Hardware-driven visibility culling

20073114 김정현

I. Introduction

The goal of the 3D graphics is to generate a realistic and accurate 3D image. To achieve this, it needs to process not only large amount of primitives but also additional data (texture) associated with the primitives. Therefore, 3D graphics rendering processes require a number of memory operations associated with the data, including stencil tests, z-test, color blending, and vertex data fetching as well as texture mapping.

Every primitive should be processed in rasterization-based rendering hardware although all of them are not visible in the final scene. Visibility culling rejects fragments or primitives that are invisible in the final scene, which means that they are occluded by other fragment or primitive. By eliminating invisible fragments or primitives, it is able to save redundant processing operations and power for them. Many algorithms have been proposed in software level to examine the visibility of a primitive. Some of them are able to be implemented with effective hardware.

There are two kinds of culling, primitive culling and fragment culling. A typical example of primitive culling is clipping. Originally, clipping means cutting out some part of a triangle which is out of the screen but in graphics pipeline, rejecting triangles which are out of screen is an important role of clipping too. There is also another kind of primitive culling method which uses depth values of the before frame. There are two choices in hardware according to the position of fragment visibility test. Conventional rendering processors perform the fragment visibility test after texture mapping to support standard API, such as OpenGL. It is also possible to perform z-test before texture mapping. However, the latter may cause wrong results and it needs more hardware resources to prevent that problem. As a result, early z-test architecture has been proposed. The early z-test (EZT) architecture performs the z-test roughly in early stage of rasterization pipeline and does it again in raster operation (ROP) unit with an accurate depth comparison. As the name implies, it tests the visibility of a fragment by using an image-space coherency before texture mapping to reduce wasteful memory accesses caused by invisible fragments.

There are two approaches in the image-space EZT methods; Z-Max method and Z-Min method. The Z-Max methods check if a fragment or primitive is occluded by the primitives drawn before. An occludee is rejected from a rasterization pipeline. As a result, the entire memory accesses (texture read (TR), depth read (ZR), depth write (ZW), color read (CR), and color write (CW)) of the invisible fragment are eliminated in a rasterization pipeline. Hierarchical Z-Buffer (HZ-Buffer) is a well-known Z-Max method.

It keeps a reduced resolution map of the external z-buffer on an on-chip memory and removes hidden fragments as early as possible; such fragments are removed away from the pipeline before texture mapping and the other per-pixel processes. The ATI Hyper-Z technique is the first implementation of the HZ-Buffer into a commercial product by using just two-level hierarchical z-buffer including the frame buffer as a lowest level. NVIDIA also integrated the HZ-Buffer by including a memory management unit to resolve the memory bottle-neck. Depth Filter (DF) Method does EZT by using depth planes. The planes contain the history of each screen position whether pixels are rendered at the front space of the corresponding position of the plane or not. Such a direct comparison between plane position and the incoming pixel position gives visibility information. On the other side, the Z-Min method checks if an incoming pixel occludes the pixel drawn before. An occluder is trivially accepted, and its ZR memory access is eliminated by skipping the z-test in a raster operation (ROP) unit. Although it reduces only ZR memory accesses, it results in a significant improvement in low depth complexity scene because the ZRs take place on every pixel unlike the other accesses such as TR or CR/CW. There is a fragment culling method using programmable unit. It makes possible culling more efficiently in case by case.

In this paper, I propose a very simple primitive culling method. This method uses the depth values of the before frame. Because there is very much coherency between continuous two frames, we can use the z-max value to cull all triangles in outside of that z-max value at the next frame. This can be implemented very simply by change the position of far-plane of clipping part. As the far-plane become closer, many triangles will be rejected. This method can be applied to restricted situation, like the FPS games, and also have some fundamental problems. Solving problems for general situation is left for future work.

II. Related Works

The image-space EZT methods can be divided into two major classes according to the target of memory accesses; Z-Max and Z-Min method. The Z-Max method focuses on removing the entire redundant memory accesses of a fragment or primitive behind stage of the EZT. The Z-Min method tries to remove only the redundant z-read (ZR) memory accesses. In addition, there were attempts to combine the Z-Max and the Z-Min method.

