한국게임학회 논문지
제5권 제3호 2005년 9월

효과적인 메모리 구조를 갖는 병렬 렌더링 프로세서 구조

김경수[0], 윤덕기, 김일산*, 박우찬
세종대학교 인터넷 공학과, *연세대학교 컴퓨터 과학과
(kskim[0],dkyoon)@rayman.sejong.ac.kr
(sany*, pwchan)@korea.com

A architecture for parallel rendering processor with by effective memory organization

Kyung-Su-Kim[0], Duk-ki-Yoon, Il-San-Kim*, Woo-Chan-Park
Dept. of Internet Engineering, Sejong University
*Dept, of Computer Science, Yonsei University

요약

현재의 거의 대부분의 3차원 그래픽 프로세서는 한 개의 삼각형을 빠르게 처리하는 구조로 되어 있으며, 향후 여러 개의 삼각형을 병렬적으로 처리할 수 있는 프로세서가 등장할 것으로 예상된다. 고성능으로 삼각형을 처리하기 위해서는 각각의 레스터라이저마다 각각의 고유한 픽셀 캐시를 가져야 한다. 그런데, 병렬로 처리되는 경우 각각의 프로세서와 프레임 메모리 간에 일관성 문제가 발생할 수 있다. 본 논문에서는 각각의 그래픽 가속기에 픽셀 캐시를 사용가능 하게 하면서 성능을 증가시키고 일관성 문제를 효과적으로 해결하는 병렬 렌더링 프로세서를 제안한다. 또한 제안하는 구조에서는 픽셀 캐시 미스에 의한 지연시간을 크게 감소시켰다. 실험 결과는 본 구조가 16개 이상의 레스터라이저에서 선형적으로 속도 향상을 가져옴을 보여준다.

Abstract

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 resolving such consistency problem effectively. Moreover, the proposed architecture reduces the latency due to a pixel cache miss significantly. The experimental results show that proposed architecture achieves almost linear speedup at best case even in sixteen rasterizer

Keywords : Computer Graphics, Rendering Processor, Parallel Rendering, Graphics Hardware.

1. 서론

현재의 대부분의 3차원 그래픽 프로세서는 다수개의 픽셀 파이프라인을 사용하여 한 개의 삼각형을 고속으로 렌더링 처리하는 구조를 가지고 있다. 최근 반도체 기술의 발전으로 인하여 다수개의 가속기를 내장한 병렬 3차원 그래픽 프로세서가 등장이 가능하게 되었다. [1]에서는 PlayStation2에 상응하는 Graphics Processing Unit (GPU)

※ 본 연구의 일부는 2005년도 반도체 설계 교육센터 IDEC 의 EDA Tool 지원에 의해서 연구되었음.

16개와 256-Mb 내장 DRAM을 한 개의 칩에서 구현한 GScube를 발표하였다. 그런데, 16개의 GPU의 출력을 한 개의 pixel merging IC에서 합쳐서 비디오 디스플레이로 보내기 때문에 다수개의 프레임 메모리를 필요로 한다. 이로 인하여 GScube에서는 큰 내장 DRAM을 사용하였다.

한 개의 프레임 메모리를 가지고 3차원 그래픽 가속을 병렬로 수행하는 경우 각각의 렌더링 처리부들과 프레임 메모리 간에 일관성 문제가 발생한다. 이를 해결하기 위해 범용 마이크로프로세서에서의 슈퍼 스칼라 방식을 이용하는 구조가 제시되었다 [2]. 그런데, [2]에서는 슈퍼 스칼라를 지원해주기 위한 별도의 하드웨어를 필요로 하며, 크기가 큰 폴리곤이나 삼각형 스트립 같이 특정한 경우 별도의 특별한 처리를 해주어야 한다. 또한, 전체 성능에 큰 영향을 미치는 픽셀 캐시에 대한 고려가 전혀 없다. 픽셀 캐시는 그래픽 가속기의 메모리 지연시간을 줄여서 성능을 높이고 메모리 대역폭을 줄이는 역할을 수행하며, 최근의 그래픽 프로세서에 필수적으로 사용된다[3, 4, 5]. 이에 대한 방안으로 모든 가속기가 접근 가능한 전역적 픽셀 캐시를 사용하는 것을 고려해볼 수 있는데, 이는 픽셀 캐시에 많은 수의 포트를 가속기 수에 비례하여 필요로 하기 때문에 성능이 떨어지고 가격이 비싸다.

