Hardware Accelerated Soft Shadows using Penumbra Quads

Jukka Arvo (*) Jan Westerholm (¤)
Department of Information Technology, University of Turku (*),
Turku Centre for Computer Science (*,¤), and Åbo Akademi University (¤)
Lemminkäisenkatu 14 A,
FIN-20520 TURKU, Finland
jarvo@cs.utu.fi (*), jan.westerholm@abo.fi (¤)

ABSTRACT

Shadow mapping is a commonly used technique for generating hard shadows in real time. However, existing shadow map based algorithms cannot render full soft shadows penumbras onto arbitrary dynamic geometry by utilizing only consumer level programmable graphics hardware. In this paper we introduce a new fully hardware accelerated penumbra map algorithm which stores additional penumbra information into separate penumbra maps. The method is capable of generating approximate soft shadows on both sides of the hard shadow boundaries at real time frame rates. The shadow generation requires neither graphics hardware read backs nor processing with the CPU while it is able to handle arbitrary shadow receivers and dynamic environments. The algorithm also has a straightforward implementation on programmable graphics hardware.

Keywords
Shadow algorithms, soft shadows, graphics hardware, shadow map

1. INTRODUCTION

One of the most challenging tasks in computer graphics is the efficient computation of *soft* shadows. Soft shadows occur because different points in the scene receive different amounts of light from an area light source. Regions which are fully in the shadow, that is, occluded from the light source, are called the *umbra*. Points that cannot fully see the light source, but can see some part of it, are in the soft transition from no shadow to fully in the shadow (the *penumbra* region). Thus the soft shadow calculation can be seen as an area visibility problem. Algorithms that model both umbra and penumbra shadows are commonly referred to as "soft shadow" algorithms.

There are two main avenues for the treatment of area light sources in hardware accelerated real-time computer graphics: shadow volumes and shadow maps. Shadow volumes [Cro77a] construct additional polygons that bound the umbra regions and a pixel is in the shadow if it is inside the bounded umbra region. Approximate soft shadows can be achieved by generating multiple shadow volumes from different sample positions of the area light source or by constructing additional penumbra geometry [Ass03a]. Shadow maps on the other hand [Wil78a] discretize the scene geometry as seen from the light source into a shadow map. The shadow map is essentially a z-buffer, and describes the light space z-coordinate of the pixels that are lit by the light source. This information is later used for determining whether a pixel in the camera view space is seen by the light source or not and thus lit or dark. Fully hardware accelerated soft shadows can be generated by rendering multiple shadow maps from various light source sample positions [Hec97a].

The soft shadow algorithm presented in this paper may be classified as a shadow map algorithm, but instead of only storing the z-coordinates for each pixel as seen from the light source, our new algorithm stores additional penumbra information into separate penumbra maps which will be used to generate the penumbra regions. In contrast to previous penumbra mapping algorithms [Kir03a, Cha03a, Wym03a], our method generates both inner and outer penumbra regions on both sides of the hard shadow edges and it does not require any runtime

Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972
WSCG'2004, February 2-6, 2004, Plzen, Czech Republic.
Copyright UNION Agency – Science Press

processing with the CPU. As in those methods, our approach produces only approximate and not physically based soft shadows, as will be discussed in section 4.3. In particular, our method tends to generate slightly larger penumbras which are however visually appealing.

We refer to the frustum of a camera as a *view frustum* and the frustum of a light source as a *light frustum*. A pixel shader is a GPU program that computes the color value of a pixel in the image space. A render target buffer is a table where the pixel shader results are gathered. A vertex shader is a GPU program that computes vertex parameters during various transformations.

The paper is organized as follows. In Section 2 related work is reviewed. In Section 3 some basic quantities relevant to the practical implementation of irradiance calculations are presented. In Section 4, we give a detailed description of our new algorithm. Then in Section 5, we discuss implementation details and present our test results obtained under a variety of conditions. We discuss some limitations in Section 6 and finally in Section 7 we point out directions for future work and present our conclusions.

2. Previous work

In this section we review the work that is most closely related to ours. For a complete review of shadow generation algorithms see [Ake02a, Has03a].

