GPU friendly, anti-aliased, soft shadow mapping

Michal Valient Tomas Bujnak
valient@sccg.sk tomasb@dataexpert.sk
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Abstract

In this paper, we present an algorithm that renders anti-aliased, soft edged shadows using a modified shadow-mapping approach. Our algorithm adds the silhouette information to the shadow map, thus allowing rendering precise and continuous shadow boundaries. Soft shadows for small spherical light sources are rendered using a variation of the *percentage closer filtering algorithm* [Reeves 1987]. The critical improvement of our algorithm over previous approaches is that it runs completely on the GPU using only two passes. This includes the silhouette detection step which is done completely in the vertex and pixel shaders using specially generated version of the mesh.

CR Categories: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism – *Color, shading, shadowing, and texture, visible line/surface algorithms.*

Keywords: Rendering, Soft shadows, Graphics Hardware, Silhouette detection.

1. Introduction and previous work

Interactive applications, such as games or interactive walkthroughs, add shadows to increase the realism of the scene. The common requirements of these applications are visually plausible (not necessarily correct) shadows that are generated using only minimal resources to allow other processing such as physics, application logic or audio. Following two most popular techniques emerged over the time.

The shadow volumes approach introduced by Crow [1977] uses the geometry to produce exact volume of the shadow and renders alias-free boundaries. This technique requires quite high fillrate and has some special cases that have to be handled to make it robust [McGuire et al. 2003].

Shadow mapping algorithm introduced by Williams [1978] brings a completely different approach as it uses a texture filled with depth values in light space to compute shadows. This method is generally faster but introduces aliasing problems related to precision and resolution of the depth texture. Several approaches were proposed to help fight mainly the resolution problems. Sen et al. [2003] and Bujnak [2004] add silhouette point information to the shadow map to improve the precision. *Percentage closer filtering* (PCF) [Reeves 1987] allows filtering of shadow test results to produce soft shadows.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SCCG'05, May 12–14, 2005, Budmerice, Slovakia.
Copyright Comenius University

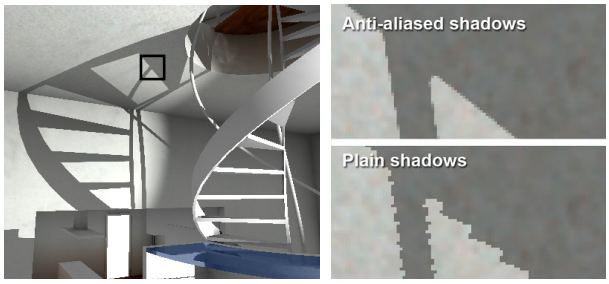


Figure 1 Left image shows hard edged shadows generated using our method. Rectangle marker specifies the comparison area shown in the right image. Right image shows magnified quality comparison of plain shadow mapping and our method. Please note that the eye position is slightly altered.

Many shadow rendering approaches use the silhouette detection. It can be performed in the image space like in recent soft shadow algorithms [Chan et al. 2004, Valient and Boer 2004], but this method is hard to apply when additional silhouette information is required. Currently most applications detect silhouettes using original geometry and the CPU, which can be slow in case of highly detailed meshes. McGuire et al. [2004] proposed a hardware based approach, which similarly to our algorithm, uses additional mesh for the edge detection and runs on GPU. Their implementation uses a more complicated mesh structure with greater memory consumption, but requires less per-pixel processing than our approach.

We propose a modification of shadow mapping algorithm. In the first pass, we extract silhouettes from the scene geometry and we store the distance to the nearest silhouette edge in the shadow map along with standard depth value. In the second pass, we take advantage of the bilinear filtering of textures on graphics hardware and extract the shadow boundary from the distance information as well as the soft shadow gradient. We use four samples and the variation of the percentage closer filtering to produce visually plausible soft shadows. The advantage of our algorithm is that it runs completely on GPU and offloads CPU to other tasks. Figure 1 shows quality of our approach.

