Geometry Image-based Shadow Volume Algorithm for Subdivision Surfaces

Min Tang, Jin-xiang Dong

College of Computer Science, Zhejiang University, P.R. China, 310027

{tang_m, djx}@zju.edu.cn

Abstract How to generate shadow volumes efficiently for subdivision surfaces remains a challenging task for computer graphics community. We present a geometry image based algorithm that runs on GPUs (Graphics Process Units). By using GPU shaders, two geometry images containing position and normal vector information will be computed from the control meshes. By detecting silhouettes and extruding shadow volumes from the geometry images using shaders, all calculations of the shadow volume algorithm can be fully accelerated by commodity GPUs. A prototype system has been implemented on a Windows 2000 system with a NVIDIA GeForce 7950GX2 card. Experiments show that the algorithm can render subdivision surfaces and their shadows in real-time under dynamic lighting. Source code is available online.

Keywords Subdivision surface · Geometry image · Silhouette detection · Shadow volume · GPU (Graphics Process Unit)

1 Introduction

As a powerful modeling tool, subdivision surfaces have, in recent years, replaced NURBS surfaces as the preferred method of modeling character and scene geometry in the film and television visual effects industries. Subdivision surfaces are easy to use and implement, they can model surfaces of arbitrary topological type, and control the continuity of the surface locally.

Fast shadow rendering for subdivision surfaces is essential for using them in real-time applications. As far as known to the authors, there is little research on shadow rendering algorithm especially designed for subdivision surfaces. The conventional methods usually need to convert subdivision surfaces to facet models according to a prescribed resolution, then shadow algorithms for polyhedron models are used.

For polyhedron models, there exist two classic shadow generation algorithms: shadow mapping algorithm [1] and shadow volume algorithm [2]. Both of them have been proved to be successful under real-time circumstances.

As an object space algorithm, shadow volume algorithm can generate more accurate shadows, while shadow mapping algorithm is well scalable as an image space algorithm.

For a complex model defined by many subdivision surfaces, tons of facets will be produced after a few subdivisions on CPU. And all the data will be transferred from CPU to GPU for model rendering and shadow rendering. This is a serious bottleneck for conventional methods, especially when the complex model is deforming. Also, the powerful computation ability of commodity GPUs is not fully exploited.

Our discussion will focus on how to effectively adapt the shadow volume algorithm, initially designed for polyhedron modes, to subdivision surfaces.

Our contribution: By packing the subdivision surfaces of a model into two geometry images, silhouette detection and shadow volume extrusion methods based on the geometry images is designed. A purely graphics hardware accelerated shadow volume algorithm for subdivision surfaces is designed. The methods can also be used for other geometry image related graphics applications.

2 Related Work

Subdivision Surface Evaluation on GPU: [3] presents a real-time kernel for subdivision surface evaluation by organizing the control mesh of subdivision in texture memory carefully. [4] uses a vertex shader to generate mesh refinement on GPUs. As an application, Curved PN-Triangles have been implemented using a refinement pattern. [5] proposes a per-pixel evaluation method of parametric surfaces by using fragment shaders. [6] uses GPU for NURBS and T-Spline surfaces tessellation, and a trim-texture is used for trimming.

Real-time Shadow Volume Algorithm: [7] enhances the robustness of the classic shadow volume algorithm by using a few useful techniques including z-fail stencil-testing scheme. [8] presents a purely hardware-accelerated shadow volume algorithm by packing mesh data into textures and by processing on GPU. [9] and [10] extend classic algorithm to support soft shadow rendering by using penumbra wedges. [11] covers the details about how to implement a real-time shadow volume algorithm with the help of GPU. [12] is an early attempt at including free-form surfaces into classic shadow volume algorithm by solving parametric equations, but its speed makes it unfeasible for real-time applications.

Geometry Image: As first introduced by Gu et al. [13], geometry images have become a powerful tool for geometry processing. By storing geometry data as GPU-

friendly textures, many operations can be accelerated using graphics hardware, such as mesh simplification, mesh deformation, global illuminations [22], etc.

Silhouette Detection is used to extract silhouettes, which will be extruded to form shadow volumes. All the papers on shadow volume algorithm will describe detection scheme [7-12]. Silhouette detection is also widely studied in NPR (Non-Photorealistic Rendering) field. [15] and [16] give out good surveys on related algorithms and implementations. Our geometry image based silhouette detection method will be presented in Section 5.

