World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:7, No:1, 2013

A Real-Time Rendering based on Efficient Updating of Static Objects Buffer

Youngjae Chun, and Kyoungsu Oh

***Abstract*—**Real-time 3D applications have to guarantee interactive rendering speed. There is a restriction for the number of polygons which is rendered due to performance of a graphics hardware or graphics algorithms. Generally, the rendering performance will be drastically increased when handling only the dynamic 3d models, which is much fewer than the static ones. Since shapes and colors of the static objects don't change when the viewing direction is fixed, the information can be reused. We render huge amounts of polygon those cannot handled by conventional rendering techniques in real-time by using a static object image and merging it with rendering result of the dynamic objects. The performance must be decreased as a consequence of updating the static object image including removing an static object that starts to move, re-rending the other static objects being overlapped by the moving ones. Based on visibility of the object beginning to move, we can skip the updating process. As a result, we enhance rendering performance and reduce differences of rendering speed between each frame. Proposed method renders total 200,000,000 polygons that consist of 500,000 dynamic polygons and the rest are static polygons in about 100 frames per second.

***Keywords*—**Occlusion query, Real-time rendering, Temporal coherence.

I. Introduction

NOW the 3D graphics technology is getting to be faster and more sophisticated with advanced rendering techniques and devices. Most of recent graphic devices can draw about 10 millions of polygon per frame. However, to render a 3D scene that consists of a huge amount of polygons still takes a long time. Rendering workload which cannot be handled in hardware need to be distributed both hardware and software.

In general, both of the real world and virtual world has much more static objects than dynamic objects. The performance might be boosted by only rendering dynamic objects and reusing static objects' information. Since rendering results of the static objects in the previous frame and the current frame are similar, we can present the static objects in the current frame with the previous frame recycling. The state of object can change from static into dynamic, so its information cannot be reused. Therefore, we have to update parts of the previous frame image. For instance, a dynamic object which was static in the past, the previous frame, is moving in the current frame, hence its information, in the previous frame, needs to be removed or modified. Some pixels of static objects can be deleted as deleting a dynamic which locates in front of it. In this case, we have to render the static objects in order to recover the deleted pixels and it takes a long time. If the new dynamic object is overlapped by many another ones, the updating cost is much more expensive and causes the sudden slowing. Real-time 3D applications such as 3D games need to avoid this situation.

Our key idea is that we can provide faster and more stable rendering speed by updating static objects' information efficiently. For this, we examine whether the object that starts to move is entirely overlapped by other static ones or not. If the object is invisible, it is unnecessary to delete that object from the static objects buffer and the renew process can be skipped. As a result, our rendering technique offers faster average rendering performance and alleviates the sudden slowing as updating static the objects buffer every time. Our method has a definite advantage where we fix the viewpoint. However, some previous 3D applications which employ top view of quarter view provided high-quality contents even if the fixed viewpoint limits user interactions.

The following sections of this paper: Section II discusses related work, and then in Section III will introduce our system overview. Data structure and algorithm details will be explained in section IV and V. After that, comparison results will be presented in section VI. Proposed algorithm is measured against the traditional rendering method and the previous TC (Temporal Coherence) method. At last, in section VII we will summarize our contribution, the presented method and applicable area.

II. Related Work

There are many advantages to use temporal coherence features. Consecutive rendering results have similar shape, color or depth information. Similarity between the previous and the current frame has been used to predict visibility among objects in the 3D scene ([1]-[3]). It also has been researched that to reduce rendering cost using temporal coherence. Intermediate images are can be interpolated using already rendered several images ([4]).

Matrix transformation is a basic concept of generating 3D virtual scene and we can perform transformation in reverse for several purposes. While we render a 3D scene through the graphics pipeline, we can store matrix information in order to unproject and reproject any object. These transformation techniques help us to render 3D scene fast which includes many polygons.

Render cache concept is introduced for the first time in [5].

Youngjae Chun is with the Department of Global Media, Soongsil University, Seoul, Korea (e-mail: dkreformer@ssu.ac.kr).

Kyoungsu Oh is with the Department of Global Media, Soongsil University, Seoul, Korea (corresponding author to provide phone: +82-10-8886-3924; fax: +82-2-822-3622; e-mail: oks@ssu.ac.kr).