2.1 Hard shadow mapping

The shadow map method was first introduced by Williams [Wil78a]. First the algorithm forms the light space image into a z-buffer, the (standard) shadow map which represents the surfaces illuminated by the light. Then the view frustum image is generated into another z-buffer and those parts that coincide with the shadow map are illuminated. Coincidence is tested by transforming each view frustum pixel into the light frustum and comparing the transformed depth value with the shadow map (Figure 1).

Because the shadow map method is based on transformations between discrete maps, it is prone to quantization and aliasing problems. Reeves et al. [Ree87a] presented the percentage closer filtering method for softening the shadow boundaries and thus reducing the shadow boundary aliasing. The intensity of a sample is determined based on the percentage of the surrounding weighted samples that are closer to the light source.

2.2 Approximate soft shadowing.

Heckbert and Herf [Hec97a] introduced an algorithm, which creates soft shadow textures for each receiver surface by combining multiple hard shadow images. Although the algorithm can utilize graphics hardware, it is mostly suitable for shadow texture pre-generation due to the large number of samples needed to achieve smooth penumbra regions.

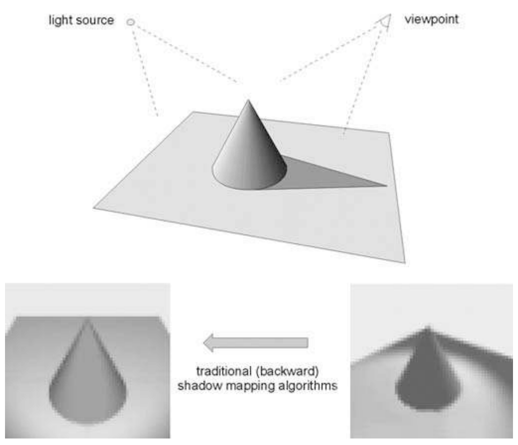

Figure 1. Two z-buffers are created, one in the light frustum and another in the view frustum. The shadow regions are determined by pixel transformations and depth comparisons between these z-buffers.

Soler and Sillion [Sol98a] used a convolution on occluder images to compute approximate soft shadows. For parallel light sources, a light source image is convolved with a hard shadow image of a clustered geometry. General configurations are handled by an error driven algorithm. Hardware accelerated implementtations would require specialized DSP features, which are not available on current consumer level graphics hardware.

A method presented by Gooch et al. [Goo99a] approximates the two-dimensional sampling of the spherical light source with a one-dimensional sampling process. This is implemented by moving the receiver surface up and down, and averaging the resulting shadows. Haines [Hai01a] presented an algorithm for creating outer penumbra regions on a flat surface by searching the silhouette edges and forming circular shadow cones at each vertex. Adjacent cones are used to interpolate the shadows along each edge. Neither of these two methods works with arbitrary receiver geometries nor do they have implementations on concurrent graphics hardware.

Heidrich et al. [Hei00a] introduced an algorithm for linear light source shadows by using a few light source samples in low frequency scenes. Separate shadow maps are generated from each light source sample point and depth discontinuities are detected. The resulting depth discontinuities are then triangulated, warped, and rendered into a visibility map. The method has been generalized to polygonal light sources by Ying et al. [Yin02a].

Brabec et al. [Bra02a] presented an area light source algorithm using a single shadow map which generates soft shadow boundaries by performing a neighborhood scan for every view frustum pixel. The distance from the hard shadow boundary then determines the illumination. The algorithm utilizes graphics hardware for the view frustum and light frustum rendering, but all soft shadow computations are CPU based.

Agrawala et al. [Agr00a] used multisampling of the area light source and layered attenuation maps for storing multiple depth samples for each pixel. They also described a coherent ray tracing based algorithm for higher quality images.

Approximate soft shadows can be divided into inner and outer penumbra regions according to whether they lie inside or outside the hard shadow region as seen from a single light source sample point. Kirsch and Doellner [Kir03a] presented an algorithm which generates approximate inner penumbra regions. Respectively, Chan and Durand [Cha03a] and Wyman and Hansen [Wym03a] introduced outer penumbra algorithms. The central contribution of Chan and Durand was to replace the fixed soft shadow width for outer penumbra regions by an approximate width depending on the distance between the occluder and the receiver. In our current work, we mimic both inner and outer penumbras, and our algorithm is fully hardware accelerated. The previously mentioned penumbra methods required runtime processing with the CPU.