Shadow Volume Algorithm for Subdivision Surfaces: In [17], a shadow volume algorithm based on SPs (Subdivision Patterns) is designed for subdivision surfaces. But there are some problems in it:

- Each patch is handled for evaluation, silhouette detection and shadow volume extrusion respectively. The latency caused by frequent render-to-texture ops (num_of_patch*3 per model) dramatically reduces its efficiency.
- The instability of its silhouette detection method occasionally causes cracks on shadow volumes.

Our improvement: The work here is an extension of [17]. By using two geometry images to pack all the patches' information, the efficiencies of silhouette detection and shadow volume generation are greatly improved (2 render-to-texture ops per model under dynamic lighting). And an enhanced silhouette detection

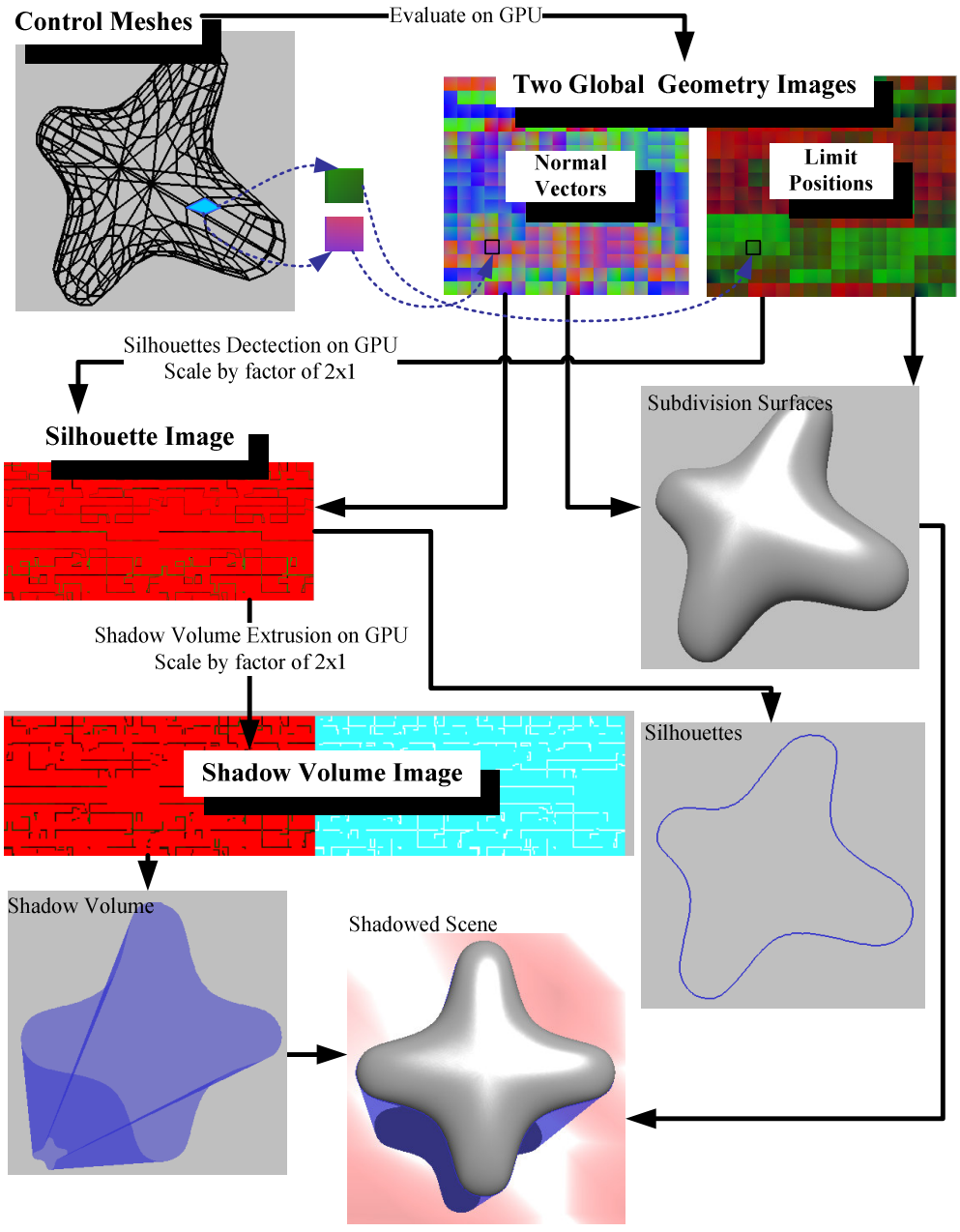


Fig. 1 The work flow of the shadow volume algorithm for subdivision surfaces

method is designed to improve accuracy and ensure robustness.

3 Overview

As illustrated by Figure 1, the shadow algorithm for subdivision surfaces is composed by following steps:

1. Subdivision surfaces evaluation

Control meshes of the model are used for evaluation by using a fragment shader. For every patch, two geometry images containing position and normal vector information are generated. The geometry images are placed sequentially into two global geometry images.

The global geometry images will be used for rendering the model and for subsequent silhouette detection.

2. Silhouette detection

Another fragment shader is used for silhouette detection on the geometry images. Assuming the size of the geometry images are W*H, a silhouette image of size (2*W)*H, containing silhouettes detected for a given light source, will be generated.

