STENCIL SHADOW VOLUMES FOR COMPLEX AND DEFORMABLE OBJECTS

Ivica Kolić, Zeljka Mihajlovic, Leo Budin

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10 000 Zagreb, Croatia

ABSTRACT

We present real-time shadow method based on the shadow volume that exploit capabilities of the modern graphics cards. Algorithm is primarily created for casting shadows of a highly concave complex objects such as trees. For those objects silhouette calculation that is usually preformed by other shadow volume algorithms is complicated and poorly justified. Instead of calculations, it is better to assume a worst case scenario and use all of the edges for construction of the shadow volume mesh, skipping silhouette determination entirely. The achieved benefit is that all procedure, the object and shadow calculation and rendering, could be done on GPU. Proposed solution for shadow casting allows open edges. Indexed vertex blending is used for shadow projections, and the only calculation required is determining projection matrices. Once created, shadow volume is treated like any other mesh.
Keywords: shadow, shadow volumes, stencil testing, vertex blending.

1. INTRODUCTION

Shadow represents one of the key elements in creating realistic images. Their value is not only in aesthetic reasons, but also in supplying additional information about object placement in scene.

Shadow volumes are one of the most powerful techniques used for real-time computer shadows [3, 5, 10]. Unlike some other popular shadowing methods (shadow mapping) they work well with all type of light sources. Because of the enormous hardware demands, this technique has never been completely accepted in real-time applications. Problem lies in today's algorithms that are still putting most of the work on CPU, thereby ignoring full potential of graphic cards. This property can be easily seen by changing video card resolution and noticing that the frame rate is the same at lower and at higher resolution setting.

With new features of graphic card being introduced, such as transformation and lighting (T&L) and vertex shaders, CPU is relived of processing lightning and geometry and allowed to do additional calculation elsewhere. However, old shadow volume algorithms do not use those new features of graphic cards.

2. PREVIOUS WORK

Idea of shadow volume was first introduced in 1977 when Frank Crow [1] published his ray-casting based shadow volume algorithm. A pixel's view ray is walked from the viewer's eye point and intersections with shadow volume are counted prior to ray hitting some solid surface. Observed pixel is shadowed in two cases. First case occurs when number of intersections is odd and eye point is located outside the shadow volume and the second case occurs if eye point is located inside the shadow volume and number of intersections is even.

In 1991 Heidmann published his stencil buffer based shadow volume algorithm which is even today the main shadow volume algorithm [2]. This time, stencil and depth buffers were used to achieve functionality of the Crow's ray-casting. Scene was first rendered with only ambient lightning updating the depth buffer. Shadowed parts of the scene are marked using shadow volume mesh and depth buffer information, and scene was rendered once more with all of the lightning only updating unmarked parts of the scene (two pass algorithm). However, most of the developers prefer to use simpler method where scene is completely drawn once and shadowed parts are marked and darkened a bit (single pass algorithm).

For more complex objects then a triangle, additional effort has to be made to determine the shape of the shadow volume mesh. This is the bottle neck of the algorithm, and the reasons why original Heidmannn's algorithm does not allow open edges. Explanation how to properly handle open edges can be found in the work of Bergeron [4]. Shadow volume has to be constructed from the object's silhouette seen from the light's point of view. Simplest and fastest way to determine objects silhouette is to separate positively and negatively oriented triangles and isolate all the edges that have one triangle oriented positively and another negatively. In a case of a closed mesh (no open edges) every edge has two triangles.

Improvement of the shadow volume method can be found in work of many authors [6 - 9, 11].

Our approach is based on defining functions performed on the stencil buffer for shadow volumes to create shadows.

3. SHADOW VOLUMES

The goal was to create an algorithm that will perform actual marking of shadowed parts of the scene no different then rendering any other mesh. If this is achieved, it will become possible to use advanced mesh deforming features of the modern graphic cards such as vertex shaders, while rendering a shadowed scene. CPU calculations of shadow volume shape will not be necessary, and shadow volume will be deformed the same way the original mesh is, using vertex shaders.

Unlike Heidmann's algorithm, stencil buffer won't be used in a form of counter increasing its values when front planes of a shadow volume are drawn or decreasing them in case of back planes. Also, attention won't be focused only on stencil-pass or z-fail events while rendering shadow volume planes. Instead, operations will be preformed in both stencil-pass and in z-fail case. Given that there is no calculation of a shadow volume by CPU, there is also no way of knowing what are back or front sides of a shadow volume, so the method should not differ them. This means that the same stencil testing parameters will be used when back and front planes of a shadow volumes are render. Therefore, entire process of marking shadowed parts of a scene can now be done in a single pass.