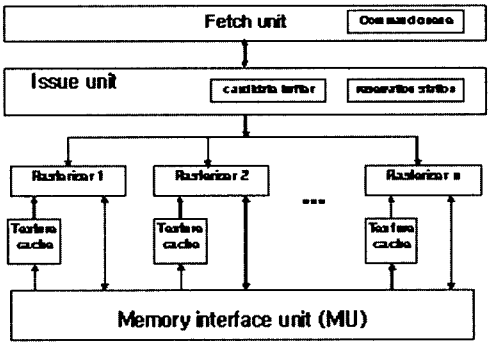
본 논문에서 제안하는 방식은 각각의 가속기 당 픽셀 캐시를 가지고 레스터라이저를 수행하는 새로운 병렬 렌더링 프로세서를 제안한다. 우리는 각각의 픽셀 캐시당 일관성 문제를 허락하나 모든 픽셀 캐시 블락이 프레임 버퍼로 쓰여질 때는 consistency-test(C-test)를 추가함으로써 일관성을 유지시켰다. 또한 제안하는 구조는 캐시 블락 미스가 발생돼서 memory interface unit(MIU)로 전송된 후에도 레스터라이제이션 파이프라인이 즉시 실행되게 함으로써 생기는 픽셀 캐시 미스에 의한 latency를 감소시켰다

본 논문에서는 3종류의 메모리 시스템을 보일 것이다. 처음으로 가장 느린 것으로 프로세서 안에서 C-test를 수행하며 프레임 버퍼는 일반적인 DRAM으로 구성되어있는 것이다. 2번째로 C-test를 프레임 버퍼 안에서 하는 것인데 3D-RAM[9,10]과 유사한 특별한 프레임 메모리가 요구된다. 이 논문에서 우리는 consistency-RAM이라 불리는 새로운 DRAM 구조를 제안한다. Embedded DRAM은 3번째 메모리 시스템에서 높은 메모리 대역폭을 얻기 위해 프레임 버퍼로 제공되었다.

제안하는 구조의 정당성을 위해 Trace-driven 시뮬레이터를 제작하였고 이것을 가지고 3가지 벤치마크로 다양한 시뮬레이션 결과를 제공하였다. 일단, 레스터라이저의 숫자를 증가시키면서 픽셀 캐시 시뮬레이션을 수행하였다. 또한 우리는 레스터라이저의 숫자에 따른 메모리 latency 감소 비율을 계산하였고 16개의 레스터라이저에서 90%의 zero-latency 메모리 시스템을 얻었다.

2. 관련 연구

[그림 1]은 [2]에서 제안된 SuperScalar Rendering Processor의 블록 도이다. 이는 다수개의 병렬 래스터라이저가 겹쳐지는 않은 삼각형들 만을 병렬로 처리하여 일관성이 유지 되도록 슈퍼 스칼라 기법을 사용하였다. 먼저, 기하학 처리 연산이 수행된 3차원 모델 데이터가 fetch 부를 통하여 그래픽 프로세서로 입력된다. Reservation station이 가용하면 삼각형이issue 부에 입력되어 의존 관계를 검사하고 scheduling을 한다. 각각의 삼각형이 수행이 끝나면 래스터라이저는 issue unit에 이를 알려주고 해당reservation station의 사용을 끝낸다.



[그림 1] SuperScalar Rendering Processor의 블록 다이어 그램

Issue 부에서 수행되는 의존 관계 검사는 래스터라이저에서 수행되는 삼각형과 입력되는 삼각형 간에 겹쳐지는 지의 여부를 검사하는 것이다. 그런데, 삼각형들을 가지고 겹쳐지는 여부를 정확하게 검사하는 것은 매우 복잡하다. 이를 쉽게 하기 위하여 삼각형을 둘러싼 사각형인 bounding

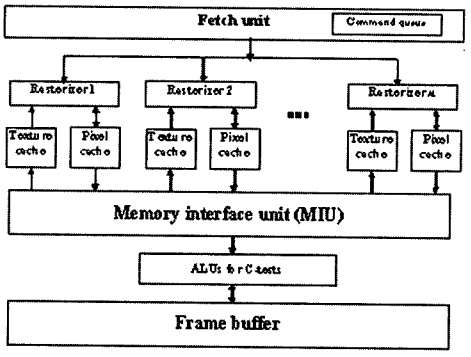
box를 대신하여 검사한다. 이로 인하여 다음과 같은 문제점이 발생할 수 있다. 주로 3차원 모델 데이터는 삼각형의 일련으로 이루어진 삼각형 스트립으로 구성되는데, bounding box를 사용하여 겹치는 여부를 검사할 경우 실제로는 겹쳐지지 않지만 겹쳐진다고 판명이 난다. 이를 완화시키기 위하여 삼각형 스트립 짝수 번째 삼각형과 홀수 번째 삼각형을 따로 처리 한다. 또한, 큰 크기의 삼각형을 처리할 경우 계속해서 겹쳐진다고 판명이 나서 병렬 프로세서가 순차적으로 수행 될 수 있다. 이를 완화하기 위하여 큰 크기의 삼각형인 경우 작은 크기의 슬 라이스로 나누어서 처리한다. 이러한 기법을 사용하면 삼각형 레벨에서의 병렬성이 95%정도 향상하는 것을 실험을 통하여 보여주었다.

3. 제안하는 구조

[그림 2]는 제안하는 병렬 3차원 그래픽 프로세서의 전체 블록 도이다. Superscalac Processor Rendering 방식의 [그림 2]와 비하여 issue 부가 제거 되었고 MIU와 프레임 버퍼 사이에 C-test를 위한 ALU가 삽입 되었다. 그리고 각각의 래스터라이저 당 지역적 픽셀캐시가 장착되었다.

