LETTER

An Efficient Method for Dynamic Shadow Texture Generation

Kyoung-Su OH[†a)] ***and*** **Byeong-Seok SHIN**[††], ***Nonmembers***

SUMMARY We propose a novel shadow texture generation method with linear processing time using a shadow depth buffer (*SZ-Buffer*). We also present a method that achieves further speedup using temporal coherence. If the transition between dynamic and static state is not frequent, depth values of static objects does not vary significantly. So we can reuse the depth value for static objects and render only dynamic objects.

key words: *shadow, shadow texture, depth buffer, temporal coherence*

1. Introduction

Shadows enhance reality and provide good clues to spatial relationships between objects. Much work has been done on real-time shadow generation. However, only a few 3D graphics systems can produce shadows in real time.

Shadow texturing is one popular method for real-time shadow generation. A shadow texture is a projected image with the position of the light source as its center of projection. The shadow texture of an object represents the shadows cast on it. Each pixel value represents the presence of interfering objects between the object and the light source: if there are objects between the pixel and the light source, the pixel color is black, otherwise it is white. This is useful for shadow representation. However, shadow generation time is proportional to the square of the number of objects in the scene, because this method for generating an object's shadow texture renders all objects between the object and the light source.

Nguyen explained how to generate shadow textures using graphics hardware when the shadow caster and receivers are identified [1]. Herf presented a method for soft shadow generation when the number of shadow receivers is small. Soler and Sillion used the Fast Fourier Transform (FFT) and its inverse to produce soft shadow textures [3]. Previous methods have not been adequate for rendering complex scenes because they quickly slow down as the scene complexity increases. This is because they require rendering of all shadow casters between a shadow receiver and the light source [2].

We present a novel method that has linear processing time using a shadow depth buffer (*SZ-buffer*). We can generate shadow textures by rendering only the shadow receiver. Each pixel of the *SZ-buffer* contains the distance from the light source of the surface closest to the light source along the corresponding ray. Pixels on the surface being rendered that are farther from the light source than the *SZ-buffer* distance are therefore changed from a light color to the shadow color.

We can also exploit temporal coherence in shadow texture generation. There were many researches to exploit temporal coherence in visibility computation under fixed viewing conditions [4]–[8]. We can exploit temporal coherence in shadow texture generation with a fixed light source. If we render objects from a light source position, the depth values of static objects can be reused without rendering the objects. We can find those values efficiently and render only the dynamic objects. As a result, the time complexity of our algorithm is asymptotically $O(N_{dyn})$, where N_{dyn} is the number of dynamic objects in the scene.

In Sects. 2 and 3, we explain how to generate shadow textures and how to improve processing speed using temporal coherence. Experimental results are shown in Sect. 4.

2. Shadow Texture Generation Using the *SZ-Buffer*

Previous shadow texture generation methods produce shadow texture for an object (shadow receiver) by rendering all the other objects (shadow casters) between the receiver and the light source (Fig. 1 (b)). We use an *SZ-buffer* to generate a shadow texture for an object. To create this data structure, we render all objects from the light source position. After setting the viewpoint as the light source po-

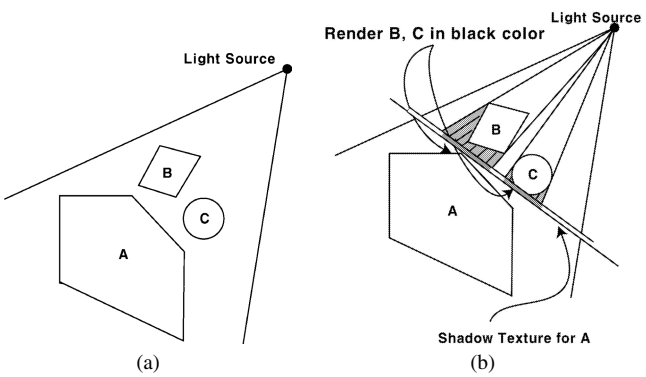

Fig. 1 Previous shadow texture generation method: (a) light source and objects in the scene. (b) To generate the shadow texture for A, render all objects between the light source and A (B and C) with shadow color.

Manuscript received September 3, 2004.

[†]The author is with the School of Media, College of Information Science, Soongsil University, Korea.

[††]The author is with the School of Computer Science & Engineering, College of Engineering, Inha University, Korea.

a) E-mail: oks@ssu.ac.kr

DOI: 10.1093/ietisy/e88–d.3.671

sition, we render all objects using a depth buffer algorithm and save the depth values in the *SZ-buffer*. As a result, each pixel of the *SZ-buffer* contains the distance from the light source to the closest object seen through it (Fig. 2 (a)).