Fig. 1. shows a 2D representation projected triangle. 2D representation of the projected front and back shadow volume sides are shown on Fig. 2. and 3. so that the areas affected by them could be easily perceived.

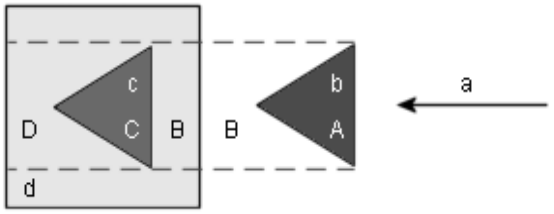


Figure 1. 2D representation of a projected triangle. A) shadowing triangle; B) area between shadow and shadowing object; C) shadow; D) shadowed object.

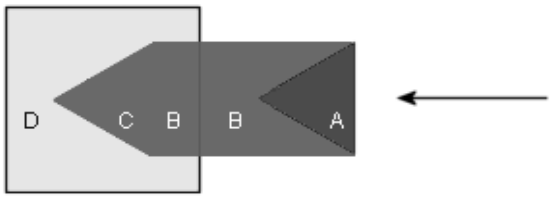


Figure 2. 2D representation of a projected front shadow volume side.

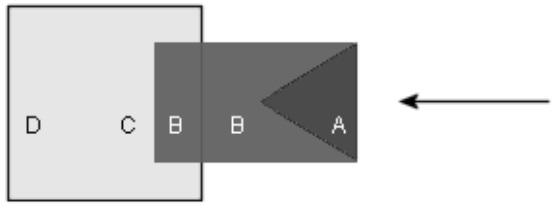


Figure 3. 2D representation of a projected back shadow volume side.

In area D operation z-fail will occur when both front and back sides of a shadow volume are drawn. On the other hand, in area B stencilpass operation will occur twice, and in area C z-fail will occur together with stencilpass operation. It is also noticeable that there is a problem with area A (shadowing triangle). If shadowing triangle is facing the viewer, then it will be missing a small part of a front shadow volume side. If the triangle is facing away from the viewer, small part of a back shadow volume side will be missed.

3.1 Required Operations on the Stencil Buffer

Before stepping into describing how algorithm works, a few guiding points will be introduced:

- When ever certain polygon castes its shadow, no other polygon can erase it. This leads to conclusion that stencil comparison function will be NOTEQUAL to referent stencil buffer value representing marked shadowed area.
- Shadow must be cast only on those areas where Z-FAIL and STENCILPASS operations have occurred together (surface of the shadowed object inside the shadow volume) – area C.
- Where Z-FAIL occurs twice (inside of the shadowed object) stencil buffer values must be equal to zero – case of area D. Therefore, two Z-FAIL operations must cancel each other out (for example Z-FAIL operation = INVERT).
- Where STENCILPASS occurs twice, stencil buffer value must also be equal to zero (area between shadowing and shadowed object) – case of area B. Therefore, two STENCILPASS operations must also cancel each other, however, the same operation used for Z-FAIL event can not be used in this case.
- On areas where operations Z-FAIL and STENCILPASS have occurred together (where shadow should be cast), ordering of these operations must not be relevant. Same result (stencil buffer value representing shadow) must be achieved whether Z-FAIL has occurred before STENCILPASS operation, or vice-versa.
- Since shadow volume is drawn only once, algorithm should not make difference between back and front side of shadow volume. Therefore, face culling should be disabled – shadow volume is rendered in a single pass (back and front sides together).

- On those areas where shadow should be, stencil buffer must have one value, and stencil buffer value of all other areas must be zero.

3.2 Functions Applied on the Stencil Buffer

We have to find right parameters for the following values:

STENCILFUNCTON	= NOTEQUAL
STENCILPASS	= ?
Z-FAIL	= ?
SETNCILFAIL	= ?
STENCILMASK	= ?
SETNCILWRITEMASK	= ?
STENCILREF	= ?
COLLMODE	= NONE
ZFUNC	= ?

As mentioned before, Z-FAIL operation must cancel itself. This can be archived by applying INVERT operation of stencil buffer value when this event happens. If zero is inverted twice, the final stencil buffer value will also be zero.

