Shadow Volumes Revisited

Stefan Roettger, Alexander Irion, and Thomas Ertl

University of Stuttgart, Faculty of Computer Science
Visualization and Interactive Systems Group
`wwwvis.informatik.uni-stuttgart.de`

ABSTRACT

We present a method to utilize the Shadow Volume Algorithm by Crow and Williams without using a stencil buffer. We show that the shadow mask can be generated in the alpha channel or even in the screen buffer, if a hardware-accelerated stencil buffer is not available. In comparison to the original stencil buffer method, a small speed up can be achieved, if the shadow mask is computed in the alpha buffer. The method using the screen buffer requires the scene to be rendered a second time after the shadow mask has been computed. Both methods are less restrictive with respect to hardware requirements, since we use only standard color blending and depth testing. In general, rasterization bandwidth is the main bottle neck when generating the shadow mask at high screen resolutions. In order to overcome this bottle neck we propose a way to compute the shadow mask at a resolution that is lower than the resolution of the screen buffer. Then the shadow mask is applied to the scene by utilizing texture mapping. The latter method might be reasonable especially in interactive entertainment, where rendering speed is traded in favour of image quality.

Keywords: Shadow Volumes, Hardware-Accelerated Rendering, Interactive Entertainment

1 Introduction

The display of shadows in three-dimensional scenes is a very useful technique to provide the viewer with a better impression of the shape and relative orientation of the objects in the scene. In an interactive environment with dynamic objects the fast computation of the shadows is a difficult problem, even for the simplest case of hard edged shadows. A straight-forward approach is to project each shadow casting polygon onto all the other polygons, but this approach does not scale well for big scenes.

2 Previous Work

Currently there exist mainly two well known hardware-accelerated methods, which solve this problem: Shadow volumes [14, 5, 12, 4, 1, 2, 7] and shadow maps [6, 15, 8, 16]. In [11] McCool et al. show a hybrid technique combining both methods, while an in depth comparison of both methods is given in [13]. Additionally, a method for generating soft shadows is presented in [9].

3 Hardware-Accelerated Generation of Dynamic Shadows

In the following we will briefly describe the basic concepts of the two hardware-accelerated methods. After a discussion of the pros and cons of these algorithms we show how to extend the shadow volume method in order to overcome the necessity of a hardware-accelerated stencil buffer and the shadow rasterization bottle neck.

3.1 Shadow Maps

The shadow map algorithm is an object-space approach to dynamic shadow casting. In order to compute the shadows, the scene is first rendered from the viewpoint of the light source and the resulting depth values are stored in a depth map. Then the scene is rendered again, but from the viewpoint of the eye. The coordinates of each rendered pixel are transformed into the local coordinate system of the light source, which enables us to retrieve the corresponding depth values from the depth map. Then each rendered pixel is shadowed, if and only if its distance to the light source is greater than the corresponding value in the depth map. One problem of this approach is to determine a suitable resolution of the depth map. A resolution, which is too low, results in blocky shadows on objects that are distant from the light source. On the other hand, the required memory increases quadratically with the resolution of the depth map. Another problem of the shadow map algorithm is that it depends on the availability of specific hardware extensions, which are not yet available every platform.

3.2 Shadow Volumes

In the following we will briefly describe the basic shadow volume algorithm, which is a screen-space approach to shadow casting. In Section 4.2 we will show that it can be employed without using a stencil buffer, if such a buffer is not available. The shadow volume of a shadow casting polygon is defined to be the half space that is bounded by the polygon itself and the set of quadrilaterals that are attached to each edge of the polygon stretching out in the direction of the light. The shadow volume algorithm now computes, whether a fragment of the visible scene is enclosed by at least one shadow volume or not. Since this is performed in screen space the shadows are computed exactly. For this purpose, the scene is first rendered from the point of view in order to obtain the correct depth values in Z-buffer. Next, depth buffer writing is disabled, but the depth test still remains active. Then the front and back faces of each shadow volume are rendered in the following fashion: Each front facing fragment that passes the depth test increases the stencil value by one, while a back facing fragment decreases the stencil value by one.