We can generate shadow textures using the *SZ-buffer* by applying the depth buffer algorithm once again, setting the viewpoint as the light source position and copying the contents of the *SZ-buffer* to the depth buffer. For each object, we set its shadow texture as the render target, and fill the color buffer (render target) with the shadow color, normally black (Fig. 2 (b)). Then we render the object as a light color using the depth buffer algorithm. Because the *SZ-buffer* contains the distances to the nearest objects, the colors of pixels closer to the light source are changed from the shadow color to light colors (Fig. 2 (e)).

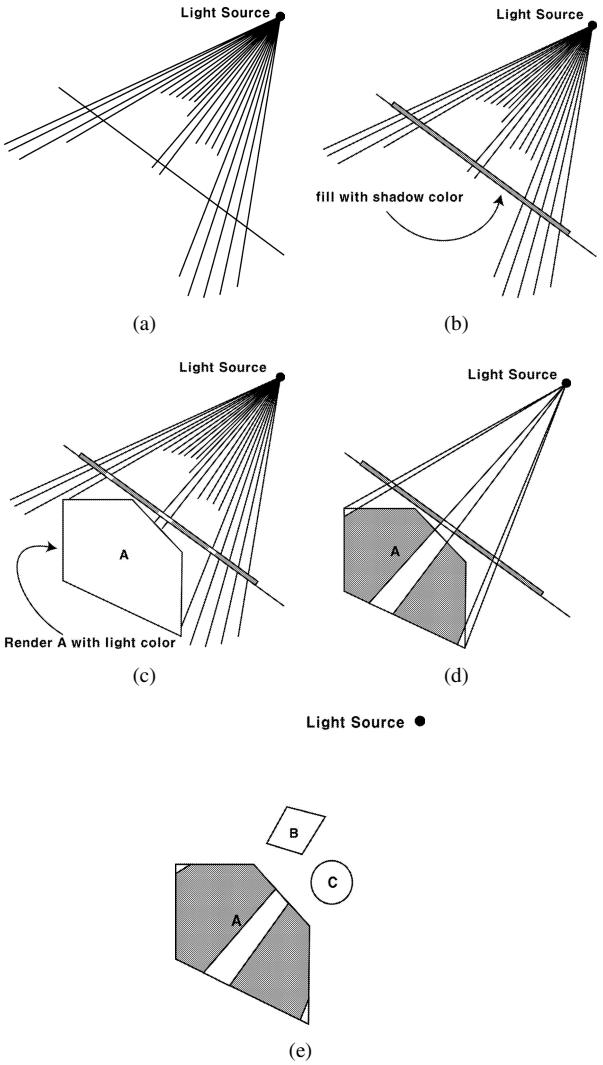

Fig. 2 Shadow texture generation using the *SZ-buffer*: (a) Contents of the *SZ-buffer*: a length of ray means the depth value. (b) Fill shadow texture with shadow color (normally black). (c) Render shadow receiver (object A) to find its shadow texture. (d) Shadows generated by the shadow texture. (e) Result of rendering all objects with the shadow texture.

3. Further Speedup Using Temporal Coherence

In most applications, light sources and objects are static in most frames. We use this coherence to speed up shadow texture generation.

3.1 Acceleration of *SZ-buffer* Generation

Each object in an animation sequence has one of two states, *static* or *dynamic*. If the transitions between the two states are infrequent, the depth values of currently static objects are similar to those of the static objects in the previous frame. When we have the depth values of static objects, we can complete rendering of the current frame by rendering only dynamic objects. In a previous paper, we proposed an efficient visibility method (called *mobility culling*) [4], which enhances the conventional Z-buffer algorithm by efficiently calculating the depth values of static objects for a fixed viewing condition. We can apply the method to generating the *SZ-buffer* when the light source is fixed.

Assume that all objects are classified into four states, SD (static-dynamic), DS (dynamic-static), SS (static-static), and DD (dynamic-dynamic) according to their mobility information. For example, objects that are static in the previous frame and dynamic in the current frame are categorized into the set SD. Figure 3 shows an example of classification of target objects. We use a *static depth buffer* that holds depth values for currently static objects and a *rendering depth buffer* to store depth values for all the objects in the current frame.

The *SZ-buffer* generation procedure using temporal coherence consists of the following three stages: (1) obtain depth values of currently static objects from the static depth buffer, (2) store the depth values of the static objects in the rendering depth buffer, and (3) render currently dynamic objects using the depth buffer algorithm.

Figure 4 shows how the content of the rendering depth buffer and the static depth buffer changes while applying our method.