각각의 레스터라이저는 지역적 텍스처 캐시와 픽셀 캐시를 가지고 레스터라이제이션을 수행한다. SRP와 다르게 의존성 검사는 수행하지 않는다. 그래서 제안하는 구조에서 삼각형 단위의 병렬성은 100%이다. 그러나 픽셀 캐시에서는 의심할 필요 없이 일관성 문제가 발생한다.

이 논문의 가장 주된 아이디어는 우리는 각각의 픽셀 캐시에서는 일관성 문제가 발생하는 것을 허용하나 프레임 버퍼에서는 일관성을 엄격하게 유지할 것이다. 픽셀 캐시 미스가 발생할 때마다 픽셀 캐시의 자료가 프레임 버퍼로 넘어간다. 또한 현재 레스터라이제이션 프레임이 끝날 때마다 픽셀 캐시가 flushing 된다. 제안하는 구조의 프레임 버퍼의 일관성 문제는 프레임 버퍼의 블락과 일치하는 캐시 블락이 전송될 때 추가적인 C-test를 수행함으로 유지한다. 제안하는 구조의 다른 아이디어로는 캐시 블록을 전송한 후에 즉시 레스터라이제이션 단계를 끊임없이 수행하는 것 보다 MIU로 miss를 발생시킨다. 그러므로 프레임 버퍼로부터 픽셀 캐시로 전송될 응답 블락의 시간을 포함한 캐시 미스에 의한 latency가 감소하게 된다. 더 나아가 레스터라이제이션 파이프라인과 C-test는 독립적으로 수행된다 이러한 목적의 새로운 픽셀 캐시 구조를 알리기 위해서 이 논문이 쓰여졌다.

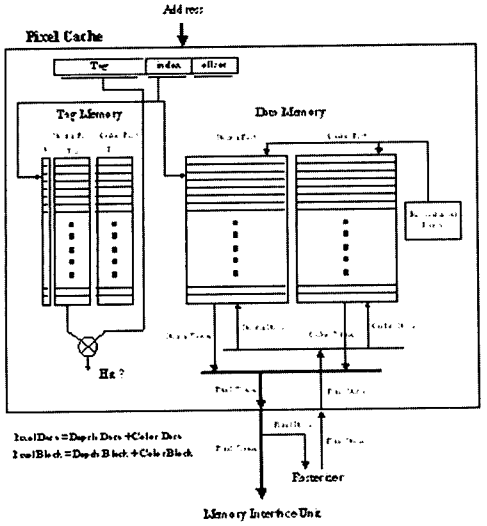


[그림 2] 제안하는 구조

3.1 제안하는 픽셀 캐시 구조

[그림 3]에서 제시한 제안하는 픽셀 캐시 구조는 valid bit (V) 부, 깊이 부분 태그 및 색깔 태그로 구성된 태그 메모리, 깊이 데이터 부분과 색깔 데이터 부분으로 구성된 데이터 메모리, 초기화를 위한 초기화 로직으로 구성된다. 일반적으로 픽셀 캐시는 깊이 값이 저장되는 깊이 캐시와 색깔 값이 저장되는 색깔 캐시로 구성된다. 깊이 캐시와 색깔 캐시는 별도로 동작할 수도 있으나, 본 논문에서는 제안하는 알고리즘의 특성상 동일한 스크린 주소를 갖는 깊이 값과 색깔 값이 함께 이동해야 되기 때문에 깊이 캐시와 색깔 캐시는 쌍으로 구성된다. [그림 1]에서 살펴 본 바와 같이 깊이 값 참조가 색깔 값 참조보다 먼저 수행되기 때문에, 래스터라이저에서의 캐시 참조 시 깊이 값 읽기에 대해서만 태그 비교를 수행하며, 색깔 값에 대한 태그 메모리는 색깔 데이터가 프레임 버퍼로 쓰여지기 위한 주소를 저장하기 위하여 사용된다.

제안하는 픽셀 캐시에서는 두 가지 경우에 대하여 데이터 이동이 발생할 수 있으며 이는 깊이 값과 색깔 값이 함께 이동 된다. 첫 번째는 [그림 1]에서와 같이 픽셀 캐시와 래스터라이져 사이에 발생하는 읽기와 쓰기 연산이다. 일반적으로 읽기 연산은 해당 캐시 블록을 모두 증폭기로 가지고 온 다음 증폭을 시킨 후 해당 픽셀 데이터 만을 가지고 오며, 쓰기 연산은 픽셀 데이터만큼을 캐시에 바로 쓰기 연산을 수행한다.



