CMPUT 610
Implementation of a Soft Shadow Algorithm

Nathan Funk

April 7, 2004

1 Introduction

The main objective of this project was to implement the real-time soft shadow volume method proposed by Ulf Assarsson and Thomas Akenine-Möller in their SIGGRAPH 2003 paper [1]. The original method is proposed as a hardware implementation. However due to time constraints, the implementation in this project does not employ a graphics accelerator card for the soft shadow rendering. Fortunately, the features of the approach could still be studied in great depth without a hardware implementation.

The original method is one of the first accurate real-time soft shadow rendering methods. Previous methods are either slower, less accurate or less flexible in terms of the shadow casters and receivers. In contrast, Assarsson and Akenine-Möller's approach is very flexible since it allows arbitrary shadow casters and receivers, and also arbitrary light sources when generalized. The novel idea of the paper uses edges of the object's silhouette to construct pieces of the shadow penumbra individually. More details follow in the Background section.

2 Background

2.1 Outline of the original method

The algorithm proposed by Assarsson and Akenine-Möller is broken down into three major steps. First the so called *penumbra wedges* are constructed for each edge on the silhouette. These define loose bounds that constrain the soft shadow volume. The second step, *visibility pass 1*, is to draw a hard shadow from the center of the light source. Finally in *visibility pass 2* the penumbra region is filled with the smooth transition into the umbra which has already been filled by the hard shadow. Note that the final step changes the visibility values of parts of the hard shadow.

An assumption that is made in the paper, is that the silhouette is calculated from a single point in the center of the light source. Of course, with an area or volume light source, the edges of the silhouette may change depending on which point within the light source the object is viewed from. This assumption is quite necessary however, since most of the calculations are based on the location of the individual silhouette edges.

The shadows are initially rendered into the *visibility buffer* which acts as a gray-scale mask. The mask acts solely on the the specular and diffuse components of the lighting. This can be accomplished by using the mask as the alpha channel on the buffer containing the specular and diffuse lighting, blending with a solid black image, then adding the ambient lit scene.

2.1.1 Penumbra Wedge Construction

One penumbra wedge is constructed for every edge along the silhouette. The first step in the construction of the penumbra wedges is to move the edge vertex furthest from the light source towards the light source. It is moved along a straight line towards the light source until it is at the same distance from the center as the other edge vertex. Four planes are constructed which barely touch the light source. Two of these (the front and back plane) contain the moved edge, the other two pass though either the first or second vertex.

Assarsson claims that without shifting the farther vertex, the penumbra wedge would not fully enclose the penumbra volume in general. By creating a wedge which is larger than the penumbra region, some unnecessary calculations are made in the subsequent steps. These are however minimal. Figure 1 shows a simple example scene with highlighted penumbra wedges.

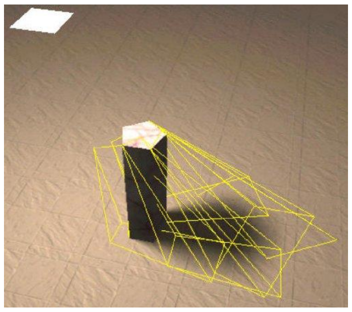

Figure 1: The penumbra wedges as constructed in the original paper [1]. Note that the area of intersection with the base plane extends beyond the penumbra region in some places. This a the result of building the wedges around the modified silhouette edges.

2.1.2 Visibility Pass 1

The first visibility pass simply fills the previously cleared visibility buffer with a hard shadow. The hard shadow is generated from a point located at the center of the light source. The common shadow volume algorithm proposed by Crow [2] is used for this purpose.

2.1.3 Visibility Pass 2

This final step creates the soft region of the shadow which extends beyond the previously drawn hard shadow. It also compensates for the overstated umbra region caused by the last step.

The concept of *visibility* is introduced here according to Drettakis and Fiume [3]. For a point **p** on the shadow receiver, the visibility is the area of the light source visible from **p** divided by the total area of the light source. In other words, when looking at the light source from **p**, the visibility is the percentage of the light source that is not occluded by the shadow caster. So by determining the visibility of the points on the shadow receiver, the intensity of the shadow in the penumbra region is determined point by point.

