perspective.

To determine if a point in the scene is in shadow, we first project it into light space. We compare the current fragment depth with the four closest shadow depth samples. If they all agree that the object is either lit or shadowed, this region does not have a silhouette boundary going through it and we shade the fragment accordingly.

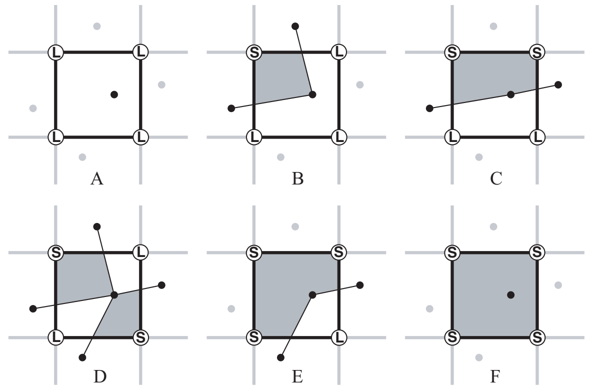

Figure 2: All possible combinations of depth test results and shadowing configurations. The depth test result at each corner is denoted by an L or an S indicating lit and shadowed, respectively. (A) All corners lit, (B) one corner shadowed, (C,D) two corners shadowed, (E) three corners shadowed, (F) all corners shadowed.

This is similar to a standard shadow map depth test. If the depth tests disagree, however, a shadow boundary must pass through this texel. In this case, we use the silhouette map to approximate the correct shadow edge.

Shadow edges, by definition, separate regions of light and shadow. Therefore, they also separate regions where the depth tests pass and fail. Given our square texel with depth tests at every corner, we observe that the shadow edge must cross the sides of the square with differing test results. By connecting the current silhouette point with the points of the appropriate neighbors, we can generate line segments that approximate the shadow edge at the current texel. This approximate silhouette contour divides the texel into shadowed and lit regions. In order to correctly shade the fragment, we note that the shading of each region matches that of its respective corner. Hence the result of the appropriate corner depth test will tell us how to shade the fragments on that side of the silhouette boundary. Figure 2 shows all the different combinations of depth test results and shadowing configurations.

Since the depth is sampled at discrete intervals and with finite precision, there might be disagreement in the depth tests which would lead the shader to believe there is a silhouette edge in a region where there is none. We resolve this problem by always placing a default point in the center of every texel. If no valid point can be found in a texel, the algorithm assumes the point is in the middle. In the worst case, this can be viewed as defaulting back to a standard shadow map, since a standard shadow map behaves like a silhouette map with every texel having a point in the center. We handle this case to ensure that our algorithm is robust, although using a small depth bias as in conventional shadow maps generally prevents this issue from arising.

Overall, our technique can be thought of as storing a piece-wise linear representation in the silhouette map as opposed to the piecewise constant representation of a shadow map. This technique is also well suited for hardware. Despite the lack of native hardware support for our method, we are able to get good performance for our interactive scenes.

3 Implementation

We implemented our algorithm on the ATI Radeon 9700 Pro using ARB_vertex_program and ARB_fragment_program shaders. Three rendering passes are needed: one to create the conventional shadow depth map, one to create the silhouette map, and one to render the scene and evaluate shadowing for each pixel in the rendered scene.

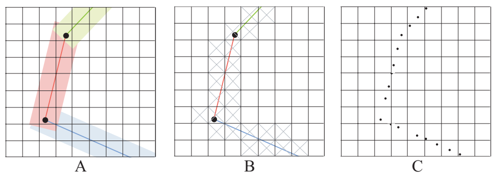

Figure 3: Generation of silhouette map using quadrangles. (A) line segments with quadrangles shown in lighter color, (B) intersection test between line segments and rasterized fragments, (C) generated points of silhouette map.

The first pass, rendering a depth map of the scene from the point of view of the light, proceeds much as in conventional shadow maps. However, we use a simple fragment program at this stage to copy the depth output into a color output so we can read it back in subsequent passes. This is needed because our hardware does not allow fragment programs to bind a depth texture directly.