The silhouette image will be used for subsequent shadow volume extrusion. It can also be used for rendering the silhouettes.

3. Shadow volume generation

The shadow volume extrusion is also carried out in a fragment shader. A shadow volume image of size (4*W)*H will be computed. It will be used for rendering the shadow volumes.

4. Rendering the shadowed scene

With the rendering of the model and the shadow volume, the Z-pass method is used to render the shadowed scene.

4 Subdivision Surface Evaluation

For a Catmull-Clark subdivision surface, the analytic evaluation method in [14] is used to evaluate it at a given resolution. By projecting the control mesh into the eigenspace, the limit position can be calculated as the weighted combination of the projected control mesh:

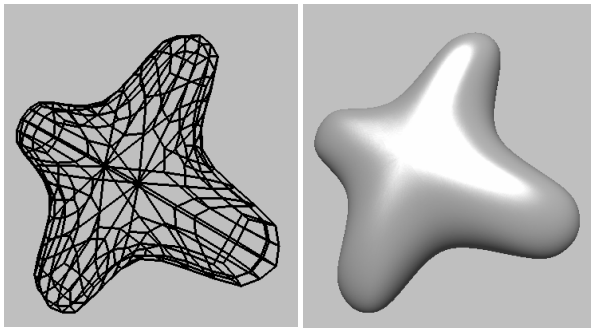
$$S_{k,n}(u,v) = (X^{-1}C_0)^T \Lambda^{n-1} (P_k \overline{A} X)^T b(u,v) \quad (1)$$

where A is the subdivision matrix, X is the matrix of eigenvector of A, and Λ is the diagonal matrix of eigenvalues. To accelerate the computation, the terms independent of the control mesh are packed into $Weight(u,v)$. We have [5]:

$$\begin{aligned} Weight(u,v) &= (X^{-1})^T \Lambda^{n-1} (P_k \overline{A} X)^T b(u,v) \\ S_{k,n}(u,v) &= C_0^T Weight(u,v) \end{aligned} \quad (2)$$

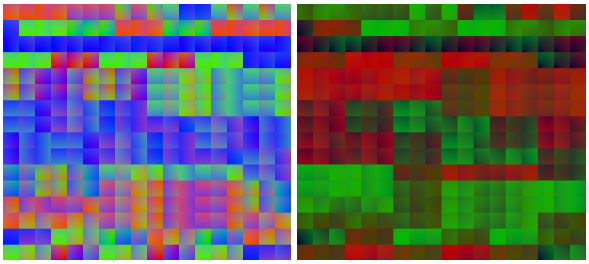
All the $Weight(u,v)$ are pre-calculated according to the parameters (u, v). The evaluation is preformed by combining the input control mesh and the weights in a fragment shader. The evaluation results are two geometry images storing limit positions and normal vectors respectively.

