Order Independent Transparency for Image Composition Parallel Rendering Machines

Woo-Chan Park[1], Tack-Don Han[2], and Sung-Bong Yang[2]

[1] Department of Internet Engineering,
Sejong University, Seoul 143-747, Korea,
pwchan@sejong.ac.kr
[2] Department of Computer Science,
Yonsei University, Seoul 120-749 Korea,
{hantack}@kurene.yonsei.ac.kr
{yang}@cs.yonsei.ac.kr

Abstract. In this paper, a hybrid architecture composed of both the object-order and the image-order rendering engines is proposed to achieve the order independent transparency on the image composition architecture. The proposed architecture utilizes the features of the object-order which may provide high performance and the image-order which can obtain the depth order of all primitives from a viewpoint for a given pixel. We will discuss a scalable architecture for image order rendering engines to improve the processing capability of the transparent primitives, a load distribution technique for hardware efficiency, and a preliminary timing analysis.

1 Introduction

3D computer graphics is a core field of study in developing multi media computing environment. In order to support realistic scene using 3D computer graphics, a special purpose high performance 3D accelerator is required. Recently, low cost and high performance 3D graphics accelerators are adopted rapidly in PCs and game machines[1,2,3].

To generate high-quality images, solid models composed of more than several millions polygons are often used. To display such a model at a rate of 30 frames per second, more than one hundred million polygons must be processed in one second. To achieve this goal in current technology, several tens of the graphic processors are required. Thus, parallel rendering using many graphics processors is an essential research issue.

According to [4], graphics machine architectures can be categorized into the three types: *sort-first architecture*, *sort-middle architecture*, *sort-last architecture*. Among them, sort-last architecture is a scalable architecture because the required bandwidth of its communication network is almost constant against the number of polygons. Thus, sort-last architecture is quite suitable for a large-scale rendering system.

P.-C. Yew and J. Xue (Eds.): ACSAC 2004, LNCS 3189, pp. 449–460, 2004.

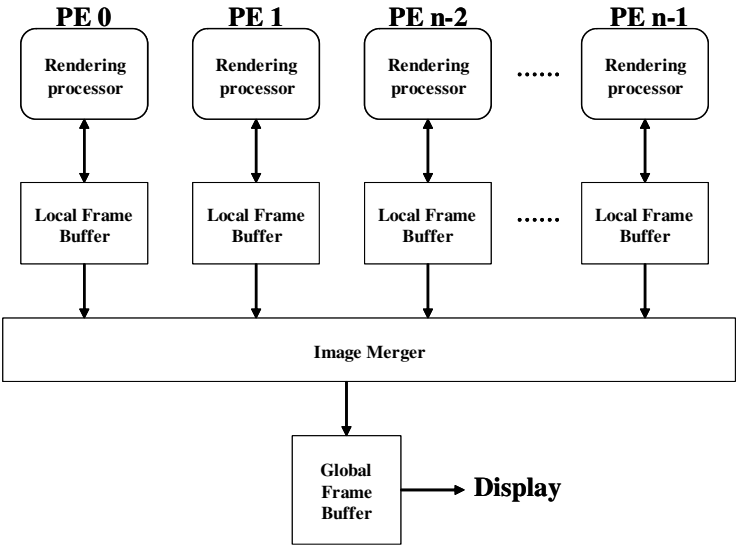

Fig. 1. An image composition architecture

One of the typical sort-last architectures is an image composition architecture [5,6,7,8,9]. Figure 1 shows the overall structure of image composition architecture. All polygonal model data are distributed into each rendering processor which generates a subimage with its own frame buffer, called a local frame buffer. The contents of all the local frame buffers are merged periodically by the image merger. During image merging, the depth comparisons with the contents of the same screen address for each local frame buffer should be performed to accomplish hidden surface removal. The final merged image is then transmitted into the global frame buffer.

In a realistic 3D scene, both opaque and transparent primitives are mixed each other. To generate a rendered final image properly with the transparent primitives, the order dependent transparency problem must be solved. That is, the processing order depends on the depth order by a viewpoint, not on the input order. Order dependent transparency problem may cause the serious performance degradation as the number of transparent primitives increases rapidly. Therefore, the order independent transparency, the opposite concept of the order dependent transparency, is a major for providing high performance rendering systems. But up until now we have not found any parallel 3D rendering machine supporting hardware accelerated order independent transparency.

