Extended shadow maps

Tomas Bujnak[1]
Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava / Slovakia

Abstract

The most popular methods for interactive hard shadows rendering are *shadow volumes* and *shadow maps.* Shadow volumes generate precise shadows but require high fill rate. Due to excessive fill rate requirements, shadow maps are probably the most widely used means for generation of shadows, despite their well known aliasing problems. In this paper, Extended Shadow Maps are introduced, as a means to reduce required shadow map resolution and enhancing visual quality of shadows by encoding additional information about silhouette edges into the shadow map by drawing quads for silhouette preservation. Recent algorithms using this technique were not suitable for computer games due to their hardware requirements, because in this types of applications, high frame rate is required. Our algorithm, Extended Shadow Maps creates anti-aliased shadows using fast fixed-point pixel shaders version 1.4. For shadowing detection only 4 arithmetic instruction slots and 3 texture pipes are used. Also, all geometry required to render anti-aliased hard shadows using our new method can be stored in static vertex and index buffers on graphics accelerator. Additionally, using one dp3 instruction, arbitrary attenuation function can be implemented.

Keywords: Shadows, Shadow Volumes, Shadow Maps, Real-time Shadows

1 Introduction

Shadows play a vital role in our space perception [1]. Without them, scenes often look unnatural and flat; and relative depths of objects in a scene can be very unclear. Shadows are indeed essential in creating visually realistic images, but they are also computationally intensive and hard to generate.

In section 2 we present some algorithms related to our work, in section 3 we introduce Extended Shadow Map algorithm, section 4 presents our implementation, section 5 gives results of this method and section 6 is discussing about limitations and advantages of our approach.

2 Previous work

Numerous different methods and algorithms have been developed for generating realistic looking shadows using acceptable amount of resources.
Review of many older shadowing techniques can be found in the survey published by Woo et al.[2] and in Watt and Watt's book [3].

Unfortunately not all techniques are suitable for hardware rasterizers. Two most common techniques for hardware accelerated complex shadowing are stenciled shadow volumes and shadow mapping for spotlights. Shadow mapping can be extended to omni directional lights using multiple passes.

2.1 Shadow volumes

Shadow volumes are polyhedral regions that form shadow of an object. They are formed by extruding the silhouette edges away from the position of light source. This auxiliary shadow volume geometry is then rendered along with the original scene. 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 it is in shadow or it is lit. Shadow volume methods have the advantage of generating precise shadow boundaries. Unfortunately, the pixel fill rate requirements for rendering shadow volumes remain substantial. Shadow volumes have been recently extended to soft-shadows [4][5]. Although this method requires excessive fill rates, it is widely supported by PC-market graphics accelerators and benefits from highly optimized rendering pipelines.

2.2 Shadow Maps

Shadow mapping was introduced by Williams [6]. Shadow maps require an extra rendering pass to generate a depth image from a 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, point position is projected into light space and tested against the corresponding shadow map depth sample. The point is in shadow if it is further from the light than the stored depth value.

Since it is an image-space technique, if offers advantages in terms of generality, speed and ease of

[1]tomas.bujnak@post.sk

implementation. Unfortunately, like any discrete image-based technique, shadow maps suffer from aliasing problems. Aliasing occurs, when the shadow map resolution used is insufficient for the rendered image. Removing aliasing in shadow maps has been a difficult problem to resolve. Percentage-closer filtering[7] is one of the solutions, but in cases of severe aliasing it can only mask the aliasing by blurring. Deep shadow maps[7] address aliasing by using jittered samples and pre-filtering them. Silhouette Maps[10] uses additional information encoded into silhouette map to reconstruct shadow edges.

Silhouette map is a texture, whose texels represent the coordinates of points that lie on the silhouettes of objects as seen from the light source. The purpose of the silhouette map is to provide information on the location of the shadow boundary so that it can be faithfully reconstructed. The shadow boundary can be approximated by a series of line segments. The piecewise linear contour is reconstructed during the second stage of the algorithm by connecting adjacent points. Dual contouring algorithm is used to reconstruct shadow boundaries from silhouette map during the rendering. Unfortunately implementing this technique in hardware using pixel shaders requires approximately 46 instructions per pixel, and 60 instructions per shadow map fragment.