3.1 Generating the Silhouette Map

This pass rasterizes the silhouette edges into the silhouette map texture as shown in Figure 3. The rasterization rules used when drawing wide lines vary between implementations and often do not guarantee that every pixel intersected by a line segment will be rasterized. To avoid problems due to these inconsistencies, we draw the region around the line segment as a quadrilateral. The size of the quadrilateral is chosen to guarantee that a fragment is generated for every texel intersected by the line segment. To draw the quad, we simply offset the vertices on either side of the line by a small amount in screen space perpendicular to the direction of the line. We also make the quads slightly longer than the line segment to guarantee that the end points of the line are rasterized as well.

Since the quadrilateral is chosen conservatively, fragments that do not overlap the line segment may also be generated. The fragment program that selects the point on the line segment must make sure the point is actually inside the texel. To perform this test, we pass the two endpoints of the line segment as vertex parameters to the fragment program. If one of the vertices is inside the texel, we know trivially that the line segment intersects the square and we store the vertex into the silhouette map — see Figure 4(A). If neither vertex is inside the texel square, we test to see if the line intersects the texel by intersecting the line segment with the two diagonals of the texel. If the line intersects the texel, at least one diagonal will be intersected inside the square. If only one of the two intersections is inside the square we store it in the silhouette map, and if both are valid we store the midpoint between them, as shown in Figures 4(B) and 4(C), respectively. If the diagonals are both intersected outside the square, the line does not intersect our texel and so we issue a fragment kill, preventing the fragment from being written into the silhouette map. This case is shown in Figure 4(D). This simple technique selects a point on a line segment clipped to the texel boundary, and can be implemented in an ARB_fragment_program.

To ensure enough precision, we represent the coordinates of the silhouette points in the local coordinate frame of the texel, with the origin at the bottom-left corner of the texel. In the fragment program, we translate the line's vertices into this reference frame before performing our intersection calculations. In addition, we also want to ensure that only visible silhouettes edges are rasterized into the silhouette map. To do this properly, the depth of the fragment is compared to that of the four corner samples. If the fragment is farther from the light than all four corner samples, we prevent it from writing into the silhouette map by issuing a fragment kill.

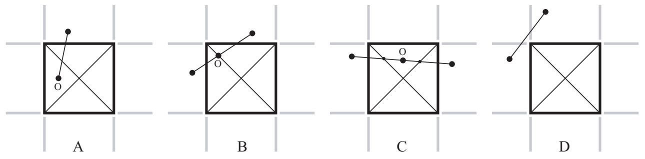

Figure 4: Intersection of line segment with texel. We label the point that will be stored in the silhouette map as "O." (A) If either vertex is inside the texel, store it. (B) Segment intersects diagonals at only one point. (C) Segment intersects diagonals in two places so store the midpoint. (D) No intersection at all, so do not store anything.

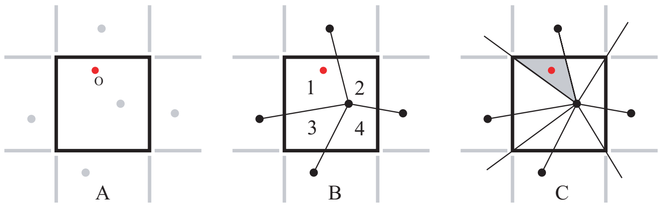

Figure 5: (A) We want to shade point "O" using silhouette map points shown in gray. (B) We must first determine which "skewed" quadrant (1–4) the point is in. (C) We test against the eight pie-shaped regions. Here we determine the point is in quadrant 1, so we shade it based on the depth sample at the top-left corner.