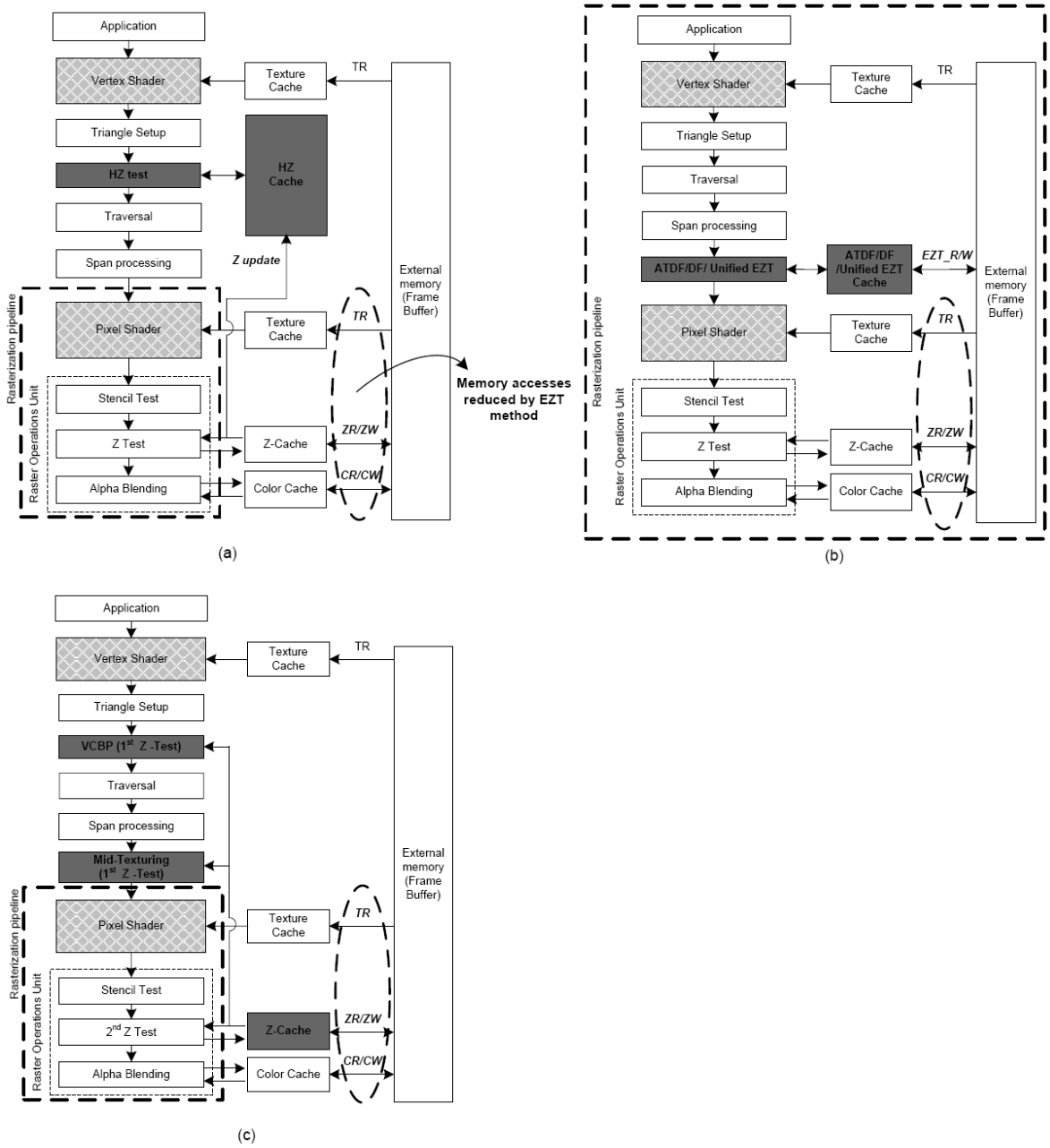

Fig. 1. Hardware architectures of a programmable 3D graphics processor adopting various EZT methods.

(a) Hierarchical Z-Buffer method.

(b) ATDF and DF method.

(c) Mid-texturing and Visibility Culling per cache Block with Prefetch (VCBP). Texture read (TR), z-read(ZR), z-write (ZW), color read (CR), and color write (CW) are the target that is reduced by EZT methods.

