David II: 효과적인 메모리 시스템을 가지는 병렬 렌더링 프로세서

이길환*, 박우찬**, 김일산*, 한탁돈*
*연세대학교 컴퓨터과학과
**세종대학교 인터넷공학과
e-mail : kiwh@kurene.yonsei.ac.kr

David II: A new architecture for parallel rendering processors with effective memory system

Kil-Whan Lee*, Woo-Chan Park**, Il-San Kim*,Tack-Don Han*
*Dept. of Computer Science, Yonsei University
**Dept. of Computer Engineering, Sejong University

요 약

Current rendering processors are organized mainly to process a triangle as fast as possible and recently parallel 3D rendering processors, which can process multiple triangles in parallel with multiple rasterizers, begin to appear. For high performance in processing triangles, it is desirable for each rasterizer have its own local pixel cache. However, the consistency problem may occur in accessing the data at the same address simultaneously by more than one rasterizer. In this paper, we propose a parallel rendering processor architecture, called *DAVID II*, resolving such consistency problem effectively. Moreover, the proposed architecture reduces the latency due to a pixel cache miss significantly. The experimental results show that DAVID II achieves almost linear speedup at best case even in sixteen rasterizers.

1. Introduction

Currently, high-performance rendering processors with moderate prices are announced and adopted in almost all of PCs. Even in the game consoles, such as Sony's PlayStation®2 [1] and MS X-Box, 3D graphics are accelerated with their own processors. Then, current rendering processors aim to process triangles (or a primitives) one at a time with their multiple pixel pipelines. However, the performance improvement by this approach is still insufficient to produce truly realistic scenes. Thus, the concurrent execution on multiple triangles at a time should be provided.

As the semiconductor technology advances, it is possible to produce a parallel rendering processor by integrating the multiple rasterizers into a single chip. Sony's GScube includes 16 graphics processing units (GPUs) integrated with 256-Mb embedded DRAM [2]. Because the output of the 16 GPUs is fed into a pixel merge IC which drives the data stream to a video display, each GPU must have its own frame buffer, which is similar to a sort-last parallel rendering machine as classified in [3]. Thus a large amount of embedded DRAM should be integrated.

Parallel rendering with a single frame buffer, which can be classified into the sort middle machine, causes the consistency problem in case more than one rasterizer accesses the data at the same address. In [4], a superscalar rendering processor (SRP) with superscalar principles for increasing the available parallelism is presented. The consistency problem is detected automatically by the dependency testing. But, the additional hardware for the superscalar execution, such as the dependency testing hardware, should be provided. Special manipulations in case of large triangles and the triangle strips are also required to boost the parallelism. Because an ideal multi-ported frame buffer is assumed, only triangle-level parallelism for the benchmarks is shown in their results.

In this paper, we propose a new parallel rendering processor architecture in which each rasterizer executes a conventional rasterization pipeline with its local pixel cache. We allow the consistency problem to arise in each pixel cache. But we maintain the consistency in the frame buffer by performing additional consistency-tests (C-tests) for all pixels within each pixel cache block, whenever it is written into the frame buffer. A C-test implies *z*-test and *alpha*-blending operations for a pixel. The proposed architecture also reduce significantly the latency due to a pixel cache miss by executing the rasterization pipeline immediately after transmitting the cache block on which a miss generated into

the memory interface unit (MIU).

To validate the proposed architecture, various simulation results with three benchmarks are given. We also calculate the memory latency reduction rates according to the number of rasterizers. We can achieve up to 90% zero-latency memory system even in sixteen rasterizers.

In the next section, we give a brief overview of a conventional rasterization pipeline flow. In Section 3, we propose our architecture. The three memory systems of the proposed architecture are discussed in Section 4. Various simulation results and performance evaluation are given in Section 5. Conclusions are presented in Section 6.

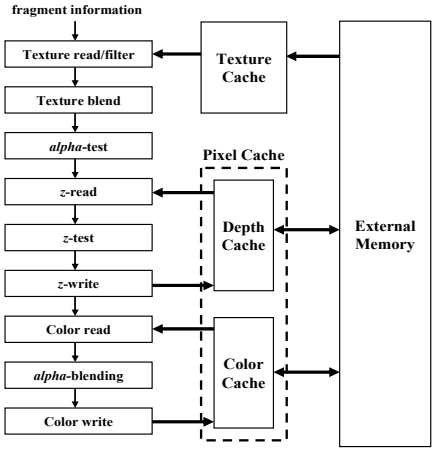


Fig. 1. A conventional pixel rasterization pipeline.

