Improving Shadow Aliasing with Smoothies

Jyun-Ming Chen, Yi-Fan Tsai
Department of Computer Science and Engineering, Tatung University, Taiwan
jmchen@ttu.edu.tw, g9306011@ms2.ttu.edu.tw

ABSTRACT

Shadows can increase realism of a scene and help a viewer to understand the relative position of each object. However, realistic shadows can be very difficult to compute in real-time.

The article focuses on the generation of soft shadow. Shadow map is used to render the hard shadow; smoothie map is used to soften the aliasing edges from shadow map. In addition, the silhouette searching algorithm has been enhanced to improve the performance.

Keywords: shadow, shadow map, silhouette, smoothie.

1: INTRODUCTIONS

Shadow can exactly show the relative position among objects. For instance, the shadow of several moving objects can express the exact position. The shadow of a bouncing ball can express when the ball hits ground.

Presently, the dominant shadow algorithms are shadow map[1][3][4][5][7] and shadow volume[1]. The shadow computed by these two methods is hard shadow. The result from shadow volume is more precise, but shadow map has higher performance. Furthermore, shadow map and shadow volume are both hard to implement in software. Shadow map is more amiable to hardware implementation. The article uses shadow map and extra computation to create soft shadow.

2: CONSTRUCTION OF SMOOTHIES

In manifold models, there are exactly two neighboring faces for each edge. When an edge is front-facing (to a light source) and the other is back-facing, this edge is a silhouette edge[8][9]. The projection of silhouette edges is usually the border of shadow. By using this property of silhouette edges, the border aliasing of shadow map can be solved. The method combines geometry-based smoothies[2] and image-based shadow map to reach the goal.

A smoothie is based on a silhouette edge. A main focus is to find more efficient algorithm to locate the silhouette edges. Our method is largely motivated by the ideas of Hall [9]. His method first locates a candidate silhouette edge. Then the neighboring faces of this edge are checked to locate subsequent silhouette edges, until a checked face is met. We check neighbor edges instead of neighbor faces and then construct smoothies. The algorithm is explained in detail as follows.

2.1: SEARCH OF NEIGHBOR SILHOUETTE

To find the complete silhouette edges, the first step is to find a starting silhouette E_0 in the model. Then, start from E_0, find its neighbor edges and check if one of them is a silhouette edge. Repeat the same steps with detected silhouette edges. The repeated steps stop when the based edge meets E_0, and a complete silhouette list S_0 is found. When the light is moving, the silhouette list will be varied with light position. The silhouette list could be adjusted by the variation in S_0.

Figure 1 is a simple example. S_0 is the silhouette list data of previous frame, and S_0' is the next frame. When the light position is changed, original silhouettes, E_2 and E_3 in S_0, are also changed. They are not silhouette edges, and have to be deleted. After deleting the two edges, check E_1's neighbor edges to find new silhouette edge. E_5 is the new silhouette edge and then check its neighbor edges. Because E_5's neighbor edges have a checked silhouette edge, the check action stops and completes a silhouette list. The remaining steps begin from silhouette edge E_4. New silhouettes E_6 and E_7 can be found in the same way. Check all silhouette lists and correct the changes.

In general, models can be classified into two types: convex models and concave models. An edge with an inner (solid) angle greater than 180 degrees is called a convex edge. A concave model is a model with at least one non-convex (concave) edge. Otherwise, it is a convex model. The method mentioned above is appropriate for convex models. A convex model has no concave edge and has only one convex silhouette. On the other hand, a concave model can have several concave silhouettes.

Figures 2(a) and 2(b) compare the two types of models. Fig. 2(a) is a convex model and has only one convex silhouette. Fig. 2(b) is a concave model and has two silhouettes. The rectangular silhouette is a convex silhouette. The other triangular one is concave. The inner angle of pentahedron's base is larger than 180 degrees that forms concave silhouette.