A. Z-Max Method

The hierarchical Z-Buffer (HZ-Buffer), shown in Fig. 1a, is the most popular algorithm that belongs to

the Z-Max method. It achieves high rejection ratio, however, it consumes additional memory bandwidth to maintain the hierarchical structure, and requires large memory size to store all hierarchy level. That is, HZ-Buffer should be updated when the z-buffer of frame buffer is updated. The update process is to search the farthest z-value in one block. Thus every z-value of a block is fetched from the Z-buffer when updating the HZ-buffer. This leads to increase the number of Z-buffer read accesses and the latency. Thus the benefit of HZ-Buffer method is largely diminished. To avoid this overhead, temporary Z-Max value is introduced . It stores the farthest z-value of a pixel that enters a block but is not rejected by HZ- Buffer method. Thus temporary Z-Max value is set to be a new Z-Max value without fetching z-values of a block from the Z-Buffer when the Z-Max value is updated. As a result, additional memory for temporary ZMax value and Mask bits are required to implement it.

Yu proposed a new Z-Max algorithm, Depth Filter (DF) shown in Fig. 1b, to overcome the problem of the HZ-Buffer that requires additional depth comparisons to find the maximum depth value when the frame buffer is updated. The DF uses a pixel-based depth plane filter which is independent of the frame buffer and efficient in terms of area. The DF is located at a position in the z coordinates and stores the history of pixels to a DF-flag buffer to cull occluded fragments. For example, if the z-value of a pixel is smaller than the DF, DF-flag is set to ‘1’ and stored in DF-flag buffer. When the next pixel in the same screen position is traversed, the corresponding DF-flag is fetched. If the z-value of the pixel is larger than that of the DF, the fragment is occluded by previous pixel, it can be culled from the 3D graphics rendering pipeline. Although DF is simple to implement with hardware and requires small on-chip memory for EZT, the performance limitation is caused by the fixed position of a DF plane within a frame.

Mid-texturing is a z-test which is performed twice at the end of rasterizer and at a ROP unit as shown in Fig. 1c. While other EZT methods have its own depth memory, it uses z-cache that is also used to perform z-test in ROP unit. First z-test at the end of rasterizer rejects fragments as EZT method, and pre-fetches corresponding data into the shared zcache when miss occurs at the first z-test. Therefore, miss latency at the second z-test is hidden in this architecture. Although it is possible to reject fragments before the texture mapping, the zread (ZR) memory access is not reduced because ZR is required to test the visibility of fragment even in the first z-test. In addition, it needs larger z-cache than other EZT methods to achieve a high hit rate at the first z-test. While other EZT methods fetch the representative depth value (ZMax or Z-Min) and reuse it for all the fragments of a block, Mid-texturing method has to read a depth value for each fragment of a block. To overcome this limitation, the new scheme, Visibility Culling per cache Block with Prefetch (VCBP) , is proposed, where depth test is performed for each span block of traversal stage. The Z-Max and Z-Min value of each span block are generated from traversal stage and compared with Z-Max value determined from a zcache. This architecture reduces ZR memory accesses unlike

original mid-texturing method; however, additional memory for Z-Max values is also required within a z-cache as well as original depth values.

B. Z-Min Method

In mobile device, the depth complexity of application scene is just two or three. The depth complexity means that how many pixels are overlapped on average, and it is evaluated by dividing the number of pixels sent to rasterizer with a resolution of screen. That is, two or three pixel are drawn at the same screen coordinate in mobile applications, which means that only one or two pixels can be rejected by drawn pixel. Therefore, HZ-Buffer method that requires large sized memory and additional memory bandwidth for determining the Z-Max value is not suitable for portable devices. To reduce the memory bandwidth effectively for mobile devices, Z-Min method was presented. It eliminates ZR memory accesses by using a Z-Min value to check the visibility of the fragment and primitive. Its operations are the same as the HZ-Buffer except that a minimum depth value (Z-Min) is used as a representative value. If the z-value of a fragment and primitive is smaller than the Z-Min value, the fragment or primitive occludes the block represented by the Z-Min value. The occluder is trivially accepted, and the pixel and primitive does not require a z-test in the raster operation (ROP) unit. In contrast to the HZ-Buffer, it is independent of the frame buffer, and additional comparison operations are not required to find a Z-Min value of the block. Although it reduces only ZR, its effect is significant in the application scene of low depth complexity. While CR/CW is performed only for the pixel that is passed through the z-test, ZR always needs to perform the z-test or EZT for every fragment. However, Z-Min method is less effective than Z-Max method as the complexity becomes higher, because it cannot reduce the other memory accesses except for the ZR memory accesses.