Assarsson and Möller [Ass03a] recently introduced a shadow volume based algorithm for generating geometrically based soft shadows. The algorithm is based on generating additional penumbra volumes where each penumbra volume pixel clips it's silhouette edge towards the area light source (for a description of silhouettes, see Section 4). Shadow volumes are prone to excessive fill-rate problems and the penumbra volumes will aggravate the situation further.

3. Preliminaries

During the calculation of soft shadows we have to address the question on how to compute the irradiance incident on a surface. The computation of the irradiance from an area light source on a surface is given by [Coh93a]:

$$E = \int_{A_{light}} \frac{L \cos\theta_i \cos\theta_l}{\pi r^2} V dA \qquad (1)$$

where $\boldsymbol{L}$ is the radiance from the light source, θ_l is the incident angle, θ_i is the angle made with the lighting normal, $\boldsymbol{V}$ is the binary visibility term, and r is the distance from the processed point on the surface to the point on the light source.

A common approximation in rapid soft shadow calculations is to separate lighting and visibility calculations. Thus, it is reasonable to compute the average visibility term that attenuates the lighting as [Agr00a]:

$$v = \frac{1}{A_{light}} \int_{A_{light}} V dA \qquad (2)$$

The irradiance at the current pixel can then be approximated by:

$$E \approx lv \qquad (3)$$

where $\boldsymbol{l}$ is the surface irradiance due to a point light source and $\boldsymbol{v}$ is the average visibility term. Our algorithm is strongly based on this assumption and we mainly concentrate on the rapid computation of the visibility term.

4. Algorithm

In this chapter we begin with a general description of our algorithm (Section 4.1). This is followed by a technical description of the *penumbra quad,* an essential concept used in the algorithm (Section 4.2). In Section 4.3 we present the algorithm in full detail by explaining how penumbra shadow maps are generated from the penumbra quads and how the umbra and penumbra shadows in the view frustum are finally rendered (Section 4.4).

4.1 Algorithm Structure

Consider the following simple geometry: A ground plane, a closed polygonal object above the ground plane, and a point light source above the object. The light from the point light source falls on the object (*occluder*) which subsequently casts a hard shadow onto the ground plane. We begin by calculating the standard shadow map (*first shadow map*) for the light source. In the shadow map the occluder is a 2-D object with a closed boundary (*silhouette*). As the occluder typically is made up from polygons, analytically the silhouette consists of a sequence of straight lines closing on itself. By construction the silhouette generates the boundary of the hard shadow. Suppose now that the point light source grows to an area light source of for instance circular cross section. The effect of this is the emergence of penumbra shadows on both the inner and the outer sides of the boundary of the hard shadow. In our algorithm we attribute the growth of the penumbra regions to the silhouette lines by assigning one quadrilateral for each silhouette line. This quadrilateral is generated by duplicating the end points of the silhouette lines, and it will be used to compute two new quadrilaterals, the *outer* and the *inner penumbra quad*, depending on whether the duplicated points are shifted to the outside or the inside of the silhouette. Thus only one geometric object will be used to generate two penumbra quads, and henceforth we will only speak of the penumbra quads. The algorithm essentially deals with a method of estimating the size, illumination and shadow effect of these quads, that is, we compute where the quads will throw their penumbra shadows. The results of these calculations are stored in two additional maps in the light space image, called penumbra maps, one for the outside quads *(outer penumbra map*) and one for the inside quads (*inner penumbra map*). The utilization of the inner penumbra map also uses a *second shadow map* which stores depth values of the second front facing surfaces as seen from the light source; the inner penumbra shadows will be cast on these objects. The final illumination in the light space image is generated from a linear combination of the four shadow maps, yielding the umbra and penumbra shadows. As will be shown, both shadow maps and both penumbra maps can be fully generated and combined in the GPU with no read-backs or processing with the CPU.

In the following subsections, we describe in more detail the penumbra quads, how the penumbra maps are generated, and the soft shadow visualization process.

4.2 Penumbra quads

The penumbra regions are located on both sides of the hard shadow boundaries which are directly related to the silhouette edges as seen from the center of an area light source. Soft shadows can thus be computed by detecting the silhouette edges with the CPU and uploading new penumbra geometry into the graphics card at each frame. However, this is often computationally rather expensive and therefore we compute the penumbra fully with the GPU by creating additional static geometry, penumbra quads, which are modified at runtime. The penumbra quads are created at start up, and both inner and outer penumbra map rendering utilize the same quadrilaterals.