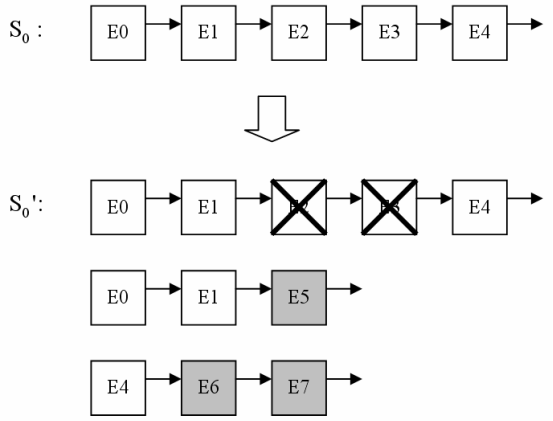
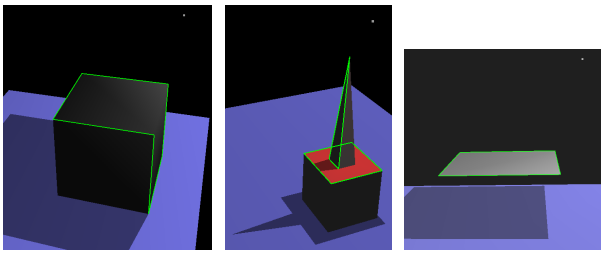


Figure 1 The process of searching silhouette neighbor edges

Before finding the silhouette edges in a concave model, concave edges have to be saved in concave list, and border edges have to be saved in border list. In Fig. 2(c), four edges of the quadrangle are border edges. Each of them has only one neighbor face. In addition, the checks of concave list and border list is necessary when detecting silhouette edges. The checked silhouette edges do the same as E_0 in Fig. 1 to find complete silhouette lists.

The method of searching neighbor silhouettes can increase performance of silhouette algorithm. The result of search is in Fig. 3. However, this method only searches neighbor silhouettes in one way that has an error. The error happens on bifurcation of edges and misses part silhouette edges. In Fig. 4, the thin black lines are checked silhouette edges and the thick lines with arrow are incomplete silhouette edges. The incomplete silhouette edges end check when they meet a checked silhouette edge. In Fig. 4, dotted lines illustrate the missing silhouette edges. Restricted by the method, the



(a) **(b)** **(c)**

Figure 2 Different kinds of silhouettes

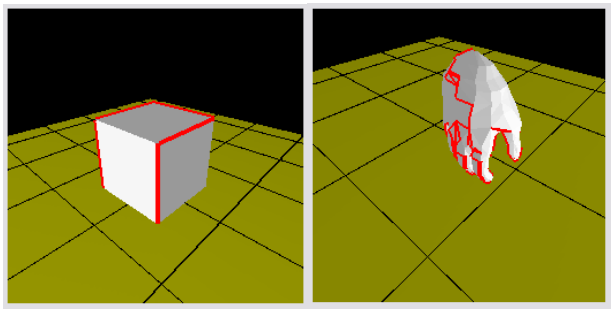


Figure 3 Results of searching neighbor silhouettes

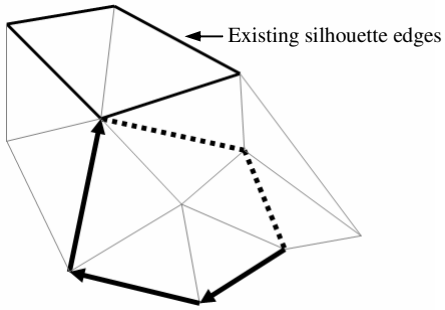


Figure 4 The error of one-way search

missing silhouette edges cannot be found. It is possible that concave models can miss silhouette edges, but not for convex models. The missing part makes incomplete soft shadow.

The error of missing silhouette edges will be accumulated as time goes on. In order to decrease the error, we search the silhouette edge again every time T. Smaller T will cause greater precision.

2.2: SMOOTHIE CONSTRUCTION

A basic smoothie is an extended rectangle with width D. It extends from a silhouette edge. The extended direction of a smoothie is decided from the bisector B of the corresponding silhouette edge, as shown in Fig 5. The direction of smoothies must be outward from the model. Inward smoothies are useless.