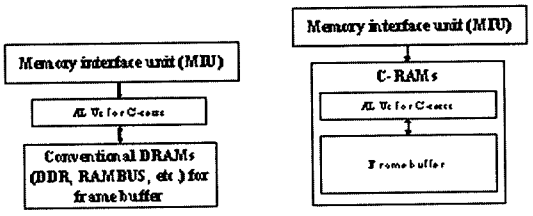
[그림 3] 제안하는 픽셀 캐시 구조

두 번째는 픽셀 캐시에 미스가 발생하여 해당 캐시 블록을 외부 프레임 메모리로 보내는 연산이다. 일반적인 픽셀 캐시 구조와는 달리 픽셀 캐시와 프레임 메모리간에 읽기 연산은 발생하지 않고 오직 쓰기 연산만 발생한다. 즉, 캐시 참조 시 미스가 나더라도 프레임 버퍼로부터 입력되는 정보가 없다는 것이다. 그 대신, 픽셀 캐시 미스가 발생한 데이터 메모리의 캐시 블록의 모든 비트를 1의 값으로 초기화 시키는 로직이 첨가되어 있다. 이는 또한 새로운 프레임이 시작되면 캐시 데이터 블록을 초기화 하는데도 사용되며 이는 프레임 버퍼 초기화와 같은 역할을 수행한다.

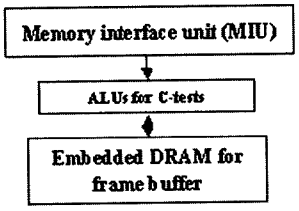
4. 제안하는 메모리 시스템 구조

[그림 4]는 제안하는 3가지 메모리 시스템을 보여주고 있다. 회색 블락은 렌더링 프로세서 칩 안에 위치하고 있고, 흰색 블락은 칩과 분리되어 있다. [그림 4]의 3가지 모습은 MIU의 출력 율에 따라 배열 하였다. [그림 4](a)가 가장 낮고 [그림 4](c)가 가장 높다.

[그림 4](a), 프레임 버퍼에 일반적인 DRAM이 사용되었고, 렌더링 프로세서 안에 C-test를 위한 ALU가 포함되어 있다. 다른 [그림 4](b) 는 C-RAM이 프레임 버퍼에 사용 되었고 C-test를 위한 ALU가 포함되어 있다. [그림 4](a)에서 프로세서와 프레임버퍼 사이의 관계는 read-modify-write 이고 [그림 4](b)는 쓰기 전용이다. 그래서 레스터라이제이션 단계에서 프레임 버퍼에 접근 시 [그림 4](b) 구조는 [그림 4](a)보다 절반의 메모리 대역폭을 요구한다. 그러나 [그림 4](a)는 비용 효과적에서 압도적으로 유리하다. 왜냐하면 [그림 4](b)의 C-RAM과 같은 새로운 형태의 메모리는 개발하기에 비싸기 때문이다.



(a) Conventional DRAM for frame buffer (b) C-RAM for frame buffer



(c) Embedded DRAM for frame buffer

[그림 4] 3가지 메모리 시스템

C-test는 캐시 블록당 수행하고, C-RAM의 처리 방법은 현재 DRAM의 기술과 유사하기 때문에 C-RAM은 DRAM에 추가적인 단순한 하드웨어 로직을 포함하여 구현할 수 있다. 그러나 3D-RAM은 내부 캐시를 포함하고 내부 캐시의 성능을 증가시키기 위해서 다른 복잡한 설계를 포함해야 한다.

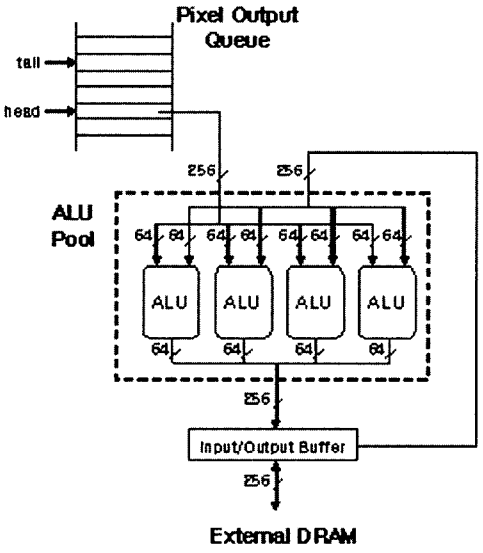
[그림 4](c) 에서 C-test를 위한 ALU와 프레임 버퍼는 렌더링 프로세서에 포함되어 있고, MIU와 프레임 버퍼 사이의 1024비트 이상의 넓은 폭의 버스는 프레임 버퍼 접근을 위한 latency를 줄여준다.

4.1 메모리 시스템의 구조

[그림 5]은 [그림 4](a) 메모리 시스템의 블록다이어그램이다. [그림 5]의 Pixel out queue는 픽셀 캐시에서 보내진 대체 캐시 블록 들을 저장하는 큐이다. ALU pool에서는 C-

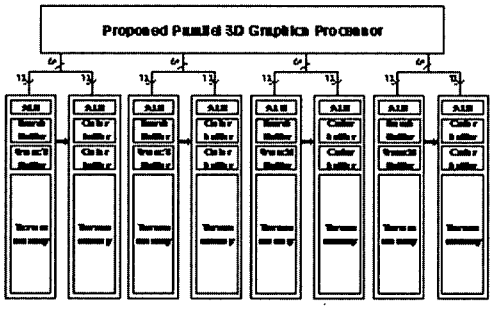
test(깊이 비교 및 알파 블렌딩)을 수행하는 ALU가 다수 개 존재한다.