2. A Conventional Rasterization Pipeline

The rendering process consists of two stages: *geometry processing* and *rasterization*. A general rasterization pipeline consists of three steps: *triangle setup*, *edge-walk*, and *pixel rasterization*. A conventional pixel rasterization pipeline is shown in Fig. 1 [6]. The first two stages read four or eight texels from the texture cache, perform either bi-linear or tri-linear filtering on them to produce a single texel, and blend the texel with the pixel color. The *alpha*-value of the current fragment is then compared with that of the filtered texel. The next two stages read the *z*-value from the depth cache within the pixel cache and compare it with that of the current fragment. Observe that the pixel cache consists of the depth cache and the color cache, as shown by a dotted box in Fig. 1. If the *z*-test is successful, a new *z*-value is written into the depth cache. Finally, we read the color data from the color cache of the pixel cache, *alpha*-blend them with the result of texture blending, and then write the final color data back to the color cache.

3. The Proposed Architecture

Fig. 2 shows the proposed parallel rendering processor architecture. The issue unit does not exist, the ALUs for C-tests are inserted in between MIU and the frame buffer, and the pixel cache is locally placed on each rasterizer.

Each rasterizer performs the rasterization with its local texture cache and local pixel cache. As opposed to SRP, the dependency testing does not exist. Thus, triangle-level parallelism of the proposed architecture is certainly 100%. But, the consistency problem occurs undoubtedly in the pixel cache.

One of the main ideas of this paper is that we allow the consistency problem to arise in each pixel cache, but we maintain the consistency strictly in the frame buffer. The data in the pixel cache are transmitted into the frame buffer whenever a pixel cache miss occurs. It is also generated in flushing the pixel cache when the rasterization of the current frame is completed. In the proposed architecture, the consistency of the frame buffer is maintained by performing additional C-tests for each transmitted block from the pixel cache with the corresponding block on the frame buffer.

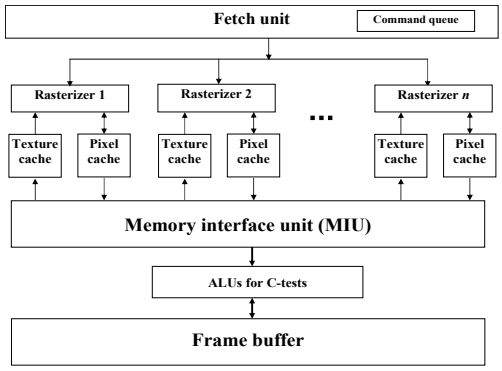


Fig. 2. The proposed architecture.

Another main idea is that the proposed architecture, even though a pixel cache miss occurs, does not wait until the cache miss handling is completed. The proposed architecture rather continues to execute the rasterization immediately after transmitting the cache block on which a miss generated into MIU. Thus, the latency due to a cache miss, which includes the time to transfer the corresponding block from the frame buffer into the pixel cache, is significantly reduced. Moreover, the rasterization pipeline and C-tests can be executed independently.

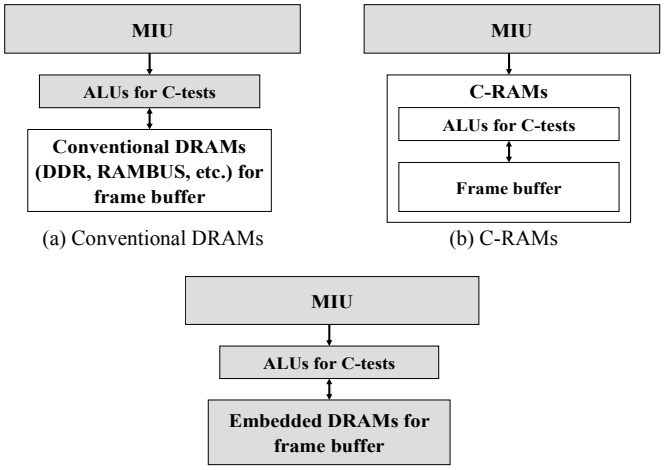
4. The Memory Systems of the Proposed Architecture

A unified memory system is widely adopted by recent rendering processors. As mentioned in [5], the biggest advantage of a single graphics memory system is the dynamic reallocation of memory bandwidth. The external bus width of a current rendering processor is either 128 bits or 256 bits. It is expected that a wider bus will be announced in the next generation rendering processors. The pixel cache and the texture cache are essentially included into a rendering processor to use a wide external bus effectively and to run the rasterization pipeline as high a rate as possible.

A conventional MIU has several queues to buffer the data transmissions between the processor and the external memory. For example, each memory controller in [5] has five request queues. The replaced cache blocks transmitted from the pixel cache are fed into the pixel output queue in MIU and then each of them is written into the frame buffer after C-tests. It is desirable for an effective memory system that the input rate of MIU should match well with the output rate of MIU.

