Computer Technology and Application 2 (2011) 638-643

Shadow Volume in Real-Time Rendering

Abd El Mouméne Zerari and Mohamed Chaouki Babahenini

Department of Computer Science, LESIA Laboratory, University Med Khider Biskra, Biskra 07000, Algeria

Received: May 16, 2011 / Accepted: June 09, 2011 / Published: August 25, 2011.

Abstract: This paper presents an optimization of shadow volume algorithm, which allow a rendering in real-time. This technique is based on previous works which makes it possible to obtain shadows in real-time, although the calculation of the silhouette requires a pretreatment of the geometry implemented on the CPU (Central Processing Unit). By using last version of the GPU (Graphic Processing Unit), the authors propose to implement the calculation of the silhouette on the GPU by using Geometry Shader. The authors present the step which made it possible to lead to a concrete implementation of this algorithm, the modifications which were made, as well as a comparative study of results, followed by a discussion of these results and choices of implementation.

Key words: Shadow volumes, silhouette, GPU (graphic processing unit), real-time, shaders.

1. Introduction

Shadows enhance the realism of computer-generated images and also provide information about the spatial relationships of objects. An intuitive way to think of the shadows is in a purely geometrical way. This approach was initially described by Crow in 1977 [1] and then implemented using graphic material by Heidmann in 1991 [2].

Shadows have a relatively long history in the young science of computer graphics. It nevertheless took more than 20 years before it was finally applicable for real-time rendering of average complexity scenes [3]. The algorithm in Ref. [3] was very fast for its time, based on very particular representations in order to adapt the computation to a graphics card. Today, this solution is mostly historical. More direct and efficient implementations are possible on the latest generations of cards. The paper is organized as follows: Section 2 discusses on shadow volumes; section 3 introduces the Stencil shadow volumes; section 4 is fine proposed model; section 5 introduces new Shaders; section 6 presents results and discussions; section 7 gives conclusions; section 8 presents future work.

2. Shadow Volumes

The approach consists in generating a polygonal shadow starting from the occulting objects of the scene opposite to a point source, then to display only certain parts of these volumes, classically, those ranging between the front faces and back of shadow volumes, as illustrated in Fig. 1. With the source, the silhouettes edges (i.e., incidental edges with two polygons, one directed towards the source, the other not) form plans which delimit the shadow volume of the object. Only the points of space not contained in such a volume are lighted by the source [3].

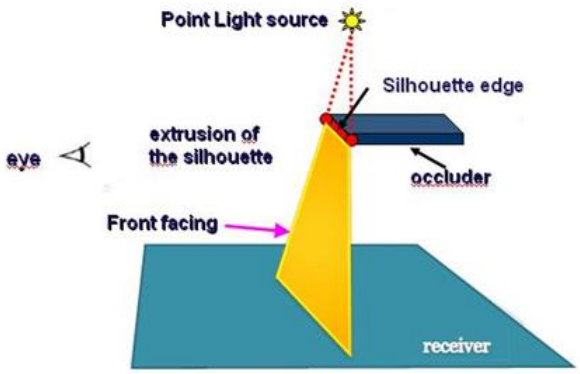

Fig. 1 Shadow volume.

Mohamed Chaouki Babahenini, Ph.D., associate professor, research field: computer graphics and vision.

Corresponding author: Abd El Mouméne Zerari, master by research, research field: computer graphics and vision. Email: sssokba@hotmail.com.

- Silhouette Edge:

A silhouette edge is an edge shared by a front and back facing polygon. Front facing is any polygon visible from the light source. Back facing is any polygon non visible from the light source.

3. Stencil Shadow Volumes

Many methods were developed to allow the fast compute of the shadows parts of a sequence of images. One of these methods is known under the name of stencil shadow volumes [4]. It uses the geometry of the scene 3D to extract volumes. These volumes are then drawn in an image called “stencil buffer” to obtain a mask indicating which pixels are in the shadow of the point light source. There exist two basic techniques to draw these shadow volumes in the stencil buffer. They are known under the names of Z-pass [2] and Z-fail [5]. The first technique (Z-pass) presents some defects whose correction is made by Z-Fail method.

3.1 Algorithm Z-Pass

In 1991, Tim Heidmann presented the basic technique for rendering of Shadow Volumes is commonly called method Z-pass [2].