The control mesh in Figure 2(a), which consist of 288 patches, has been evaluated on GPU. The evaluation information for each patch is stored in the 16x18 = 288 lattices of two global geometry images (Figure 2(c) and Figure 2(d)). By using the positions and normal vectors in the two geometry images, the subdivision surfaces are displayed (Figure 2(b)).



(a) Control meshes

(b) Rendering using the two geometry images



(c) The geometry image storing normal vectors

(d) The geometry image storing limit positions

Fig. 2 Subdivision surface evaluation on GPU

5 Silhouette Detection

Before give out our silhouette detection method, we make a brief review on existing techniques.

[18] describes a graphics hardware accelerated method for silhouette extraction by using a vertex shader to enlarge the degenerate quads attached on edges to proper width when silhouettes are detected. By adding one degenerate quad to each edge, it aggravates the load of GPU. [19] uses the new features of DirectX 10 to detect silhouettes and extrude shadow volumes on a geometry shader.

[8] determines contour edges on the fragment processor using vertices encoded as color values. For each edge, indices for two end points and two adjacent points are read from an edge texture. Then the indices are used to get vertices from a vertex texture. The multiple dependent texture reads compromise its efficiency.

[20] uses normal maps to find out pixels on geometry images, corresponding to vertices on silhouettes, and links them by searching nearest neighbors in image space. The method is feasible for NPR applications, but is inadequate

for shadow volume algorithm, which need silhouettes to be neatly linked in geometry space.

In [17], an edge is marked as a silhouette by using the visibilities of the two facets meeting at it. As an object-space algorithm, silhouettes are properly linked. But the numerical instability occasionally causes jagged artifacts and gaps in silhouette chains.

Here a new method for silhouette detection based on the two geometry images is designed:

- The visibility of each vertex is calculated by using the position and normal vector information in the two geometry images;
- For each quad, corresponding to a grid on the geometry images, by using a directed visibility interpolation method in Figure 4, a silhouette will be computed. The two end points of the silhouettes will be saved in a silhouettes image. If no silhouette on the quad, a degenerated silhouette with two coincidence end points is produced. So for the geometry images of size W*H, a silhouette image of (2*W)*H containing the end points of silhouettes will be produced (Figure 3(a)).
- The silhouette image will be used for silhouettes rendering (Figure 3(b)), and subsequent shadow volume generation.

(a) a silhouette image (b) silhouettes

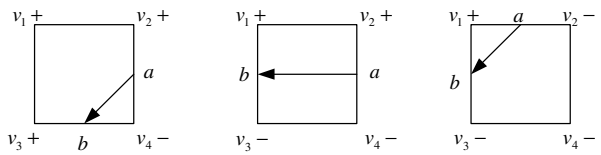
Fig. 3 Silhouette detection on GPU

Directed Visibility Interpolation:

To enhance the robustness of our silhouette detection method, we used a directed visibility interpolation method, which is inspired by the method dealing with smooth surfaces in [21].

For a quad, we check the visibilities for its four vertices. If:

- all vertices are visible/invisible, then a degenerated silhouette is generated;
- if one is visible/invisible, other three are invisible/visible, a silhouette is linked as shown in Figure 4(a)/Figure 4(c);



(a) $a=(v_2+v_4)/2$, $b=(v_3+v_4)/2$ (b) $a=(v_2+v_4)/2$, $b=(v_1+v_3)/2$ (c) $a=(v_1+v_2)/2$, $b=(v_1+v_3)/2$

Fig. 4 Directed visibility interpolation for silhouette extraction (+ for visible, - for invisible)

- if two are visible, and two are invisible, a silhouette is linked as shown in Figure 4(b).

Note that the orientation of the silhouettes in Figure 4 is decided by the right-hand rule, i.e., the visible part of the model is on the right of a silhouette. The orientation rules guarantee that the silhouettes are linked into directed loops.

Figure 5 illustrates a comparison between silhouettes obtained from [17] and from our method. It is clear that the silhouettes in Figure 5(b) are linked more smoothly, and are more adhere to the variation in visibility.

Figure 6 shows some experiment results of [17] and our method. By comparing Figure 6(a) and Figure 6(b), the silhouettes detected by our method is more accurate, and the gap in Figure 6(a) is removed. The cracks in Figure 6(c) caused by numerical instability are mended in Figure 6(d), and more smooth silhouettes computed.