STENCILZFAIL	= INVERT

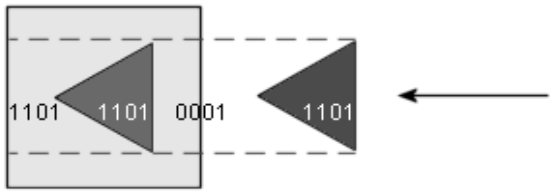
A choice of a STENCILPASS operation also has a nullification nature. This time the same operation as used for Z-FAIL case cannot be used since it is necessary that this two operations working together cause stencil buffer to reach final stencil value representing the shadow. Operation INCR is chosen, but limited on only first bit (inverting the first bit). This is achieved by setting STENCILWRITEMASK to 1101, so that the second bit can never become 1. In example, if beginning stencil buffer value of a current pixel is 0, when STENCILPASS operation occurs it will become 1. If STENCILPASS occurs for the second time, stencil buffer value of that pixel will once again become 0. It should actually become 2 but since the second bit can't be written, 0 will be written instead. So, to conclude:

STENCILPASS	= INCR
STECILWRITEMASK	= 1101b
SETNCILMASK	= 1101b

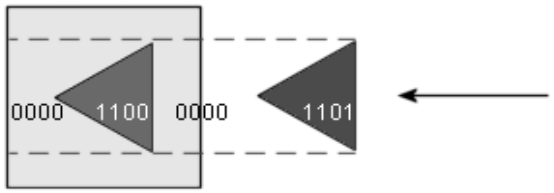
These parameters are actually solution to a shadow marking problem. On those areas where STENCILPASS operation occurs together with STENCILZFAIL operation, a unique stencil buffer value (representing a final shadow value) will be reached. This value is 1100, and it occurs only on shadowed parts of a scene – area C and in some cases area A (depending of a triangle orientation).

STENCILREF	= 1100b

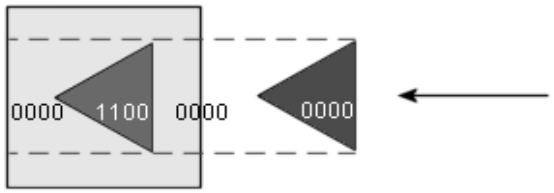
Example of a triangle casting shadow will be explained next. Two examples are possible. In first case back side of shadow volume is drawn first as showed on Fig. 4, and in second case front side of shadow volume is drawn first. It is important to consider that at the beginning all stencil buffer values are set to zero.



a) Back side of shadow volume is drawn first



b) Then the front side of shadow volume is drawn



c) Then, a shadowing triangle is redrawn once more to correct its stencil values.

Figure 4. Stencil buffer values after rendering shadow volume when back side of a shadow volume is drawn first.

For every triangle casting the shadow, seven additional shadow volume triangles has to be drawn. If shadow casting triangle is facing the light it should not be shaded and its stencil buffer value should be zero. In Fig. 4. the triangle facing the light was marked by its back shadow volume side with value 1101. The STENCILZFAIL operation happens once again when a shadow casting triangle is drawn as a part of a shadow volume. During rendering, a ZFUNCTION parameter (z-buffer comparison function) must be set to LESS. This will always induce Z-FAIL event, and set its stencil buffer values to zero.

4. RESULTS

Testing included comparison between Heidmann's and proposed algorithm while dropping shadow of a small branch shown on Fig. 5. Model is composed of 6790 vertices and 2878 triangles. Testing was done in three most common screen resolutions. During the testing of a Heidmann's algorithm no checking was done whether the camera was located within shadow volume which additionally speeded up its algorithm. Results are shown in Table 1. followed by graphs on Fig. 6 and 7.

Testing configuration:

- Intel Pentium 4 2.8 GHz, 1 GB RAM, Windows XP;
- ATI Radeon 9700, 128 MB, DirectX 8.1a;

Figure 5. Testing model.

Table 1. Testing results are in frames per seconds (FPS).