[그림 5]에서 외부 DRAM과의 데이터 버스는 256 비트라고 하고 픽셀 캐시 블록의 크기는 n256이라고 가정하면 [그림 5]의 동작은 다음과 같다. Pixel output queue의 head가 가리키는 대체 캐시 블록에서 처음 256 비트를 ALU pool로 보내고 이와 동시에 같은 주소를 같은 목표 데이터 블록의 256 비트를 외부 메모리로부터 읽어 온다. 256 비트는 4개의 픽셀 정보인 4개의 깊이 값과 4개의 색깔 값으로 구성된다. ALU pool에서는 4개의 픽셀에 대하여 C-test 을 수행한다. 이러한 과정을 거친 4개의 픽셀 데이터는 output buffer를 통하여 외부 메모리에 쓰여진다. 위의 과정을 n 번 수행하면 한 개의 대체 캐시 블록에 대한 연산이 종료된다.



[그림 5] 메모리 인터페이스 부의 구성도

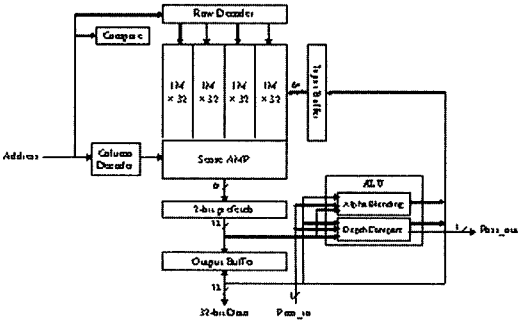
C-test를 수행하는 도중에 프레임 버퍼로부터 burst로 읽어오고 쓰여지게 된다. Burst 읽기 연산이 끊난후에 즉시 burst 쓰기 연산이 실행되는 2개의 연산이 실행하기 위해서 끊임없이 프레임 버퍼로 접근하는 바람직한 것이다. 우리는 ALU의 파이프라인 단계 숫자가 (k+1)이라고 가정을 했다. Burst 읽기 연산과 burst 쓰기 연산 사이에 setup latency가 요구된다. 이 latency를 1이라고 가정한다. 파이프라인단계에 따르면 깊이 비교 파이프라인과 알파 블렌딩 파이프라인은 직렬 혹은 병렬적으로 연결되어 있다. 그래서 프레임 버퍼로부터 256비트의 읽기 연산이 k번 수행됨과 동시에 ALU 파이프라인이 실행된다. 그 후 C-test를 수행한 후 처음 256비트는 (k+1) 사이클 이후에 출력 버퍼에 나온다. 마지막 256비트가 나오는데 까지는 (2k+1) 사이클이 걸린다.



[그림 6] 256 비트 버스 크기를 갖는 경우에 메모리 구성도

4.2 C-RAM의 메모리 구조

[그림 6]은 256 비트의 버스를 갖는 제안하는 병렬 렌더링 프로세서의 메모리 구성을 보여주고 있다. 3D-RAM에 프레임 자료가 있는 동안 C-RAM안에 프레임 자료와 텍스처 자료가 조화롭게 있다. 3D-RAM에서 색깔 C-RAM과 깊이 C-RAM에 색깔 자료와 깊이 자료가 분리돼서 저장되어 있다. 그래서 [그림 5]가 [그림 6]보다 ALU의 숫자가 절반으로 작지만 [그림 6] 구조의 모든 ALU 성능은 [그림 5]의 성능과 같다. 색깔 C-RAM에서 두 개의 색깔 버퍼는 더블 버퍼 링(Double buffering)을 지원해주기 위하여 두 개 존재한다. 그 외에 스텐실(stencil) 효과를 주기 위한 스텐실 버퍼가 깊이 C-RAM에 존재한다.



[그림 7] C-RAM의 상세한 블록 다이어그램의 예

4.3 상세한 C-RAM의 예

[그림 7]은 C-RAM 칩의 블록 다이어그램의 예이다. 최근 렌더링 프로세서에서 널리 쓰이는 128M DDR 그래픽 DRAM으로 변경돼서 디자인 되었다. 내부 버스의 폭은 64 비트 임으로 32 비트 데이터는 출력 버퍼로 한번에 전송될 수 있다. 텍스처 메모리와 프레임 버퍼는 칩 안에서 독립적인 메모리 공간을 차지하고 있다. Row 디코더로 들어오는 Row 주소를 check 한 후에 메모리의 어느 부분에 접근하는지를 결정할 수 있다. 만약 텍스처 메모리에 접근한다면 C-RAM의 행동은 일반적인 DDR DRAM으로 동작을 수행한다. 픽셀 캐시 블락이 MIU로부터 전송되고 C-RAM이 C-test를 수행하게 변경된다. 이 경우에, 파이프라인 구성과 ALU의 실행 방향은 3D-RAM과 유사하다.