Finally, our implementation must handle the case where the silhouette line passes through the corner of a texel. In these situations, problems occur if we store the point in only one of the texels adjacent to the corner. To avoid artifacts and ensure the 4-connectedness of our representation, we store lines that pass near corners (within limits of precision) in all four neighboring texels. To do this, the clipping square is enlarged slightly to allow intersections to be valid just outside the square region. When the final point is computed, the fragment program clamps it to the texel square to ensure that all the points stored in a texel are always inside the texel.

The algorithm was implemented as an ARB_fragment_program of 60 instructions, four of which are texture fetches to get the required depth samples.

3.2 Shadow rendering

In this pass we decide if a point is in shadow using the depth and silhouette maps. The depth tests are first performed and if they all agree we shade the point based on their result. If they disagree, the silhouette map is used to approximate the shadow edges passing through this texel and shade the point accordingly.

Line segments between the current point and these four neighbors divide the texel into four "skewed" quadrants, as shown in Figure 5. Each quadrant is shaded in the same manner as its corner point, so we must determine which quadrant the current fragment is in to properly shade it. We divide the texel into eight pie-shaped regions (two for every quadrant) and perform simple line tests to place the point in the correct quadrant. The fragment is then shaded appropriately based on the result of the quadrant's depth test.

This step of the algorithm is implemented in 46 instructions of an ARB_fragment_program. The remaining instructions are used to shade the fragment. There are a total of nine texture fetches in the shadow determination portion, four to gather the depth samples and five to get the silhouette point and its neighbors.

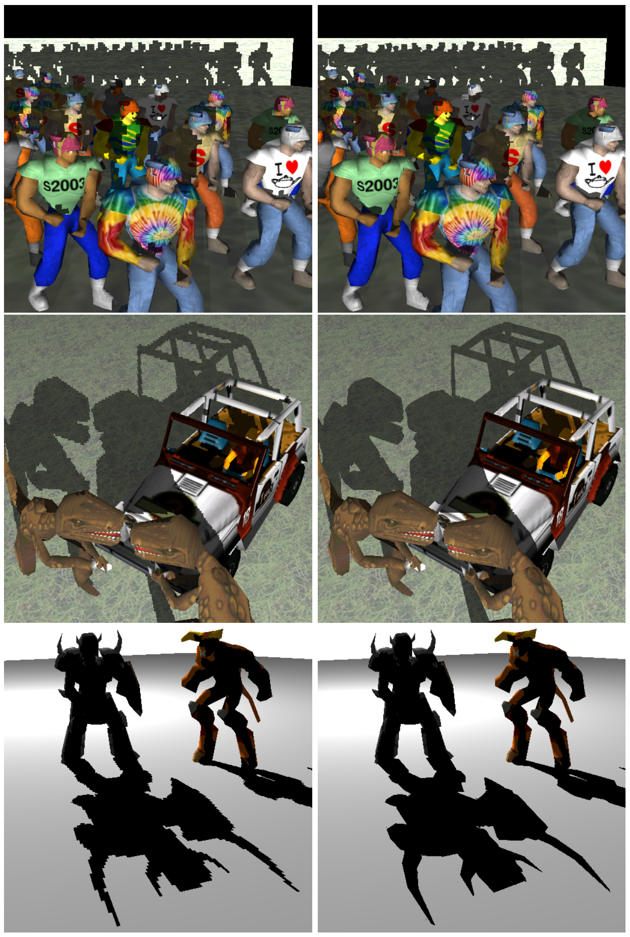

Figure 6: Test scenes (standard shadow map on the left, silhouette map on the right). From top to bottom: CROWD, JEEP, CASTLE.

4 Results

We test the silhouette map algorithm using the scenes shown in Figure 6. These scenes are dynamic and the silhouette map is recomputed for every frame. Statistics for these scenes are given in Table 1. The first scene, CROWD, consists of twenty-four characters casting shadows onto a distant wall. When viewed from certain directions, the depth complexity of the shadow volumes in this scene is very high. The second scene, JEEP, and the third scene, CASTLE, are examples typical of interactive entertainment applications. We also experiment with another scene, STRIKE, to illustrate shadows on curved objects (see Figure 9). Here, each bowling pin is tessellated to 450 triangles.