The calculation of the visibility is the true innovation presented in this paper. It is performed on silhouette edges individually so the shadow can be computed in a robust fashion independently of the shape of the entire silhouette. Assarsson assumes that the light source appears rectangular from a point on the shadow receiver. For most points within the penumbra wedges, the shadow caster will partially occlude the light source. The amount of occlusion is determined by first projecting the current silhouette edge with its associated hard shadow onto the light source. Figure 2 shows the projected hard shadow quad on the light source. The overlapping area is determined by first clipping the edge on the light source. The two resulting points (x_1, y_1) and (x_2, y_2) can be used to calculate the size of the area using Green's theorem.

The plane of the hard shadow quad Q divides space into a negative and positive half space. Depending on which side of Q the point **p** is located the coverage value is either added or subtracted from the total coverage. Details of why this is necessary are given in the original paper [1].

The area calculation is not completely trivial and would impede the performance of the algorithm if it was performed repeatedly for every point and edge. Since the area is constant for two specific clipped points, the area can be precomputed and stored in a lookup table. This table would be 4D (since each area value depends on four parameters), but it can flattened into a 2D table by combining dimensions. This is done so that the table can be stored as a texture on the graphics card in a hardware implementation. The texture is referred to as *coverage texture* and is precomputed and loaded before the coverage calculations.

In the implementation described in the paper, they use $n = 32$ values for each lookup parameters so the resulting texture is 1024 x 1024. If the light source is one solid color, the coverage texture can be gray scale. The method however also allows colored, textured and video light sources. For colored textured light sources, the coverage texture contains three color channels.

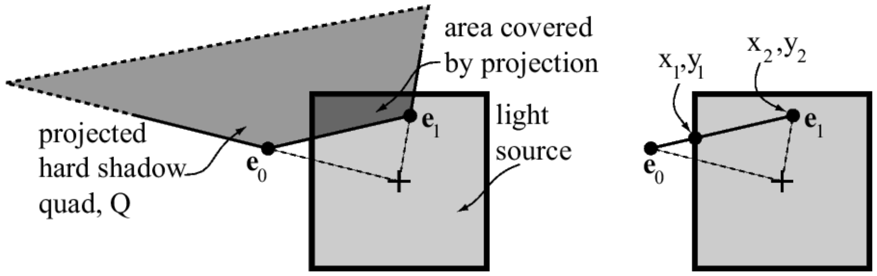

Figure 2: Left: View of the light source from a point **p** on the shadow receiver. The area covered by projection needs to be determined for the coverage calculation. Right: The silhouette edge is clipped against the light source border. This results in points (x_1, y_1) and (x_2, y_2) which are used as an index in the lookup of the coverage value. (From the original paper [1])

3 Implementation

The general structure of the algorithm is maintained in this implementation, however some modifications were made. The calculation of the hard shadow is completed using the stencil buffer approach from assignment 4. Although originally assignment 4 was implemented for a directional light source, only a few simple modifications were necessary for generating the shadow of a point light source.

The following section outlines the most important differences between the original and the implemented algorithms.

3.1 Differences from original algorithm

3.1.1 No penumbra wedges

During the implementation it was recognized that implementing the complete algorithm would take too much time to complete. For this reason, it was decided to omit the construction of the penumbra wedges. Since the main purpose of the penumbra wedges is to restrict the area within which the visibility is computed, it is not essential to the functioning of the algorithm. The most major expected effect due to this omission was decreased performance, since the coverage calculations need to be performed on many more points than in the original implementation. Indeed, this effect is quite noticeable in the final performance of the shadow computation. More details are provided in the Performance section.

Another side effect of omitting the penumbra wedge construction is shadows might be cast on objects that can not physically receive shadows. For example if the point **p** is nearer to the light source than the silhouette edge, the projection of the edge onto the light source might still result in clipping and coverage calculation. The resulting artifacts are discussed in the Artifacts section.

3.1.2 Circular light source

The second major difference, is that a circular light source was implemented rather than a rectangular one as outlined in the paper. Initially this decision was made to simplify the task of constructing the penumbra wedge, and after deciding not to construct the penumbra wedges, it was still maintained since it also simplifies the clipping slightly.