Fig. 3 shows the three memory systems for the proposed architecture: *conventional DRAMs for the frame buffer (CDFB)*, *C-RAMs for the frame buffer (CRFB)*, and *embedded DRAMs for the frame buffer (EDFB)*. The shaded blocks reside within a rendering processor chip. The non-shaded blocks can be organized as separate chips. The three figures in Fig. 3 are arranged according to the output rate of

MIU; that is, conventional DRAMs in Fig. 3 (a) have the lowest output rate, C-RAMs in Fig. 3 (b) are the next, and embedded DRAMs in Fig. 3 (c) have the highest rate.



(a) Conventional DRAMs (b) C-RAMs

(c) Embedded DRAMs

Fig. 3. The three memory systems.

In CDFB, conventional DRAMs are used for the frame buffer and the ALUs for C-tests are included within the rendering processor. On the other hand in CRFB, C-RAMs are used for the frame buffer and the ALUs for C-tests are included within C-RAMs. Note that the relationship between the processor and the frame buffer in CDFB is read-modify-write, while that in CRFB is write-only. Thus, in accessing the frame buffer for rasterization, CRFB requires only a half amount of the memory bandwidth of CDFB. However, CDFB has an overwhelming advantage over CRFB in terms of the cost-effectiveness, because C-RAMs are too expensive to develop.

Because C-tests are performed per cache block, the processing style of C-RAMs is similar to that of current DRAMs. Thus, C-RAMs can be implemented by adding simple hardware logics into current DRAMs, while 3D-RAMs include an internal cache and other complex schemes to improve the performance of the internal cache.

In EDFB, because both the ALUs for C-tests and the frame buffer are included in the rendering processor, a very wide bus width, for example more than 1024 bits, between MIU and the frame buffer is available and the latency to access the frame buffer is also reduced. Note that Sony's PlayStation®2 and GScube are typical rendering processors with embedded DRAMs for the frame buffer.

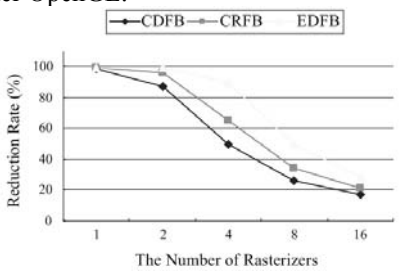
5. Experimental Simulation Results

In order to validate the proposed architecture, various simulation results are given in this section. A trace-driven simulator has been built for the proposed architecture. The traces are generated with three benchmarks, *Quake3* demo I, *Quake3* demo II, and *Lightscape* for 1600×1200 screen resolution by modifying the Mesa OpenGL compatible API.

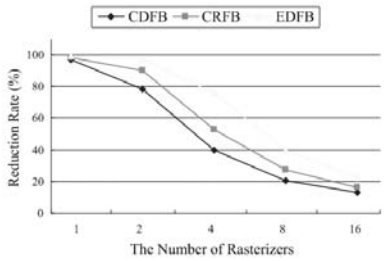
For each benchmark, 100 frames are used to generate each trace. The model data of each benchmark are evenly distributed into the given number of rasterizers by round-robin fashion. For example, if the number of the rasterizers is n, the first triangle, the (n+1)-th triangle, and so on are inputted into the first rasterizer.

With these traces, the pixel cache simulations are performed by modifying the well-known DineroIII cache simulator [7]. The memory latency reduction rates shown in Section 5.1 are also calculated for each trace.

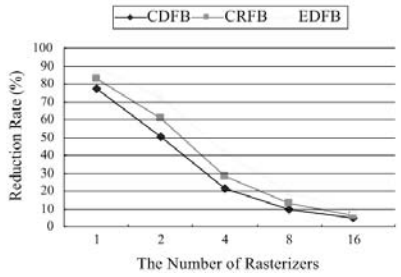
Quake3 in particular is one of typical current video games and is frequently used as a benchmark in other related works for their simulations. *Lightscape* is a product of SPECviewperf™ and is an industrial standard benchmark for measuring the performance of 3D rendering systems running under OpenGL.



(a) *Quake3* I



(b) *Quake3* II



(c) *Lightscape*

Fig. 4. The memory latency reduction rates**.**

5.1 The memory latency reduction rates

A replaced cache block from the pixel cache is stored into the tail entry of a pixel output queue indicated by the tail pointer. When the replaced block reaches the head entry indicated by the head pointer, it is written into the frame buffer. The overall pipeline does not stall as long as the pixel output queue is not full. Thus, with a buffer of infinite size, the proposed architecture is able to achieve a zero-latency memory system.

Fig. 4 shows the memory latency reduction rates for the three memory systems. Note that the reduction rate of 100% represents a zero-latency memory system. If the reduction rate is 0%, the full memory latency is required for a pixel cache miss. We assume that the numbers of cycles to complete C-tests for a pixel cache block for CDFB, CRFB,

and EDFB are 16, 12, and 8, respectively. The number of cycles can be determined according to the block size of a pixel cache, the number of ALUs, the DRAM performance, etc. We also assume that the number of entries in a pixel output queue is fixed to 128, because the simulation results on the reduction rates for various numbers of entries, which is not provided in this paper, show that the numbers of entries from 4 up to 1024 affect the reduction rates under 8%.