Recently, Eric Chan and Frédo Durand [11] independently developed a soft shadow algorithm that is using similar principles as ours. They combine shadow maps with geometric primitives called smoothies to render fake soft shadows. We compare their approach to ours in Section 6.3.

3 Extended Shadow Maps

Standard shadow maps perform well in most areas of the image, but suffer from aliasing artifacts near shadow boundaries. Similarly as in Silhouette map algorithm, standard shadow map is augmented with additional information, which is then used to reconstruct shadow boundaries.

Like standard shadow map algorithm, extended shadow map has two stages. The first pass of algorithm uses the light source as a view point and shadow depth map and additional information about silhouettes are generated. Silhouette information is encoded inside shadow map using one or two quads per silhouette edge, depending on the local curvature. Shadow depth map and silhouette information is encoded into the same 32bit texture. In the second stage, scene is rendered from the viewer's perspective and shadow determination is performed. Extended Shadow Map is then used to improve the quality of shadow near the shadow boundaries. This is done differently as in [10] [11]. The main criteria for selecting edge representation was, that the algorithm should be as fast as possible, it should use as little as possible texture fetches and pixel shader arithmetic instructions and the required pixel shader version should be as small as possible. Next, reconstructed shadow need not to be exact, but should look well.

3.1 Silhouette representation

Silhouette edge representation uses the observation, that bilinear interpolation of coplanar vertices forms a planar surface. We can look at a texture as a height field, where actual texel values represents heights. When four neighboring texels are coplanar, then bilinear interpolation of these texels will form planar surface. So if we store signed distances to silhouette edge to the near texture fragments, edge can be represented by isosurface of bilinear interpolation of these values, i.e

$$f(u,v) = (u,v)\begin{pmatrix} A_{i,j} & A_{i,j+1} \\ A_{i+1,j} & A_{i+1,j+1} \end{pmatrix}\begin{pmatrix} 1-u \\ 1-v \end{pmatrix}$$

(*i,j* corresponds to *u,v*), so if *f(u,v) = 0, u,v* will be silhouette boundary. Unfortunately, Z-buffer removes any fragments rendered inside objects and original shadow map algorithm creates shadow that is due to discrete sampling sometimes 0.707 pixels wider. This can be overcome using more passes and 2.0 pixel shader, which would reduce performance to level that is not acceptable for computer games. Instead of this, we use near clip plane manipulation to perform long-distance Z-biasing, and actual shadow is extended 0.707 pixels in shadow map space. Z-biasing alone is not always working properly, because some fragments are still not rendered and some hidden silhouette edges can emerge. To minimize artifacts of this method, hidden silhouette edges should not be rendered. For fast determination, if fragment being rendered is behind the edge, silhouette depths are also stored into the extended shadow map.

To generate silhouette information, the silhouette edges of the geometry must be identified using any of the techniques for shadow volumes. Only silhouette edges formed from concave facets need to be processed, because the other will not smooth any shadow boundary. From every edge one or two quads are created according to local curvature. This can be preprocessed, and the whole silhouette determination and quads construction can be done in graphics hardware using programmable vertex pipeline.

3.2 Shadow determination

To determine, if a point in the scene is in shadow, it is first projected into the light space. Its depth is then compared with shadow map depth sample and silhouette edge depth sample. This depths are point filtered. Results of this comparisons are treated as flags and are encoded together with bilinearly interpolated silhouette edge distance into *u* coordinate of a texture. *v* coordinate is left unused, it can be used to encode arbitrary attenuation function. Then a dependent lookup is performed to

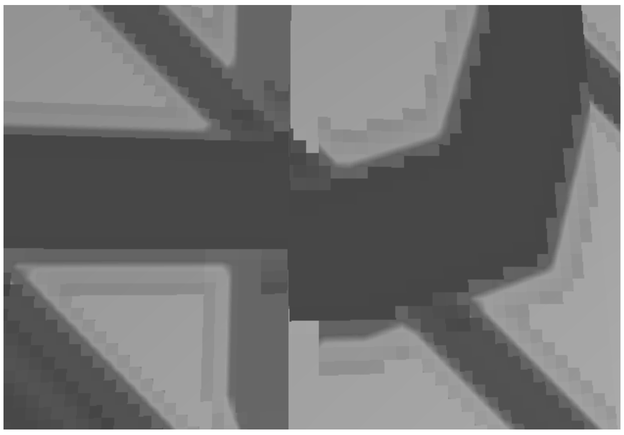
Figure 1: Shadow boundaries reconstruction from extended shadow map. Shadow boundary is shifted 0.707 pixels in shadow map space from its orginal position

transform flags, edge distance and attenuation factor to light intensity.