Depending on the depth value of each fragment, three cases, which are shown in Figure 1, are possible: If the front and back faces of the shadow volume are both behind a fragment, the stencil value remains unchanged for this fragment, because the depth test fails for both the front and the back faces (Case 1). The stencil buffer also remains unchanged, if both the front and the back face are in front of a fragment, since the stencil value is first increased and then decreased again (Case 2). If the fragment lies between the front and the back faces the stencil value is increased, because the front face passes the depth test, while the back face does not pass the test (Case 3). To sum up, the stencil value of a fragment is increased, if and only if the fragment is enclosed by the shadow volume. After all shadow volumes have been rendered, the stencil value of a pixel is greater than zero, if the pixel is shadowed by at least one polygon. Otherwise the stencil value is zero.

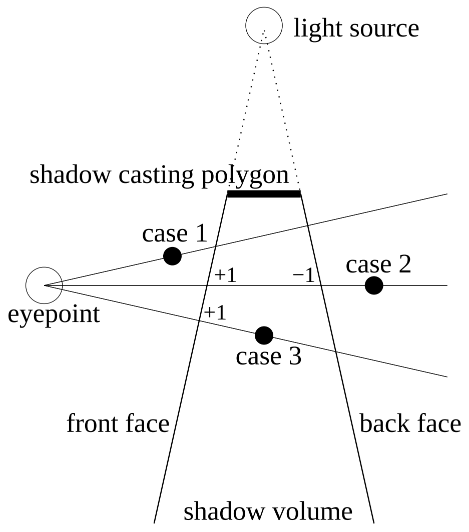


Figure 1: The shadow volume algorithm: The black dots indicate a fragment that lies either in front (Case 1), behind (Case 2) or inside (Case 3) the shadow volume. Only in Case 3 the value of the stencil buffer is increased by one. Therefore a pixel is shadowed, if and only if the stencil value is non-zero.

In order to apply the generated shadow mask to the scene a polygon covering the entire window is rendered with the stencil test passing, if the stencil value is greater than zero. If the viewer is positioned inside a shadow volume, the correct stencil values can be obtained by clipping the shadow volume at the near plane and by rendering the resulting polygon as a front face. Now every shadowed pixel can be either attenuated by using a constant blending factor or can be simply set to black. In the latter case, an ambient lighting term can be added by rendering the scene a second time. For the case of convex polyhedrons, however, an important improvement is possible: Instead of generating one shadow volume for each surface polygon, it is sufficient to construct a single shadow volume from the contour edges of the polyhedron as seen from the position of the light source.

The main advantage of the described shadow volume algorithm is that the computation of the shadows is per-pixel exact and that it can be performed almost entirely by the graphics hardware. However, one significant drawback is the potentially large amount of overdraw when rasterizing the shadow volumes. In the worst case, each shadow volume entirely covers the window, thus each pixel is at least rasterized once for each shadow volume. Even the average size of the rasterized regions must not necessarily be small. Therefore the minimum achievable frame rate is limited primarily by the rasterization bandwidth of the graphics hardware.

4 Shadow Mask Generation without using the Stencil Buffer

In interactive entertainment dynamic shadows are becoming more and more popular. To give an example, here is a quote from the game developer magazine *Gamasutra* [3]: “Now that 8-bit stencil buffers are appearing on a wide assortment of graphics accelerator cards, shadow-casting methods can be employed for generating real-time shadows with only a minimal performance hit“.

4.1 Restrictions of the Traditional Shadow Volume Algorithm

At the *NVIDIA* home page [10] it is presumed that shadow volumes “requires a hardware stencil buffer for fast performance”. In many other publications the words “shadow volume” and “stencil buffer” are mentioned in the same breath leading to the wrong conclusion that fast shadow volumes are impossible without stencil buffer support. Here, our initial motivation for this paper was to show that real-time shadows can be utilized not only on the Sony PS2, for example, which has no dedicated stencil buffer, but even on first generation graphic cards like the *3dfx* Voodoo 1.

In the following we will present an extension to the shadow volume algorithm by Crow [5] and Williams [14], which enables us to perform interactive shadow casting without using the stencil buffer. In conclusion, we were also able to widen the main bottle neck of the shadow volume algorithm by reducing the spatial resolution of the shadows. This is interesting especially in interactive entertainment where speed is considered vitally. On platforms that already support a stencil buffer our method is also rewarding, because in some cases our method is even faster.

