Adaptive Tile Depth Filter for the Depth Buffer Bandwidth Minimization in the Low Power Graphics Systems

You-Ming Tsao, Chi-Ling Wu, Shao-Yi Chien, and Liang-Gee Chen
DSP/IC Design Lab and Media IC and System Lab
Graduate Institute of Electronics Engineering and Department of Electrical Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

***Abstract*— Depth buffer bandwidth minimization with depth filter plays an important role in designing low power graphics processors. In this paper, an efficient adaptive tile depth filter (ATDF) algorithm is proposed. The key concept is to consider more occlusion conditions in order to achieve better performance. Two existing algorithms, Z_{max} and Z_{min} algorithms, are integrated to detect both occluded and non-occluded fragments. Moreover, two new techniques, coverage mask and adaptive tile mode, are proposed to further improve the performance. Experiments show that the proposed algorithm can filter out up to 85% fragments to reduce 40% memory bandwidth for test applications with complex scenes, which outperforms the previous works.**

I. Introduction

In recent years, the market of mobile electronics grows rapidly. Meanwhile, 3D graphics becomes more and more important in mobile or portable devices, where energy efficiency of the mobile graphics processors is the most important design challenge [1]. To reduce the power consumption, it is shown that the amount of external memory access is the most crucial factor [2]. Moreover, large amount of external memory access in graphics systems causes not only the energy consuming problem but also the performance bottleneck [3]. In the rendering pipeline of graphics processors, there are five types of memory access, Texture Read (TR), Depth Buffer (Z-Buffer) Read (ZR), Depth Buffer (Z-Buffer) Write (ZW), Color Read (CR), and Color Write (CW) [1]. Among these memory bandwidth demands, the ZR bandwidth occupies at least 40% [3]. This means the depth buffer bandwidth is the first critical part that should be optimized in a low power rendering pipeline.

Many depth buffer bandwidth minimization algorithms have been proposed in literatures. The Hierarchical Z-Buffer [4] is a well-known high efficient algorithm to cull occluded fragments; however, since this algorithm is too complex and has irregular operations, it is not suitable for hardware implementation. Morein proposed an algorithm to simplify the Hierarchical Z-Buffer [3], where the screen is segmented into several tiles, and the maximun z-value of each tile, Z_{max}, is stored in a Z_{max}-Buffer as a low resolution Z-Buffer. When a new tile is traversed, the corresponding Z_{max} will be fetched. If the smallest z-value of the new tile is larger than Z_{max}, all the fragments of the current tile will be culled. Compared with conventional Z-test, Z_{max} test can remove redundant memory bandwidth and operation power of the the occluded tiles. Möller proposed a new algorithm in different point of view [1], where the minimun z-values, Z_{min}, are stored in a Z_{min}-Buffer instead. When a new tile is traversed, the Z_{min} of the corresponding tile will be fetched. If the largest z-value of the new tile is smaller than Z_{min}, the whole current tile should be visible. That is, this tile definitely occludes the previous tile. Thus the pixel based Z-Buffer test can be ignored, which can reduce the ZR bandwidth. Möller also indicates that in a normal scene, whose depth complexity is below four, the Z_{min} test can reduce more memory bandwidth than Z_{max} test. Relative to the tile base scheme, Yu and Kim proposed a pixel-based depth plane filter (DF) algorithm [5]. It setups a DF by a z-value in screen coordinates. This algorithm can be described with an example. If the z-value of a fragment is smaller than the DF, a DF-flag is set as “1” and stored to a DF-flag buffer. When the next fragment in the same screen position is traversed, the corresponding DF-flag will be fetched. If the z-value of the current fragment is greater than the DF, a DF-flag is set as “0” and compared with the previous DF-flag fetched from the buffer. In this example, current DF-flag is smaller than previous DF-flag, which represents that the current fragment is behind the previous fragment and can be culled from the pipeline. Note that in this algorithm, the position of the depth plane is the most crucial parameter to affect the performance. In realtime interactive applications with dynamic scenes, however, it is hard to gather the statistics of the depth information of all fragments to derive the optimal depth plane.