ALU는 z-test와 알파 블렌딩 파이프 라인을 포함한다. 각각의 파이프라인은 여러 단계로 되어있다. 예를 들어 [6]에서 깊이 비교는 2단, 알파 블렌딩의 3단 파이프 라인이다. 처음에는 MIU와 프레임 버퍼의 깊이 값을 각각의 C-RAM 깊이 비교 파이프 라인에서 비교한다. 만약 깊이 비교 시 성공이면 깊이 값을 input buffer에 보내 이를 저장하게 하고 pass_out에 성공에 대한 신호를 넣어주며, 실패이면 pass_out에 이에 대한 신호를 넣는다. 메모리로 입력되는 데이터와 메모리에 저장된 데이터를 가지고 알파 블렌딩을 수행한 후 입력되는 pass_in 신호에 따라 input buffer로 보내어 알파 블렌딩이 수행된 값을 메모리에 저장하거나 버린다. 이러한 깊이 비교와 알파 블렌딩의 처리는 burst한 데이터에 대해서 처리하기 때문에 외부로부터 입력되는 데이터의 속도와 DRAM 내부에서 메모리 참조 속도를 같게 할 수 있음으로, ALU는 파이프라인 정지 없이 수행될 수 있다.

5. 시뮬레이션 결과

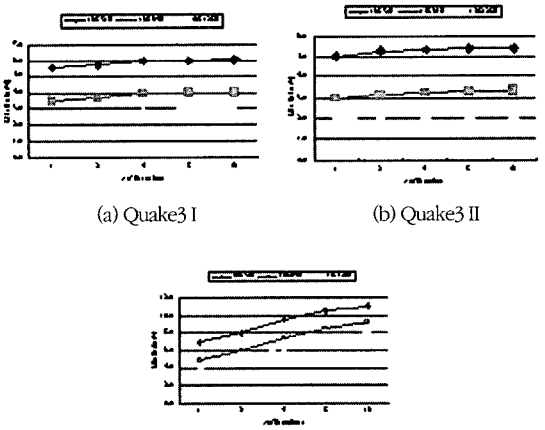
제안하는 구조의 타당성을 위해서, 이번 단락에서는 다양한 시뮬레이션 결과를 주어질 것이다. Trace-driven 시뮬레이터를 제안하는 구조에서는 사용되었다. 1600x1200 해상도에서 Mesa Opengl 호환하는 API로 Trace를 3가지 벤치마크(Quake3 demo I, Quake3 demo II, Lightscape) 를 사용하여 만들었다.

(a) Quake3 I (b) Quake3 II (c) Lightscape

[그림 8] 3가지 벤치마크

각각의 벤치마크에서 100 프레임이 각각의 trace를 생성하기 위해 사용되었다. 각각 벤치마크의 모델 데이터는 라운드 로빈 방식으로 레스터라이저에게 분배 되었다. 예를 들어 만약 레스터라이저 숫자가 n 이라면 처음 삼각형은 (n+1)번째 삼각형이고, 그러므로 처음 레스터라이저의 입력이 된다. 이러한 trace로 픽셀 캐시 시뮬레이션은 잘 알려진 Dinero III 캐시 시뮬레이터[20]를 변경하여 수행하였다. 메모리 대기 감소 비율은 각각의 trace를 계산해서 5.2 단락에서 보여진다.

[그림 8]은 벤치마크를 캡처한 것을 보여진다. 특히 Quake3는 현재 비디오 게임 중에 하나이고 시뮬레이션에서 벤치마크를 위해서 사용된다. Lightscape는 SPECviewperfTM 의 작품으로 OpenGl 환경 아래서의 3D 렌더링 시스템 벤치마크 표준으로 사용된다. Lightscape는 이 논문에서 다른 SPECviewperfTM 의 제품들과의 높은 장면 복잡도를 벤치마크하기 위해 사용되었고, 그것의 명백한 픽셀 캐시 미스 분포를 [10] 다른 벤치마크들과 비교하기 위해 사용되었다.



(a) Quake3 I (b) Quake3 II

(c) Lightscape

[그림 9] 픽셀 캐시 미스 비율

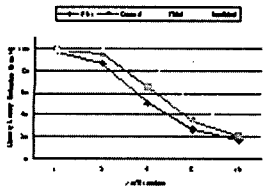
5.1 픽셀 캐시 시뮬레이션

[그림 9]에서 direct-mapped 방식의 16 Kbytes의 깊이 캐시와 1개에서 16개 까지 레스터라이저를 증가 시키면서 3가지 다른 불락 사이즈에서 픽셀 캐시 미스 비율을 보여준다. 왜냐하면 삼각형은 라운드 로빈 방식으로 많은 레스터라이저에 나누어 져 있기 때문이고, 많은 레스터라이저의 각 픽셀 캐시의 locality는 1개의 레스터라이저 locality 보다 감소하기 때문이다. 시뮬레이션 결과는 미스 비율이 [그림 9](a) 와 [그림 9](b)에서 레스터라이저 숫자가 증가하는 것 만큼 상당히 느리게 증가한다. 그러나 [그림 9](c)의 미스 비율은 이전 2가지 경우와 비교될 만큼 빠르게 증가한다.