Figure 6 illustrates how to build smoothies. The vertices v_0 and v_1 extend two vertices v_0' and v_1'. The four vertices form a smoothie. Their calculations are mentioned below.

$$B = (a,b,c) \tag{1}$$

The direction of $\overrightarrow{v_0v_0'}$ and $\overrightarrow{v_1v_1'}$ are both B, so:

$$\overrightarrow{v_0v_0'} = \overrightarrow{v_1v_1'} = B = (a,b,c) \tag{2}$$

Equation 2 could be rewritten as a parametric equation Eq. 3.

$$\begin{aligned} v &= (x,y,z) \rightarrow \\ v' &= (x',y',z') = (x+at, y+bt, z+ct) \quad t \in R \end{aligned} \tag{3}$$

v is a vertex on the silhouette edge and v' is the extended vertex. The width of $\overline{v_0v_0'}$ and $\overline{v_1v_1'}$ is D which is decided by user. Eq. 4 is derived from these information and distance equation.

$$\overline{vv'} = \sqrt{(x'-x)^2 + (y'-y)^2 + (z'-z)^2} \tag{4}$$

Then, substitute D and parametric equations of v_0 and v_0' for parameters in Eq. 4. The solved answer is t value. See Eq. 5.

$$t = D / \sqrt{a^2 + b^2 + c^2} \tag{5}$$

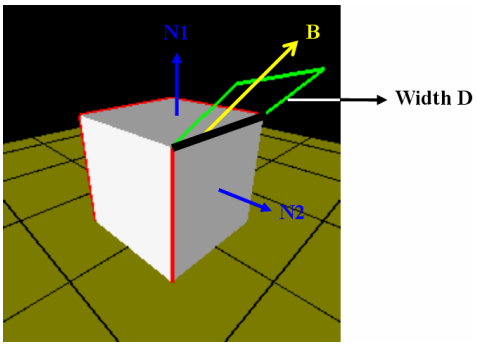


Figure 5 The extended direction of a smoothie

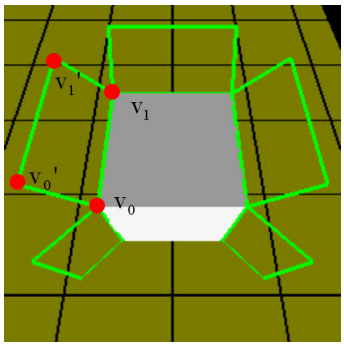


Figure 6 Illustration of smoothies

Finally, substitute t for parameters in parametric equation and solve v_0'. The value of v_1' can be solved in the same way. After these computations, the four vertices are known. The basic smoothie is formed by linking v_0, v_0', v_1, and v_1'. Fig. 7 shows an example of cube. The outlined rectangles are the smoothies.

Basically, the width of smoothies depends on the size and complexity of models. Larger models have wider width; more complex models have thinner width. Smoothies are not really better when they are wider. Suitable D is the best. A small model with wide smoothies will make the soft shadow much bigger than the model. On the contrary, too thin smoothies will be hard to improve aliasing problem. The width of smoothies is not absolute. Different widths have different effects. Figure 8 illustrates the result which uses two different widths of smoothies. In Fig. 8(a), the width of smoothies is twice the width of cube. In Fig. 8(b), the width is one third time the width of cube. Two results are quite different.

Figure 7 shows incomplete smoothies. Just link the two vertices of neighbor smoothies. The result is in Fig. 9.

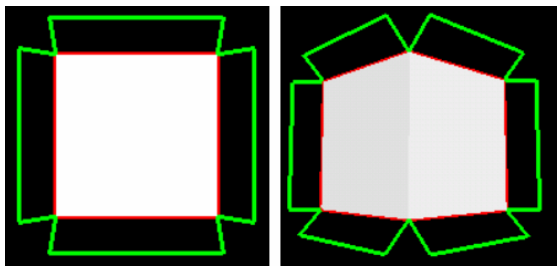
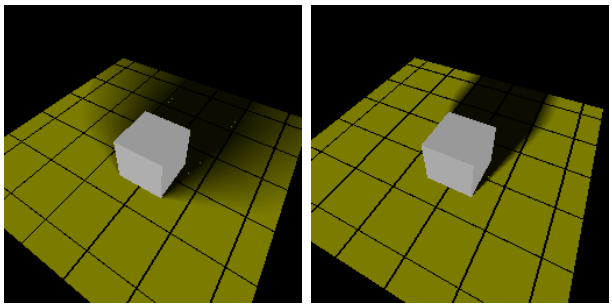