The light source appears as a circle from any view point giving the impression of a spherical light source. There is however a fine difference between an actual spherical light source and the implementation however. For a spherical light approximated with a large set of uniformly distributed point light sources, the density of these point light sources is highest in the center of the light, when projected onto a 2D surface. This implementation assumes a uniform distribution in the 2D plane, equivalent to a spherical light source with a lower density in the center. The results show assumption has negligible effects on the appearance of the shadow.

Clipping of each silhouette edge on the light source boundary needed to be implemented. Although line-rectangle clipping algorithms are readily available and well documented, line-circle clipping algorithms do not appear to be as popular. The clipping algorithm was developed from a parametric representation of the line and the equation for a unit circle in Cartesian coordinates ($x^2 + y^2 = 1$). The potential intersection points can be expressed as solutions to the quadratic equation:

$$\begin{aligned} & t^2((x_{e1} - x_{e0})^2 + (y_{e1} + y_{e0})^2) \\ & + t(2x_{e0}(x_{e1} - x_{e0}) + 2y_{e0}(y_{e1} - y_{e0})) \\ & + x_{e0}^2 + y_{e0}^2 - 1 = 0, \end{aligned} \tag{1}$$

where t is the parametric value of points on the line from $e_0 = (x_{e0}, y_{e0})$ to $e_1 = (x_{e1}, y_{e1})$. Figure 3 shows an example clipping configuration. By solving Equation 1, two values for t are obtained. In the case where the results would be complex (the term under the square root of the solution equation is negative), the line does not intersect the circle. This case is excluded before the calculation of t. Other special cases are when the line does intersect the circle, but the intersection is not between e_0 and e_1. As in Figure 3 the line continued past e_0 and e_1 intersects the circle at two locations. Only one of them is relevant however.

Tests were performed on the clipping algorithm to ensure that all special cases are handled properly.

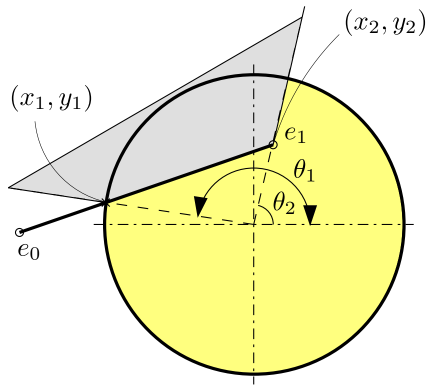

Figure 3: An example clipping configuration. Since the implemented light source is circular, the clipping of the edges against the light source boundary needed to be modified.

3.1.3 Coverage calculation

Since the light source is circular, the coverage calculation also needed to be modified. A MATLAB script was written to generate a 1024x1024 sized coverage texture (shown in Figure 4). Instead of using Cartesian coordinates for the lookup parameters, polar coordinates are used. Each clipped point is described by an angle θ (see Figure 3) and a radius r. Lookup in the texture is performed with the coordinates $(nr_1 + r_2, n\theta_1 + \theta_2)$, where $n = 32$ is the number of discrete values for each parameter.

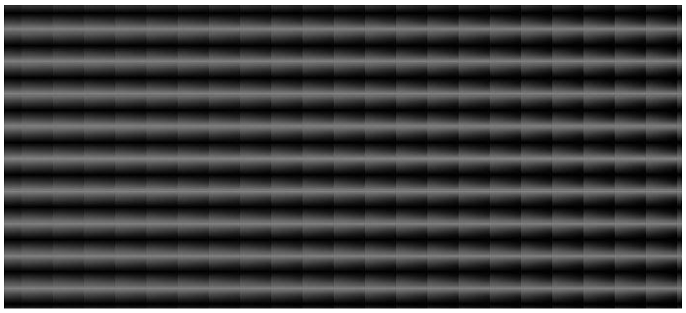

Figure 4: A sample section of the coverage texture. r is indexed in the horizontal, θ in the vertical direction. Note that towards the left side, there is little intensity variation in the horizontal direction. This is due to the coverage area not varying much for low values of r_1.

The coverage area is computed by subtracting the triangle area defined by the two clipped points and the center of the circle from the pie section between the clipped points.

As the sample of the coverage texture shows, some areas show little variation in the horizontal direction. For example when r_1 is small (the left side of the texture), r_2 has little effect on the area. Geometrically this is sound, but it points out that there is a significant amount of redundancy in the texture. The same holds for the original texture although it is not quite as apparent there.