For these figures, the shadow and silhouette maps are generated at a resolution of 512 by 512 texels. In all test scenes, there is substantial improvement in shadow quality between standard shadow maps and silhouette maps. Our tests also indicate that the problems with depth bias are no worse in silhouette maps than shadow maps.

We also compare silhouette maps to an implementation of shadow volumes based on NVIDIA's publicly available shadow volume demo [Kilgard 2000]. This code identifies silhouette edges and extrudes shadow volumes. We test the two implementations on the scenes CROWD, JEEP and CASTLE. The quality of shadows computed using silhouette maps is similar to that of shadow volumes and the frame rates are comparable. In both algorithms, the actual frame rate is limited by the geometry issue rate of the application running on the host.

Although the silhouette map algorithm requires more arithmetic operations than the other algorithms, it is more important to compare the bandwidth requirements of the algorithms. We develop a simple model for the bandwidth consumed by all three algorithms, and measure scene statistics in order to compare them.

First, we compare shadow maps to silhouette maps. In the generation step, both algorithms generate a depth. Silhouette maps also require the silhouette be drawn. However, there are many fewer silhouette texels than interior texels, so the incremental cost of silhouette maps is small.

During shadow determination, shadow maps read 1 depth value. This is really the worst case, since for a magnified shadow map less than 1 depth value is read on average. Although silhouette maps use 4 neighboring depth values, on average only one value is read because of the spatial locality of texture accesses. This is similar to the case of mipmaps, which typically require only 1 texel to be read on average [Hakura and Gupta 1997]. To shade points near the silhouette edge, we need to read 5 silhouette points (the current silhouette point plus the 4 neighbors). There is less spatial locality in this case since the silhouette is a curve, so we will assume on average 3 silhouette points will be read. Thus, the average bandwidth for depth values will be 4 bytes per pixel, and the average bandwidth for vertices will be $24P$, where P is the percentage of pixels on silhouettes. If $P < \frac{1}{6} \approx 16\%$, then silhouette maps will consume less than twice the bandwidth of shadow maps. The results in the table show that this is always the case. It should be noted that P is artificially high in the table because the statistics correspond to cases where we are zoomed in on the shadow. We conclude that silhouette maps and shadow maps consume roughly the same bandwidth.

Next, we compare shadow maps to shadow volumes. As stated previously, shadow volumes require high fill rates; in our test scenes, the number of shadow volume fragments is between 4 and 155 times the number of rendered pixels. We call this the overdraw factor O. To compute the bandwidth requirements, we assume each shadow volume fragment must read 1 depth value and 1 stencil value (1 byte). If the shadow fragment passes the depth test, the stencil value is incremented and stored in the stencil buffer; this consumes on average an additional 1/2 byte of bandwidth. Thus, the ratio of stencil volume to shadow map bandwidth is $O \times \frac{5.5}{4}$. Therefore, if $O > 4/5.5$, the shadow volume algorithm will consume more bandwidth than the shadow map algorithm. In the scenes we tested, this is always the case.

5 Discussion

We conclude by comparing silhouette maps to existing methods as well as discussing the artifacts and limitations of our technique. In addition, we discuss the connection between the silhouette map and discontinuity meshing and propose future work.

Comparison with existing techniques: Silhouette maps offer improved visual quality over conventional shadow maps. They are also complementary to Stamminger's perspective shadow maps. While Stamminger's technique optimizes the distribution of shadow samples to better match the sampling of the final image, our technique increases the amount of useful information provided by each sample. The two techniques could be easily combined to yield the benefits of both.

While we have shown that silhouette maps use substantially less bandwidth than shadow volumes, our implementation did not show improved performance. Currently, both our prototype and the shadow volumes implementation are limited by the speed at which we can generate the silhouette, which is done on the host. In addition, our algorithm was implemented using the first generation of fragment shaders. In contrast, shadow volumes benefit from established, carefully-tuned hardware paths supporting stencil buffer operations. In the next section, we will discuss ways in which our algorithm could benefit from greater hardware support.