Figure 7 The results of computing smoothies



(a) **(b)**

Figure 8 Result of different widths of smoothies

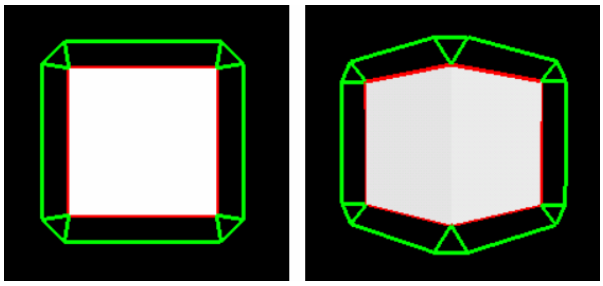


Figure 9 Linking corner

2.3: ALPHA VALUE

Another important part in this work is to compute smoothie alpha values. The gradual alpha values on smoothie can simulate the intensity in penumbra and soft aliasing problem. Alpha values also decide the shape of penumbra. The smoothie pixel close to the model has smaller alpha value. Oppositely, the alpha values are bigger. The program relies on the rule to compute alpha values by linear interpolation. See Fig. 10.

Then, this rule has a contact error when objects touch the floor. The same width of smoothies makes penumbra unnatural as Fig. 11. There has an unnecessary part of shadow. Actually, the shadow should be gradually extended from light source. To correct the mistake, alpha remapping is needful.

Alpha remapping adjusts the alpha values. It uses Eq. 6 to compute new value α':

$$\alpha' = \frac{\alpha}{1 - b/r} \tag{6}$$

α presents original alpha values computed by linear interpolation. b is the distance from light to smoothies. r is the distance from light to receiver. See Fig. 12.

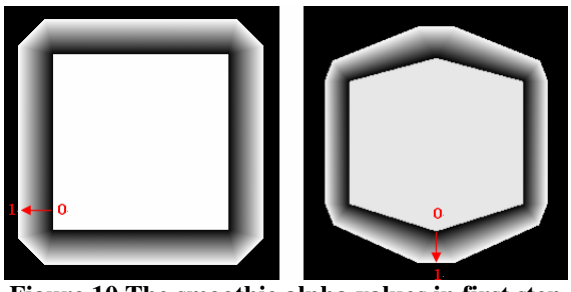


Figure 10 The smoothie alpha values in first step

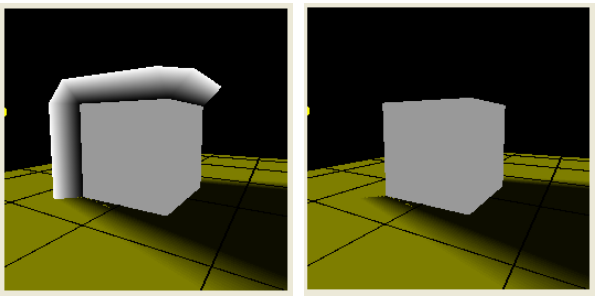


Figure 11 Contact error

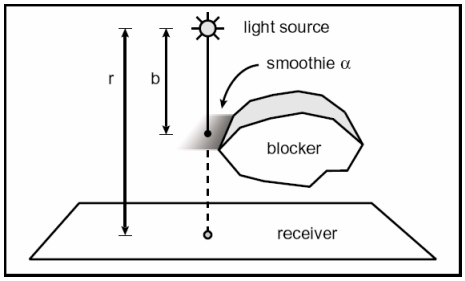


Figure 12 Illustration of equation parameters [2]

Equation 6 uses the relation between b and r to adjust original alpha values. If the smoothie is closer to receiver, the value of 1-b/r will become smaller. Then, α becomes larger. Moreover, the alpha values close to the model are nearly 0 that have little effect after remapping.

After remapping, alpha values will be adjusted as in Fig. 13. The contact error is corrected. Fig. 14 is also the result of alpha remapping. The two figures have different light height. The light in right figure is lower; the light in left figure is higher. Higher light makes smoothies wider; Lower light makes smoothies thinner. These variations result in different size of penumbra as shown in Fig. 14.