This technique is well suited for hardware. Small changes in hardware can increase rendering performance (fetching filtered and not filtered texel from the same texture), but with standard pixel shaders version 1.4 we are able to get real-time performance for today computer games environments.

Figure 1 visualizes reconstruction of anti-aliased shadow boundaries from extended shadow map.

4 Implementation

We implemented this algorithm on the ATI Radeon 9600 Pro using programmable pipeline of the Microsoft Direct X 8.1 API. We have implemented version for both point lights and spot lights. Point light were rendered using 6 shadow maps together with stenciled shadow volume to mask geometry outside shadow map being rendered. Also, static lightmap pass was used in our implementation, but lightmaps were only used to add ambient factor to rendered scenes. Three rendering passes were needed per single shadow map, one to create conventional shadow depth map, one to render silhouette quads, and one to render scene and evaluate shadowing for each image fragment.

The first pass, rendering a depth map of the scene from the light point of view proceeds as in conventional shadow maps, but instead of using Z-buffer values, fragment depth is stored into green color channel. Also, 0.75 is stored into edge distance channel (red) and *maximal distance* is stored into edge depth channel (blue). Alpha is used to mask non-silhouette depths, it is initialized to zero. Although this is possible to be done using fixed function rendering pipeline, small pixel shader was used, in order to minimize switches between legacy and programmable pipelines. Fragment program uses one transformation of texture coordinates into color values instruction and one arithmetic slot for *move* instruction to store these values to output register. Alpha channel is initialized in parallel to zero from pixel shader constant. Figure 2 shows extended shadow map using 32 bit rendering, 8 bits per color channel.

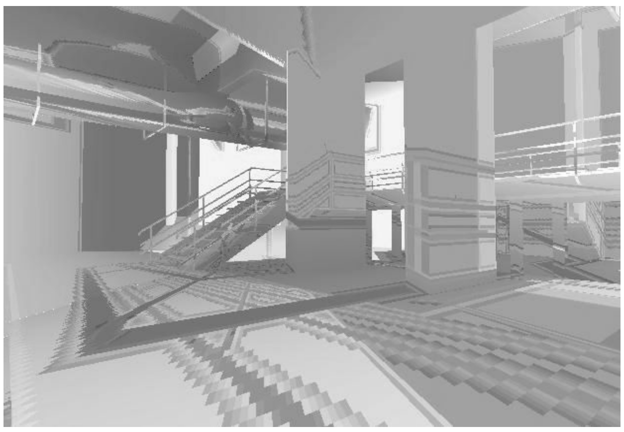
Figure 2: Extended shadow map projected to scene.

4.1 Rendering silhouette quads

Silhouette quadrilaterals are used to compute signed distances from silhouette edges in the shadow map space. Silhouette distances are transformed through texture, because using only linear distances was not sufficient. The size of the quadrilateral is chosen to guarantee that the fragment is generated for every texel that would be needed to correctly interpolate edge distances.

Due to shadow shift, no fragments inside shadow occluder need to be generated, but doing so reduces visual defects of this method in cases, where two or more silhouette edges crosses. Because we cannot guarantee that some fragments generated inside occluder will not be removed during depth test, some distortions can arise. For these texels only silhouette quad depth and alpha are changed. Mentioned distortion is not critical, since shadow edges are shifted 0.707 pixels away.