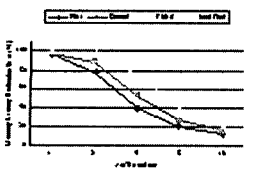
5.2 메모리 latency 감소 비율

[그림 10] 에서는 다음 단락에서의 보여질 레스터라이저 숫자에 따른 3개 메모리 시스템과 변경된 구조에서의 메모리 감소 비율을 보여준다. 100%에서 감소비율은 zero-latency 메모리 시스템을 나타낸다. 0% 감소비율의 경우에는 모든 메모리 latency가 픽셀 캐시 미스를 요구되어 진다. 우리는 3가지 메모리 시스템에서 완전한 C-test를 하기 위한 사이클 수가 각각 16, 12, 8 이라고 가정한다. 픽셀 캐시 불락 사이즈, ALU의 개수, DRAM의 성능, 기타를 결정하기 위해선 많은 수가 걸린다. 우리는 또한 전체 픽셀 출력 큐가 변수가 아니 128로 고정되었다고 가정한다. 왜냐하면 전체 수에 따른 감소 비율의 시뮬레이션 결과는 이 논문에서는 제공하지 않고 4이상 1024는 최고 8% 감소비율에 영향을 미친다고 보여진다.

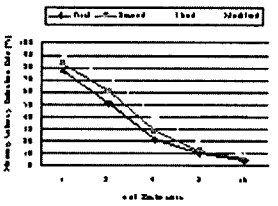
시뮬레이션 결과는 1개의 레스터라이저와 2개의 레스터라이저의 2번째와 3번째 메모리 시스템에서 거의 zero-latency 메모리 시스템을 만들 수 있다고 보여진다. 4개의 래스터라이저에서 중요한 감소 비율을 3번째 메모리 시스템에서 얻었다. 왜냐하면 동시에 MIU로 입력되어진 교체되어진 블록은 레스터라이저 수의 비율을 증가시킨다. 레스터라이저가 증가할수록 감소 비율은 감소한다 래스터라이저가 8개에서 16개 이면 감소 비율은 중요하지 않다.



(a) Quake3 I



(b) Quake3 II

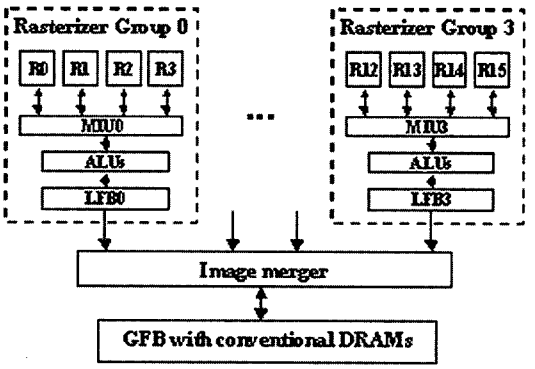


(c) Lightscape

[그림 10] 메모리 latency 감소 비율

5.3 8개 이상의 레스터라이저에서의 성능 향상

[그림 11]에서 16개의 레스터라이저로 변형된 구조를 보여준다. 효과적인 메모리 시스템은 레스터라이저를 4개 이상 유지해야 된다. 우리는 4개의 레스터라이저를 1개의 그룹으로 통합한다. 각각의 그룹은 [그림 4]의 4개의 레스터라이저를 가지는 제안된 구조와 같다.



[그림 11] 16개의 레스터라이저를 가지는 변경된 구조

각각의 레스터라저 그룹은 local frame buffer(LFB)라 불리는 전체 스크린 프레임 버퍼의 작은 이미지를 만들어낸다. 모든 LFB의 내용은 image Merger에 의해서 파이프라인 방식에 따라서 CRT scan 속도로 합성된다. 마지막 합성된 이미지는 global frame buffer(GFB)에 전송된다. LFB와 GFB의 더블 버퍼링을 위해 레스터라이제이션 과 이미지 합성 단계는 중첩되게 실행된다. 각각의 레스터라이저 그룹은 독립적으로 실행되고, 8개와 16개의 레스터라이저에서 memory latency reduction은 4개의 레스터라이저와 같다. 그러나 LFBs(2개의 8개 레스터라이저 와 4개의 16개 레스터

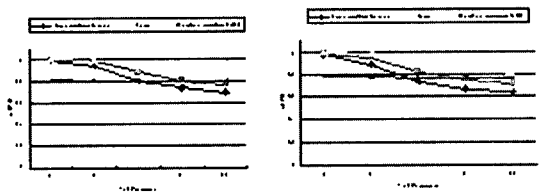
라이저)와 image merger는 렌더링 프로세서에 내장된다.