With the smooth linked silhouettes, cracks on the shadow volume in Figure 6(e) are eliminated in Figure 6(f).

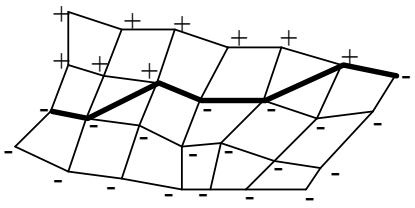
The whole procedure of silhouette detection is done on GPU with the help of a fragment shader. As a natural continuation of previous evaluation process, data are efficiently transferred among GPU shaders as textures.

Our method is good to manipulate geometry stored as textures. For using the methods in [18] and [19], another calculation for converting textures to vertex array is needed. Comparing to [8], multiple dependent texture reads is avoid by accessing data locally with the regular data layout in the geometry images.

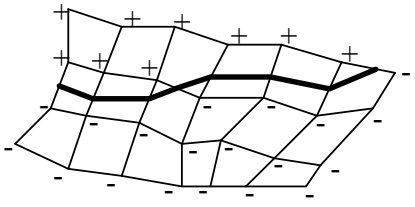
6 Shadow Volume Generation

As shown in Figure 7, a fragment shader is used to generate a shadow volume image from the silhouette image.

For a silhouette image of size (2*W)*H, it is scaled by a

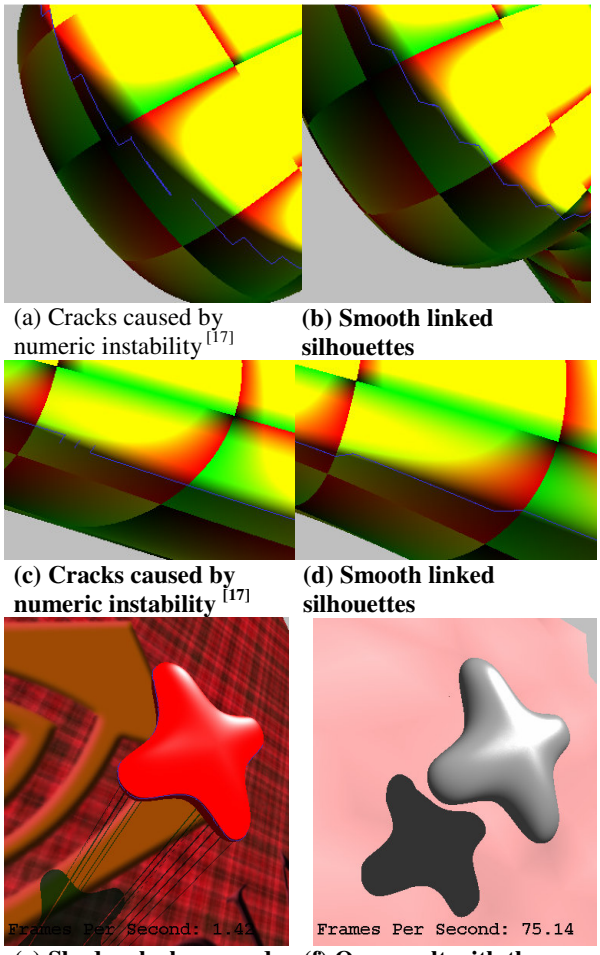


(a) Silhouette detection using normal vector of adjacent facets in [17]



(b) Silhouette detection using the directed visibility interpolation method

Fig. 5 Comparison of silhouettes getting from [17] and our method



(a) Cracks caused by numeric instability [17]