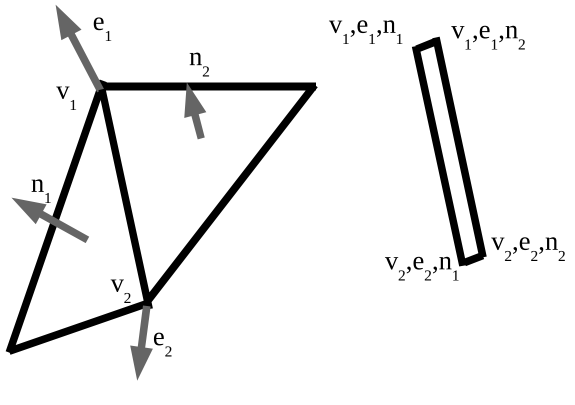

Figure 3: (Left) two triangles sharing potential silhouette edge e_1e_2, (Right) created degenerated quad

All potential silhouette quads are encoded as degenerated quads. Every vertex consists of an position, plane equation and extrusion vector as shown in Figure 3. For silhouette edges detection on graphics hardware, a

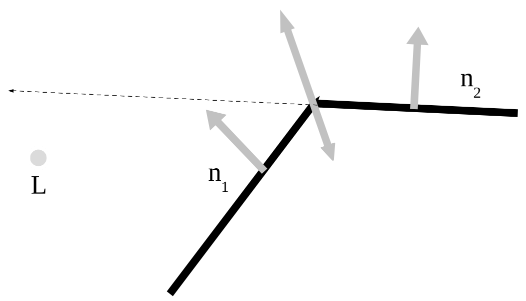

Figure 4: Light source L is behind plane n_2, appropriate vertexes are shifted against extrusion vector, L is in front of plane n_1, appropriate vertexes are shifted according extrusion vector,resulting quad is not degenerated.

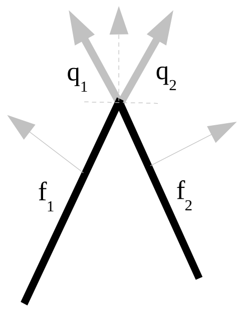

Figure 5: Angle between facets f_1 and f_2 is less than 90 degrees, two degenerated quads (extrusion vectors are q_1 and q_2) are generated for their common edge. Degenerated quad is dotted.

similar algorithm as for shadow volumes algorithm is used.

To draw a quad, we simply offset the vertices on either side of the line by a extrusion vector. Length of extrusion is such, that after projection to shadow map, all texels that would be needed are processed. Extrusion length is evaluated from vertex *w* multiplied with a constant. Direction of extrusion is specified according the position of light from given plane equation. If the extrusion vector, if it is behind, it is moved in opposite direction half distance. These shifts automatically generate non-degenerated quads only from silhouette edges, because a valid quad is only generated iff the light is in front of one plane and behind the other. See Figure 4 for details.

To minimize complexity of vertex shader, generated silhouette quadrilaterals are not rotated to be parallel to screen space. Instead, if mutual angle of corresponding facets is less than 90 degrees, two silhouette quads are generated for single edge. Imaginary degenerated facet is created with plane equation set to beveling plane formed from sum of corresponding plane equations. Both quads are now created as for normal case from degenerated quad and corresponding facet. See Figure 5 for details. We preferred this approach for two reasons: First, there are not many facets meeting this conditions and quad overhead is much smaller than performance loss caused by more complex shader. Second, introduced error was small to notice. In detail, maximal error we introduced with this strategy is less than 30%. Worst case happens if angle between two facets is exactly 90 degrees, and light is just above one facet. Angle between light direction and quad normal is 45 degrees. In this worst case, length of quadrilateral is shrinked 70.7%.

To minimize errors caused when two or more silhouette quads overlaps, we move visual defect to the edge which is closer to light, because near light there is higher shadow map sampling density, so error is minimized in this way. This is achieved with setting D3DRS_BLENDOP to D3DBLENDOP_MAX, D3DRS_SRCBLEND to D3DBLEND_ONE and D3DRS_DESTBLEND to D3DBLEND_DESTALPHA.