2. New algorithm

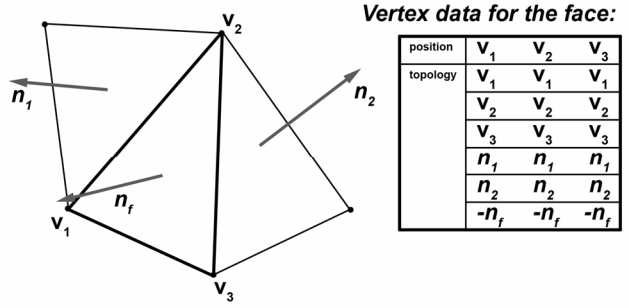
Like the standard shadow mapping algorithm, our approach consist of two stages. In the first stage, scene is rendered from a point of view of the light and depth values as well as silhouette information are stored. In the second stage, scene is rendered from the point of view of the viewer and previously created shadow map is used in the actual shadow computation.

2.1 Shadow map generation

Shadow map is generated in two steps. The first step renders standard shadow map depth values [Williams 1978] and the

second step generates the silhouette information. Both results are stored in the depth map texture.

Current graphical hardware does not provide topology information in the vertex shader because the shader is allowed to read data only for one vertex. We use a new mesh structure called *topology aware mesh (TAM)* where we actually store topology information into data of each vertex in the vertex stream. Each item of the vertex stream represents one vertex and three successive items represent one face. We construct each TAM vertex to contain seven fields of data with information about the entire face and three adjacent faces. One field is filled with the vertex position. Three fields are filled with position of all three vertices of the corresponding face. The last three fields are filled with normals of the adjacent faces. We store the negative value of the face normal for each triangle edge which does not have a neighbor. This ensures that such edge is always considered a part of the silhouette during the computation. The TAM face structure is shown on Figure 2. The first position field is used to produce screen space coordinates during the rendering. The information about normals is used to identify silhouette edges in the vertex shader and the three position fields are used to draw correct edges during the rendering.



Vertex data for the face:

position	V_1	V_2	V_3
topology	V_1	V_1	V_1
	V_2	V_2	V_2
	V_3	V_3	V_3
	n_1	n_1	n_1
	n_2	n_2	n_2
	$-n_f$	$-n_f$	$-n_f$

Figure 2: Triangle $V_1V_2V_3$ with normal n_f has two adjacent faces with normals n_1, n_2. The table shows part of TAM vertex stream for this triangle. Columns represent three items of the vertex stream (vertices) and rows represent the information stored in each item. Negated face normal $-nf$ is used for the edge V_1V_3 that is without a neighbor face.

McGuire and Hughes [2004] introduced a structure called *edge mesh* where they added 4 additional vertices for each edge. Using half-precision in the vertex data, this mesh consumes about 9 times the storage of the original mesh for typical closed meshes. In the same conditions, our *topology aware mesh* introduces only 3 vertices per face. Each one contains 48 bytes of information using half precision and TAM consumes only 1.53 times of memory of the original mesh. Using the stream frequency feature of current hardware we can lower the memory consumption to 0.77 times of memory of the original mesh.

2.1.1 Silhouette rendering

The vertex shader first detects the silhouette edges of the face. It computes the normal vector of current face n_F using the vertex position fields and computes the orientation to the eye vector v_{EYE}. For each edge, it also computes the orientation of adjacent face using normal vector of adjacent face n_E and compares it to the orientation of the current face. Edge of the triangle is marked as a part of the silhouette, only if the following condition is met:

$$\text{sign}(n_F \bullet v_{EYE}) \neq \text{sign}(n_E \bullet v_{EYE}). \quad (1)$$

Triangle that does not contain silhouette edge is placed in front of the near clipping plane and collapsed into a single point – thus it is automatically rejected by the graphics card before entering the costly rasterization process.

The key feature of our algorithm is that it does not introduce new faces. Therefore, the vertex shader has to scale the screen representation of each face to contain all required silhouette features after the rendering. All following operations are performed in the post-perspective projection shadow map space.