(b) Smooth linked silhouettes

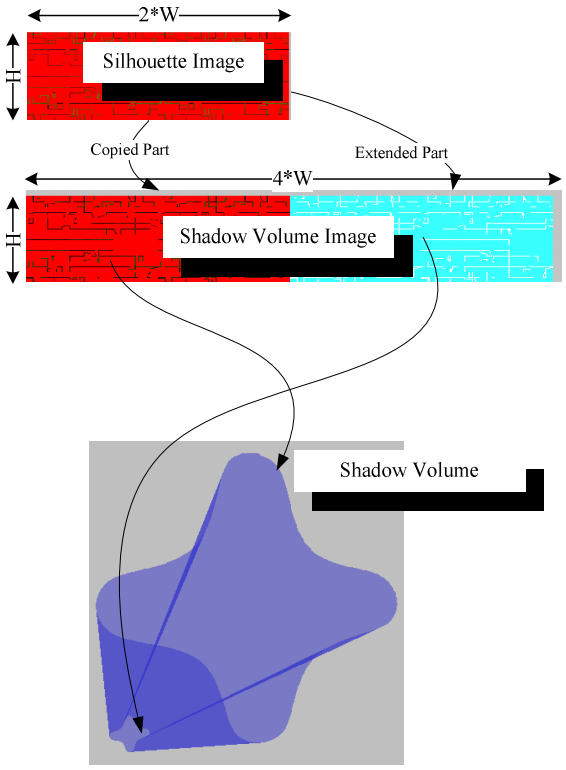
(c) Cracks caused by numeric instability [17]

(d) Smooth linked silhouettes

(e) Shadow leaks caused by the cracks [17]

(f) Our result with the same light source

Fig. 6 Experiment results of [17] and our method



factor of 2 to generate a shadow volume image of size (4*W)*H, which can is composed by two parts of size (2*W)*H The left part is a copy of the silhouettes image, and the right part contains new vertices by extending the vertices in the silhouette image:

$$Y = X + (X - O) * \infty \qquad (3)$$

where X are the vertices on silhouettes, O is the light source, and ∞ stands for infinite distance[1]. By connecting the vertices on the silhouettes and the vertices extended to infinite, shadow volumes are built. With the shadow volume image, the shadow volume can be displayed.

7 Implementation and Results

The shadow volume algorithm has been implemented on a Windows 2000 system with a NVIDIA GeForce 7950GX2 card. We use OpenGL as 3D API, and use Cg and GLSL to write GPU shaders. The FBO (Frame Buffer Object) extension is used for render to texture, and the PBO (Pixel Buffer Object) extension is used for construct vertex arrays from textures. An optimization is made by combining the silhouette detection and shadow volume extrusion in one fragment shader. All the source codes of our implementation are available online at the address listed at the end of this paper.

GeForce 7950GX2 supports textures with maximum size of 4096x4096. So the maximum dimension of shadow volume textures can be 4096*4096. The dimension of silhouette textures can up to 2048*4096, and the geometry images can up to 1024*4096. If the patches are evaluated at resolution of 10*10, approximately 1024*4096/100 ≈ 40K patches can be handled in one pass. If patch number is bigger than 40K or the maximum size is limited by the underlying graphics hardware, multi-pass rendering techniques can be used.

Comparison with conventional methods:

For conventional methods, the model is faceted on CPU according to a prescribed resolution. Massive data need to transfer from CPU to GPU. As shown in Table 1, for the model in Figure 1, if the subdivision level is 5, there are about 3.5M float data need to transfer to GPU.

[1] OpenGL provides support for extending vertices to infinite distance by setting the w component of homogeneous coordinates to 0.

Table 1 Data transferred from CPU to GPU in conventional methods

GLfloat	Usage
32*32	Resolution for each patch (2^5*2^5)
*288	Totally 288 patches
*2	Position and normal vector
*3	x, y, z
*2	Silhouette detection/shadow volume extrusion using vertex shader
~3.5M	

For our method, only the control mesh needs to transfer to GPU. As shown in Table 2, there are about 28K float data will be transferred.

Table 2 Data transferred from CPU to GPU in our method

GLfloat	Usage
288	Patches
*16	Control mesh
*2	Position and normal vector
*3	x, y, z
~28K	

Another benefit of our method is that the subdivision level of the model can be dynamic adjusted. This will be useful for LOD scheme.

Comparison with the method of [17]:

If the model is stationary, for a moving light source, we only need to update the shadow volume textures, i.e., 1 render-to-texture ops per moving of the light. So comparing to the (3*num_of_patches) render-to-texture ops per frame in [17], the speed is greatly improved.

Figure 8(a)-(c) demonstrate some of our results. For the models defined by Catmull-Clark subdivision surfaces, its silhouettes and shadows are displayed in real-time. For the model in Figure 8(a), result of [17] is about 15 fps, while our result is about 75 fps.