In this paper, we propose a new method of the hardware accelerated order independent transparency for image composition in the parallel rendering architecture. To achieve this goal, we suggest a hybrid architecture composed of both the object order and the image order rendering engines. The proposed mechanism utilizes the features of the object order which may provide high performance and the image order which can obtain the depth order of all primitives from a viewpoint for a given pixel.

The proposed architecture has n object order rendering engines, from PE 0 to PE n-1 as in Figure 1 and one image order rendering engine, PE n, where n is the number of PEs. All the opaque primitives are allocated with each object order rendering engine. All the transparent primitives are sent to the image order rendering engine. Thus subimages generated from PE 0 to PE n-1 are merged into a single image during image merging. This merged image is fed into the image order rendering engine, PE n. With this single merged image and the information of the transparent primitives, PE n calculates the final image with order independent transparency. We also provide a scalable architecture for the image order rendering engines to improve the processing capability of transparent primitives and load distribution technique for hardware efficiency.

In the next section, we present a brief overview of the object order rendering, the image order rendering, the order independent transparency. Section 3 illustrates the proposed image composition architecture and its pipeline flow. We describe how the proposed architecture handles the order independent transparency. The timing analyses of the image merger and the image order rendering engine, and the scalability are also discussed. Section 4 concludes this paper with future research.

2 Background

In this section, we present a brief overview of the object order rendering and the image order rendering methods. We also discuss the order independent transparency.

2.1 Object Order and Image Order Rendering Methods

Polygonal rendering algorithms can be classified into the object order rendering and the image order rendering according to the rendering processing order for the input primitives [10]. In the object order rendering the processing at the rasterization step is performed primitive by primitive, while in the image order it is done pixel by pixel. According to their features, an object order rendering system is suitable for high performance systems and current design approaches [1,2,3,5,9], while the image order is low cost systems and later approaches [11,12, 13,14]. In most of parallel rendering machines, an object order rendering engine is adopted for each rendering processor for high performance computation.

The rendering process consists of geometry processing, rasterization, and display refresh steps. In an object order rendering system, the processing order at the rasterization step is performed primitive by primitive. Thus the geometry processing and the rasterization steps are pipelined between primitives and the object order rendering and the display refresh steps are pipelined between frames. An object order rendering system must have a full-screen sized frame buffer, consisted of a depth buffer and a color buffer, for hidden surface removal operations. To overlap the executions of the rendering and the screen refresh

steps, double buffering for the frame buffer are used. The front frame buffer denotes the frame buffer used in the rendering step and the back frame buffer is used by the screen refresh step. Figure 2 shows the pipeline flow of the object order rendering system between two frames.

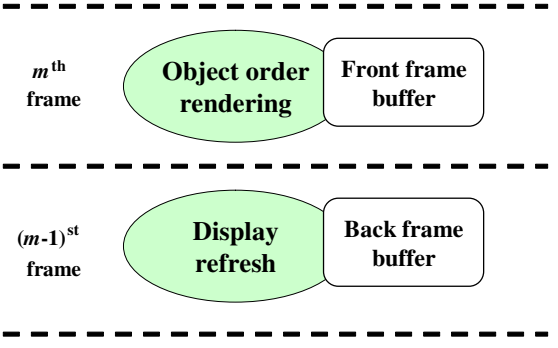

Fig. 2. The pipeline flow of the object order rendering method between two frames

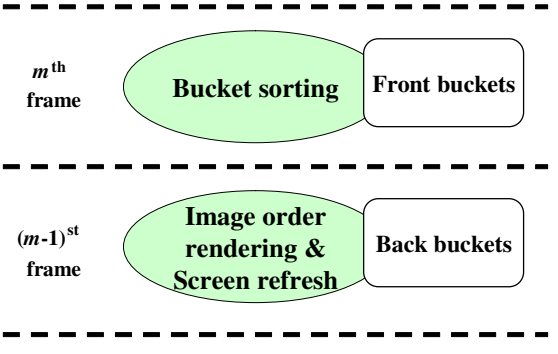

Fig. 3. The pipeline flow of the image order rendering method between two frames

In an image order rendering system, the processing order at the rasterization step is performed pixel by pixel. For example, if the screen address begins from (0, 0) and ends with (v, w), the rasterization step is accomplished from (0, 0) to (v, w). In rendering pixel by pixel, the lists of the all primitives, after geometry transformation, overlaying with a dedicated pixel must be kept for each pixel. These lists are called the *buckets* and bucket sorting denotes this listing. Thus the bucket sorting step including geometry processing and the image order rendering step including screen refresh are pipelined between frames. Figure 3 shows the pipeline flow of the image order rendering system between two frames.