For each object and each edge, we generate a quad that has two sides consisting of copies of the original edge, *the inner* and *the outer edge,* and two opposite sides of zero length. The edges have different

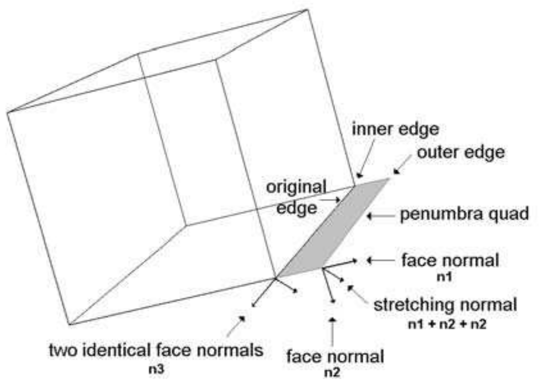

Figure 2. The penumbra quad is enlarged to illustrate the basic concepts. Every edge of the original object is replaced by a penumbra quad which has two edges parallel to the original edge (inner- and outer edge) and two opposite sides of zero length. All penumbra quad vertices have two face normals from their sharing polygons. The stretching normal is an average of the object normals sharing the vertex.

behavior because the inner edges always remain at the silhouette and only the outer edges are moved. Therefore we add two face normals and one stretching normal into every penumbra quad vertex (Figure 2), where the outer edge face normals are taken from the original mesh normals and the inner edge face normals are set to equal value (we will see later why). The stretching normal is always an average of the face normals sharing the current vertex in the original mesh. As the result, penumbra quads can be modified with the vertex shader at runtime and only the outer edges are detected as silhouettes. The inner edges are necessary ingredients in forming the penumbra quads as geometrical objects.

It should be noted that every shadow casting object has to be collapsed into a single two-manifold mesh, where all redundant back-to-back triangles are removed and the remaining mesh is welded together before generating the penumbra quads. For example, if the wing and the fuselage of an airplane are separate closed objects, then the wing should be connected to the fuselage in such a way that back-to-back triangles in the junction area are removed. Otherwise, false penumbra regions may be generated, because the vertex shader will falsely classify edges in the junction as silhouettes. The same restriction applies to the GPU based shadow volume generation [Eng02a] as well, but the resulting artifacts are usually less distracting with hard shadows.

4.3 Penumbra map generation

The average visibility information for both penumbra maps is generated by rendering penumbra quads at the silhouette edges. This requires that one has to detect the silhouette edges, stretch the corresponding penumbra quads to become visible, and compute the average visibility value for each rendered penumbra map pixel.

To accomplish this, we first detect the silhouette edges with the vertex shader by computing two dot products between the two penumbra quad face normals and the vector from the light to the vertex. The vertex is concluded to be part of the silhouette if the results have opposite signs. Then we stretch these silhouette penumbra quads by projecting each silhouette vertex into the light source's far plane and back towards a point determined by the stretching normal and the boundary of the light source (Figure 3). We compute this back stretching point from the angle of the stretching normal and the boundary of the light source. Notice that by putting equal face normals for the inner edges, these will not be detected as silhouette edges.

After the penumbra quad vertices have been processed they are sent to the pixel shader, which computes the average visibility values for the visible penumbra quad pixels. The computation utilizes penumbra quad vertex visibility values (***v*** in Eq. 3), that are ***v*** *= 0.0* at the outer edge (vertices) of the inner penumbra polygons and ***v*** *= 0.5* at the inner edges. Similarly, for the outer edge of the outer penumbra polygons, the visibility value is ***v*** *= 1.0* and for the inner edge ***v*** *= 0.5*. The visibility value ***v*** *= 0.0* means that the point is fully occluded from the light source and ***v*** *= 1.0* indicates that the pixel can "see" all points on the light source. This configuration gives a perspective correct linear transition between the fully lit and the fully shadowed regions and it can be easily computed with the pixel shader. The result is not physically based but it generates acceptable soft shadow boundaries, and both inner and outer penumbra quads are rendered by using the same penumbra quad geometry with due care paid to the stretching direction. Figure 4 illustrates the result of calculating the inner and outer penumbra maps with corresponding shading.