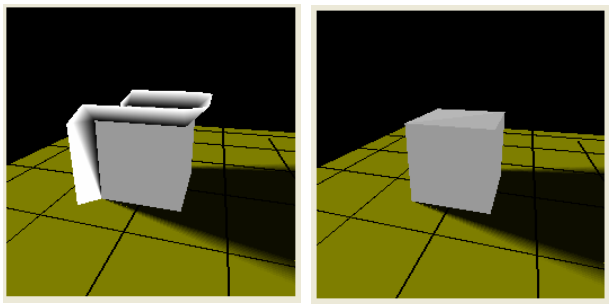


Figure 13 Alpha remapping

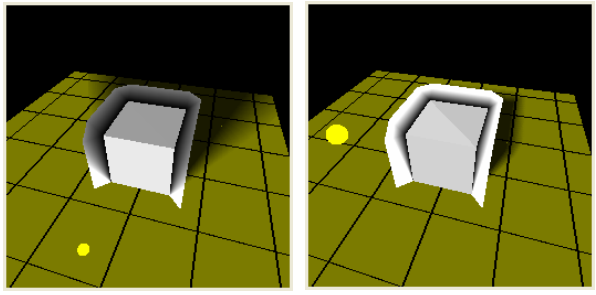


Figure 14 Alpha remapping–adjust light height

2.4: DISCUSSION

In Sec. 2.2, the outline of smoothies is constructed. Nevertheless, the smoothies will intersect with each other and cause error. If two neighbor silhouettes have inner angle larger than 180 degrees, the smoothies will intersect, and the corner will link wrong. See Figs. 15 to 16. In Fig. 15, dotted lines illustrate the linking error. Fig. 16 is a real example of smoothie error. To simplify the illustration, the corners are not linked. Taking notice of right figure in Fig. 16, the smoothies in black line intersect with each other.

The linking error may be resolved with computing inner angles, but it can not completely solve the problem and wastes time to do additional computation. A compromising way is to shrink the width of smoothies that reduce the overlap parts. See Figs. 17 to 18.

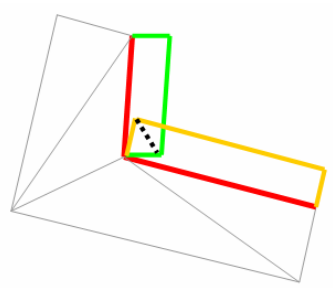


Figure 15 Smoothie error

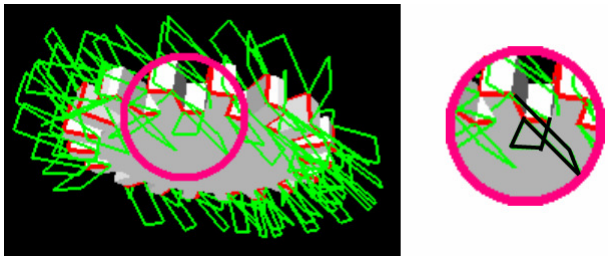


Figure 16 An example of smoothie intersection

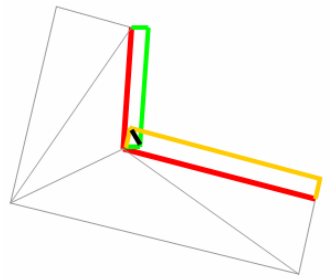


Figure 17 Correction of smoothie error

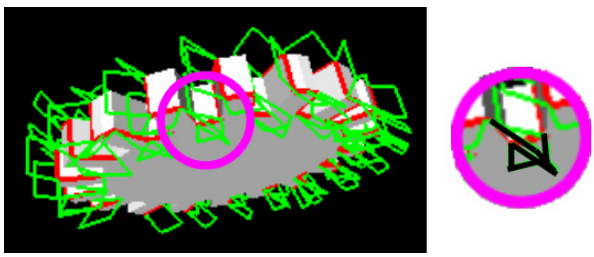


Figure 18 An example of smoothie correction

The alpha values of smoothies will be saved and used to compute shadow intensity. When saving the alpha values, they can not be blended. The action of blend will make value larger and make a big mistake. A better method is only save the nearest pixel's alpha value.

3: SHADOW COMPUTATION

Shadow map and smoothie map are the two maps used to compute shadow. Compare the sample pixel in the scene with relative point in maps. The result determines generation of shadow.

Smoothie map is similar to shadow map. It has two parts: smoothie depth map and smoothie alpha map. Using both shadow map and smoothie map can improve aliasing problem in original shadow map.