C. Adaptive Tile Depth Filter

The adaptive tile depth filter (ATDF) combines the Z-Max and Z-Min method to minimize the memory bandwidth in the 3D graphics rendering pipeline as shown in Fig. 1b. Both Z-Max value and Z-Min value of a tile are used to reject and accept fragments. Mask bit of DF [17] is also used for eliminating ZR memory accesses when a pixel is drawn first. Since z-test is based on a pixel level in this method, location of EZT in 3D graphics processor is the same as DF. ATDF uses two Z-Max/Z-Min pairs for a single tile, and saves large amount of memory accesses by using Z-Max, Z-Min, and Mask. The Z-Max and Z-Min value for half tile are determined after all the flags of half tile are set to ‘1’ and fixed within a frame. The Z-Max and Z-Min value of another half tile are also determined when all of the Mask bits of a corresponding half tile are set to ‘1’. To reduce the memory size, newly determined Z-Max and Z-Min

values are stored at the place where Mask bits are previously stored. However, since Z-Max and Z-Min values are fixed in a frame, if Z-Max value is determined as a large value (far from eye position), the rejection ratio of Z-Max method greatly decreases in high depth complexity applications.

D. Programmable Culling Unit

Graphics hardware still lack efficient and flexible support for culling. To improve the situation, Jon introduced the programmable culling unit, which is as flexible as the fragment program unit and capable of quickly culling entire blocks of fragments. Fig. 2 represents the position of PCU. PCU is located before pixel shader and PCU runs another shader program. This Cull Program is created automatically. By the result of that program, a tile is culled or not. Because PCU use tile based processing, many pixels are treated at once. This technique improves the efficiency of culling for each scene.

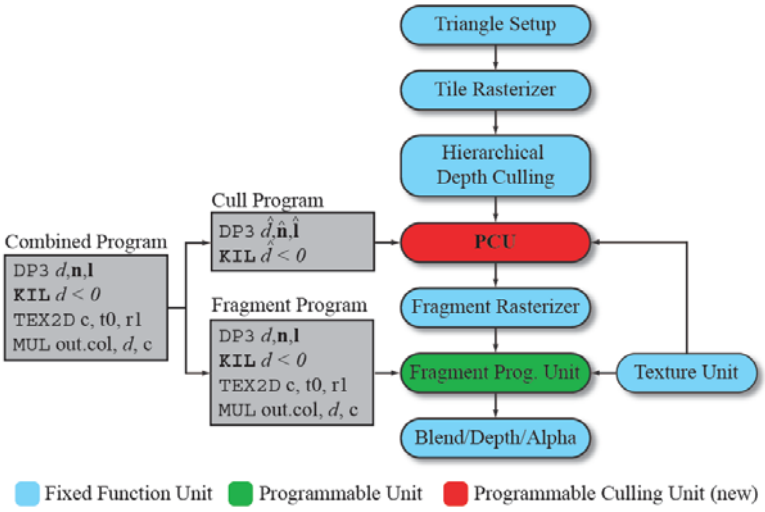

Fig. 2 Behavioral model of our proposed rasterization architecture. a new programmable unit is added between the depth culling unit and the fragment-producing unit. This unit executes a cull program, and decides whether a tile can be culled or not based on the results of the program.

III. Overview

Because of a large amount of memory operations, rasterizer engine (RE), per-pixel processor, needs to access external memory considerably. For example, texture read (TR), depth read (ZR), depth write (ZW), color read (CR), and color write (CW) are required for each fragment. Due to these high memory accesses, the performance of a rendering processor highly depends on the bandwidth of the memory system, and large amount of power is consumed by the memory accesses. To resolve this problem, visibility culling methods are needed.

As I mentioned before, there are many visibility culling techniques and we can take more than one technique because each technique is not much complex. Hence adding more culling technique is

beneficial for performance and power consumption. In that way, I add on another simple primitive culling technique. What the proposed method does is adjust the clip-far-plane. Clipping is the simplest culling technique and it is already implemented in graphics pipeline. So, using this means this technique is very simple to implement and rarely become overhead. We should not add hardware to cull just a few triangle and we should not use lazy scheme which makes overall performance lower. In other words, if hardware addition is ignorable and it does not take much time, we can add a number of scheme to cull more triangles, even a triangle, then before. At this perspective proposed scheme is valuable. Even though this scheme is not perfect yet, it may be able to become a useful technique.