The initial z-buffer for outer penumbra map rendering is the standard shadow map because it prevents penumbra region leaking, that is, a false assignment of receiver surface. The problem here is that the GPU vertex shader does not identify which surfaces (and silhouettes) are lit by the light source and which are simply in the shadow of the occluders and therefore should not be assigned any penumbra quads. Similarly the inner penumbra map is rendered by second depth values (second shadow map) as the initial depth buffer [Nvi02].

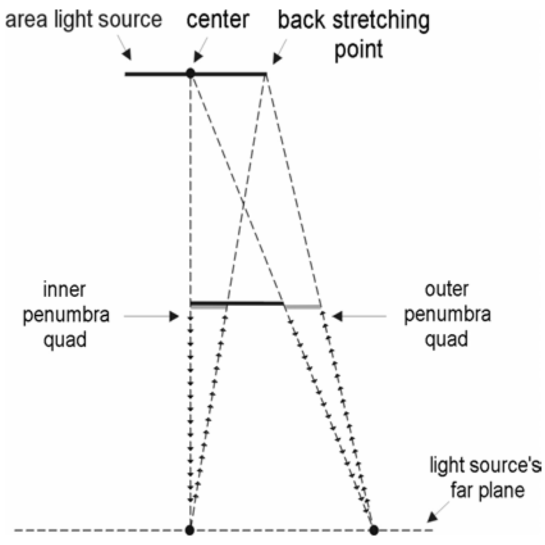

Figure 3. Every silhouette vertex is projected all the way down to the far plane and then back towards the area light source's silhouette. The back stretching direction towards the back stretching point is determined with the stretching normal. This process is performed separately for both the inner and the outer penumbra quads.

Our algorithm cannot use the exact distance between the occluder and the receiver, that is, the first depth map value for the outer penumbra quad vertices and the second depth map value for the inner penumbra quad vertices due to hardware restrictions. This may however change in the near future, when a fast copy from the frame buffer to a vertex attribute array on the GPU becomes available [Bra03a]. This will enable the rendering of depth values as seen from the light source for each penumbra quad vertex and they can be used directly for the stretching distances. Due to the lack of this feature, we are forced to use the light source's far plane for the stretching distance. Additionally, if our procedure in its present form yields too large penumbra quads, it can be easily modified by computing additional smoothie values for the penumbra regions [Cha03a].

We note in passing that although penumbra quads are assigned to all edges, the presented algorithm detects and uses only the silhouette edges of shadow casting objects as seen from a single point light source. Similar approximations have been used before [Bra02a, Ass03a] and we discuss some limitations of this approximation in Section 6.

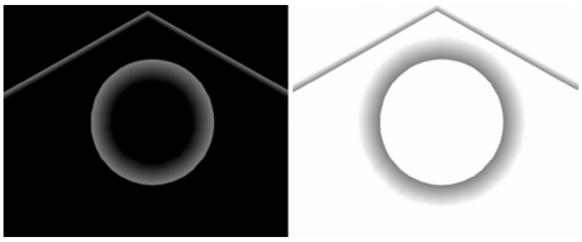

Figure 4. A simple example of inner (left) and outer (right) penumbra maps where the scene consists of a sphere and a ground plate. The inner penumbra map has $v = 0.0$ (black) visibility value adjusted to the outer vertices and $v = 0.5$ to the inner vertices. Respectively, the outer penumbra map has $v = 0.5$ visibility value adjusted to the inner vertices and $v = 1.0$ (white) to the outer vertices.

4.4 Soft shadow visualization

The soft shadow visualization process utilizes the first shadow map, the second shadow map and the penumbra maps for rendering the umbra and penumbra regions.

During the soft shadow rendering pass, each view frustum pixel is transformed into the light space and the first shadow map depth test is performed. If the pixel passes this test, the pixel is lit according to the first shadow map, and the outer penumbra map is thus sampled in order to determine the illumination. In contrast, when the first depth test fails, which indicates that the current pixel is fully in the shadow according to the first shadow map, the depth of the transformed pixel is compared against the second shadow map. The inner penumbra value is used to brighten the illumination when the pixel passes the second shadow test. Otherwise, only the ambient value is added.