We think better depth filter performance can be achieved with considering more occlusion conditions of a tile. Therefore, in this paper, we propose an algorithm, Adaptive Tile Depth Filter (ATDF), to reduce the depth buffer bandwidth for low power graphics systems. It can classify the fragments into four categories, occluded, non-occluded, first-drawn, and uncertain in the beginning stage of the rendering pipeline. If a fragment is found to be occluded, this fragment will be rejected from the pipeline to save redundant operation power and memory bandwidth. For a non-occluded or first-drawn

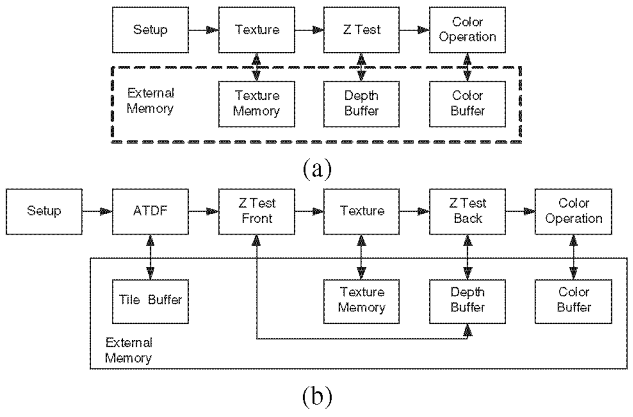

Fig. 1. Illustration of (a) conventional rendering pipeline and (b) rendering pipeline with ATDF.

fragment, the depth buffer test will be ignored to save the ZR bandwidth. In addition to save memory accesses during depth test, the bandwidth for clearing memory in the initialization stage can also be prevented with considering the first-drawn situation as HyperZ system does [3]. Finally, if the category of a fragment is uncertain, it means the ATDF fails to classify the fragment, and all the rendering operations will be performed in the render pipeline.

The organization of this paper is as follows. The proposed algorithm is described in Sec. II. Next, in Sec. III, the experimental results will be shown. Finally, a short conclusion is given in Sec. IV.

II. Adaptive Tile Depth Filter

A simplified conventional rendering pipeline can be illustrated in Fig. 1(a), while Fig. 1(b) shows the rendering pipeline with ATDF. In Fig. 1(b), for the normal graphic scene without alpha test, stencil test, or texture transparency, Z-Buffer test can be performed before texturing (Z Test Front); otherwise, the Z-Buffer test after texturing (Z Test Back) should be performed. In the normal scene with Z Test Front, the memory bandwidth can be described as [1]:

$$\begin{aligned} M = {} & \alpha \times ZR + \beta \times ZR \\ & + \beta \times (ZW + CR + CW + TR), \end{aligned} \tag{1}$$

where α is the number of the occluded fragments, and β is the number for the non-occluded fragments. On the other hand, for the scenes requiring Z Test Back, the memory bandwidth will be changed to:

$$\begin{aligned} M = {} & \alpha \times (ZR + TR) + \beta \times ZR \\ & + \beta \times (ZW + CR + CW + TR). \end{aligned} \tag{2}$$

The Z_{max} and DF algorithms could avoid the $\alpha \times ZR$ or $\alpha \times (ZR + TR)$ cost in terms of memory bandwidth, while the Z_{min} algorithm could avoid the $\beta \times ZR$ cost. ATDF provides the better filter rate to avoid the redundant memory bandwidth of both occluded and non-occluded fragments by use of both Z_{max} and Z_{min} concepts. In addition, with two new techniques, coverage mask and adaptive tile mode, the depth filter performance can be further improved. The details of the proposed algorithm will be described in the next subsections.

A. *Coverage Masks And Tile Decomposition*

The proposed depth filter algorithm can be illustrated in Fig. 2. In each figure, it shows the fragment distribution along the z axis in a tile, where the horizontal axis represents x-y axis of the screen, and the circles represent the fragments in the tile. When a triangle is rendered, the tiles covered by it are traversed one-by-one. For one of the tiles, in the following descriptions, the “z-values of the previous tile” denotes the z-values of the rendered fragments in the tile before rendering the current triangle, and the “z-values of the current tile” denotes the fragment z-values of the current triangle in the tile. Figure 2(a) shows how the Z_{max} and Z_{min} values can be generated from z-values of the previous tile. With Z_{max} and Z_{min}, the z-space can be separated into three regions: non-occluded, uncertain, and occluded. For the fragments of the current tile, whose z-value distribution is shown in Fig. 2(b), all fragments in the occluded region could be rejected from the pipeline, and the fragments in the non-occluded region could be rendered without checking the Z-Buffer to avoid the ZR bandwidth. These two kinds of the filtered fragments are shown as the solid circles in the figure. For the fragments in the uncertain region, which are shown as the hollow circles, they will be passed into the pipeline to perform all the pixel-based operations including ZR.