The principle of Z-pass is the following:

- Render front faces of volume, by incrementing the stencil when the test of depth is true, else anything. To disable the display of volume;

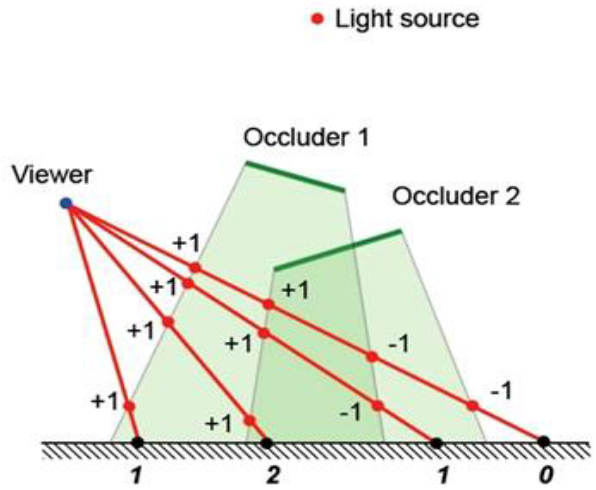

Fig. 2 Z-Pass technique.

- Render faces back of volume, by decrementing the stencil when the test of depth is true, else nothing. To disable the display of volume.

A point having a value of stencil different from zero is in the shadow (Fig. 2).

3.2 Algorithm Z-Fail

The opposite version was proposed in 2000 per John Carmack [5]. Indeed, Z-pass does not functioning if the observer is located in the shadow volume.

The principle of Z-fail is the following:

- Render faces back of volume, by incrementing the stencil when the test of depth is false, else anything. To disable the display of volume;
- Render front faces of volume, by decrementing the stencil when the test of depth is false, else anything. To disable the display of volume.

3.3 Comparison of Z-Pass and Z-Fail

The Z-Pass method has a higher performance than the Z-Fail method. But the Z-Pass algorithm fails when the shadow volume intersects the near clipping plane. This near clipping problem was the reason for the development of the Z-Fail technique, which processes shadow volume fragments that fails (instead of pass) the depth test. This approach moves the problems from the near to the far clipping plane which can be handled robustly by moving the far plane to infinity. However, this robustness comes at the expense of performance since in the Z-Fail case the shadow volumes must be closed at both ends.

3.4 Shadow Volume Disadvantages

Shadows volume require that the object which projects shadow have to be closed (each edge of the grid must be divided exactly by two polygons). Several pass of rendering are necessary to generate shadow volume what results in consuming much fill-rate. The calculation of silhouette can charge the CPU for dynamic scenes. The information of adjacencies between the primitive basic ones (calculated by the CPU) is required too.

3.5 Shadow Volumes Optimizations

Recently, some works tried successfully to improve the method of shadows volume based on optimization of fill-rate, Culling [6], the clipping, and Clamping [7], or by rendering optimizing shadow volume by using the shaders [8]. And finally silhouette determination optimizations by simplified occluder geometry or using Geometry Shader (our contribution).

4. Proposed Model

Our work is based on Ref. [8] the technique of Z-Fail and the extrusion of the silhouette is implemented by program GPU (shader), but in Ref. [8] the stage of silhouette generation (Create the face connectivity information, i.e., store the neighboring faces for every polygon in the mesh) is implemented in the CPU, i.e., it requires a pre-process of the geometry, because the information of adjacency between the primitives is absent.

Our analysis made on the limits of the method [8] enabled us to propose a method faster for generation of the silhouette, by an improvement using the new performances of the recent material graphic.

There was several research works to find an effective manner of detection the silhouette in real-time [4, 9-11], although the majority of the algorithms require a pre-process of the geometry or multiple rendering passes [12]. In the preceding cases there is no support geometry underlying the silhouette.

With the new stage of the geometry shaders of GPUs, a new possibility is open, since the information of adjacencies is available. Thanks to Geometry Shaders we can produce geometry of silhouette.

Our contribution consists in implementing the stage of generation of the silhouette on the GPU by using Geometry Shaders.

5. Shaders

The Shaders Programs are programs carried out by graphics card (GPU). They are used to replace the fixed functions implemented in the graphics card (cabled functions) by other functions written by the programmer [13]. They intervene at various levels of the graphic pipeline. The most used shaders are the Vertex and Shaders Fragment. In the modern graphics cards, the stage Geometry Assembly of the pipeline became programmable what gives to the programmers more flexibility to manage the vertices and their connectedness (addition or removal of vertices or edges). The Direct3D and OpenGL graphic libraries use three types of shaders:

- Vertex shader: They are carried out at the treatment of each vertex of the primitives;
- Fragment shader: Still called Shader Pixel, they intervene for the treatment of each pixel to display;
- Geometry shader: Geometry shaders can add and remove vertices from a mesh. Geometry shaders can be used to generate geometry procedurally or to add volumetric detail to existing meshes that would be too costly to process on the CPU. If geometry shaders are being used, the output is then sent to the rasterizer.