We find the center of the incircle $v_{incenter}$ for the face and radius of the incircle w_O. We identify the minimum silhouette width w_S, which is chosen conservatively to be always bigger than the expected width of the silhouette (specified as a rendering parameter). We compute the w_S by add two times the length of the pixel's diagonal to the expected width. The reasons for such approach are the GPU rasterization rules. GPU renders a pixel only if the center of the pixel lies inside the triangle. By adding two times the length of the pixel's diagonal to the expected silhouette width we make sure that the centers of pixels at the silhouette borders are inside the triangle and they are properly rasterized. The precise shape of the silhouette is computed later in the pixel shader. We define w_{SC} as a half of the w_S. Equation 2 is used to compute the factor required to scale the triangle so that each silhouette is completely covered.

$$f_{SCALE} = (w_O + w_{SC})/w_O \quad (2)$$

We scale the triangle using the scaling center in the $v_{incenter}$ and the scale factor f_{SCALE}. The process is illustrated in the Figure 3.

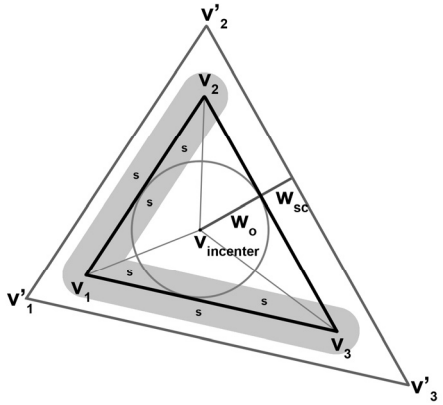


Figure 3: Scaling of the triangle in the vertex shader. Triangle $V_1V_2V_3$ represents the original triangle. Triangle $V'_1V'_2V'_3$ represents triangle scaled using center in $v_{incenter}$ and scale factor f_{SCALE}. Grey area marked with *s* represents the final silhouette.

Since the incenter computation and scaling is performed in the image space we have to deal with the situation when the triangle degenerates after the perspective projection. Such triangle would not be rendered at all because it is not expanded by our algorithm. Vertex shader solves this case by moving one vertex of such triangle slightly in the reverse direction of the normal that belongs to the opposite face edge. Our implementation performs this correction step for all triangles, where inradius w_O is smaller than

the half of the diagonal of the pixel. The correction step is performed before the scaling and incenter computation.

The main purpose of the pixel shader is to clip all pixels that do not belong to the silhouette. The nearest point on silhouette is found for each pixel. Pixels that do not belong to thick silhouette are clipped. Signed distance to the silhouette point and its depth are rendered for pixels that belong to the thick silhouette. Pixels inside the triangle have positive distance and pixels outside the triangle have negative distance. Later in the rendering we use bilinear texture filtering capabilities of the hardware to quickly render precise shadow boundaries [Bujnak 2004].

We use depth test and alpha test to blend the capsules correctly together. Alpha and depth test are set to LESSEQUAL mode and only pixels with the depth less or equal to the current depth and minimal distance to the silhouette edge are rendered.

2.2 Final rendering

First, to speed up the shadowing determination process, we need to offset shadow boundary as done in [Bujnak 2004], so that rendered shadow boundary does not interfere with boundary rendered using standard shadow map, because shadow mapping is used to shade non boundary parts of shadow. With introduction of this offset, we can render smooth shadow boundaries by adding one simple texture read: we determine whether to use edge distance for shadowing or whether the point is fully lit or fully occluded based on the edge and occluder depths. In most cases the introduced error is invisible to the user and becomes noticeable only on objects casting thin shadows.

Shadow determination works similar to normal shadow mapping. We use additional information we have previously encoded into shadow map to create the anti-aliased shadow boundaries. To determine, whether a point in the scene is in shadow, we first project it to the shadow map space. Point's depth is first compared with the shadow map depth. If the shadow map depth is smaller than point's depth, point is declared to be in a shadow. Otherwise we compare point's depth to silhouette edge depth. If silhouette edge is behind the tested point, the point is fully lit. Otherwise we use a bilinearly filtered edge distance to determine shadowing by thresholding.

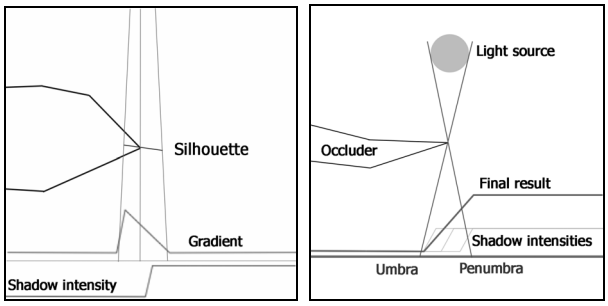


Figure 4: Shadows intensity is determined from depths and silhouette distance using texture lookup (left). Soft shadows computation for small spherical light source (right).