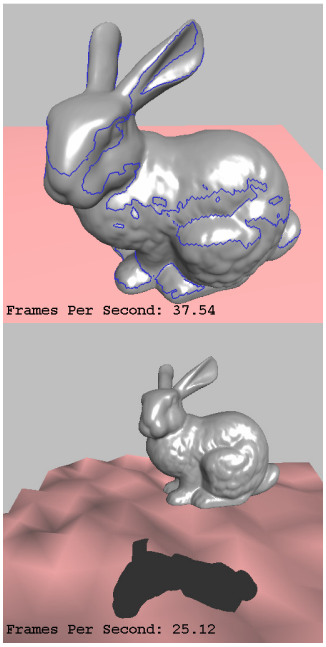


Fig. 9 Silhouettes detection and shadow rendering for the bunny model defined by a geometry image.

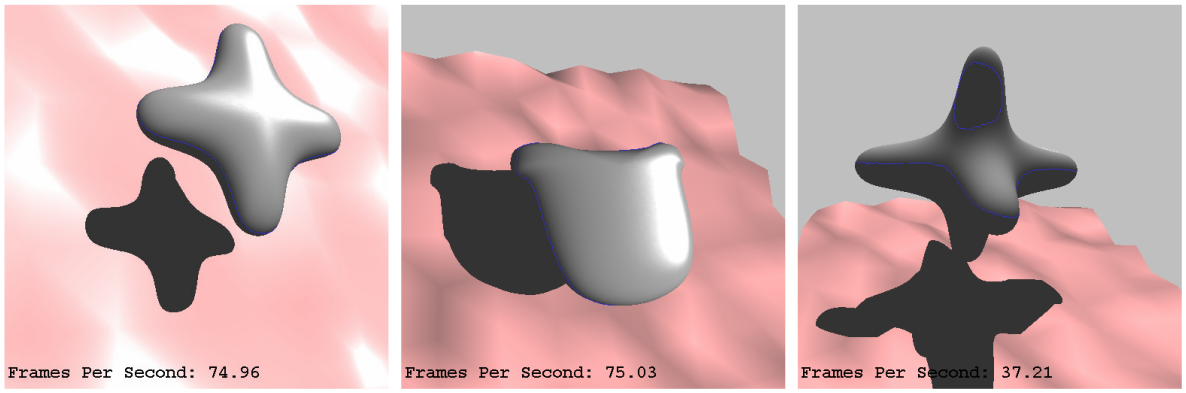
8 Extensions

The application of our algorithm is not limited to subdivision surfaces. It can also be used for other surfaces which have a parametric evaluator, e.g. Bézier surfaces, B-spline surfaces, etc.

Figure 9 shows the silhouettes and shadows of the bunny model, which is defined by a geometry image.

Figure 10 shows the silhouettes extracted for cloth simulation.

For other models which can be defined using geometry images, such as terrains, water surfaces, etc., our method can also easily adapted.



(a) Shadows and silhouettes of the mode in Figure 1 (288 patches), ~75fps

(b) shadows and silhouettes of a face-shaped model (288 patches), ~75fps

(c) shadow and silhouettes of a complex model (480 patches), ~37fps

Fig. 8 Some experiment results

9 Conclusion and Future Work

In this paper, two geometry images are used to store the evaluation information of subdivision surfaces. Geometry image based silhouette detection and shadow volume extrusion methods are designed. A directed visibility interpolation method is used to enhance the robustness of silhouettes detection method.

Experiments show that the GPU accelerated algorithm can fulfill the demand for real-time rendering, and our methods can also be extended to other geometry images defined models.

The efficiency bottleneck of our algorithm is the phase of "construct vertex array from texture". In our implementation, we used PBO (Pixel Buffer Object) extension, which is superior to VTF (Vertex Texture Fetch) method. But as the size of a geometry image is growing, its efficiency is rapidly dropped down. We expect this limitation could be solved in future GPUs.

Future works include geometry image-based capping on GPU. So Z-fail method could be implemented. Geometry image-based soft shadow rendering may be another interesting topic for further study.

Acknowledgements

This work was supported in part by Science Supporting Project of Hangzhou (20062422B05), and Chinese National Key Basic Research and Development Program (2006CB303106).