4.2 Generating the Shadow Mask in the Screen or in the Alpha Buffer

When using the screen or alpha buffer instead of the stencil buffer the main problem commonly is the lack of a subtraction operation. As a replacement for incrementing and decrementing the stencil buffer we use equivalent blending operations that double or halve the destination buffer values. In OpenGL notation the operation that halves the destination values can be performed by choosing the blend function `glBlendFunc(GL_DST_COLOR, GL_ZERO)` with the vertex brightness set to $\frac{1}{2}$. To perform the double operation we choose the blend function `glBlendFunc(GL_DST_COLOR, GL_ONE)` with a vertex brightness of 1. This holds for both the alpha and the screen buffer method.

In analogy to the stencil buffer method, the scene is first rendered to place the depth values in the depth buffer. All pixels of the shadow mask buffer (whether alpha or screen buffer) are initially set to the value $\frac{1}{4}$. In order to render a shadow volume the front faces are drawn first by using the double operation. Next the back faces are rendered using the halve operation. This requires two state changes for each shadow volume, but the time spent by the state changes is neglectable compared to the time spent by the rasterization of the shadows. After the first shadow volume has been rendered the initial value of $\frac{1}{4}$ will remain, if a pixel is not shadowed, whereas the value of a shadowed pixel will double to $\frac{1}{2}$. The remaining shadow volumes are rendered subsequently, but now the value of a shadowed pixel can be either $\frac{1}{2}$ or 1, The latter is true, if a pixel is shadowed by more than one object. If the pixel is behind a shadow volume and its value has already been raised to 1, it falls back to $\frac{1}{2}$. This is due to the fact that the double operation has no effect because of color clamping. Nevertheless, this is no general algorithmic restriction, because in any case, the value of a pixel that is not shadowed will be $\frac{1}{4}$ after all shadow volumes have been drawn. Otherwise the shadow mask’s value will be $\frac{1}{2}$ or 1, if the corresponding fragment is enclosed by at least one shadow volume. Figure 2 shows these three possible states of a fragment. An example of applying this algorithm (frame buffer method) to a simple test scene is given in Figure 3. The scene consists of two shadow casting objects and a ground floor. Except the two shadow volumes, the objects of the scene are rendered in wire frame mode to allow a better perception of the generation of the shadow mask, which is illustrated step by step. The first image shows the contents of the frame buffer after drawing the front faces of the shadow volume cast by the cylinder. In the next image the back faces of the cylinder were rendered. The subsequent two images show the shadow mask after the front and back faces of the shadow volume cast by the rectangle have been drawn. The next two images show the shadowed regions and the final rendered scene. The conversion of the three valued shadow mask into a black and white mask and the application of the shadow mask to the scene are described in detail in the next section.

4.3 Normalizing and Applying the Shadow Mask to the Scene

The final step of image generation is the application of the shadow mask to the scene. Before this can be accomplished the three valued shadow mask has to be converted into a bilevel image. The fastest way to achieve this normalization is to draw three rectangles that entirely cover the window. The first white rectangle is rendered by using the blend function

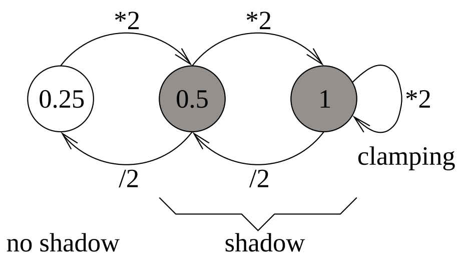


Figure 2: The shadow mask states: The state $\frac{1}{4}$ corresponds to unshadowed pixels, while the other two states stand for shadowed fragments. The front faces of a shadow volume evoke *2 transitions, while back faces evoke /2 transitions. If a pixel is enclosed by a shadow volume, its initial state of $\frac{1}{4}$ changes to $\frac{1}{2}$. Afterwards the state of the pixel can only alter between $\frac{1}{2}$ and 1, but it cannot fall back to $\frac{1}{4}$.