By using the second shadow map we are able to guarantee that the soft shadow regions will not penetrate to the consecutive third, fourth, etc. depth order surfaces. This method is not a physically correct way to treat the soft shadow attenuation, but the artifacts seem to be acceptable with relatively small area light sources. The possible limitations of the used sampling technique are discussed in more detail in Section 6.

The visualization process can be further accelerated by bounding the light frustum regions more accurately [Arv03a]. As a result, the number of pixels on which the shadow map test is performed will diminish.

5. Implementation and results

We have implemented the algorithm described in the previous sections using DirectX 9.0. The main objective of the implementation was to verify that the algorithm generates plausible soft shadows at real time frame rates, including inner and outer penumbras, by utilizing only off the shelf graphics hardware. Thus, it will require some future work to compare our method in more detail against other existing soft shadow algorithms.

The tests were run on a 1.4GHz AMD Athlon with an ATI Radeon 9700 Pro graphics card. Both the screen and the floating point render target resolutions were 1024x1024. The floating point format increased the memory bandwidth consumption significantly, but gave more accurate depth values and smoother average visibility transitions. The shadow maps and the penumbra maps were rendered from scratch for every frame in order to simulate a highly dynamic environment. The irradiance due to a point light source as formulated in Section 3 was computed for each pixel according to the Phong model [Pho75a].

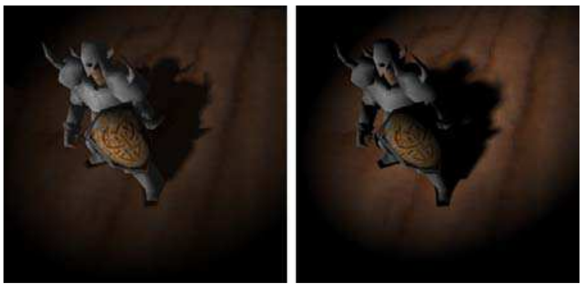

Figure 5. A test scene with non-trivial shadow casting objects (2148 triangles). The left image is generated with penumbra maps while the right image is generated with 128 supersampled shadow map images.

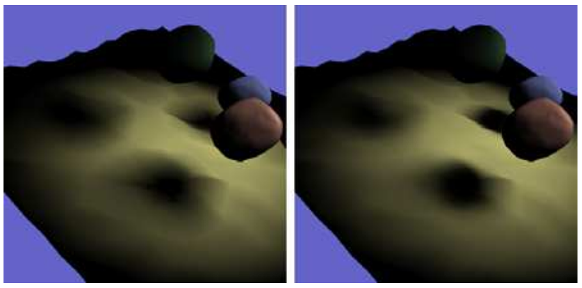

Figure 6. A test scene (5064 triangles) with clouds where shadow casting objects have relatively large inner areas. The left image is generated with penumbra maps and the right image is generated with 512 supersampled shadow map images.

Our implementation first rendered the shadow maps and the penumbra maps. We then rendered the z-buffer from the view frustum before performing the actual soft shadowed rendering. This additional rendering pass reduced the soft shadowing pixel shader commands to the view frustum's visible pixels. The rest of the implementation straightforwardly adapted the above description of the algorithm.

We measured the frame rates for two test scenes using the shadow map multisampling [Hec97a], and our new algorithm. In practice, shadow maps are best suited for local light sources and therefore we chose two scenes that have local light sources. Both test scenes (Figs. 5, 6) ran at approximately 25 frames per second for our algorithm, while the multisampling approach ran slower than one frame per second (128-512 samples). As can be seen from the Fig. 5, the penumbra regions cannot be very large with objects which have relatively small inner areas. Proportionally, the penumbra regions can be relatively large when the shadow casting objects have large inner areas (Fig. 6).

6. Discussion

Here we will discuss the possible artifacts of our algorithm due to approximations used. The artifacts can be classified into three classes: discrete sampling artifacts, object overlap artifacts, and single silhouette artifacts.

The first artifact occurs due to the discrete sampling representation for storing the average visibility values. As long as there is enough render target area for storing the average visibility values, our algorithm is able to produce plausible soft shadows. However, when a shadow casting object is lit by a large area light source and the distance between the occluder and the receiver surface is large, our algorithm may generate aliased soft shadow boundaries because multiple view space pixels are transformed into a single penumbra map pixel.