There is a considerable amount of symmetry in the area calculation. For example the area is the same independent of the order of the clipped points. Considering this would already reduce the texture size by one half. Rotating the clipped points in the original approach by $90°$, $180°$ or $270°$ also results in the same area. Assarsson and Akenine-Moeller express the desire to minimize the size of the coverage texture in their Performance Results section as well as when mentioning video light sources. Considering some of these symmetries would assist in this.

The implemented method is also not optimal in the coverage value lookup. The use of polar coordinates does simplify some aspects of the implementation, but it is also computationally more expensive compared to leaving the coordinates in Cartesian form.

3.1.4 Ambient lighting

This implementation does not consider possible ambient lighting. Adding ambient lighting would simply involve blending the masked diffuse and specular scene with the same scene rendered under ambient lighting without shadows.

4 Artifacts

A number of artifacts were observed during the implementation of the method. Some of these could be fixed before completing the project, others were left unresolved. This section outlines the artifacts, describes how they are caused, and how they can be avoided.

4.1 Thick lines in the penumbra

Figure 5 shows an example of a thick line artifact encountered during the implementation. Finding the bug causing this artifact was difficult, because the exact location of the artifact was not obvious. The exact position was finally discovered by adjusting the view position to be located immediately above the artifact region.

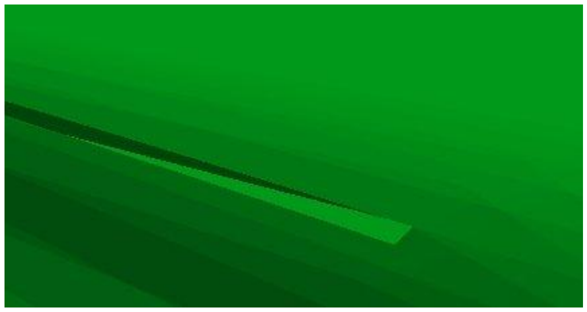

Figure 5: The thick bright line artifact in the center of the image was caused by a bug in the lookup value calculation.

With the coordinates of the points with incorrect shadow values, the bug was finally traced back to a rounding error. One of the θ angle parameters was rounded above 2π when it should have instead been wrapped back to above 0. This resulted in a wrong lookup position in the coverage texture causing unpredictable results.

4.2 Thin edge surrounding hard shadow region

An unexpected artifact was encountered after finishing the implementation of the coverage calculation. As Figure 6 shows, a thin edge is present in the center of the penumbra region. In fact, it is located at the hard shadow boundary.

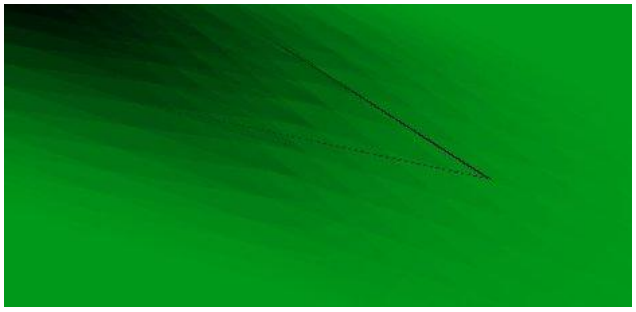

Figure 6: The thin line at the tip of the shadow is an artifact caused by inconsistent representations of the hard shadow volume.

This artifact is caused by a slight difference in the calculation of the hard shadow boundary in the stencil buffer compared to the hard shadow boundary calculation necessary for the coverage value calculation.

There are several options for avoiding this artifact. The main concept would be to render the hard shadow and the coverage values using the same description of the hard shadow planes. Although the artifact might be reduced by increasing the accuracy of the coverage calculations (`float`s were used where `double`s could be used instead), this would not necessarily guarantee no artifacts. Instead, modifying the stencil buffer approach to use the same representation of the hard shadow planes as the coverage calculations is more likely to achieve good results.

4.3 Intersecting volumes

Figure 7 was generated using an airplane model in which the wing volume intersects the volume of the fuselage. The implementation does not handle this case appropriately. By separating the two volumes and performing the entire algorithm separately on each volume, this result could be avoided.