Web Information

Binaries, source files, and some AVI files are available from this link:

http://www.cs.wichita.edu/~tang/SubShadow/subShadow.htm.

References

1. Williams, L. 1978. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, 270-274.
2. Crow, F.C. 1977. Shadow algorithms for computer graphics. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, 242-248.
3. Shiue, L-J., Jones, I., and Peters, J. 2005. A Realtime GPU Subdivision Kernel. In *Proceedings of SIGGRAPH Computer Graphics Proceedings,* 1010–1015.
4. Boubekeur, T. and Schlick, C. 2005. Generic Mesh Refinement On GPU, *In Proceedings of ACM SIGGRAPH/Eurographics Graphics Hardware*.
5. Kanai, T., and Yasui, Y. 2004. Per-pixel evaluation of parametric. surfaces on gpu. In *Proceeding of ACM Workshop on General Purpose Computing on Graphics Processors.*
6. Guthe, M., Balázs, Á., and Klein, R. 2005. GPU-based trimming and tessellation of NURBS and T-Spline surfaces. *ACM Trans. Graph.* 24(3): 1016-1023.
7. Everitt, C. and Kilgard, M. J. 2002. Practical and Robust Stencil Shadow Volumes for Hardware Accelerated Rendering. Austin, Texas. NVIDIA. Referenced: 5.4.2002. http://developer.nvidia.com/object/robust_shadow_volumes.html
8. Brabec, S. and Seidel, H. 2003. Shadow Volumes on Programmable Graphics Hardware. In *Proceedings of Eurographics (2003)*, vol. 22, pp. 433-440.
9. Assarsson, U., and Akenine-Möller, T. 2003. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3), 2003.
10. Laine, S., Aila, T. Assarsson, U., etc. 2005. An Improved Physically-Based Soft Shadow Volume Algorithm. *Computer Graphics Forum 25(3) (EUROGRAPHICS 2006)*, pp. 303-312.
11. MGuire, M. 2004. Effective Shadow Volume Rendering, *GPU Gems*, Chap. 9, Addison Welsey, 2004, pp. 137-166.
12. Hefline, G. and Elber, G. 1993. Shadow Volume Generation from Free Form Surfaces. In *Proceedings of CGI'93*.
13. Gu, X., Gortler, S.J., Hoppe H., 2002. Geometry Images, in *SIGGRAPH(2002)*, pp. 355-361.
14. Stam, J. 1998. Exact Evaluation of Catmull-Clark Subdivision Surfaces at *Arbitrary* Parameter Values. In *Computer Graphics Proceedings*, Annual Conference Series, 395-404.
15. Isenberg, T., Freudenberg, B., Halper, N., et al. 2003. A developer's guide to silhouette algorithms for polygonal models. *Computer Graphics and applications*, IEEE, 2003,23(4): 28-37
16. Wang, A-Y., Tang, M., and Dong, J-X. 2004. A survey of silhouette detection techniques for non-photorealistic rendering, *Proceedings of Third International Conference on Image and Graphics*, Dec 18-20 2004, Hong Kong, 434-437.
17. Tang, M., Dong, J-X., and Chou, S-C. 2006. Real-time Shadow Volumes for Subdivision Surface Based Models, *Proceedings of Computer Graphics International 2006*, pp. 538-545, June 26-28, 2006
18. McGuire, M. and Hughes, J.F. 2004. Hardware Feature Edges, In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, 35-147.
19. Tariq, S. 2006. DirectX10 Effects, http://developer.download.nvidia.com/presentations/2006/siggraph/dx10-effects-siggraph-06.pdf, NVIDIA Crop. 2006.
20. Yuan, X., Nguyen, M.X., Zhang, N., Chen, B. 2005. Stippling Silhouettes Rendering in Geometry-Image Space, in *Proceedings of Eurographics Symposium on Rendering*, 193-200. Konstanz, Germany June 29- July 1, 2005.
21. Hertzmann, A. 1999. Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines, *Non-Photorealistic Rendering (Siggraph 99 Course Notes)*, S. Green, ed., ACM Press, 1999.
22. Carr, N.A., Hoberock, J. Crane, K., and Hart J.C. 2006. Fast GPU ray tracing of dynamic meshes using

geometry images. *In Proceedings of Graphics Interface*.
A.K. Peters, 2006.