`glBlendFunc(GL_DST_COLOR, GL_ONE)`, which effectively doubles the shadow mask values. Due to clamping only the values $\frac{1}{2}$ and 1 are remaining. The second white rectangle is rendered by using the blend function `glBlendFunc(GL_ONE_MINUS_DST_COLOR, GL_ZERO)`. This inverts all values of the shadow mask. Now all shadowed pixels correspond to a value of 0, whereas the value of unshadowed pixels remains $\frac{1}{2}$. At this point we have three choices of applying the shadow mask to the scene: We can either set the shadowed regions of the frame buffer to black, redraw the shadowed parts of the scene adding an ambient lighting term or attenuate the shadowed regions by a constant factor. For the case of black or ambient shadows a third white rectangle is rendered by using the blend function `glBlendFunc(GL_DST_COLOR, GL_ONE)`, which doubles the shadow mask values leaving only zero and one (see also Figure 4). For the case of attenuating the brightness of the shadowed regions by a factor of $\frac{1}{2}$, for example, the third rectangle is rendered by using the blend function `glBlendFunc(GL_ONE, GL_ONE)` and a vertex brightness of $\frac{1}{2}$ leaving shadow mask values of $\frac{1}{2}$ and 1.

Now we are ready to multiply the shadow mask with the scene. In the case of using the alpha buffer the scene has already been rendered into the frame buffer, so we simply draw another last rectangle by using the blend function `glBlendFunc(GL_ZERO, GL_DST_ALPHA)`. The screen buffer method requires one more rendering pass, since the scene is not already available in the frame buffer. For that purpose we are using the blend function `glBlendFunc(GL_DST_COLOR, GL_ZERO)`. Another optional rendering pass is required to add a ambient lighting term to the frame buffer (blend function `glBlendFunc(GL_ONE, GL_ONE)`). A com-

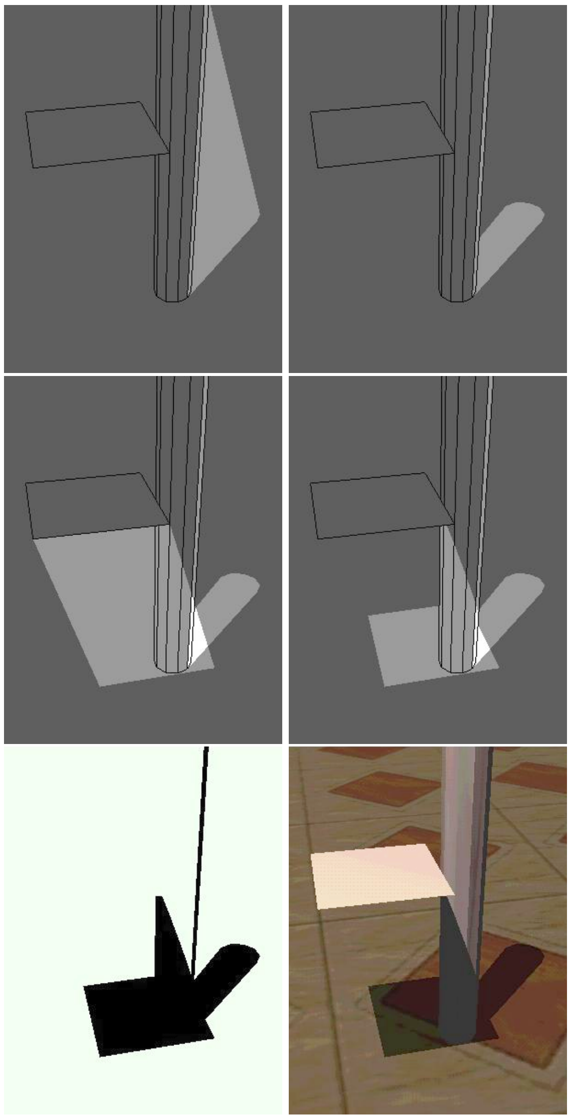


Figure 3: Shadow mask generation: The first image shows the shadow mask after rendering the front faces of the shadow volume cast by the cylinder. The next image shows the mask after rendering its back faces. The following two images show the shadow mask after the front and back faces of the shadow volume cast by the rectangle have been rendered. Next, the shadow mask is shown after its three possible values have been normalized. Finally, the resulting black and white shadow mask is applied to the scene using ambient shadows on a *3dfx* Voodoo 2 graphics accelerator.