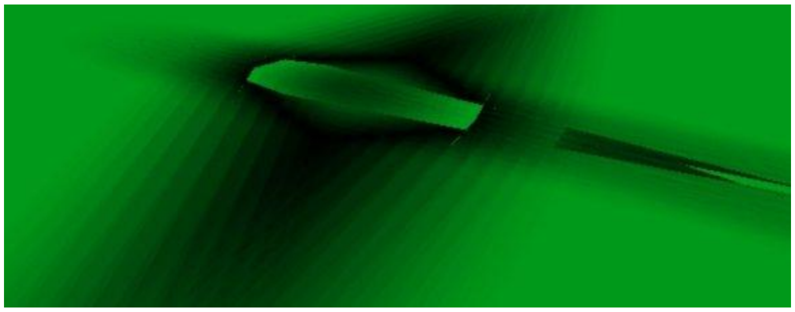

Figure 7: The wrong shadow calculation in this image is the result of improper handling of intersecting volumes.

4.4 Steps in the soft shadow

As Figures 5, 6 and 7 show, the soft shadow is not very smooth when viewed close-up. This is a result of the discrete lookup of coverage values. Assarsson and Akenine-Möller mention that they experimented with bilinear filtering while doing the lookup. More precisely, as they point out, a quadlinear lookup would need to be performed.

Bilinear filtering was incorporated after the making of the project video. However the results only improved the appearance slightly. Some steps were still visible. Full quadlinear interpolation would likely be worthwhile in a demanding implementation if shadows are viewed close up, if light sources are large, or if the distance between the shadow caster and the receiver is large.

4.5 Artifacts outside shadow region

As mentioned earlier, since the penumbra wedges are not constructed, the coverage calculation is performed on all visible pixels. Most points are entirely outside the shadow region, but for others the coverage is non-zero even if they would not physically receive shadows. For example at points located between the light source and the silhouette, the silhouette edges are still projected on the light source and successfully clipped. This was partially avoided by ignoring points that are closer to the light source than both of the current edge points.

Of course the proper solution to this problem is the use of penumbra wedges to restrict the area over which coverage values are computed.

5 Performance

Assarsson and Akenine-Möller report frame rates of 0.4 fps at a resolution of 512x512 for a alien figure model in their software implementation. This implementation renders at approximately 0.25 fps at a resolution of 720x480 for a simple airplane model. This relatively low result is expected since a `gluUnProject()` call is made for every pixel in the frame to determine its associated 3D position. Restricting the coverage calculations within the penumbra wedges would greatly improve the performance.

Of course these numbers are not particularly relevant since the algorithm is intended for a hardware implementation using a pixel shading language. Since the original paper was published, the algorithm was implemented on both the NVidia GeForce FX and the ATI 9700.

The computation time of the 1024x1024 coverage texture was about 5 minutes on a 1 GHz Pentium 3. Since the implementation was performed in MATLAB it is by no means optimized. Assarsson and Akenine-Möller's implementation takes less than 3 minutes on a 1.7 GHz Pentium 4.

6 Conclusions

This implementation shows the impressive flexibility of Assarsson and Akenine-Möller's work. In comparison to previous methods such as Heckbert and Herf's [4], it is performance and accuracy are very promising.

A few modifications were made to the original approach. Omitting the penumbra wedge construction was necessary for the project to be completed on time. The use of a circular light source demonstrates the flexibility of the approach in that it is not bound to the rectangular light source as implemented in the original paper. Through the resulting modified coverage texture it was also discovered that the coverage textures both in this implementation and the original one contain a significant amount of redundant information. Considering the symmetries in the area computation would reduce the size of coverage textures.

This paper provides approaches to avoid all the artifacts encountered. Incorporating these improvements and the addition of penumbra wedges would result in a considerably robust and flexible soft shadow implementation.

References

[1] U. Assarsson and T. Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. In *Proceedings of ACM SIGGRAPH 2003*, pages 511–520. ACM, 2003.

[2] F. Crow. Shadow algorithms for computer graphics. In *Proceedings of ACM SIGGRAPH 1977*, pages 242–248. ACM, 1977.

[3] G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using back projection. In *Proceedings of ACM SIGGRAPH 1994*, pages 223–230. ACM, 1994.

[4] P. Heckbert and M. Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, Carnegie Mellon University, January 1997.