Silhouette quads are rendered with alpha set to one. Alpha is used to ignore depths and distances generated for non-silhouette texels. In the case, where more silhouette quads overlap, maximal depth and minimal edge distance (maximal value) is chosen. Edge distance and edge depth values generated for non-silhouette texels are masked using alpha blending. This is accomplished with modulating back buffer with alpha, which is for these texels set to zero. For the first edge rendered to single pixel, alpha is set to zero, so MAX blending function with these settings only passes source values. When more edges are rendered to single pixel, alpha is set to one from previous quad, and MAX function returns maximal depth and maximal inverted distance.

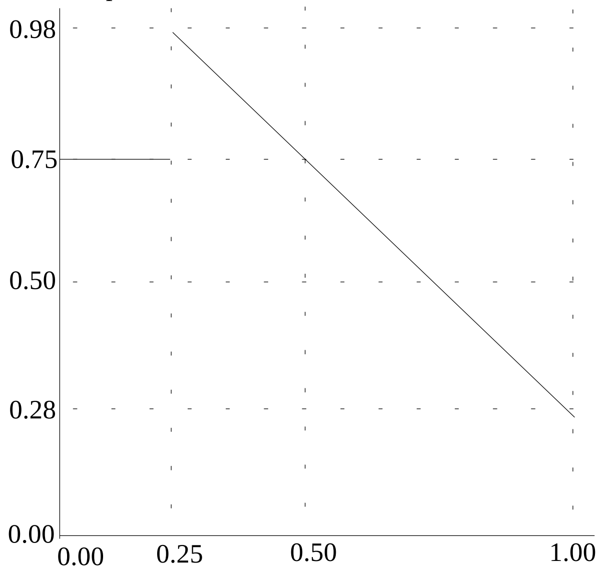

Figure 6: Transformation texture that transforms linear edge distances.

In case, when combination of this blending operation together with these blending factors are not supported, silhouette rendering can be done in two passes, in the first pass, silhouette quads are rendered only to overwrite values rendered in non-silhouette texels, then second pass

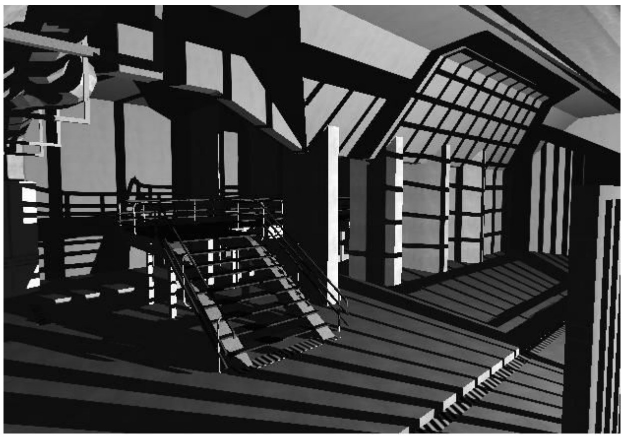

Figure 7: Image rendered with Extended Shadow Maps at 62fps, 4.2MB texture memory in 8 shadow maps was used for whole scene. Scene contained 40.000 triangles. Per pixel attenuation was used.

is used to select maximal values for depth and edge distances using MAX blending.

4.2 Encoding edge distances

Distances from silhouette edges cannot be encoded to ESM directly as linear gradients. To reduce some artifacts that can arise espetialy when shadow map resolution is low and more silhouette edges crosses, there must be *guard zone* on in-occluder side of silhouette edge. One dimensional texture is used to encode values with quard zone. This transformation texture is shown on figure 6. Guard zone is part of the texture between zero and 0.25. Without this section, end of one silhouette quad can repulse shadow from another quad below it. As a consequence, small lit holes between shadow created from opaque triangles and shadow created from silhouette quads will emerge (Figure 9).

Silhouette quads are scaled so, that edge passes through 0.33, and 0.5 is mapped 0.707 pixels away from edge.

5 Results

All of the images presented in this section and in the accompanying video were generated at resolution of 640x480 on a AthlonXP 2800+ system with an ATI Radeon 9600 PRO graphics card. Scene was lit with point light, per-pixel bump mapping and per-pixel linear attenuation was used.

Figure 7 shows typical in-door game scene of a factory with about 11300 triangles lit by one point light, materials contain diffuse and bump map. Only diffuse lighting was used. With our method, we have achieved 88fps with shadow map updated every frame, 135 frames per second was achieved without updating shadow map information. Second frame rate is important, because for computer games typical case, only a small fraction of lights need to be updated. Also shadow need not to be updated more than 25-30 times per second.