Render cache is a point-based data structure, which stores 3D position or shading result of the previous frame. Authors used render cache to generate the current frame image. This method shows faster performance than a traditional method. Reference [6] implemented render cache on GPU to accelerate computation. Similar to [5], a new concept so called reprojection cache is proposed in [7] and [8]. Reprojection cache doesn't contain point but visible pixel color and depth. Authors of [7], [8] reprojected a pixel in the current frame into the previous frame using reprojection information. After then, obtained pixel by reprojection has shading color and was used to decide a pixel color in the current frame.

Some kind of image-based rendering methods store the previous frame as image and use it as input for the current frame. Frame to frame temporal coherence is introduced in [9] and [10]. They reused image generated from the previous frame. Reference [11] separated object in a 3D scene with several layers and focused on update of layers which have fast moving object.

Temporal coherence can be used to shadow generation. Pixel-corrected shadow map presents high-quality hard shadow. It accumulated shadow map result for a several frames ([12]). Reference [12] expended their work to make soft shadow in [13]. An efficient shadow map generation method using temporal coherence is introduced in [14]. This work drew static objects into an image and reused it. Sometimes the image is needed to be modified. Based on [14], we introduce an advanced method which updates the static object image efficiently by avoiding unnecessary renew work.

III. Rendering System Overview

All scene objects rendered by our method are divided into dynamic objects and static objects according to transformation state. After rendering the static objects into a static objects buffer, we use the buffer many times without update. Yet, it needs to be modified if there is any static object is transformed by translation, rotation and scaling. Instead of updating the static object buffer if there is any change ([14]), we skip unnecessary renewing of the static objects buffer. According to the visibility of the new dynamic objects against the static objects buffer, we will decide that we have to update parts of the buffer or skip. In the meanwhile, dynamic objects are rendered into the dynamic buffer and it is refreshed every frame. After all, buffers will be merged to generate final rendering result. Each buffer stores depth value, the nearest pixel from a camera, for the visibility test. Fig. 1 shows our rendering progress depending on frame frequency axis.

TABLE I
Notion Rules for the Transformation State of Objects

Previous frame	Current frame	Notion
Static	Static	SS
Static	Dynamic	SD
Dynamic	Static	DS
Dynamic	Dynamic	DD