The first step of the algorithm has the following four substeps: (1-1) Erase the extents of currently dynamic and previously dynamic objects (erase DD objects); (1-2) Erase the extents of currently dynamic and previously static ob-

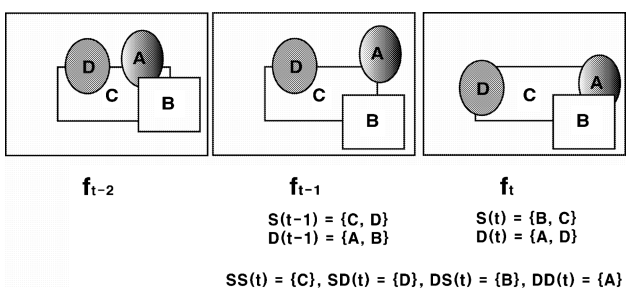

Fig. 3 Examples of objects classified into four categories according to their mobility in the last two frames.

jects (erase DS objects); (1-3) Render currently static and previously static objects (draw SS objects); and (1-4) Render currently static and previously dynamic objects (draw SD objects).

Figure 5 shows the result of each of these substeps. Before we start our algorithm, the depth buffer contains depth values from the previous frame. The first and second substeps erase dynamic objects, and the third and fourth substeps render static objects that are overwritten while erasing the currently dynamic objects.

In the first substep, copying from the static depth buffer erases objects that were dynamic in the last two frames. In the second substep, filling with the background depth value erases objects that are currently dynamic and previously static. In the third substep, currently and previously static objects are rendered. These objects were not erased in the second substep, so only those objects that overlap with one or more of the currently dynamic and previously static objects are rendered. In the fourth substep, currently static and previously dynamic objects that overlap with currently dynamic objects are rendered.

3.2 Shadow Texture Generation

When the light source is fixed, only dynamic objects and the objects overlapping with the dynamic objects require updates of shadow textures. If an object moves in the current frame, its shadow texture must be regenerated. If an object overlaps with one or more extents of moving objects, its shadow texture must also be generated. Most objects are not in these two cases and do not require shadow texture update. Currently dynamic objects and the objects overlapping with dynamic objects were identified while constructing the *SZ-buffer*. We generate shadow texture for these objects with the *SZ-buffer* using the algorithm explained in Sect. 2. Since we consider only part of entire objects, texture generation time can be reduced.

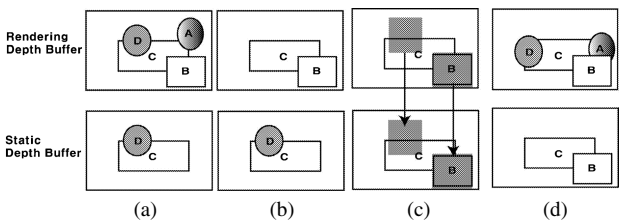

Fig. 4 Contents of the rendering depth buffer and the static depth buffer in each step of our algorithm. (a) Initial state. (b) After computing the depth values of static objects. (c) Copy to static depth buffer. (d) Render dynamic objects.

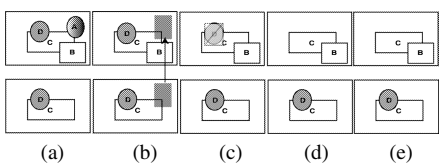

Fig. 5 Content of the rendering depth buffer (top row) and the static depth buffer (bottom row) after each sub-step of the first step. (a) Before starting. (b) Erase currently and previously dynamic objects. (c) Erase currently dynamic and previously static objects. (d) Render currently and previously static objects. (e) Render currently static and previously dynamic objects.

4. Implementation and Experimental Results

We implemented the previous shadow texturing method and ours using Direct3D on a PC equipped with a 2.66 GHz Pentium 4 CPU, 1 GB RAM and an ATI Radeon 9800 XT graphics card. We generated sphere models in random positions and assigned them random movements. Each sphere model has 2,037 vertices and screen resolution is 800 by 600 pixel. Figure 6 shows some images in an animation sequence produced by our method.

We measured rendering times of the previous method and our method without using temporal coherence. Figure 7 compares rendering times for our method with those for the previous method. As the graphs show, rendering time with

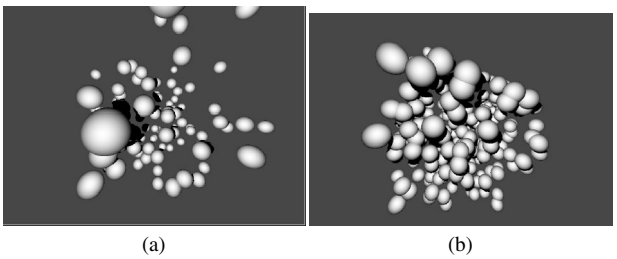

Fig. 6 Some sampling images for different models: (a) 100 spheres. (b) 300 spheres.