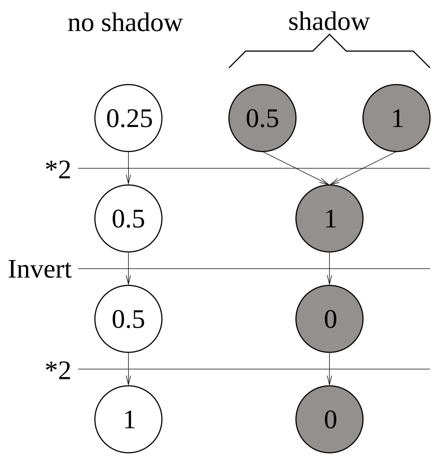


Figure 4: Normalization of the shadow mask: Three rectangles are rendered across the entire window to subsequently perform the operations *2, Invert and /2. The final values are zero, if the pixel is shadowed, and one otherwise.

	black shadows	ambient term	attenuated shadows
alpha buffer	1 / 4	2 / 4	1 / 4[2]
screen buffer	2 / 3	3 / 3	2 / 3[1]

Table 1: Tabular listing of the number of rendering passes and the number of rectangles drawn across the window for normalization. Numbers in brackets denote the number of drawn rectangles by utilizing the maximum blend equation.

plete listing of all the required passes for the cases described above can be found in Table 1.

Strictly speaking, the attenuation of a shadowed pixel is not correct in a physical sense, since the brightness of shadowed regions should not depend on the position or orientation of the light source. To compute shadows with a constant ambient (or diffuse brightness for infinite light sources) an additional rendering pass is required as described above. In a direct visual comparison we have found that this additional rendering pass can be saved in favour of using attenuated shadows, because the subjective appearance of the scene is altered just slightly. In fact, we have found that the differences become discernible in a direct comparison only.

4.4 Improving the Normalization of the Shadow Mask

On platforms which support a maximum blend equation there exists a faster way to normalize the three

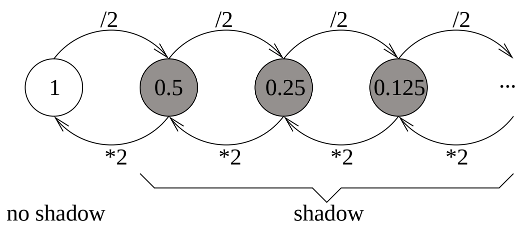


Figure 5: Alternative mask normalization using the maximum blend equation: The initial state is 1. Front and back faces halve or double the value of the shadow mask. If a pixel is shadowed, its value is less than or equal to $\frac{1}{2}$ or 1 otherwise. After all shadow volumes have been rendered, one rectangle with brightness $\frac{1}{2}$ is drawn across the entire window. As a result, shadowed regions are attenuated by a factor of $\frac{1}{2}$.

valued shadow mask. This implies a slight modification of the shadow mask generation as well: The screen or alpha buffer is initialized with the value 1. For the front and back faces of the shadow volumes the halve or double operation is applied, respectively. After all shadow volumes have been rendered, the value of shadowed pixels is less than or equal to $\frac{1}{2}$, while the value is 1 for unshadowed pixels. Figure 5 shows the corresponding modified state transition graph. Instead of drawing the former three now only one rectangle needs to be drawn across the window in order to rise values of less than $\frac{1}{2}$ to a minimum value of $\frac{1}{2}$ (see also Table 1). In OpenGL notation this can be accomplished by using the blend equation `glBlendEquationEXT(GL_MAX_EXT)` and by additionally setting the brightness of the rectangle to $\frac{1}{2}$. Multiplying the scene with this shadow mask will now attenuate shadowed pixels by a factor of $\frac{1}{2}$.