The scan-line algorithm is a typical method of the image order rendering [10]. All primitives transformed into the screen space are assigned into buckets provided per scan-line. To avoid considering primitives that do not contribute to the

current scan-line, the scan-line algorithm requires primitives to be transformed into the screen space and to sort the buckets according to the first scan-line in which they appear. After bucket sorting, rendering is performed with respect to each scan-line. Because the finally rendered data are generated by scan-line order, screen refresh can be performed immediately after scan-line rendering. Thus no frame buffer is needed and only a small amount of buffering is required. In this paper, the scan-line algorithm is used in the image order rendering engine.

2.2 Order Independent Transparency

In a realistic 3D scene, both opaque and transparent primitives are mixed each other. To generate a final rendered image properly, the order dependent transparency problem should be solved. Figure 4 shows an example of order dependent transparency with three fragments for a given pixel.

In Figure 4, A is a yellow transparent fragment, B is a blue opaque fragment, and C is a red opaque fragment. We assume that A is followed by B and B is followed by C in the point of depth value for the current view point. Then the final color results in green which is generated by blending the yellow color of A and the blue color of B. However, if the processing order of these three fragments are C, A, and B, then a wrong color will be generated. That is, when C comes first, the current color is red. When A comes next, the current color becomes orange and the depth value of the current color is that of A. When B comes last, B is ignored because the current depth value is smaller then depth value of B. Therefore, the calculated final color is orange, which is wrong. Therefore, the processing orders should be the same as the depth order to achieve the correct final color.

The order dependent transparency problem may cause serious performance degradation as the number of transparent primitives increases rapidly. Therefore, the order independent transparency is a crucial issue for high performance rendering systems. But, up until now we have not found any high performance 3D rendering machine supporting hardware accelerated.

Several order independent transparency techniques for object order rendering based on the A-buffer algorithm have been proposed [15,16,17,18,19]. But, these algorithms require either, for each pixel, the infinite number of lists of all the primitives overlaying with the dedicated pixel or multiple passes which are not suitable for high performance. On the other hand, image order rendering

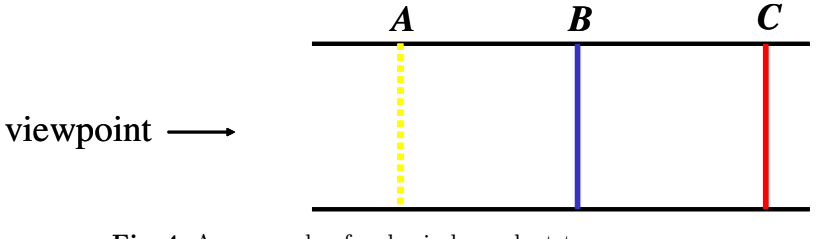

Fig. 4. An example of order independent transparency

techniques to support order independent transparency based on the A-buffer algorithm have been proposed in [13,14]. However, high performance rendering cannot be achieved with image order rendering technique.

3 Proposed Image Composition Architecture

In this section, an image composition architecture supporting hardware accelerated order independent transparency is proposed. We discuss its execution flow, preliminary timing analysis, and scalability.

3.1 Proposed Image Composition Architecture

Figure 5 shows the block diagram of the proposed image composition architecture, which can be divided into rendering accelerator, frame buffer, and display subsystems. The proposed architecture is capable of supporting hardware accelerated order independent transparency. To achieve this goal, hybrid architecture made up of the object order and the image order rendering engines are provided. The proposed mechanism utilizes the advantages of both the object order rendering and the image order rendering methods.