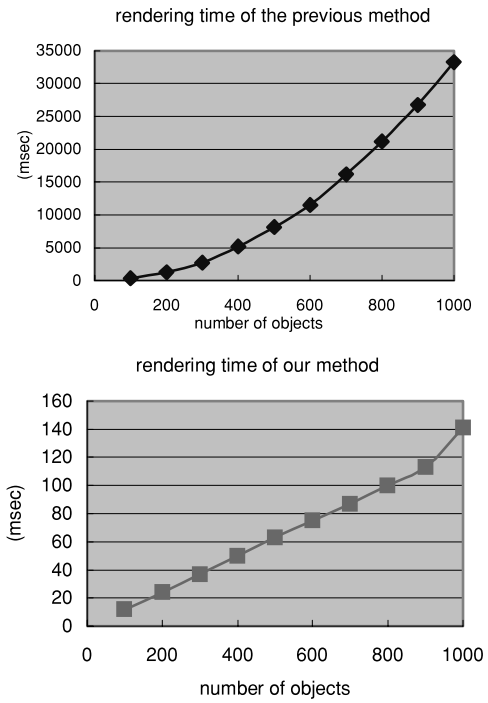

Fig. 7 A comparison of rendering times with the previous method (top) and our method (bottom).

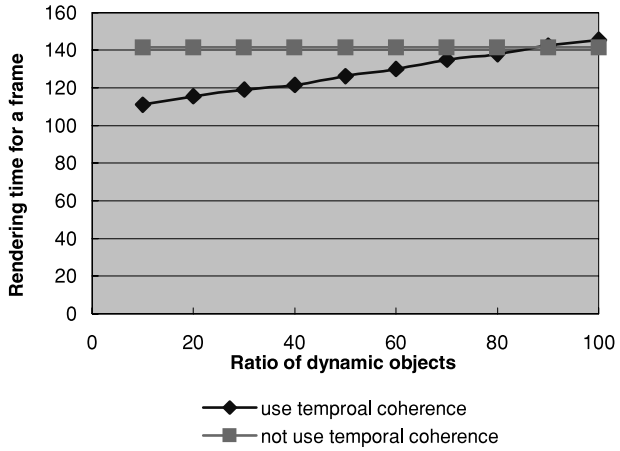

Fig. 8 A comparison of rendering times when exploiting temporal coherence and when not using coherence, according to the percentage of dynamic objects.

the previous method is $O(N^2)$, while rendering time with our method is $O(N)$.

To show the effect of exploitation of temporal coherence, we measured the rendering times of the two methods while varying the percentage of dynamic objects. As the fraction of dynamic objects becomes smaller, speedup from temporal coherence increases, as shown in Fig. 8. If the fraction of dynamic objects is 10%, the speedup is 23.4%. This is based on the total rendering time. The speedup of shadow texture generation time is much greater.

5. Conclusion

In this paper we have proposed a constant-time shadow texture generation algorithm. In addition, we have presented an enhanced method that does not generate shadow textures for static objects. Experimental results show that the performance of our method is linearly proportional to the number of shadow receivers that require shadow texture generation. Our method is an order of magnitude faster than the previous method and the speedup is greater if the fraction of dynamic objects is small.

Acknowledgments

This work was supported by the Soongsil University Research Fund.

References

[1] H.H. Nguyen, "Casting shadows on volumes," Game Developer, vol.6, no.3, pp.44–53, March 1999.
[2] M. Herf, "Efficient generation of soft shadow textures," CMU-CS-97-138, CS Dept, Carnegie Mellon U., May 1997.
[3] C. Soler and F. Sillion, "Fast calculation of soft shadow textures using convolution," Proc. SIGGRAPH98, pp.321–332, July 1998.
[4] K.S. Oh, B.S. Shin, and Y.G. Shin, "Mobility culling: An efficient rendering algorithm using temporal coherence," J. Vis. Comput. Anim., vol.12, no.3, pp.159–166. Sept. 2001.
[5] G. Scaufer, "Exploiting frame to frame coherence in a virtual reality system," Proc. Virtual Reality Annual International Symposium, pp.95–102, Santa Clara, March 1996.
[6] J. Chapman, T.W. Calvert, and J. Dill, "Spatial–temporal coherence in ray tracing," Proc. Graphics Interface., pp.196–204, Helifax, May 1990.
[7] S. Badt, Jr., "Two algorithms for taking advantage of temporal coherence in ray tracing," Vis. Comput., vol.4, no.3, pp.123–132, Sept. 1988.
[8] D. Jevans, "Object space temporal coherence for ray tracing," Proc. Graphics Interface, pp.176–183, Vancouver, May 1992.