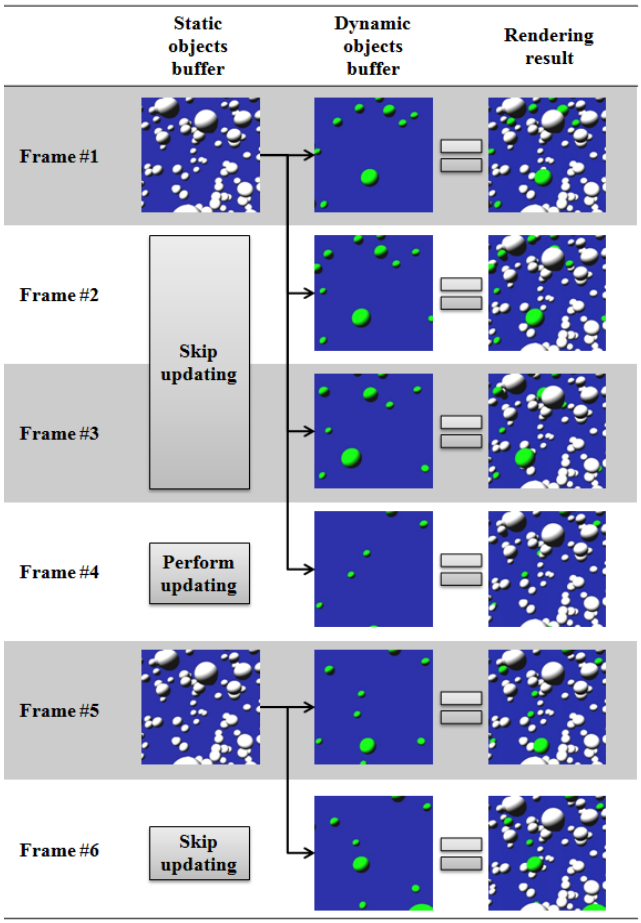

Fig. 1 Rendering overview

IV. Data Structure

We use 2 image buffers (dynamic objects buffer, static objects buffer) for our rendering method. We named the dynamic objects buffer and the static objects buffer as D-buffer and S-buffer respectively. Each buffer is a 4 channel texture and it allocates 8 bits for each channel. RGB channels store color of primitives and Alpha channel stores depth value. We render dynamic objects into D-buffer which is very similar to the common frame buffer, because we update D-buffer every frame. The rendering result of static objects is stored in S-buffer which will be reused later for the next frame information. For this work, the object's state is separated by 4 types. Table 1 shows the notion for each object according to the matrices of previous and current frame. We update S-buffer using SD and DS objects.

V. Rendering Process

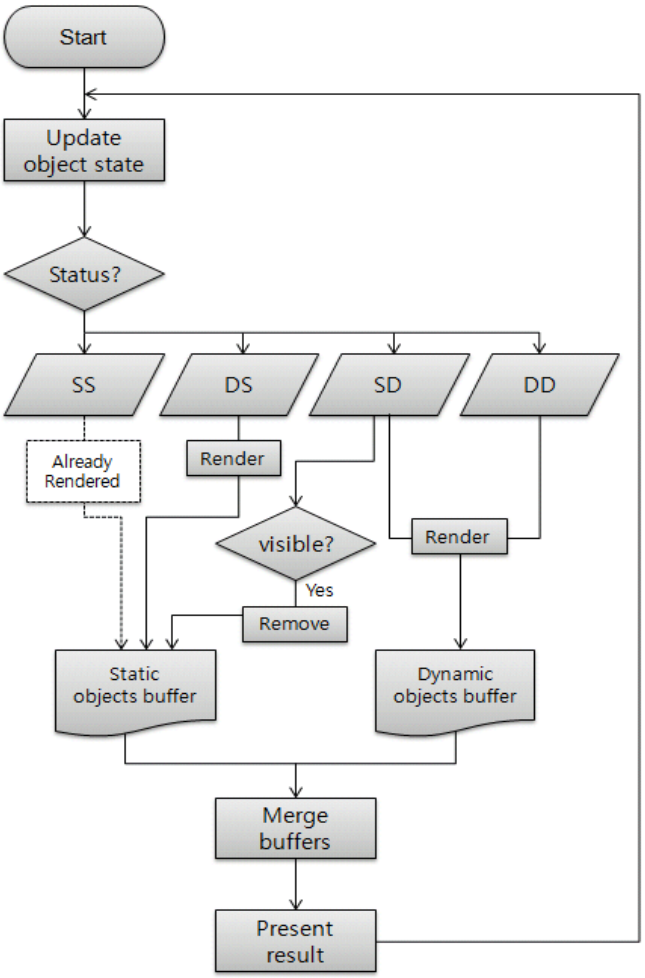

Fig. 2 Rendering flow chart

At the beginning, all objects are set as DD objects, then the rendering process will be performed as Fig. 2.

A. Update State of All Objects

At first, we select objects to make them move. Chosen objects keep the state DD and others become DS objects. Theses DS objects are used to initial the static objects buffer.

Each object stores own local-to-world transformation matrix for the next updating. In the next update stage, we compare the matrices at the current frame with the matrices at the previous frame. The objects which have difference between two matrices are set to SD or DD according to the previous state.

B. Render or Renew the S-Buffer (Optional)

Since we have no previous static objects buffer at the first time, we have to render DS objects in S-buffer to merge it with D-buffer for final rendering and reuse later. S-buffer stores an enormous amount of static objects so pixels in the S-buffer are almost filled by the static ones. Therefore lost of static objects are overlapped by other ones and invisible. Renewing process consists of the following sub stages:

1) Delete the new dynamic object from S-buffer
2) Recover partially deleted static objects due to the new dynamic one.