The second type of artifact may occur when several objects, or several parts of a single concave object overlap in the light space. Our algorithm treats these objects independently and may therefore combine their shadowing contribution incorrectly. This limitation is hard to overcome, because the current graphics hardware is not capable of reading and writing into same render target component during a single rendering pass. Thus, we are only able to perform blending operations for outer penumbra quads before the first shadow map [Cha03a, Wym03a]. Additionally, the inner penumbra quads may sometimes be rendered falsely due to the second shadow map attenuation. This can be the case with an object which has a deep concave corner between the first and the second shadow map. Nevertheless, this can be corrected by subdividing the concave object into convex sub-objects, assigning separate ID values for each sub-object, and storing the back facing polygon IDs of the shadow casting object into the second shadow map. Thus, the inner penumbra quad pixels are only rendered into the inner penumbra map if the ID in the second shadow map equals the ID of the shadow casting penumbra quad. It should be noted that the approximation used here can work flawlessly with relatively small area light sources, and without the second depth limitation the inner penumbra regions could be falsely cast onto the consecutive receiver surfaces. Figure 7 shows an example with object overlap artifacts.

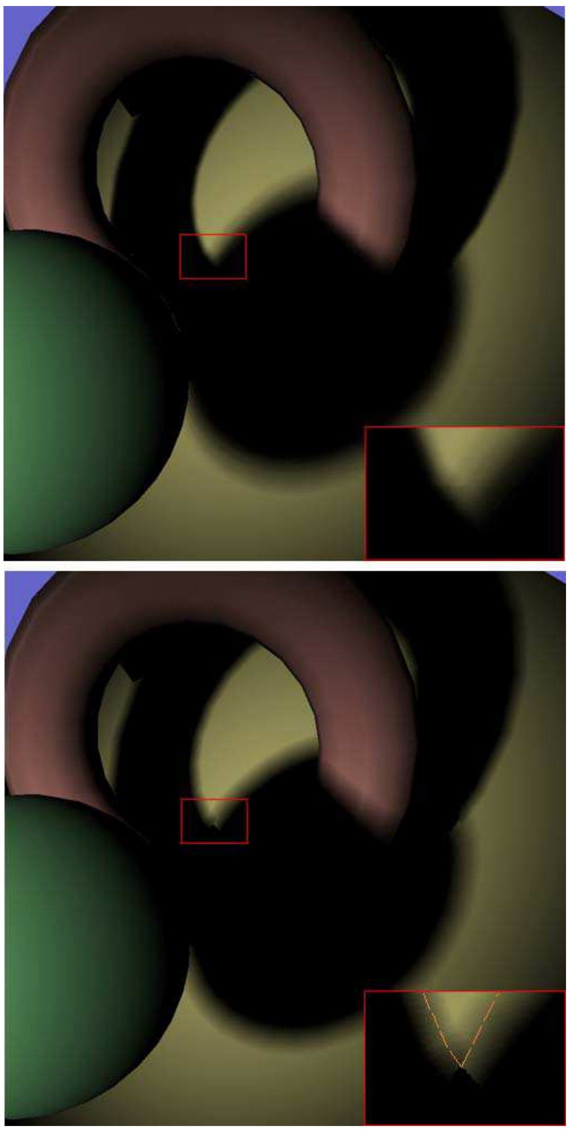
Figure 7. The top image is generated with 1024 supersampled shadow maps and the bottom image is generated with penumbra maps. As can be seen from the magnification, the overlapping shadow boundaries are not physically correct due to second depth attenuation for inner penumbra quads. The dashed line indicates approximate hard shadow boundaries.

Thirdly, every physically correct soft shadow algorithm must consider the change in the silhouette boundary according to sample location. The change at the silhouette boundary is based on object properties and ignoring it may cause artifacts into the shapes of the shadow regions. A relevant discussion of the limitations of single sample point artifacts is given by Assarsson and Akenine-Möller [Ass03a].

7. Conclusions and future work

We have presented a new soft shadow algorithm that is an extension to the standard shadow map algorithm. The shadow penumbra quad is a new shadow generation primitive that we have introduced and the algorithm is able to render approximate soft shadows on arbitrary geometries using these penumbra quads. The presented algorithm has a straightforward implementation on the latest generation of consumer level graphics hardware.

In future work, we want to apply different functions for calculating more realistic average visibility transitions. We also believe that the presented penumbra maps can be divided into multiple layers to mimic the illumination attenuation in a more authentic way. It would also be valuable to compare other existing soft shadowing techniques with ours in more detail.