Scene Information				Shadow Maps	Shadow Volumes			Silhouette Map		
Scene Name	Triangles	Silhouette edges	Scene Fill	fps	fps	Volume fill	O	fps	Silhouette fill	P
CROWD	14,836	4,800	148.9k	11.5	9.2	22.96M	154.2	9.5	19.8k	13.3%
JEEP	1,732	729	298.3k	75	72	1.66M	5.55	58	37.5k	12.6%
CASTLE	1,204	466	268.5k	97	95	1.08M	4.03	80	19.20k	7.2%

Table 1: Performance comparison of the three techniques. The scene information lists the number of triangles in each scene as well as the number of silhouette edges processed by the shadow volumes and silhouette map techniques. The fill-rate columns indicate the number of fragments written in each algorithm for various steps. Scene fill is the number of fragments rendered to draw the scene without shadows. Volume fill is the number of fragments drawn when rendering the shadow volume geometry. The ratio of volume fill to scene fill is the overdraw factor *O*. For shadow and silhouette maps the fill rate is the scene fill. The silhouette fill is the number of fragments that pass through silhouette; *P* is the corresponding percentage of scene fill.

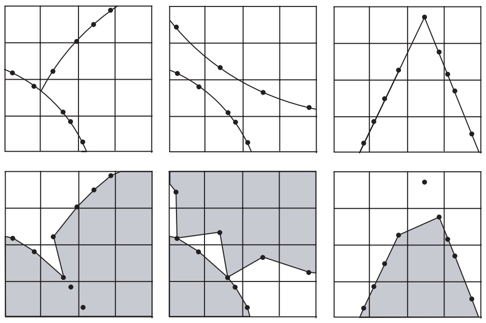

Figure 7: Artifacts of our algorithm. The top row shows silhouettes and selected silhouette points, the bottom row shows the shadows determined by our algorithm. (Left) two silhouettes intersect at a pixel, (Center) two silhouettes pass through a pixel but do not touch, (Right) a sharp corner is lost because of depth sampling resolution.

Hardware support: There are three parts of the algorithm that could be accelerated in hardware:

- Currently, we determine potential silhouette edges using code from the NVIDIA shadow volume demo [Kilgard 2000]. In our results, the performance of both silhouette maps and shadow volumes was limited by the cost of this silhouette determination on the host. Both methods would benefit from a faster mechanism for determining potential silhouette edges.
- In generating the silhouette map, our current implementation must draw thin quads and use a fragment program to identify the silhouette points. One could imagine a rasterization mode that implemented this process directly.
- When determining shadows, our fragment program must perform unnecessary operations because of the lack of conditional execution at the fragment level. Conditional execution of arithmetic and texture fetches could reduce the work of each shading pass by nearly a factor of four.

Eventually, one could imagine supporting the entire silhouette map algorithm as a primitive texture operation.

Artifacts: By choosing to store only a single silhouette point in each texel, we have limited the complexity of the curves that can be represented in the silhouette map. For example, when two disjoint silhouettes pass through a single texel, it is not possible to store a single point that is representative of both contours (see Figure 7). Because of the fixed resolution of the silhouette map, silhouettes with high curvature at the texel level will have artifacts. Increasing the tessellation of the geometry, however, does not affect the visual quality. In practice we found the artifacts introduced to be acceptable (see Figure 8) especially when considering the substantial improvement in quality relative to conventional shadow maps.