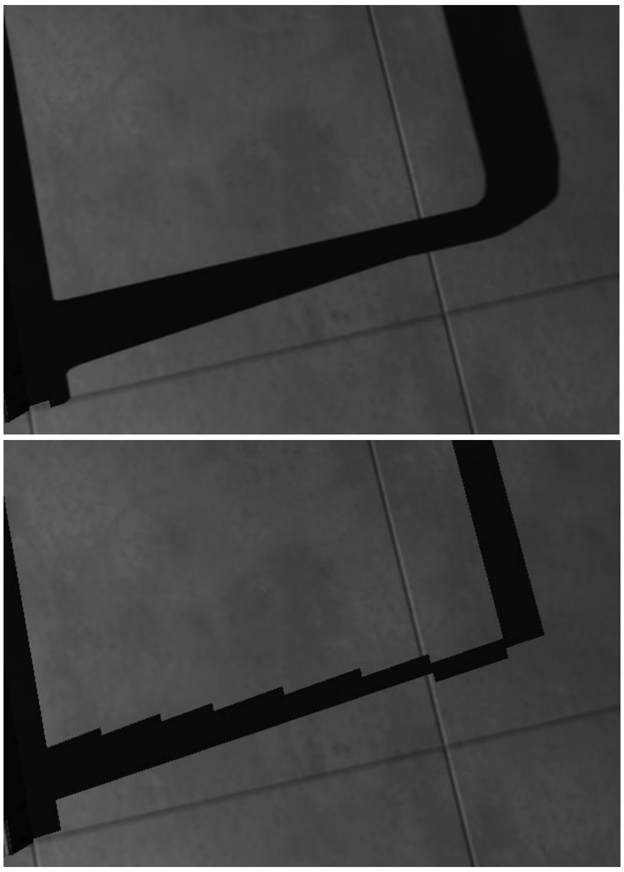

Figure 8: Image rendered with Extended Shadow Maps (top), same scene with an ordinary shadow maps (bottom). Shadow map resolution for both images was the same.

Presented frame rate can be even higher. Our implementation was not optimized, the number of Direct3D API function calls is much higher than recommended by graphics chips manufacturers (approximately ten times higher).

Figure 8 shows comparison of our method to an ordinary shadow map algorithm. Shadow map resolutions were the same for both images. Ordinary shadow map algorithm was not implemented, scene was rendered with the ESM, but with rendering of silhouette quads disabled.

Our implementation used for surfaces having diffuse and bump map 15 pixel shader instructions (6 texture fetches, 2 transformation of texture coordinates to colors and 7 arithmetic instruction slots). This amount is still high, but is significantly smaller than in existing methods. Without diffuse and bump texture, only 9 instructions were required and frame rate without extended shadow map regeneration was approximately 70% higher.

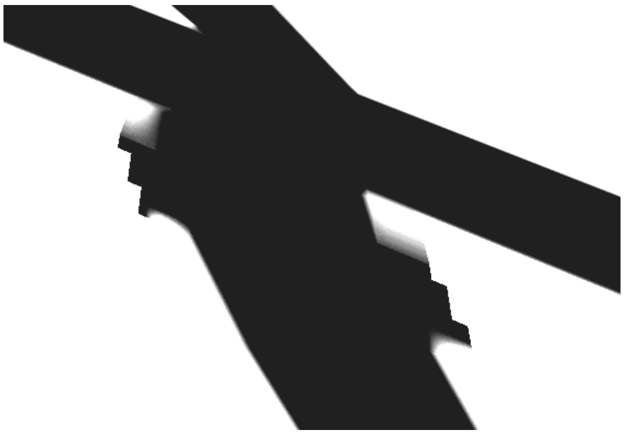

Figure 9: Minimal depth and minimal distance strategy creates wider shadow near region where two or more silhouette quads overlap. Image contains two crossed pipes casting shadow on the wall behind them.