8. REFERENCES

[Ake02a] Akenine-Möller, T., and Haines, E. Real-Time Rendering, 2nd edition. A.K. Peters Ltd., 2002.

[Agr00a] Agrawala M., Ramamoorthi, R., Heirich, A., and Moll, L. Efficient image-based methods for rendering soft shadows. *In Proceedings of SIGGRAPH '00*, pp. 375-384, 2000.

[Arv03a] Arvo, J., and Aila, T. Optimized Shadow Mapping Using the Stencil Buffer. Journal of Graphics Tools, accepted for publication.

[Ass03a] Assarsson, U., and Akenine-Möller, T. A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware. ACM Transactions on Graphics, vol. 22, no. 3, 2003.

[Bra02a] Brabec, S., Seidel, H-P. Single Sample Soft Shadows Using Depth Maps. *In Proceedings of Graphics Interface*, pp. 219-228, May 2002.

[Bra03a] Brabec, S., Seidel, H-P. Shadow Volumes on Programmable Graphics Hardware. *Eurographics '03* (Computer Graphics Forum), 2003.

[Cha03a] Chan, E., and Durand, F. Rendering fake soft shadows with smoothies. *Eurographics symposium on rendering*, 2003.

[Coh93a] Cohen, M. F., and Wallace, J. R. Radiosity and Realistic Image Synthesis. Academic Press Professional 1993.

[Cro77a] Crow, F. Shadow Algorithms for Computer Graphics. *In Proceedings of SIGGRAPH '77,* pp. 242-248, July, 1977.

[Eng02a] Engel, W. F. (Editor). Direct3D ShaderX: Vertex and Pixel Shader Tips and Tricks. Wordware Publishing, Inc, 2002.

[Goo99a] Gooch, B., Sloan, P-P., Gooch, A., Shriley, P., and Riesenfeld, R. Interactive technical illustration. *ACM Symposium on Interactive 3D Graphics*, pp. 31-38, April 1999.

[Hai01a] Haines, E. Soft planar shadows using plateaus. Journal of Graphics Tools, vol. 6, no. 1, pages 19-27. 2001.

[Has03a] Hasenfratz, J-M., Lapierre, M., Holzschuch, N., and Sillion, F. A survey of real-time soft shadow algorithms. Eurographics state-of-the-art report, 2003.

[Hec97a] Heckbert, P., and Herf, M. Simulating soft shadows with graphics hardware. CMU-CS-97-104, Carnegie Mellon University, 1997.

[Hei00a] Heidrich, W., Brabec, S., and Seidel, H-P. Soft Shadow maps for linear lights. *Eurographics Workshop on Rendering*, pp. 269-280, 2000.

[Kir03a] Kirsch, F., and Doellner J. Real-time soft shadows using a single light source sample. Journal of WSCG, vol. 11. no. 1, 2003.

[Mor00a] Moerin, S. ATI Radeon - HyperZ Tecnology. *SIGGRAPH / Eurographics Graphics Hardware Workshop, Hot3D session,* 2000.

[Nvi02] Nvidia corporation. Order independent transparency white paper. http://developer.nvidia.com\object\order_independent_transparency.html.

[Pho75a] Phong, B-T. Illumination for computer generated pictures. Communications of the ACM, vol. 18, no. 6, pp. 311-317, 1975.

[Ree87a] Reeves, T., Salesin, D., and Cook, R. Rendering Antialiased Shadows with Depth Maps. *In Proceedings of SIGGRAPH '87*, pp. 283-291, July 1987.

[Sol98a] Soler, C., and Sillion, F. Fast Calculation of Soft Shadow Textures Using Convolution. *In Proceedings of SIGGRAPH '98*, pp. 321-332, 1998.

[Wil78a] Williams, L. Casting Curved Shadows on Curved Surfaces. *In Proceedings of SIGGRAPH '78*, pp. 270-274, August, 1978.

[Wym03a] Wyman, C., and Hansen, C. Penumbra maps. *Eurographics symposium on rendering*, 2003

[Yin02a] Ying Z., Tang M., and Dong J. Soft shadows maps for area light sources. In 10th Pacific Conference on Computer Graphics and Applications, pp. 442-443. 2002