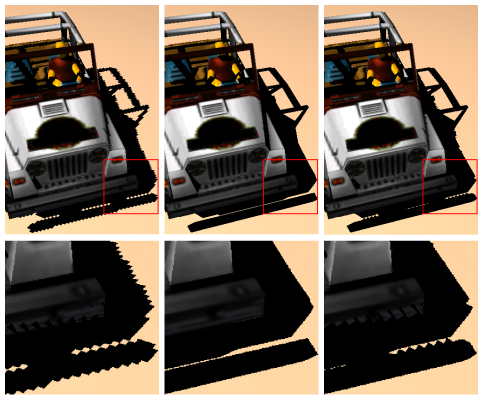

Figure 8: A closeup reveals artifacts of our algorithm when the resolution of the silhouette map is too low. (Left) standard shadow map and associated artifacts. (Center) Shadow volumes yields correct shadow. (Right) Silhouette map at the same resolution as the shadow map shows artifacts. In this case, the fender of the jeep yields an artifact like that in the center diagram of Figure 7.

Discontinuity Mesh: We have presented a new algorithm that extends shadow maps by providing a piecewise-linear approximation to the silhouette boundary. This method is easy to describe as a 2D form of dual contouring. Alternatively, we can think of the silhouette map algorithm as similar to discontinuity meshing of a finite element grid. We start with a regular grid of depth samples – the shadow map – where each grid cell contains a single value. We locally warp this grid so that the boundaries of grid cells are aligned with discontinuities formed by the shadows (See Figure 10). This is what our algorithm allows us to do using fragment programs on the graphics hardware. Note that when silhouette map points are at the center of the texels, the regular grid is undeformed, leading to behavior identical to that of conventional shadow maps.

Alternative Silhouette Representations: Prior to adopting a single point as the silhouette map representation, we explored two other techniques: triangle lists and edge equations. Initially, we considered using Haines' light buffer [1986], which stores a list of the potential shadowing triangles for each texel. We used a fragment program ray tracer to test shadow rays against this list of triangles [Purcell et al. 2002]. Unfortunately, even for simple scenes, we found we needed a long, variable-length list of triangles to get good results. This introduced many consequences which made it impractical. We next investigated using line equations to approximate silhouettes. Unfortunately, it was difficult to maintain continuity between the line segments. In the end, we stored points along the silhouette edge in our map as described in our paper. We store only two parameters (the relative x and y offsets) per silhouette map texel. Had the hardware allowed it, it would have been enough to

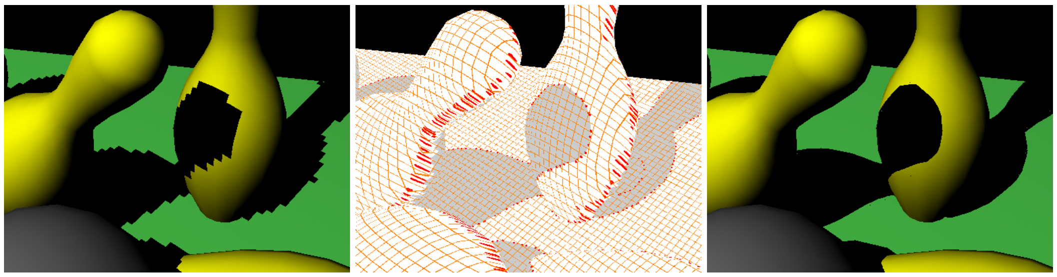

Figure 9: (Left) Scene rendered using a standard shadow map. (Middle) A visualization of the silhouette map superimposed on top of the scene. The red circles indicate the positions of silhouette points. On the edge of the bowling pin they are elongated into ellipses because they are being projected from the point-of-view of the light. (Right) Scene rendered using the silhouette map.

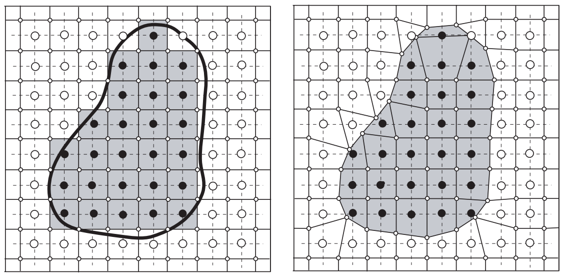

Figure 10: Silhouette maps can be thought of as warping a regular grid so that grid lines align with boundaries in the geometry. (Left) The two grids. In the background, shown in dotted lines, is the silhouette map texture. In the foreground is the depth map offset by (1/2, 1/2). The large circles indicate the depth values. Superimposed on that is the silhouette, drawn in black. (Right) We warp of the depth map grid so that it better matches the silhouette.

store a single byte to encode both of these offsets since 4 bits per offset would be enough resolution to faithfully represent points on the silhouette edge.