5.1 Geometry Shader

These last years, vertex and fragment shaders profited from a strong passion on behalf of the graphic community. Compared to CPU or old version of the GPU, they have the advantage of making it possible to accelerate computing times and to improve quality of rendering. The vertex shader does not receive information of connectivity (triangles or quads), it receives only the coordinates of the points, independently from/to each other.

The fourth version of the shader model introduced a new kind of treatment named geometry shaders [13], which manages the connectivity of the polygons like their subdivisions or simplifications. They are available on the very recent graphics cards. In the chain of operations of OpenGL, this last takes seat between the vertex shader and the phase of clipping. The objective of this shader is to be able to handle, on GPU, unquestionable primitive by making it possible to transform them, duplicate or to act on the vertices.

Being given a whole of elements in entry, the geometry shader makes it possible to obtain primitives like points, lines or triangles.

5.2 Using Geometry Shader to Implement Generation Silhouette

The first step is to create a mesh with adjacency information. This is done by creating a vertex buffer with three vertices per primitive, and then creating an index buffer containing the adjacent vertices in the proper winding order. The primitive-type triangle with adjacency must be declared in both the host code and the geometry shader constructor. As a result, the geometry shader gets access to vertex information from three triangles: the primary triangle and the three adjacent triangles for a total of six vertices, as it is shown in Fig. 3.

With this information we should test the primary triangle to see if it's front-facing by calculating the dot product of the face normal and the view direction. If the result is less than zero, we have a front-facing triangle and need to check whether it contains a silhouette edge.

If the current triangle is facing the light source. If it isn't, there's no need to perform the rest of the shader as the current triangle wouldn't be affected by light anyway, and thus the geometry shader can fall through with no output corresponding to the current triangle. If the triangle did face the light source, then we'd perform a few steps more in the geometry shader. For each edge of the triangle, it is determined whether that particular edge is on the silhouette of the geometry with regard to the light position. If it is, new triangles are created inside the geometry shader to construct the extrusion faces for that edge.

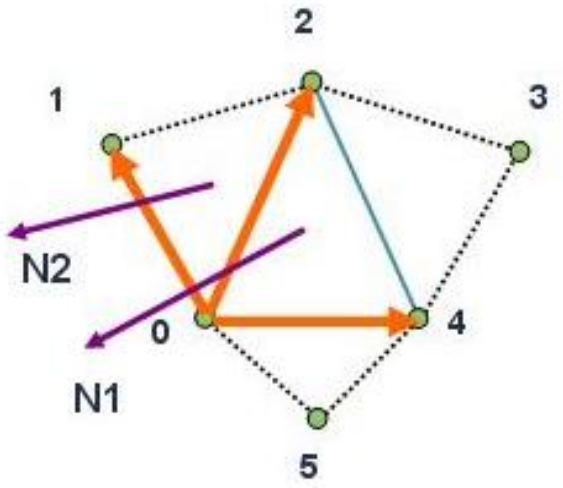

Fig. 3 The triangle currently processed is the one determined by vertices (0,2,4). The ones determined by vertices (0,1,2), (2,3,4) and (4,5,0) are the triangles that share an edge with the current triangle.

5.3 Geometry Shader Pseudo Code

Algorithm that extrudes the silhouette edge using the Geometry Shader pseudo code:

```
Calculate the triangle normal and view direction
    if triangle is front facing then
      for all adjacent triangles do
          Calculate normal
          if triangle is back facing then
            Extrude silhouette
          end if
           end for
          end if
```

6. Results and Discussion

The test results are base on the Intel (R) Pentium (R) D CPU 3.00GHz, nVidia GeForce 9400M GT, version OpenGl Core: 3.3 and version GLSL: 3.30 and 1024 MB memory with Windows operating system.

After having implemented the calculation of the shadow volume, we had the following results (Figs. 4-5).