4.5 Computing the Shadow Mask at Lower Resolutions

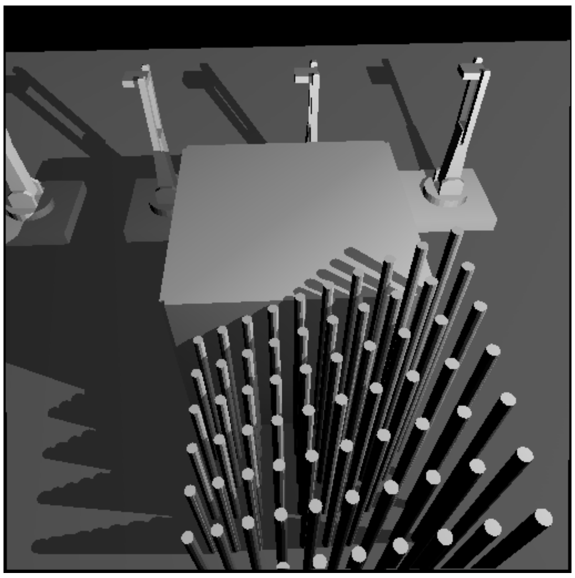
The main drawback of the shadow volume algorithm is the bottle neck of rasterizing the shadow volumes. Usually, the cost for rendering the shadows exceeds the cost for rendering the scene, in particular at high resolutions. By trading image quality for rendering performance this bottle neck can be widened by utilizing texture mapping hardware. The shadow mask is rendered as described above, but at a lower resolution. After that, it is copied from the alpha or screen buffer into a texture of the same resolution. Then the scene is redrawn at full resolution and the magnified texture is multiplied onto the frame buffer by drawing a textured rectangle across the entire window. Transfering the contents of the stencil buffer into a texture is not directly possible. Copying from the screen or alpha buffer, however, is supported on most graphics systems (by means of using `glCopyTexImage2D`). A drawback of rendering the shadow mask at a lower resolution is that the scene has to be rendered twice, since the depth values of the scene must be available in the resolution of the shadow mask. Another obvious drawback is the reduced resolution of the shadows, but bilinear texture filtering can be used to smooth the blockiness. Figure 6 shows two screen shots, which were generated using a shadow mask at full and half resolution, respectively. It turns out that the image quality is acceptable when choosing half resolution shadow masks. On the other hand, the number of rasterized pixels is reduced to one fourth, which significantly speeds up the shadow mask generation. In particular, the minimum frame rate is increased by almost a factor of four.

5 Results

Table 2 shows the average frame rate achieved by our algorithms during an animation of the scene shown in Figure 6. The test was performed on a AMD 800MHz PC with a NVIDIA GeForce 2 MX graphics card. The resolution of the window is given in the first column. The next three columns show the number of frames per second either using the screen, alpha or stencil buffer method. The fifth column shows the frame rate for using a shadow mask of half the screen resolution. The frame rate for rendering the unshadowed scene is given in the last column. The results illustrate that computing the shadow mask in the screen or alpha buffer can be even faster than rendering the shadows into the stencil buffer. Using a shadow mask of half the screen resolution did improve the worst case performance by an approximate factor of four. The performance at higher window resolutions was increased as well, because the pixel fill rate is the dominating factor at high window resolutions. Reducing the size of the shadow mask to a quarter of the screen size, for example, increased the frame rate from 10.2 to 23.6 Hertz at a window resolution of 1024×1024.

6 Conclusion

We have presented a variety of extensions to the original shadow volume algorithm by Crow and Williams. A screen or alpha buffer can be used for to compute the shadow mask. This is a necessity, if a stencil buffer is not available, for example on the Sony PS2, or already in use for other purposes. The increment and decrement operations, which are required by the shadow volume technique, are replaced by a double and halve operation that can be realized by employing standard blend functions. The screen or alpha buffer methods are slightly faster than using



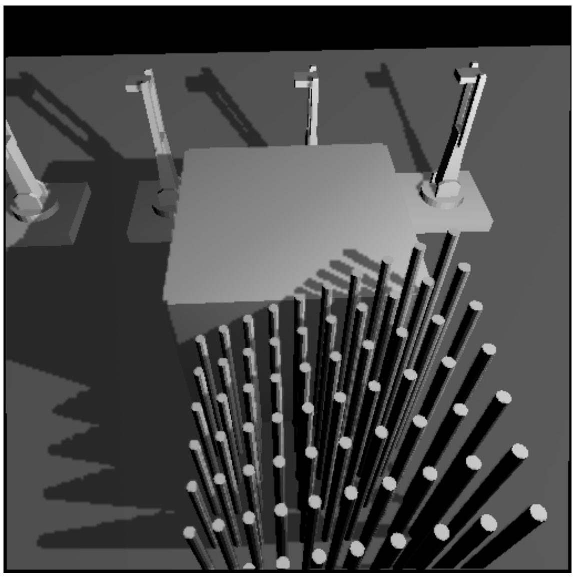


Figure 6: Final rendered scene using a full and half resolution shadow mask.