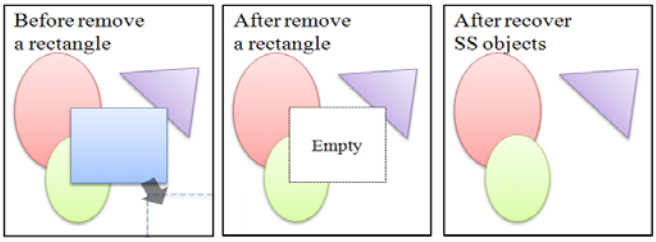

Fig. 3 Update process of the static objects buffer

Reference [14] always renewed the S-buffer when they had a SD object even if it was overlapped entirely. As mentioned above, the renew process takes a long time and subsequent processes have to wait until finish update the S-buffer. The more static objects are stored in the S-buffer, the more expensive the cost will be. It doesn't correspond with the temporal coherence concept. For this reason, passing over unnecessary renew work can be proposed. For example, the new dynamic object in the first sub-stage might be invisible then we can skip the first sub-stage and thus there is no deleted pixel. It is more efficient when we store more and more static objects in the S-buffer.

We perform the visibility test for the new dynamic objects against the S-buffer. Occlusion query is used to implement the visibility test and it returns the number of visible pixels for an object queried. If the return value equals to 0, we skip the renew process. Otherwise, we just follow the sub-stages.

C. Render the D-Buffer

The process C. is very similar to the traditional rendering. In this stage, we render DD and SD objects into the D-buffer. We assume that there are few dynamic objects in the scene, so this stage takes a bit cost.

D. Merge Two Buffers and Display

Finally, we merge two buffers for the final presentation. Each buffer stores depth value in the alpha channel. We used the pixel shader to get the nearest pixel from the current viewpoint, then display it as the current rendering result.

VI. Experiments

We render 40,000 objects and each one consists of about 4,500 polygons for experiments. As you know, the initial state of an object is DD. Randomly chosen object moves along a random direction in an arbitrary time interval. The object that begins to stop becomes DS objects and it is drawn in the S-buffer. In the next frame, it becomes SS object unless it starts to move again. Namely, all chosen objects change its state as a sequence of "SS-SD-DD-DS-SS". We increase the number of dynamic objects from 10 to 120 and once an object stops, we select another one to move it.

DirectX 9 is used as graphics library and nVidia GeForce GTX 460 is used for our experiments. We render the 3D scene using three types of algorithm. The first algorithm draws all objects in every frame, second algorithm is from [14] and the last one is ours. The conventional rendering algorithm cannot

handle 40,000 objects in real-time even if we set just 10 dynamic objects. Reference [14] shows about 68fps (frames per second) and draws objects 460 times per frame while keeping 120 dynamic objects. In the same situation, our method shows about 140fps and draws objects about 130 times per frame (Fig. 4.). Since the proposed algorithm performs an occlusion query for the SD objects, additional cost is required but gains more benefit from skipping to update S-buffer. Additionally, we use extra 6 S-buffers to distribute static objects and this work facilitates much faster and more stable rendering performance (Fig. 4 and Fig. 5).