The simulation results show that an almost zero-latency memory system can be achieved with CRFB and EDFB with one rasterizer and two rasterizers. With four rasterizers, significant reduction rates are achieved for EDFB. Because the number of replaced blocks fed into MIU at the same time increases as the number of the rasterizers increases, the reduction rates decrease as the number of rasterizers increases. The reduction rates are not sufficient when the number of rasterizers is eight or sixteen.

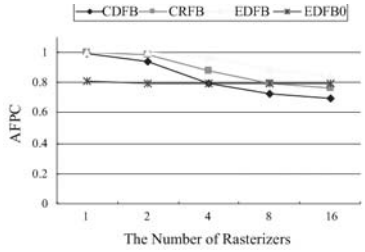
5.2 Performance evaluation

To evaluate the performance analytically, we calculate the average fragments per cycle (*AFPC*) with a rasterizer. In [6], the miss penalties due to both the pixel cache and the texture cache are assumed to degrade the overall performance. In this paper, we assume that only the memory latency due to the pixel cache can degrade the performance. Hence, *AFPC* can be calculated as follows.

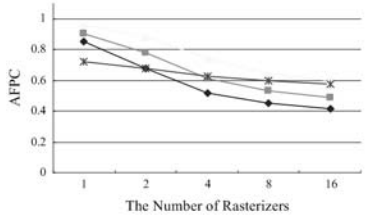
$$AFPC = 1/(1 + Miss\ Rate \times Latency \times (1 - reduction)),$$

where *Miss_Rate* is the miss rate of the pixel cache, *Latency* is the cycle times of the memory latency due to a pixel cache miss, and *Reduction* is the reduction rates shown in Fig. 4. The denominator of the above equation represents the average cycles per fragment with a rasterizer.

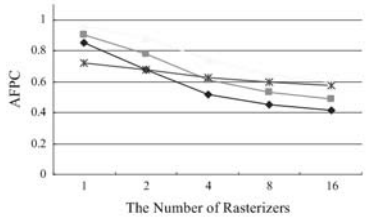
Fig. 5 shows *AFPC*s for the proposed architecture with different numbers of rasterizers and five different configurations. EDFB with 0% reduction rate is denoted by *EDFB0*. The AFPC of EDFB0 is provided to compare it with those of other four proposed configurations. For example, for four rasterizers in Fig. 5 (a), the AFPC of EDFB0 is almost the same as that of CDFB. The performance increment for *n* rasterizers can be calculated easily by multiplying *n* with *AFPC* of the architecture with *n* rasterizers.



(a) *Quake3* I



(b) *Quake3* II



(c) *Lightscape*

Fig. 5. *AFPC*s of the proposed architecture.

6. Conclusions

This paper proposes a new parallel rendering processor architecture solving the consistency problem of the pixel cache and significantly reducing the memory latency due to the pixel cache miss. As a future work, we would like to develop the prototype of the proposed architecture.

References

[1] M. S. Suzuoki et al., “A microprocessor with a 128-bit CPU, ten floating-point MAC's, four floating-point dividers, and an MPEG-2 decoder,” IEEE Journal of Solid-State Circuits, vol. 34, pp. 1608-1618, 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*, vol. 36, no. 11, pp. 1775-1783, Nov. 2001.

[3] S. Molnar, M. Cox, M. Ellsworth, and H. Fuchs, "A sorting classification of parallel rendering," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, pp. 23-32, July 1994.

[4] A. Wolfe and D. B. Noonburg, "A superscalar 3D graphics engine," In *Proceedings of MICRO 32*, pp. 50-61, 1999.

[5] J. McCormack, R. McNamara, C. Gianos, L. Seiler, N. P. Jouppi, K. Correl, T. Dutton, and J. Zurawski, “Neon: a (big) (fast) single-chip 3D workstation graphics accelerator,” Research Report 98/1, Western Research Laboratory, Compaq Corporation, Aug. 1998 (revised July 1999).

[6] W. C. Park, K. W. Lee, I. S. Kim, T. D. Han, and S. B. Yang, “An effective pixel rasterization pipeline architecture for 3D rendering processors,” *IEEE Transactions on Computers*, Vol. 52, No. 11, pp. 1501-1508, Nov. 2003.

[7] M. D. Hill, J. R. Larus, A. R. Lebeck, M. Talluri, and D. A. Wood, “Wisconsin architectural research tool set,” *ACM SIGARCH Computer Architecture News*, vol 21, pp. 8-10, Sep. 1993.