The proposed architecture has n object order rendering engine(ORE)s from PE 0 to PE n-1, and PE n with an image order rendering engine(IRE). Each PE consists of geometry engine(GE), either ORE or IRE, and a local memory which

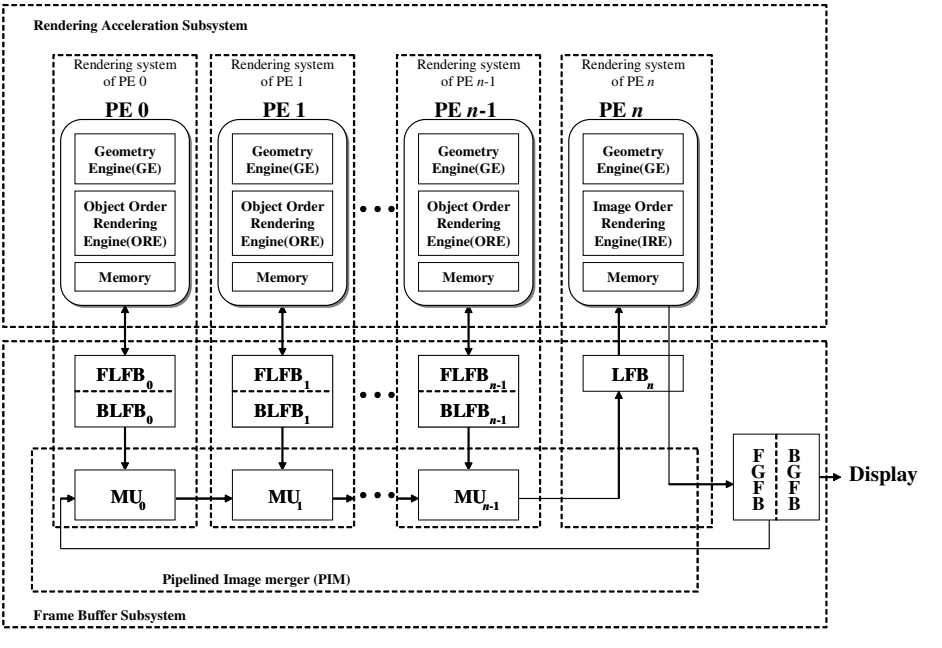

Fig. 5. Block diagram of the proposed image composition architecture

is not a frame memory but a working memory for the PE. The rendering system from PE 0 to PE n-1 is identical to that of the conventional sort-last architecture, while PE n is provided to achieve OIT. The local frame buffer(LFB) of each ORE is double-buffered so that the OREs can generate the image of the next frame in parallel with the image composition of the previous frame. One buffer is called the front local frame buffer(FLFB) and the other is called the back local frame buffer(BLFB). This double buffering scheme is also used in the buckets of IRE and in global frame buffer(GFB).

In the proposed architecture, the rendering systems from PE 0 to PE n-1 perform rendering all opaque primitives with an object order base and PE n performs rendering all transparent primitives with an image order base. Subimages generated from PE 0 to PE n-1 are merged into a single image which is transmitted into LFB_n. With this single merged image and the information of the transparent primitives, IRE calculates the final image with order independent transparency.

Pipelined image merger(PIM), provided in [6], is made up of a linearly connected array of n merging unit(MU)s. It performs image merging with n BLFBs and transmits the final merged image into GFB in a pipelined fashion. Each MU receives two pixel values and outputs the one with the smaller screen depth. We let the screen address begins from (0, 0) and ends with (v, w). BLFB_0 denotes the BLFB of PE 0, BLFB_1 is the BLFB of PE 1, and so on. MU_0 denotes the MU connected with PE 0, MU_1 is the MU connected with PE 1, and so on.

The execution behavior of PIM can be described as follows. In the first cycle, MU_0 performs depth comparison with (0, 0)'s color and the depth data of BLFB_0 and (0, 0)'s color and depth data of FGFB, respectively, and transmits the results of color and depth data into MU_1. In the next cycle, MU_0 performs depth comparison with (0, 1)'s color and depth data of BLFB_0 and (0, 1)'s color and depth data of FGFB, and transmits the results of color and depth data into MU_1. Simultaneously, MU_1 performs depth comparison with (0, 0)'s color and depth data of BLFB_1 and the result values fed into the previous cycle, and transmits the results of color and depth data into MU_2. As PIM executes in this pipelined fashion, the final color data can be transmitted into FGFB.