Branch testing		640x480	800x600	1024x768
Full screen	New	52,88	45,7	32,82
	Heidmann	43,23	36,36	28,19
	No shadow	76,22	74,4	73,53
Windowed	New	82,1	61,54	42,66
	Heidmann	52,91	46,73	34,98
	No shadow	641,03	534,76	427,53

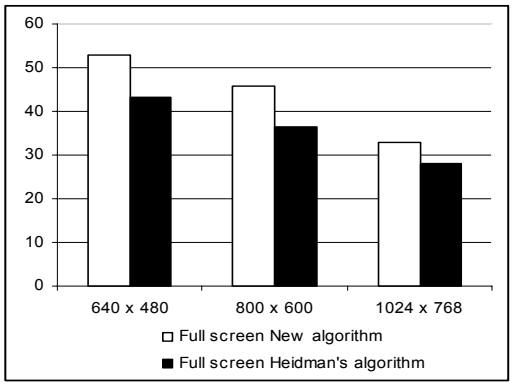


Figure 6. Testing in full screen mode.

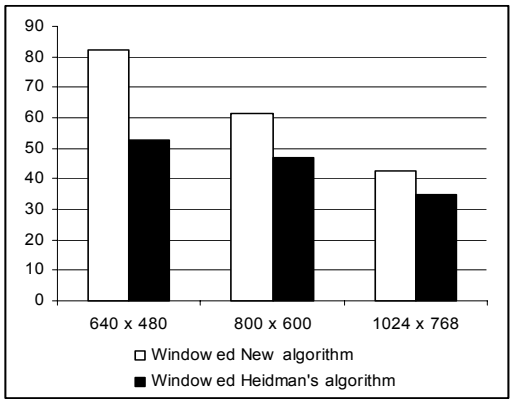


Figure 7. Testing in windowed mode.

Microsoft's X-Box has CPU that runs at only 720 MHz. If we had used a CPU similar to the one of the X-Box, results of a Heidmann's algorithm would be approximately three times slower. On the other hand X-Box's has a powerful GPU similar to the one we used, so the results of our new algorithm wouldn't be much different then the one we achieved. Because of a slow central processor X-Box should always prefer using this new algorithm rather then Heidmann's.

5. CONCLUSION

Our stencilled shadow volume algorithm is highly robust, simple to implement and supports even most advanced hardware functionality available today. It is finally possible to use shadow volumes in contest of hardware accelerated vertex blending or vertex shaders in generally. Testing showed that Haidmann's algorithm is slower even though it's doing less then a half of the graphic work preformed by algorithm that we propose. We can conclude, that even thought the extremely fast CPU was used it was not good enough for the Heidmann's algorithm. So, passing work from CPU to a GPU is important concept.

6. REFERENCES

[1] Frank Crow, "Shadow Algorithms for Computer Graphics", *Proceedings of SIGGRAPH*, 1977, pp. 242-248.

[2] Tim Heidmann, "Real Shadows Real Time", *IRIS Universe,* Number 18, 1991, pp. 28-31.

[3] Harlen C. Batagelo and Ilaim C. Junior, "Real-Time Shadow Generation Using BSP Trees and Stencil Buffers," *XII Brazilian Symposium on Computer Graphics and Image Processing*, Campinas, Brazil, Oct. 1999, pp. 93-102.

[4] Philippe Bergeron, "A General Version of Crow's Shadow Volumes," *IEEE Computer Graphics and Applications*, Sept. 1986, pp. 17-28.

[5] Jason Bestimt and Bryant Freitag, "Real-Time Shadow Casting Using Shadow Volumes," *Gamasutra.com web site*, Nov. 15, 1999.

[6] David Blythe, Tom McReynolds, et.al., "Shadow Volumes," *Program with OpenGL: Advanced Rendering*, SIGGRAPH course notes, 1996.

[7] Lynne Brotman and Norman Badler, "Generating Soft Shadows with a Depth Buffer Algorithm," *IEEE Computer Graphics and Applications*, Oct. 1984, pp. 5-12.

[8] Mark Kilgard, "Improving Shadows and Reflections via the Stencil Buffer," *Advanced OpenGL Game Development* course notes, *Game Developer Conference*, March 16, 1999, pp. 204-253.

[9] Michael McCool, "Shadow Volume Reconstruction from Depth Maps," *ACM Transactions on Graphics*, Jan. 2001, pp. 1-25.

[10] Mark Kilgard, "Robust Stencil Volumes," *CEDEC 2001 presentation,* Tokyo, Sept. 4, 2001.

[11] Andrew Woo, Pierre Poulin, and Alain Fournier, "A Survey of Shadow Algorithms," *IEEE Computer Graphics and Applications*, Nov. 1990, pp. 13-32.