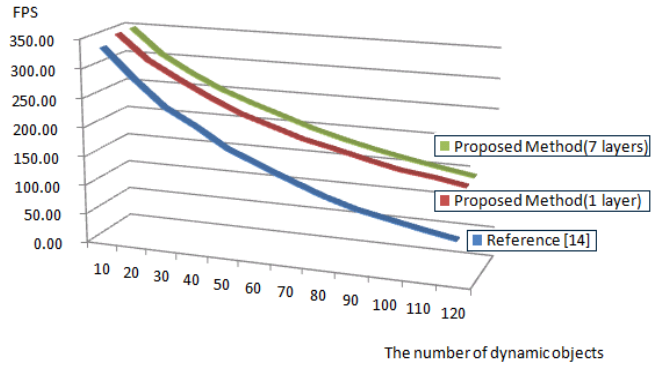

Fig. 4 Rendering performances(fps) of three methods

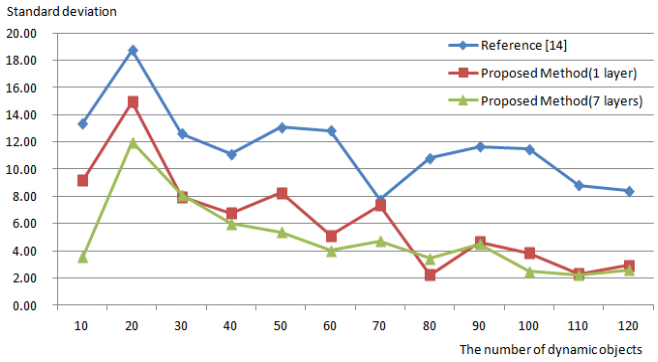

Fig. 5 Standard deviations of three methods

VII. Conclusion

In this paper, we introduced a real-time rendering method which draws the 3D scene with huge amount of polygons using a static objects buffer updated efficiently. Most of 3D applications always render all objects every frame regardless of whether objects are dynamic or not. Various graphics methods using temporal coherence have been researched to enhance rendering or representation of special effects such as motion blur or image sequence compression. Since temporal coherence feature can help the applications reduce rendering workload, we focused on acceleration and efficiently renewed the static objects information.

The previous rendering result of static object will be stored in a buffer and decrease refresh frequency of the buffer to draw many static objects in the current frame quickly and stably. When a certain static object becomes a dynamic one, we check whether the object is visible or invisible in the static object buffer. If the object is invisible due to other static ones, we just skip updating the static objects buffer. As a result, we reduced the number of the updating process and got faster, more stable rendering performance. Many 3D applications such as video games used static object image as background scene but it's cannot be changed easily. Our method facilitated that we can change static objects to dynamic objects without considerable rendering workload.

Acknowledgment

This paper was supported by BK 21 Program in Soongsil University.

References

[1] Ivan E, Sutherland, Robert F. Sproull, and Robert A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," ACM Comput. Surv, 6(1):1-55. 1974.
[2] H. Hubschman and S. W. Zucker, "Frame-to-frame coherence and the hidden surface computation: constraints for a convex world," ACM Trans. Graph. 1, 2, pp.129-162. Apr. 1982.
[3] Satyan Coorg and Seth Teller, "Temporally coherent conservative visibility (extended abstract)," In Proceedings of the twelfth annual symposium on Computational geometry (SCG '96), ACM, New York, NY, USA, pp.78-87. 1996.
[4] Shenchang Eric Chen and Lance Williams, "View interpolation for image synthesis," Proceedings of the 20th annual conference on Computer graphics and interactive techniques, pp.279-288, Sept. 1993.
[5] Walter B, Drettakis G, Parker S, "Interactive rendering using the render cache," In Eurographics Workshop on Rendering, Rendering Techniques, Springer-Verlag, pp.19-30. 1999.
[6] Zhu. T, Wang. R, Luebke D, "A GPU accelerated render cache," In Pacific Graphics (short paper), 2005.
[7] Diego Nehab, Pedro V. Sander, and John R. Isidoro, "The real-time reprojection cache," In ACM SIGGRAPH 2006 Sketches (SIGGRAPH '06), ACM, New York, NY, USA, Article 185. 2006.
[8] Diego Nehab, Pedro V. Sander , Jason Lawrence , Natalya Tatarchuk, John R. Isidoro, "Accelerating real-time shading with reverse reprojection caching," Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, Aug 04-05, 2007.
[9] Meister Eduard Gröller, "Coherence in computer graphics," PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 1992.
[10] Gernot Schaufler, "Exploiting Frame to Frame Coherence in a Virtual Reality System," In Proceedings of the 1996 Virtual Reality Annual International Symposium, IEEE Computer Society, 1996.
[11] Jed Lengyel and John Snyder, "Rendering with coherent layers," In Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH '97), ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp.233-242. 1997.
[12] D. Scherzer, S. Jeschke, and M. Wimmer, "Pixel-correct shadow maps with temporal reprojection and shadow test confidence," In: Kautz, J., Pattanaik, S. (eds.) Rendering Techniques 2007 Proceedings Eurographics Symposium on Rendering, Eurographics, pp.45-50. Eurographics Association, 2007.
[13] D. Scherzer, M. Schwärzler, O. Mattausch, and M. Wimmer, "Real-time soft shadows using temporal coherence," Lecture Notes in Computer Science (LNCS), 2009.
[14] Kyoungsu. Oh, and Byeongseok. Shin, "An Efficient Method for Dynamic Shadow Texture Generation," IEICE Transactions on Information and Systems 2005, vol. E88-D., pp.671-674, Mar. 2005.