3.2 Execution Flow of the Proposed Image Composition Architecture

All opaque primitives are allocated to each ORE and all transparent primitives are sent to IRE. A primitive allocation technique for load balance on the opaque primitives has been provided in [6]. In the first stage, from PE 0 to PE n-1, object order rendering is performed for the dedicated opaque primitives with geometry processing and rasterization steps. Simultaneously, PE n executes the bucket sorting for image order rendering through geometry processing with the transparent primitives. As a processing result, the rendered subimages for the opaque primitives are stored from FLFB_0 to FLFB_{n-1} and the bucket sorted result for the transparent primitives are stored in buckets, which reside in the local memory of PE n.

In the next step, the subimages stored from BLFB_0 to BLFB_{n-1} are merged by PIM according to the raster scan order, from (0, 0) to (v, w), and the final merged image is transmitted into LFB_n with a pipelined fashion. Simultaneously, IRE performs image order rendering with LFB_n, which hold the rendered result of all opaque primitives and the bucket sorted results of all transparent primitives. To perform simultaneously both the write operation from MU_{n-1} and the read operation from IRE for two different memory addresses, a two-port memory should be used in LFB_n.

Using the scan-line algorithm for image order rendering, IRE should check all the color and depth values of the current scan-line of LFB_n. Therefore, they should be transmitted completely from MU_{n-1} before performing rendering operation for the current scan-line. Thus, IRE cannot perform rendering operation until all color and depth values of the 0^{th} scan-line in LFB_n are transmitted completely from MU_{n-1}. But the transfer time between MU_{n-1} and LFB_n is too short to affect overall performance, as shown in Section 3.5. The final rendering results of IRE are generated and transmitted into GFB scan-line by scan-line order. Finally, GFB performs the screen refresh operation.

3.3 Pipeline Flow of the Proposed Image Composition Architecture

Figure 6 shows a pipelined execution of the proposed image composition architecture with respect to three frames. If the current processing frame is m, object order rendering is performed from PE 0 to PE n-1 with all the opaque primitives and PE n executes the bucket sorting for image order rendering with all the transparent primitives. The rendering results for the opaque primitives are stored from FLFB_0 to FLFB_{n-1} and the bucket sorted results for the transparent primitives are stored in front buckets. Simultaneously, for the $(m\text{-}1)^{st}$ frame, image merging from BLFB_0 to BLFB_{n-1} is performed by PIM according to the raster scan order, and the final merged image is transmitted into LFB_n with a pipelined fashion. Simultaneously, IRE of PE n performs image order rendering with LFB_n and the back bucket. The rendering results of IRE are generated and transmitted into GFB scan-line by scan-line. For the $(m\text{-}2)^{nd}$ frame, screen refresh is performed with a final merged image in FGFB.

3.4 Example of the Order Independent Transparency

Figure 7 shows an example of the order independent transparency for the proposed image composition architecture and illustrates the input and output values of OREs and IRE for a given pixel. For the current viewpoint, the depth values of A, B, C, D, E, F, and G are in increasing order, i.e., A is the nearest primitive and G is the farthest primitive. We assume that A is yellow and transparent, B is blue and opaque, C is red and transparent, D is blue and opaque, E is yellow and opaque, F is black and opaque, and G is green and transparent.

Among the opaque primitives (B, D, E, and F) generated in OREs, B is the final depth result of depth comparison. With the transparent primitives (A, C,