Fig. 4 represents the result of implementation of the algorithm of the shadow volume [8] with the method of Z-Fail and the detection of the silhouette by the CPU, in this case our scene generates 17 FPS with the shadows activates. Fig. 5 represents the result of our implementation of the algorithm of the shadow volume with the method of Z-Fail and the detection of the silhouette by the GPU and silhouette generation implemented by Geometry Shader, in this case our scene generates 25 FPS with the shadows activates.

Here we see very well shadow volumes as well as the self-shadow.

Fig. 4 Implementation of the method [8].

Fig. 5 Implementation of our contribution.

Fig. 6 shows the values of the FPS according to many triangles in the scene, knowing that our scene in this case is made up of 10 objects. This same scene is render while using three methods different, the first by deactivating shadow volumes, the second by activating shadow volumes and by using the method of [8] (Fig. 4), and the third by activating shadow volumes and by using our proposed model (Fig. 5). The evaluations are represented in the graph of Fig. 6.

Compared with the CPU silhouette method, the GPU silhouette method has the same scalability but has much lower performance, which only could get about 50% FPS compared with the CPU one. As a creative technique the GPU silhouette did show a new approach to let the GPU do some general computation, but it is still not mature.

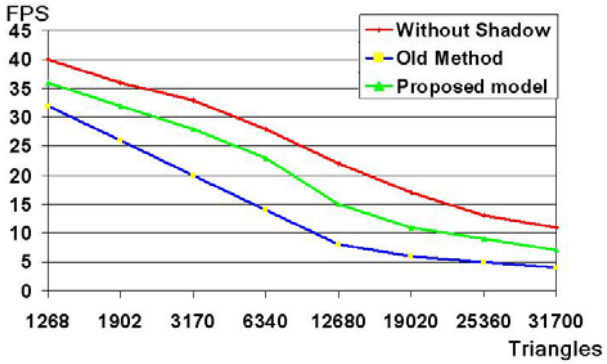

Fig. 6 Scalability when triangles increase in CPU and GPU shadow volume.

7. Conclusions

The integration of the geometry shaders for calculates the silhouette for shadow volumes to show its effectiveness in the optimization of the computing time, which enabled us to obtain the generation of the scenes in real-time. The contribution that we presented in this paper constitutes a first stage, thus we plan in the future to extend our study for other more complex scenes, and integration of the levels of details in the generation of the shadows.

8. Future Work

We are currently exploring some optimizations of our method by adding some features like infinite view frustums, occluder cullings, clamping and choosing z-pass or z-fail automatically. Another suggested future work is to explore the performance of GPU silhouette in other use like NPR, etc.

References

[1] F.C. Crow, Shadow algorithms for computer graphics, in: Proceedings of SIGGRAPH 77, Juillet 1977, pp. 242-248.

[2] T. Heidmann, Real shadows real time, Iris Universe 18 (1991) 23-31.

[3] S. Brabec, H.P. Seidel, Shadow volumes on programmable graphics hardware, Computer Graphics Forum 22 (2003) 433-440.

[4] J. Kainulainen, Stencil shadow volumes, Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, April 15, 2002.

[5] J. Carmack, On Shadow Volumes, available online at:

http://developer.nvidia.com/attach/5628, 2000.

[6] N. Govindaraju, B. Lloyd, S. Yoon, A. Sud, D. Manocha, Interactive shadow generation in complex environments, ACM Transactions on Graphics 22 (2003) 501-510.

[7] B. Lloyd, J. Wendt, N. Govindaraju, D. Manocha, CC shadow volumes, in Proc. Eurographics Symposium on Rendering, Eurographics, Eurographics Association, 2004.

[8] G. Wallner, Geometry of real time shadows, Scientific and Professional Journal of Croatian Society for Geometry and Graphics, No. 10, 2006, pp. 37-45.

[9] J. Doss, Inking the cube: edge detection with direct3D 10, August 2008.

[10] C. Dyken, M. Reimers, J. Seland, Realtime gpu silhouette refinement using adaptively blended bézier patches. Computer Graphics Forum 27 (2008) 1-12.

[11] Y. Shi, Performance comparison of CPU and GPU silhouette extraction in a shadow volume algorithm, Master's Thesis in Computing Science, Department of Computing Science, Sweden, January 27, 2006.

[12] J.R. Randi, L.K. Bill, OpenGL Shading Language, 3rd ed., Addison-Wesley, 2009.

[13] B. Lichtenbelt, P. Brown, EXT_gpu_shader4 Extensions Specifications, NVIDIA, 2007.