6 Discussion

The above results illustrate the advantage of extending shadow map with edge distance information. Although we were limited to encode only one silhouette edge per fragment, we achieved significant gain of quality and maintain high frame rates. Using on-graphics hardware silhouette detection and silhouette quadrilaterals construction, we can take advantages of vertex and index buffers and thus store complete geometry on graphic accelerator. This approach can be also used for skinned models and key-framed animations, so it is perfect-fit for todays' computer games. Also, not generating silhouettes on host processor allows us to use remaining time to be used for other purposes (i.e. AI, physics...), which is especially important in computer games. Algorithm can be simply extended to cast shadows from alpha-keyed textures and to support shadows cast by particle systems.

We have tested rendering point lights as six separate spot lights and due to high precision of rendered shadow we achieved good results, although point lights were more than six times slower.

We have tried to reduce artifacts created when two or more silhouette quads overlaps with two strategies. The differences were only in choosing edge depths and distance values. First, we tried to store minimal depth and minimal distance, as it is done in [11]. This approach created cracks near shadow boundaries due to bilinear filtering which was highly disturbing espetialy when shadow map resolution is low, which is required for maximizing performance. Visual defect is shown in figure 10. We tried to solve this with extending silhouette quads inside occluders, but this resulted in undesired side-effect as shown in figure 9. Second, maximal depth and minimal distance were tested, and this strategy was chosen. Figure 11 shows the same scene rendered with second strategy. We have not tried to encode more than one silhouettes inside one fragment, because this would require more rendering passes and switches of render targets.

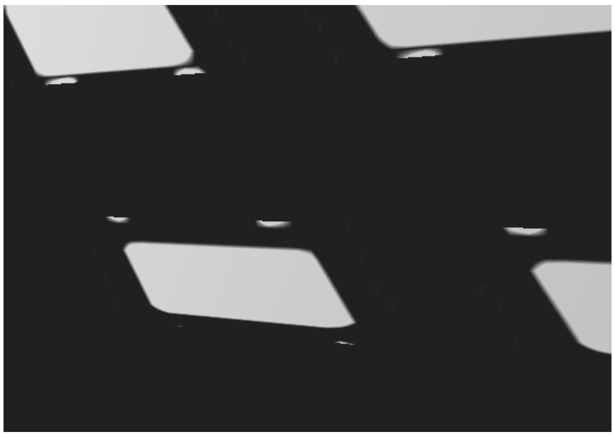

Figure 10: Crossed silhouette quads. Cracks created due to bilinear filtering. Silhouette quads are not extended inside occluders.

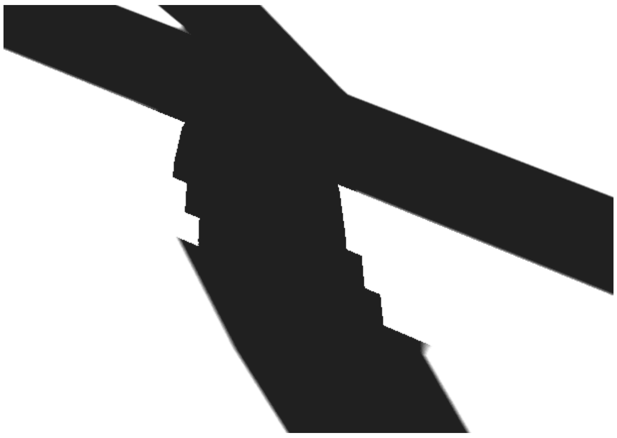

Figure 11: Maximal depth and minimal distance strategy behaves almost like standard shadow mapping near region where two or more silhouette quads overlap.

6.1 Limitations

Major limitation inherited from shadow maps is the use of discrete depth buffer samples, which leads to aliasing at shadow edges. Due to to performance limitations we can use only 32bit rendering, so maximally ten bits can be used to store depth values, so only small depth range was possible. This problem can be overcome with the use of floating point render targets, but this introduces performance penalty due to higher memory transfers. Also, as we are not using hardware support for shadow maps, which is not available on all hardware, we have to solve self-shadow aliasing on our own. We have used depth biasing and read values one shadow map unit above rendered surface. This introduces 3 instructions in vertex shader. Limitation is also number of stored silhouette edges per fragment.