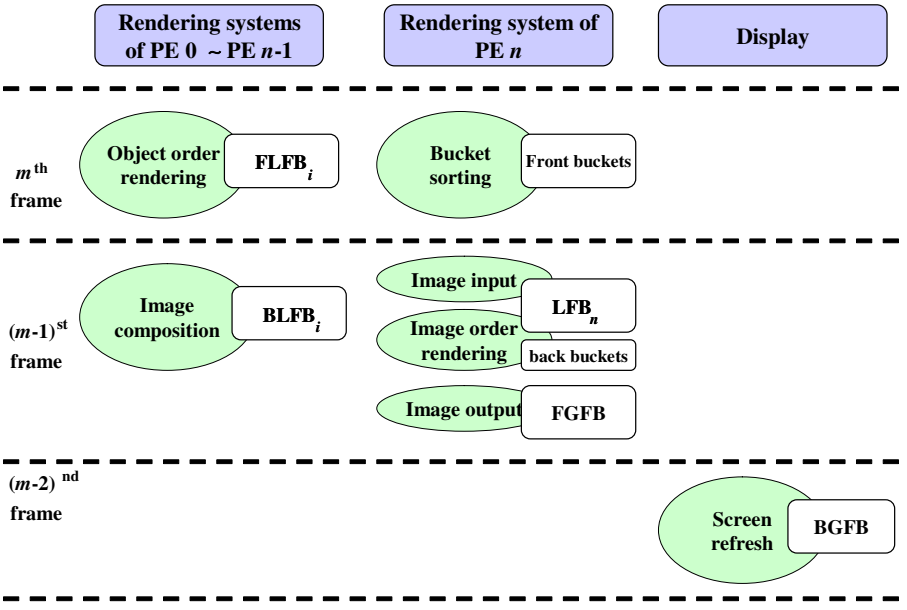

Fig. 6. Pipelined execution of the proposed architecture

and G) and B, image order rendering is performed in IRE. Therefore, green is generated as the final color with order independent transparency.

3.5 Timing Analysis of PIM and Image Order Renderer

In [6] each MU performs depth comparison and data selection through a 4-stage pipeline. Therefore, PIM, as a whole, constructs a $4n$-stage pipeline, where n represents the number of PEs. The time, T_m, needed for merging one full-screen image is equal to $(v + \beta) \cdot w \cdot t + 4 \cdot n \cdot t$, where v and w are horizontal and vertical screen sizes, respectively, t is the PIM clock period, and β is the number of PIM clocks required for overhead processing per scan-line in the GFB unit. The first term represents the time for scanning the entire screen and the second term represents the pipeline delay time. In [6] those parameters are $v = 640$, $w = 480$, $n = 16$, $\beta = 20$, and $t = 80$ nsec. Therefore, $T_m = 25.3$ msec, which is shorter than the target frame interval(33.3 msec).

Because the current screen resolution exceeds several times the resolution considered in [6] and the clock period of PIM can be shortened due to the advances in the semiconductor and network technologies, those parameters are not realistic for the current graphics environment and semiconductor technology. These parameters will be fixed after developing a prototype by the future work. However, considering the current technology, the parameters can be estimated reasonably as $v = 1280$, $w = 1024$, and $t = 20$ nsec, where n and β are not considered because the effect of those parameters is ignorable. Therefore, $T_m = 26.7$ msec, which is still sufficient time to support 30 frames per second.

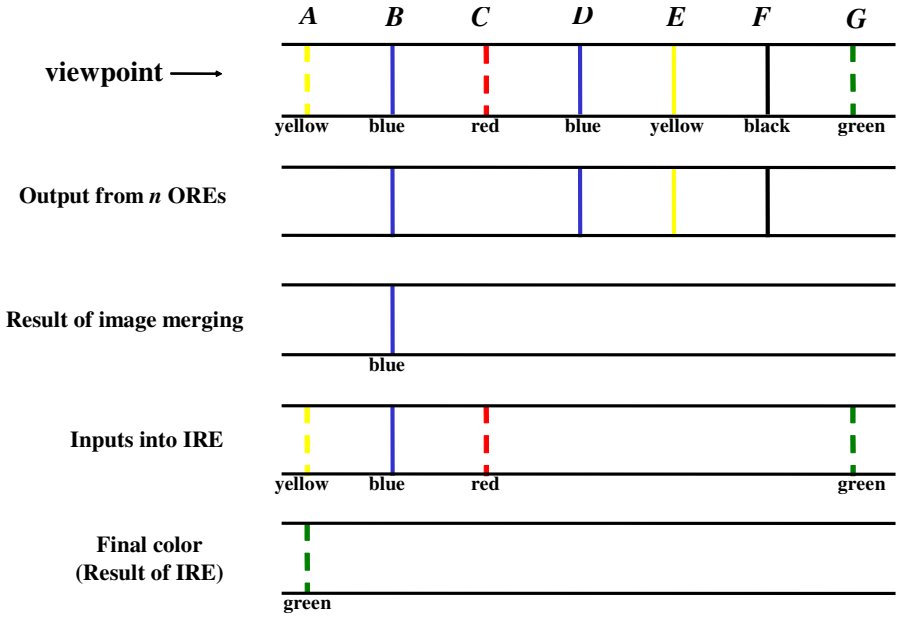

Fig. 7. An example of the order independent transparency on the proposed architecture