The whole process can be programmed very efficiently using a single lookup to gradient 1D texture (Figure 4 left). The texture coordinate for this lookup is computed using Equation 4 where d_E is distance to the edge, z_O is the occluder depth, z is the depth of current pixel and z_E is the depth of the edge. All depths are specified in the light space and we treat results of the boolean expressions as 0.0 for false and 1.0 for true.

$$coord = \left[\frac{1}{4} \bullet \left(d_E + \left(z_O < z\right)\right) + \frac{1}{2} \bullet \left(z_E < z\right)\right] \quad (4)$$

The whole shadowing determination can be done in four arithmetic instructions. By changing content of 1D texture we can adjust softness of the shadow boundary.

2.2.1 Simulating soft shadows

Shadow mapping technique described previously can be easily extended to approximate shadows cast by small spherical light sources. This can be achieved by sampling more samples and by changing 1D lookup texture to 2D. As second texture coordinate we use scaled ratio between edge depth and tested point depth. For bigger light sources, we also need to scale silhouette gradients. Because edge softness can be controlled by lookup texture, number of samples can be very low. For every ratio, we store different gradient in lookup texture. This lookup texture is dependent on the light source radius.

Sampling the extended shadow map once can render only shadows that are slightly bigger than should be. Therefore we use additional 3 samples to sample surrounding of shaded pixel. For each of four samples, we compute shadowing similarly to hard shadow method. We fill second texture coordinate for texture lookup scaled ratio to soft hard shadow algorithm. Figure 4 right visualizes the method. Three visible shadow intensities are summed together into final shadow intensity. The image shows linear penumbra – which is not correct. The almost correct cosine like result can be achieved by adjusting lookup textures.

Taking four samples generates more visually plausible umbra and penumbra than using only single sample and also reduces artifacts caused by touched silhouettes. It is possible to use even more samples. We found out, that using 4 samples is a good compromise between performance and shadow quality.

3. Results and discussion

We discovered that our implementation is about 5 times slower than the plain shadow algorithm while maintaining visual appearance similar to shadow volumes (Figure 5).

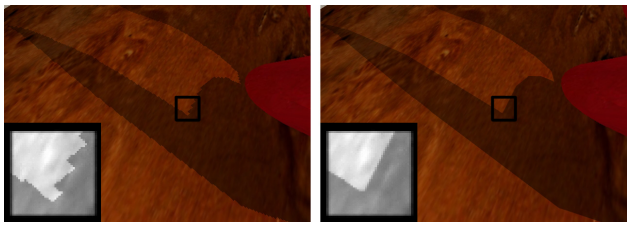


Figure 5: Quality comparison of plain shadow mapping (left) and our hard shadow approach (right). Small rectangles shows zoomed-in parts of the image (contrast and brightness enhanced)

We've tested our implementation of the algorithm on the Pentium 4 2.4GHz processor with 1GB RAM. The testing card was ATI Radeon X800XT Platinum edition. Most of the time is spent during the silhouette rendering. The main reasons are the