The creation of smoothie depth map and alpha map is the same as shadow map. The only difference is smoothie map just needs to save smoothies. Smoothie depth map saves depth value, and alpha map saves alpha value. The alpha value can not be blended. It will become larger and cause error. A better way is to save the nearest pixel's alpha value.

The compared process has two stages.

Stage 1: Determine if the sample pixel is in shadow after depth comparison with shadow map. If the result is in shadow, stage 2 is unnecessary and compared process ends. If it is illuminated, do stage 2 continuously.

Stage 2: Compare the sample pixel with smoothie depth map and determine if the pixel is in shadow. If it is, the sample pixel's intensity equals the relative alpha value in alpha map. If it is illuminated, the sample pixel's intensity is 0.

Figure 19 illustrates the comparison of maps. There are three different results.

(1)There is no occluder between light and receiver. Point a is illuminated. Its intensity is 1.

(2)The ray of light passes through the smoothie. Then, point b is partly in shadow, and its intensity equals the relative alpha value in smoothie alpha map.

(3)There has an occluder between light and receiver. This causes point c completely in shadow. The intensity of point c is 0.

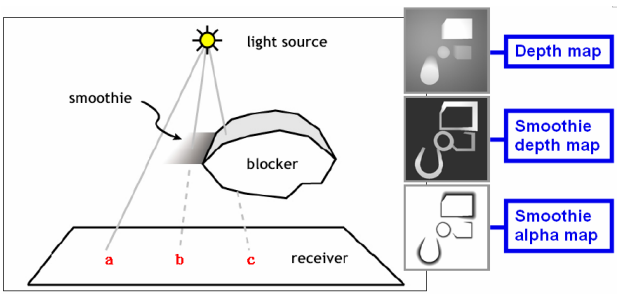


Figure 19 Test of each map [2]

4: IMPLEMENTATION

The algorithms mentioned above are implemented on the following platform: operate system is Microsoft Windows XP Home Edition Version 2002 Service Pack 2; CPU is Intel Pentium M processor 1.73GHz; memory is 1G DDR2; display card is ATI MOBILITY RADEON X700 PCI Express /64MB; development environment is Microsoft Visual C++ 6.0; system main programming language is C, FLTK, OpenGL v2.0, and GLSL v1.10. The program uses OFF (Object File Format)[6] to implement. OFF saves number of vertices, edges and faces, coordinates of vertices and point indices of faces. Point indices of faces are saved counterclockwise. OFF has enough information to construct smoothies.

Figures 20 to 23 illustrate the effectiveness of smoothie map in shadows. Smoothies really improve the aliasing problem generated by shadow map.

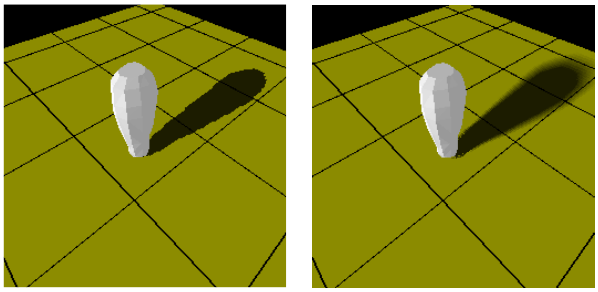


Figure 20 Shadow created with smoothie map (right) or not (left) – stick

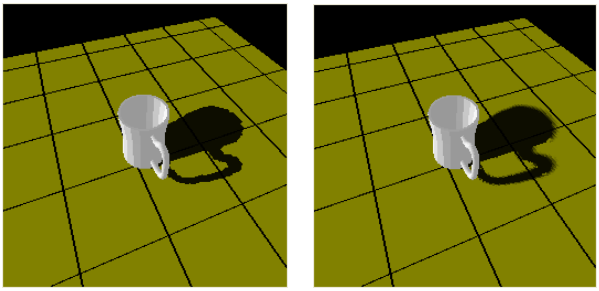


Figure 21 Shadow created with smoothie map (right) or not (left) – cup

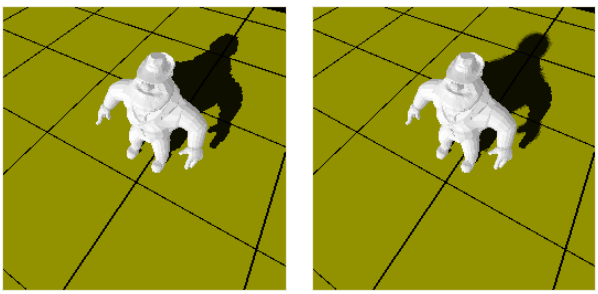


Figure 22 Shadow created with smoothie map (right) or not (left) – a man known as "Al".