window size	screen buffer	alpha buffer	stencil buffer	half res.	no shadows
256^2	40.8	42.4	42.4	39.1	90.4
512^2	29.9	33.6	29.1	30.5	89.2
1024^2	9.3	10.2	8.9	15.7	44.6

Table 2: Average frame rate using different resolutions and shadow volume methods: For high resolutions the computation of the shadow mask in the alpha or screen buffer was slightly faster than using the stencil buffer. Computing the shadow mask at half the screen resolution improved the performance for higher window resolutions.

the original stencil buffer algorithm. What is more, the computation of the shadow mask can be computed at lower resolutions, which enables us to substantially reduce the impact of generating the shadow mask at high window resolutions. In interactive entertainment, where speed is vitally, our proposed extensions allow to speed up the shadow volume algorithm and to support shadow volumes on a broader range of graphics accelerators.

7 Acknowledgements

We would like to thank *3dfx* for not providing a hardware-accelerated stencil buffer when designing their Voodoo 1, 2 and 3 graphics accelerators. We would also like to thank *3dfx* for making the alpha and depth buffer mutually exclusive.

REFERENCES

[1] P. Bergeron. Shadow Volumes for Non-Planar Polygons: Extended Abstract. In *Proc. Graphics Interface '85*, pages 417–418, May 1985.

[2] P. Bergeron. A General Version of Crow's Shadow Volumes. *IEEE Comput. Graph. Appl.*, 6(9):17–28, September 1986.

[3] J. Bestimt and B. Freitag. Real-Time Shadow Casting Using Shadow Volumes. *Gamasutra*, November 1999. `http://www.gamasutra.com/features/19991115/bestimt_freitag_01.htm`.

[4] J. F. Blinn. Jim Blinn's Corner: Me and My (Fake) Shadow. *IEEE Comput. Graph. Appl.*, 8(1):82–86, January 1988.

[5] F. C. Crow. Shadow Algorithms for Computer Graphics. *Computer Graphics (Proc. SIGGRAPH '77)*, 11(2):242–248, July 1977.

[6] E. Haines and D. Greenberg. The Light Buffer: A Shadow-Testing Accelerator. *IEEE Comput. Graph. Appl.*, 6(9):6–16, September 1986.

[7] T. Heidmann. Real Shadows Real Time. *Iris Universe*, 18:28–31, 1991.

[8] W. Heidrich. High-Quality Shading and Lighting for Hardware-Accelerated Rendering. *Ph.D. Dissertation. University of Erlangen, Germany*, 1999.

[9] W. Heidrich, S. Brabec, and H.-P. Seidel. Soft Shadows Maps for Linear Lights. In *Rendering Techniques '00, Proc. of the 11th Eurographics Workshop on Rendering*, pages 269–280, Wien, June 2000. Springer.

[10] M. Kilgard. Creating Reflections and Shadows Using Stencil Buffers (NVIDIA Technical Demonstration). *Game Developer Conference*, November 1999.

[11] M. D. McCool. Shadow Volume Reconstruction from Depth Maps. *ACM Transactions on Graphics*, 19(1):1–26, January 2000.

[12] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast Shadows and Lighting Effects Using Texture Mapping. In *Proc. SIGGRAPH '92*, pages 249–252, July 1992.

[13] M. Slater. A comparison of Three Shadow Volume Algorithms. *The Visual Computer*, 9(1):25–38, 1992.

[14] Lance Williams. Casting Curved Shadows on Curved Surfaces. *Computer Graphics (Proc. SIGGRAPH '78)*, 12(3):270–274, 1978.

[15] A. Woo. Efficient Shadow Computations in Ray Tracing. *IEEE Comput. Graph. Appl.*, 13(5):79–83, September 1993.

[16] Hansong Zhang. Forward Shadow Mapping. In G. Drettakis and N. Max, editors, *Proc. EuroGraphics Rendering Workshop '98*, pages 249–252, June 1998.