IRE cannot perform rendering operation until all color and depth values of the 0^{th} scan-line in LFB_n are transmitted completely from MU_{n-1}. T_{in} denotes the time taken for the transmission. T_{out} denote the time needed to transmit the rendered results of the final scan-line into GFB. T_{in} and T_{out} can be estimated as $(v+\beta)\cdot w\cdot t+4\cdot n\cdot t$ and $(v+\beta)\cdot w\cdot t$, respectively. Then, the actual time to perform the image order rendering at IRE is 33.3msec$-(T_{in}+T_{out})$. With $n=16$, $T_{in}=0.058$msec and $T_{out}=0.053$msec in the case of [6] and $T_{in}=0.038$msec and $T_{out}=0.026$msec in the case of the estimated parameters for the current technology. Therefore, the performance degradation due to T_{in} and T_{out} with $n=16$ is about $0.4\sim0.2\%$, which is negligible. Moreover, with n=1024, T_{in}=0.38msec and $T_{out}=0.053$msec in the case of [6] and $T_{in}=0.083$msec and $T_{out}=0.026$msec in the case of the estimated parameters. Therefore, the performance degradation due to T_{in} and T_{out} with $n=1024$ is about $1.3\sim0.4\%$, which is also negligible.

3.6 Scalability on the IRE

In Figure 5 only one IRE is used. Thus, when the number of the transparent primitives is so large that all transparent primitives cannot be processed within the target frame interval, the bottleneck point is the performance of IRE. Figure 8 shows the proposed image composition architecture with r IREs. To achieve scalable parallel processing for IRE, the per-scan-line parallelism is used for the scan-line algorithm.

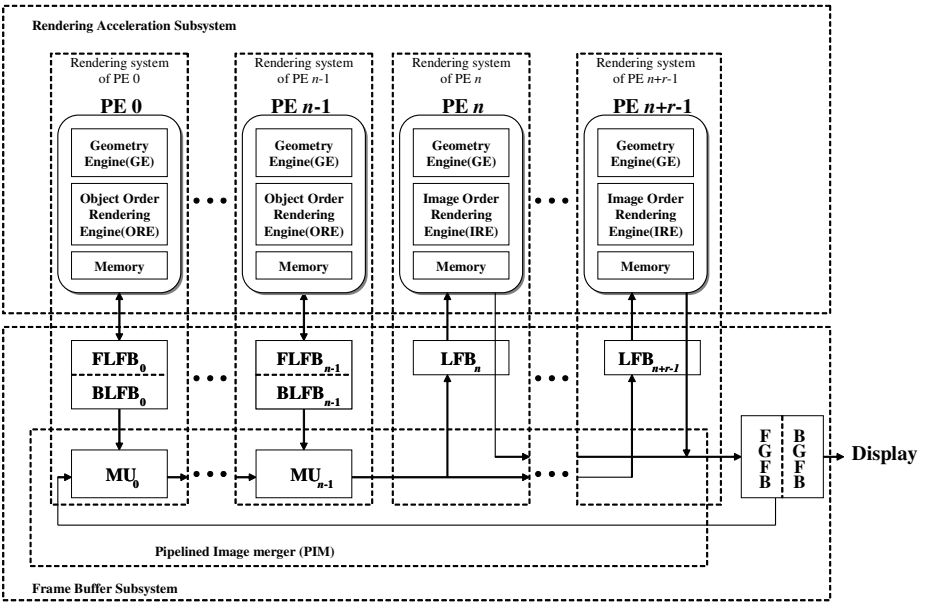

Fig. 8. Block diagram of proposed architecture with scalability on the IRE

In case that the full-screen consists of $r \cdot k$ scan-lines, each IRE has a copy of all transparent primitives and performs bucket sorting for all transparent primitives with dedicated k buckets instead of $r \cdot k$ buckets. Then, each IRE executes the image order rendering with an interleaving fashion. That is, IRE of PE n performs scan-line rendering with the 0^{th} scan-line, the r^{th} scan-line, and so on. IRE of PE n+1 performs scan-line rendering with the 1^{st} scan-line, the $(r+1)^{st}$ scan-line, and so on. By this sequence, all scan-lines can be allocated to r IREs. Simultaneously, subimages stored in BLFB_0 to BLFB_{n-1} are merged by PIM according to the raster scan order. Then, the merged image is transmitted into each LFB of IRE with an interleaving fashion. Therefore, overall performance of the image order rendering can be achieved with scalability. Finally, the rendered result of r IREs are also transmitted into GFB with an interleaving fashion.