To further filter more fragments in the uncertain region, we propose a technique by use of coverage masks to record if each pixel is drawn or not until now with one bit. When a new tile is traversed, not only the Z_{max} and Z_{min} will be fetched, but also the coverage mask will be read. Figure 2(c) shows the Z_{max} and Z_{min} values and the coverage mask of the previous tile, where “0” denotes that “the pixel has been drawn,” and “1” denotes that “the pixel has not been drawn until now (empty pixels).” With the coverage masks, we could filter more fragments in the uncertain region. As shown in Fig. 2(d), the solids circles in the uncertain region are at the positions of the empty pixels; therefore, we can render these fragments without ZR.

For the worst case shown in Fig. 2(e), where all the fragments of the current tile belong to uncertain region, both the Z_{max} and Z_{min} algorithms cannot filter out any fragments. The tile can be decomposed into to sub-tiles as shown in Fig. 2(f). Each sub-tile has its own Z_{max} and Z_{min} values, ZL_{max}, ZL_{min}, ZR_{max}, and ZR_{min}. With this tile decomposition technique, the fragments represented by the solid circles can be filtered, and only a few hollow circles are in the uncertain region. Tile decomposition in this case can achieve a great improvement of the filter rate.

B. *Tile Mode*

With the hardware resource and power constraints, it is reasonable to assume that the hardware platform for mobile

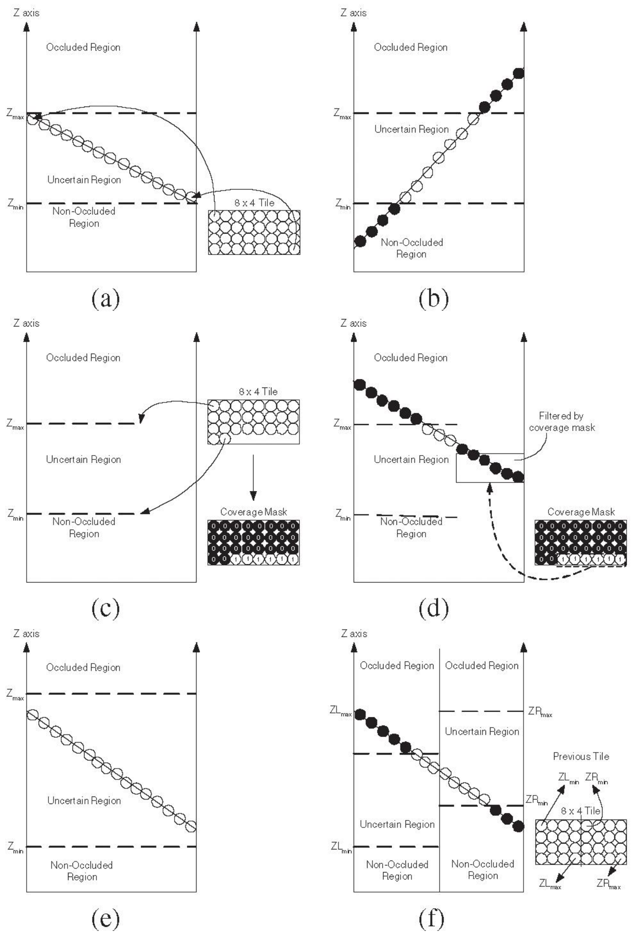

Fig. 2. Illustration of the proposed ATDF algorithm.

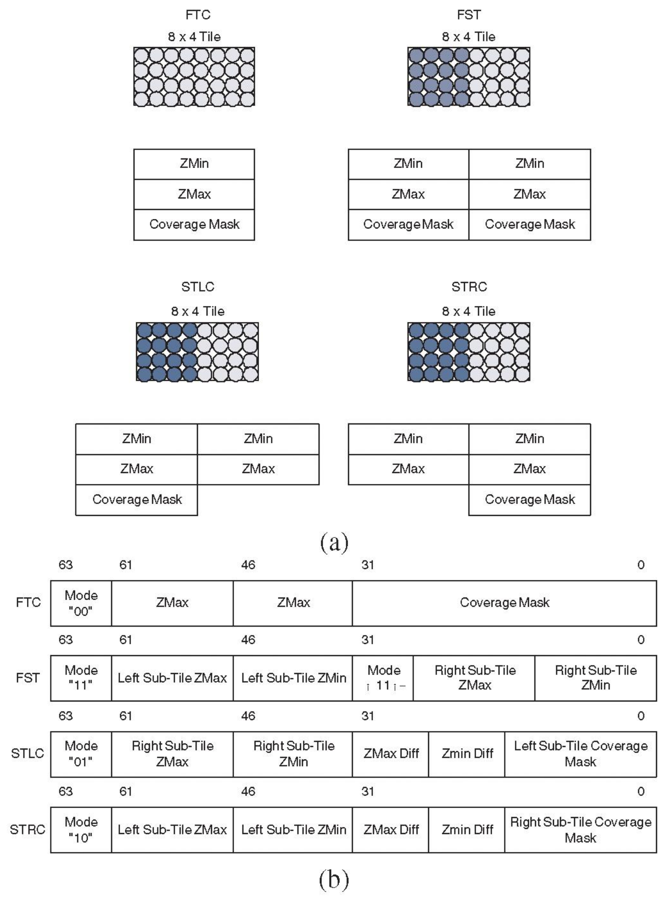

Fig. 3. (a) Illustration of the four tile moded. (b) The associated data format.