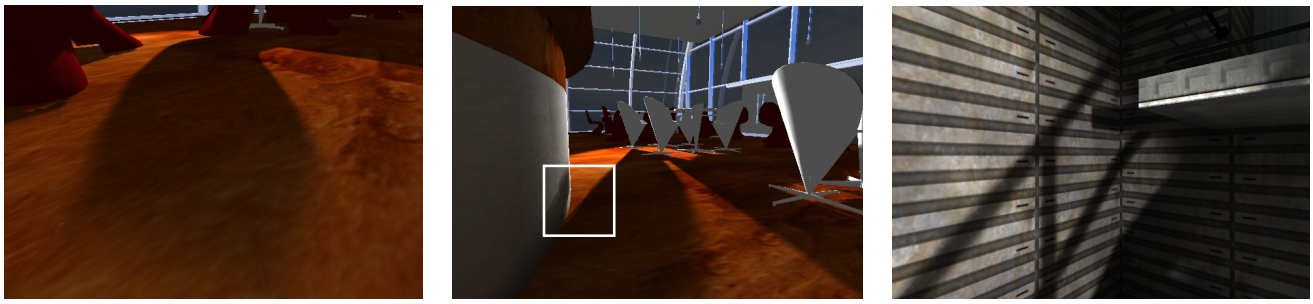


Figure 6: Soft shadow version of our algorithm. First two images show restaurant scene. Third image shows part of a game scene. Middle picture shows soft shadow errors where silhouettes touch (the error is visible inside the white rectangle).

complicated vertex and pixel shaders. Pixel shader also uses *texkill* and custom depth value writing and therefore the GPU cannot use the early z-cull technique and clip the pixel before the shader is executed. Our algorithm is however still very fast. We obtained average framerates of 150 FPS for scenes with 30000 polygons and our approach is therefore suitable for frame-rate intensive applications such as games or interactive walkthroughs.

The number of TAM vertices can be lowered for meshes that would not be deformed. In these cases we are able to identify and remove such triangles and all segments that cannot contribute to the silhouette can be removed from TAM.

Our algorithm can be adapted for animated meshes (i.e. matrix skinning) but this information has to be specified for each vertex of the TAM. This would of course slightly increase the memory requirements of the mesh and the cost of vertex processing of such vertex. We plan to work in this area in the future as well as we plan to optimize the silhouette rendering shader not to use the depth writes for better performance.

The soft shadow extension of our algorithm produces visually plausible results for most situations, but as can be seen on Figure 6 center, it also produces errors in the areas where silhouettes cross or touch. The errors can be lowered using more samples or using alternative projections when generating the shadow map to produce more details in these areas. We actively work on improvements of the soft shadows.

4. Conclusion

We have presented an algorithm that produces anti-aliased hard or soft shadows using only two GPU passes. It uses single preprocessing CPU pass and then runs completely on the graphics hardware.

There are some drawbacks of our approach. It adds additional mesh structure that requires about 0.77 times of the original mesh memory usage and additional vertex and pixel processing that is however well handled using latest generation of graphics cards. Final rendering also adds additional processing cost because it uses multiple shadow map reads in the shader which is considered as one of the most expensive operation in the shaders because of the high latency.

5. Acknowledgements

We would like to thank Andrej Ferko and Pavol Elias for their valuable comments and corrections. This research was supported in part by the VEGA grant reg.no.1/0174/03.

6. References

BUJNAK, T. *Extended Shadow Maps.* Central European Seminar on Computer Graphics 2004 Proceedings, pp. 167-173, 2004.

CHAN, E. AND DURAND, F. *Rendering fake soft shadows with smoothies*, Proceedings of Eurographics Symposium on Rendering, 193-194, 2003.

CROW, F., *Shadows Algorithms for Computers Graphics. Computer Graphics*, Vol. 11, No.3, Proceedings of SIGGRAPH 1977, July 1977.

MCGUIRE, M., HUGHES, J.,F. *Hardware-determined feature edges.* In Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering, 35-147.

MCGUIRE, M., HUGHES, J. F., EGAN, K., KILGARD, M. J., AND EVERITT, C. *Fast, Practical and Robust Shadows*. Brown Univ. Tech. Report. October 27, 2003.

REEVES, W., T., SALESIN, D., H., COOK, R., L. *Rendering antialiased shadows with depth maps*. In Computer Graphics (SIGGRAPH '87 Proceedings), pages 283–291, July 1987.

SEN, P., CAMMARANO, M., HANRAHAN, P. *Shadow Silhouette Maps.* ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH), vol. 22, no. 3, pp. 521-526, July 2003

VALIENT, M., AND W.H. DE BOER. *Fractional-Disk Soft Shadows* in W.F. Engel, ed., *ShaderX3: Advanced Rendering with DirectX and OpenGL*, Charles River Media, pp. , Hingham, MA. Oct 2004.

WILLIAMS, L., *Casting curved shadows on curved surfaces*. Computer Graphics (Proc. of SIGGRAPH 78), 12(3):270–274, 1978.