4 Conclusion

In this paper, the order independent transparency problem for image composition in parallel rendering machines has been resolved by using hybrid architecture composed of both the object order rendering and the image order rendering engines. The proposed architecture is a scalable one with respect to both the object order rendering and the image order rendering engines.

References

1. M. Oka and M. Suzuoki. Designing and programming the emotion engine. IEEE Micro, 19(6):20–28, Nov. 1999.
2. A. K. Khan et al. A 150-MHz graphics rendering processor with 256-Mb embedded DRAM. IEEE Journal of Solid-State Circuits, 36(11):1775–1783, Nov. 2001.
3. Timo Aila, Ville Miettinen, and Petri Nordlund. Delay streams for graphics hardware. In Proceedings of SIGGRAPH, pages 792–800, 2003.
4. S. Molnar, M. Cox, M. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. IEEE Computer Graphics and Applications, 14(4):23–32, July 1994.
5. T. Ikedo and J. Ma. The Truga001: A scalable rendering processor. IEEE computer and graphics and applications, 18(2):59–79, March 1998.
6. S. Nishimura and T. Kunii. VC-1: A scalable graphics computer with virtual local frame buffers. In Proceedings of SIGGRAPH, pages 365–372, Aug. 1996.
7. S. Molnar, J. Eyles, and J. Poulton. PixelFlow: High-speed rendering using image composition. In Proceedings of SIGGRAPH, pages 231–240, July 1992.
8. J. Eyles, S. Molnar, J. Poulton, T. Greer, A. Lastra, N. England, and L. Westover. PixelFlow: The realization. In Proceedings of SIGGRAPH/Eurographics Workshop on graphics hardware, pp. 57–68, Aug. 1997.
9. M. Deering and D. Naegle. The SAGE Architecture. In Proceeddings of SIGGRAPH 2002, pages 683–692, July 2002.
10. J. D. Foley, A. Dam, S. K. Feiner, and J. F. Hughes. Computer Graphics, Principles and Practice. Second Edition, Addison-Wesley, Massachusetts, 1990.
11. J. Torborg and J. T. Kajiya. Talisman: Commodity Realtime 3D graphics for the PC. In Proceedings of SIGGRAPH, pages 353–363, 1996
12. M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt. The triangle processor and normal vector shader: A VLSI system for high performance graphics. In Proceedings of SIGGRAPH, pages 21–30, 1988.
13. M. Kelley, S. Winner, and K. Gould. A scalable hardware render accelerator using a modified scanline algorithm. In Proceedings of SIGGRAPH, pages 241–248, 1992.
14. M. Kelley, K. Gould, B. Pease, S. Winner, and A. Yen. Hardware accelerated rendering of CSG and transparency. In proceedings of SIGGRAPH, pages 177–184, 1994.
15. L. Carpenter. The A-buffer, and antialiased hidden surface method. In Proceedings of SIGGRAPH, pages 103–108, 1984.
16. S. Winner, M. Kelly, B. Pease, B. Rivard, and A. Yen. Hardware accelerated rendering of antialiasing using a modified A-buffer algorithm. In Proceedings of SIGGRAPH, pages 307–316, 1997.
17. A. Mammeb. Transparency and antialiasing algorithms implemented with virtual pixel maps technique. IEEE computer and graphics and applications, 9(4):43–55, July 1989.
18. C. M. Wittenbrink. R-Buffer: A pointerless A-buffer hardware architecture. In Proceedings of SIGGRAPH/Eurographics Workshop on graphics hardware, pages 73–80, 2001.
19. J. A. Lee and L. S. Kim. Single-pass full-screen hardware accelerated anti-aliasing. In Proceedings of SIGGRAPH/Eurographics Workshop on graphics hardware, pages 67–75, 2000.