devices has one 16MB external memory with 32-bit bus. To efficiently utilize the limited bandwidth, the depth, color, and texture data format are often 16-bit for each sample. According to the above system configuration, without increasing a lot of extra bandwidth overhead, the tile size of ATDF is 8×4, and each tile uses 64 bits to store Z_{max}, Z_{min}, and coverage mask information.

For an 8×4 tile, there are four different tile modes, as shown in Fig. 3(a), and the associated tile data format for each mode is shown in Fig. 3(b). Except for the first mode, 8×4 full tile mode, the other three modes are the sub-tile modes, where each sub-tile uses 32 bits to store the related information. As shown in Fig. 3(b), the tile mode is recorded with the most significant two bits. “00” denotes the first mode called Full Tile with Coverage Mask (FTC). In the FTC, the Z_{max} and Z_{min} values are stores with 15-bit precision, respectively, and the 32-bit coverage mask is used to indicate if each pixel in the tile is covered (drawn) or not. The second type called Full Sub-Tile mode (FST) is represented by the code “11.” In this mode, there is no coverage mask, and two set of Z_{max} and Z_{min} values are used for the two sub-tiles. The other two modes called Sub-Tile with Left/Right Coverage Mask (STLC/STRC) are encoded as “01” or “10.” In these two modes, we still would like to limit the data size to 64 bits per tile. Therefore, the differential pulse code modulation (DPCM) and quantization concepts are adopted to compress the Z_{max} and Z_{min} information. For example, in STLC, 15-bit Z_{max} and Z_{min} values are stored for the right sub-tile, and the Z_{max} and Z_{min} values of the left sub-tile can be stored with 8-bit difference between the values of the right sub-tile and the left sub-tile. Similar to FTC, the 16-bit sub-tile coverage mask is used to indicate if each pixel in the 4×4 region is covered or not.

ADTF adaptively changes the tile mode according to the coverage masks statistic. Figure 4 shows the finite state machine for the tile mode decision. At first, the tile buffer is cleared to FTC mode. Note that, compared with the conventional approaches to clear the Z-Buffer, we can clear the tile buffer instead, whose required memory bandwidth is only one eighth. In the early stage of drawing a scene, it is efficient to use only the coverage mask to determine if ZR is required for the fragments. When the coverage mask bits in FTC mode are all “0,” which means all the pixels of the tile are covered, the coverage mask become useless, and the tile is changed to FST mode. At this time, more Z_{max} and Z_{min} information of the two sub-tiles can be used for more effective depth filter. In another case when the coverage mask bits of the left sub-tile

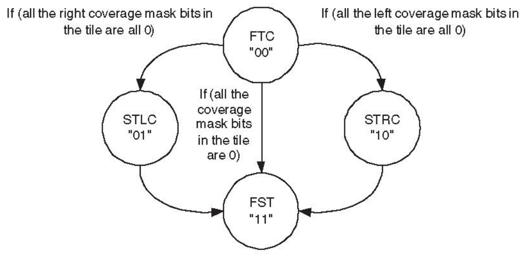
If (all the right coverage mask bits in the tile are all 0)
FTC "00"
If (all the left coverage mask bits in the tile are all 0)
STLC "01"
If (all the coverage mask bits in the tile are 0)
STRC "10"
FST "11"

Fig. 4. Finite state machine of tile mode decision.

TABLE I
FRAGMENT PROFILE WITH ATDF.

Scene (Depth Complexity)	Occluded(%)	Non-occluded(%)	Un-covered(%)	Un-certain(%)	Total pixels
DolphinVs (0.97)	0.4	44	6	50	116,953
Billboard (2.63)	4	84	5	7	316,419
Random Cylinder (6.3)	60	19	6	15	502,753

in FTC are all "0," it means the 16 bits of the left sub-tile's coverage mask are useless, and the bandwidth will be wasted for fetching these bits. To prevent the bandwidth waste, the tile mode should change to STRC. The left sub-tile still retains the same Z_{max} and Z_{min} precision, while the right sub-tile can take advantage of the coverage mask to improve the depth filter rate. Similarly, the mode will be changed from STLC/STRC to FST when bits of the coverage mask are all "0."