Future Work: Silhouette maps offer several directions for future research. A thorough exploration of different silhouette representations might yield insight into better representations for specific geometries. In addition, one might consider extending our current work on hard shadows to soft shadow algorithms. Finally, the implementation of the silhouette map shadow algorithm in hardware should be investigated.

6 Conclusion

We have presented an algorithm that significantly improves the quality of shadow mapped images. The basis of our algorithm is the silhouette map, a piece-wise linear representation of the shadowing geometry. The algorithm is simple, runs in real-time, and should work well in hardware.

Acknowledgments

We would like to thank Frank Losasso, Kurt Akeley, Kekoa Proudfoot, and Ren Ng for detailed comments and lively discussions during the course of this research. Mike Houston kindly proofread final drafts. We are also grateful to Mark Kilgard, who allowed us to use the NVIDIA shadow volumes demo which greatly aided our work. The models used in this work can be found at www.polycount.com. Finally, special thanks go out to James Percy of ATI for providing critical technical support during the implementation process. Funding for this work has been provided by the DARPA Smart Memory Project. Pradeep Sen was supported by a research grant from ATI. Mike Cammarano was supported by a 3Com Corporation Stanford Graduate Fellowship and a National Science Foundation Fellowship.

References

BRABEC, S., AND SEIDEL, H. 2001. Hardware-accelerated rendering of antialiased shadows with shadow maps. In *Computer Graphics International*, CGI, 209–214.

CROW, F. C. 1977. Shadow algorithms for computer graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 77)*, ACM SIGGRAPH, 242–248.

EVERITT, C., AND KILGARD, M. J. 2002. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. Webpage. `http://developer.nvidia.com/view.asp?IO=robust_shadow_volumes`.

FERNANDO, R., FERNANDEZ, S., BALA, K., AND GREENBERG, D. P. 2001. Adaptive shadow maps. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 387–390.

FUCHS, H., GOLDFEATHER, J., HULTQUIST, J. P., SPACH, S., AUSTIN, J. D., BROOKS, F. P., EYLES, J. G., AND POULTON, J. 1985. Fast sphere, shadows, textures, transparencies, and image enhancements in pixel-planes. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, ACM SIGGRAPH, 111–120.

HAINES, E. A., AND GREENBERG, D. P. 1986. The light buffer: a shadow-testing accelerator. *IEEE Computer Graphics and Applications 6*, 9, 6–16.

HAKURA, Z. S., AND GUPTA, A. 1997. The design and analysis of a cache architecture for texture mapping. In *24th International Symposium on Computer Architecture*, 108–120.

JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. 2002. Dual contouring of hermite data. *ACM Transactions on Graphics 21*, 3, 339–346.

KILGARD, M. J. 2000. Quake2 model bump-mapping with volumetric shadows. Webpage. `http://developer.nvidia.com/view.asp?IO=q2_bm_vol_shadows`.

PURCELL, T., BUCK, I., MARK, W. R., AND HANRAHAN, P. 2002. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics 21*, 3, 703–712.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering antialiased shadows with depth maps. In *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, ACM SIGGRAPH, 283–291.

STAMMINGER, M., AND DRETTAKIS, G. 2002. Perspective shadow maps. *ACM Transactions on Graphics 21*, 3, 557–562.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of ACM SIGGRAPH 78)*, ACM SIGGRAPH, 270–274.

WOO, A., POULIN, P., AND FOURNIER, A. 1990. A survey of shadow algorithms. *IEEE Computer Graphics and Applications 10*, 6, 13–32.