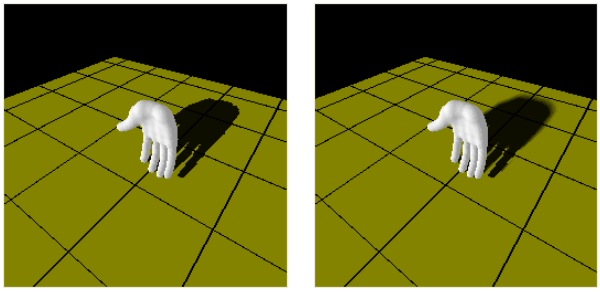


Figure 23 Shadow created with smoothie map (right) or not (left) - hand

5: CONCLUSION AND FUTURE WORK

Shadow in a scene can express the relative position between objects and create visual effects. The article focuses on the augmentation of shadow map with smoothie map to create realistic soft shadows. Although smoothies are geometric and complex to compute, they completely improve aliasing problem. A smoothie includes a silhouette edge and two smoothie vertices. Increasing the performance of finding silhouette edges is the main way to increase performance. For this reason, only check the neighbor edges of a silhouette edge. This method just finds probable edges and saves searching time. The performance will increase obviously when the model has more than 10000 faces.

Shadow map can be accelerated in modern graphics hardware. The prototype implementation uses GLSL to implement shadow map and smoothie map and also use fragment shader to blend the color of shadow and receiver.

The method of searching neighbor silhouette edges suffers from bifurcation problem. Part silhouette edges will lose if there are bifurcations. Bifurcation problem is partially solved by periodic exhaustive search. The action only decreases the error. A probable method to solve the problem is to adjust the end condition. When meeting a checked silhouette edge, the search does not stop and continuously search neighbor edges until all neighbor edges are checked.

Another problem is smoothie intersection. As the models become more complex, there may have smoothie intersection. Smoothie intersection makes smoothie broken. There have two ways to solve the problem. First, reduce the width of smoothies. The method can avoid some intersection but not all. Secondly, save the smallest alpha value when creating smoothie map. This method may improve the discontinuous alpha value caused by intersected smoothies.

REFERENCES

[1] Andrew V. Nealen, “Shadow Mapping and Shadow Volumes: Recent Developments in Real-Time Shadow Rendering”, Project report for Advanced Computer Graphics: Image Based Rendering (CS514) in University of British Columbia, 2002

[2] Eric Chan and Frédo Durand, “Rendering Fake Soft Shadow with Smoothies,” Proceedings of the 14th Eurographics workshop on Rendering, 2003, Pages: 208-218

[3] GAME TUTORIALS, http://www.gametutorials.com/gtstore/pc-321-1-shadow-mapping-with-glsl.aspx

[4] Marc Stamminger and George Drettakis, “Perspective Shadow Maps,” Proceedings of the 29th annual conference on Computer graphics and interactive techniques, 2002, Pages: 557-562

[5] Mark J. Kilgard, “Shadow Mapping with Today’s OpenGL Hardware,” NVIDIA SDK White Paper, http://developer.nvidia.com/object/gdc2001_shadows.html, 2001

[6] Object File Format (.off) , http://shape.cs.princeton.edu/benchmark/documentation/off_format.html

[7] Pradeep Sen, Mike Cammarano, and Pat Hanrahan, “Shadow Silhouette Maps,” ACM Transactions on Graphics (TOG) 22, 3, July 2003, Pages: 521-526

[8] Tobias Isenberg, Bert Freudenberg, Nick Halper, Stefan Schlechtweg, and Thomas Strothotte, “A Developer's Guide to Silhouette Algorithms for Polygonal Models,” IEEE Computer Graphics and Applications 23, 4, July 2003, Pages: 28-37

[9] Tom Hall, “Silhouette Tracking,” www.geocities.com/tom_j_hall , May 2003