6.2 Hardware improvements

Higher performance can be achieved, when there was support for percentage closer filtering to be used on an arbitrary texture. Then, a similar principle for “behind edge” detection can be used as in [10]. Also, in a pixel shader, we require to read the extended shadow map twice, once with linear filtering to read interpolated distance values and once with point filtering to read depths. This reduces number of textures that can be used in pixel shader. This method would benefit from setting different filtering methods for every color channel. One can also imagine to implement our method directly on hardware as a single instruction.

6.3 Comparsion with smoothies maps

First, we have to mention, that smoothies maps were designed to render pseudo-soft shadows instead of hard anti aliased shadows generated by our method. Our approach is designed to be used with low-resolution shadow maps, on the other hand, smoothies maps give not as good results when shadow map resolution is low. Figure 12 shows images rendered with smoothies map algorithm. Notice the distortion near occluder when smoothie size is 0.2 and shadow map resolution is 256x256. When compared to figure 8, our method gives better results when shadow map resolution is low. Also, hardware requirements are different in favor of our method. The same is the idea to encode additional data inside shadow map using geometry generated from silhouette edges, which was developed independently from our method.

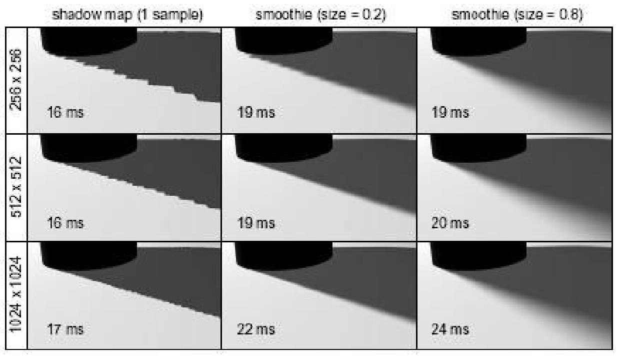

Figure 12: Shadows generated with smoothies maps algorithm with different shadow map resolutions and smoothies sizes. Image is taken from [11]

7 Conclusion

We have presented new anti aliased hard shadow generation algorithm based on shadow maps. We have achieved high frame rates for spot lights and point lights. Our algorithm is able to use arbitrary shadow casting and receiving objects based on triangular meshes. We believe that this algorithm can be successfully implemented in computer games and thus increase realism without high performance penalties known from stenciled shadow volumes algorithms.

Acknowledgments

Thank to Martin Bujňák for creating models for this paper. Also thank to Marek Zimányi to guiding me when writing this document.

References

[1] L.R.Wanger, J.A.Ferwerda, D.P.Greenberg. *Perceiving Spatial Relationship in Computer-generated Images*. IEEE Computer Graphics and Applications, May 1992

[2] A.Woo, P.Poulin, and A.Fournier. *A survey of shadow algorithms*. IEEE Computer Graphics and Applications, November 1990

[3] Alan Watt, Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice,* Addison-Wesly Pub. Co., October 1992, ISBN: 0201544121

[4] T.Akenine-Möller, U.Assarsson. *Appriximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges*, The Eurographics Association, 2002

[5] Ulf Assarsson, Tomas Akenine-Möller. *A Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware*, SIGGRAPH, 2003

[6] L.Williams. *Casting Curved Shadows on Curved Surfaces*. Computer Graphics (Proceedings of SIGGRAPH 87),August 1978

[7] W. T. Reeves, D. H. Salesin, R. L. Cook. *Rendering Antialiased Shadows with Depth Maps*. Computer Graphics (Proceedings of SIGGRAPH 87), July 1987

[8] M.Samminger, G.Drettakis. *Perspective Shadow Maps*, SIGGRAPH, July 2002

[9] R.Fernando, S.Fernandes, K.Bala, D.P.Greenberg. *Adaptive Shadow Maps*, Proceedings of SIGGRAPH, 2001

[10] Pradeep Sen, Mike Cammarano, Pat Hanrahan. *Shadow Silhouette Maps*, SIGGRAPH, 2003

[11] Eric Chan, Frédo Durand. *Rendering Fake Soft Shadows With Smoothies*. The Eurographics Association, 2003