III. EXPERIMENTAL RESULTS

Figure 5 shows the sketches of the three test applications, DolphinVS, Billboard [6], and Random Cylinder. With these test applications, the fragment profile with ATDF is shown in Table I. In the worst case, DolphinVS, ATDF still can achieve 50% filter rate, which composes of the occluded, non-occluded, and uncovered fragments. Table II shows the memory bandwidth reduction compared with the previous works. The work $Z_{max} + Z_{min}$ is the method combining the two schemes proposed by Möller [1] and Morein [3]. The DF algorithm [5] with three depth planes is also selected for comparison. The result shows that ATDF outperforms the other works no matter in low depth complexity scene, such as DolphinVS, or high depth complexity scene, such as Billboard and Random Cylinder.

Although all the three depth filter schemes have the extra tile data bandwidth overhead, it could be ignored while compared to the bandwidth reduction when using ATDF. On the other hand, in DF or $Z_{max}+Z_{min}$ scheme, if the depth filter cannot achieve good filter rate, the extra bandwidth will become the burden in the whole pipeline, such as the case of applying DF on DolphinVS. Note that, in our simulation platform, we use a 32 bank full associated cache to reduce the bandwidth overhead in every work.

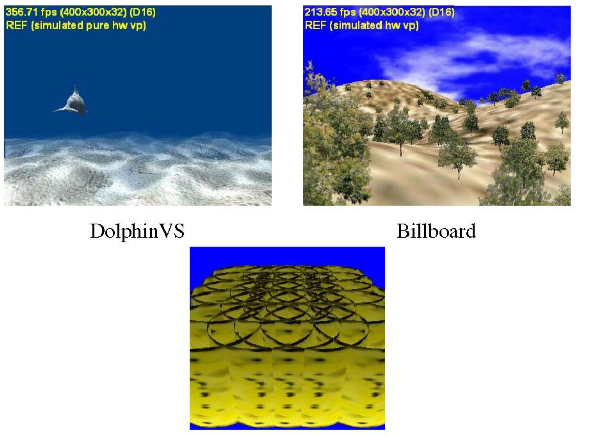
356.71 fps (400x300x32) (D16)
REF (simulated pure hw vp)
213.65 fps (400x300x32) (D16)
REF (simulated hw vp)
DolphinVS
Billboard
Random Cylinder

Fig. 5. Test application scenes.

TABLE II
BANDWIDTH REDUCTION COMPARISON.

Scene	Reduction of $Z_{max} + Z_{min}$ (%)	Reduction of DF (%)	Reduction of ATDF (%)
DolphinVs	5.00	-2.99	5.66
Billboard	15.15	12.83	16.84
Random cylinder	5.11	17.92	39.94

IV. CONCLUSION

In this paper, we propose an ATDF filter algorithm to filter out the occluded, non-occluded, and first-drawn fragments, with combining Z_{max} and Z_{min} algorithms with coverage mask and adaptive tile size techniques. By classifying the fragments with the proposed algorithm in the early stage of the rendering pipeline, the system could reduce the redundant operation power and memory bandwidth. Experiments show that the filter rate of more than 50% can be achieved, and the bandwidth reduction is much higher than those of the previous works, especially for complex scenes. The ATDF filter algorithm not only could achieve great memory bandwidth reduction in low power systems but also could fit to the normal PC-based graphic systems.

REFERENCES

[1] T. Akenine-Möller and J. Ström, "Graphics for the masses: A hardware rasterization architecture for mobile phones," in *Proc. ACM SIGGRAPH 2003*, July 2003, pp. 801–808.
[2] R. Fromm, S. Perissakis, N. Cardwell, C. Kozyrakis, B. McGaughy, D. Patterson, T. Anderson, and K. Yelick, "The energy efficiency of IRAM architectures," in *Proc. International Symposium on Computer Architecture*, June 1997, pp. 327–337.
[3] S. Morein, "ATI Radeon HyperZ technology," in *Hot3D Proc. ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, Aug. 2000.
[4] N. Greene, M. Kass, and G. Miller, "Hierarchical Z-Buffer visibility," in *Proc. of ACM SIGGRAPH 1993*, 1993, pp. 231 – 238.
[5] C.-H. Yu and L.-S. Kim, "A hierarchical depth buffer for minimizing memory bandwidth in 3d rendering engine: Depth Filter," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS'03)*, May 2003, pp. II–724 – II–727.
[6] (2005) Microsoft DirectX SDK. [Online]. Available: http://msdn.microsoft.com/directx/