5.4 성능 평가

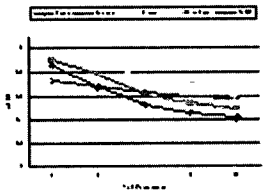
분석적인 성능 평가를 위해서 우리는 레스터라이저에서 사이클당 평균 프레그먼트 사이클(AFPC)을 계산하였다. [7]에서 픽셀 캐시와 텍스처 캐시의 miss penalty는 모든 성능의 감소를 가져온다. 이 논문에서 우리는 픽셀 캐시에 의한 메모리 latency가 성능 감소를 가져온다고 가정한다. 그 후 AFPC는 다음과 같이 계산한다.

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

Miss Rate 은 픽셀 캐시를 미스할 비율이고, Latency 는 픽셀 캐시 미스에 따른 메모리 대기 시간 사이클 타임이다. 그리고 reduction 은 [그림 10]에서 보여지는 감소 비율을 나타낸다. 이 방정식의 분모는 레스터라이저에서 사이클당 평균 프레그먼트를 나타낸다.



(a) Quake3 I (b) Quake3 II



(c) Lightscape

[그림 12] 제안하는 구조의 AFPCs

[그림 12]는 레스터라이저 수에 의한 AFPCs 수와 5개의 다른 메모리 구성을 나타낸다. 0%의 감소비율을 가지고 있는 3번째 메모리 시스템의 AFPC는 full8(전체 8개의 사이클의 픽셀 캐시 미스)를 나타낸다. AFPC의 full8은 다른 4개의 제안된 구성과 비교하도록 제공된다. 예를 들어 [그림 12](a)의 4개의 레스터라이저를 보면 AFPC의 full8은 처음 메모리 시스템과 같다. n 개의 레스터라이저에서의 성능 증가는 n을 AFPC와 함께 곱하기 위해 쉽게 계산된다. 그리하여 [그림 12](a)의 변경된 구조의AFPC는 16개의 레스터라이저에서 선형으로 증가를 하는 제안된 구조로 보여진다.

6. 결론

이 논문에서 제안하는 새로운 병렬 처리 프로세서 구조는 픽셀 캐시와 픽셀 캐시 미스로 인한 메모리 latency 감소의 consistency 문제를 해결할 수 있다. 우리는 제안하는 구조의 trace-driven 시뮬레이터를 만들었다. 실험 시뮬레이션은 제안하는 구조가 16개의 레스터라이저에서 15.5배의 속도 향상을 가져온다. 우리의 나머지 과제는 제안하는 구조의 프로토타입을 만드는 것이다.

참고문헌

[1] 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.

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

[3] M. Woo, J. Neider, T. Davis, and D. Shreiner, OpenGL programming guide, Addison-Wesley, Third edition, 1999.

[4] R. Bar-Yehuda and C. Gotsman, "Time/space tradoffs for polygon mesh rendering," ACM Transactions graphics, vol. 15, no. 2, pp. 141-152, 1996.

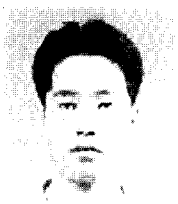
[5] Kai Hwang, Advanced computer architecture: parallelism, scalability, programmability, McGraw Hill, 1993.

[6] K. Inoue, H. Nakamura, and H. Kawai, "A 10b Frame buffer memory with Z-compare and A-bending units," IEEE Journal of Solid-State Circuits, vol. 30, no. 12, pp. 1563-1568, Dec. 1995.

[7] W. C. Park, et al, "An effective pixel rasterization pipeline architecture for 3D rendering processors," IEEE Transactions on Computers, Vol. 52, No. 11, pp. 1501-1508, Nov. 2003.

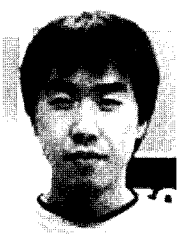
김 경 수

1999 ~ 2005 세종대학교 컴퓨터공학과 학사 졸업
2005 ~ 현재 세종대학교 컴퓨터공학과 석사과정
관심분야 : 3D Graphics HW/SW



김 일 산

1993 ~ 2000 연세대학교 컴퓨터과학과 학사 졸업
2000 ~ 2002 연세대학교 컴퓨터 과학과 석사 졸업
2002 ~ 현재 연세대학교 컴퓨터 과학과 박사과정
관심분야 : 3D graphics



윤 덕 기

1999 ~ 2005 세종대학교 컴퓨터 공학과 학사 졸업
2005 ~ 현재 세종대학교 컴퓨터 공학과 석사과정
관심분야 : 3D Graphics HW

박 우 찬

1989 ~ 1993 연세대학교 컴퓨터 과학과 학사 졸업
1993 ~ 1995 연세대학교 컴퓨터 과학과 석사 졸업
1995 ~ 2000 연세대학교 컴퓨터 과학과 박사 졸업
2000 ~ 2001 연세대학교 아식설계 공동연구소 연구원
2001 ~ 2003 연세대학교 연구 교수
2003 ~ 현재 세종대학교 조교수
관심분야 : 3D Rendering Processor Architecture
Parallel Rendering
High Performance Computer Architecture
Computer Arithmetic and ASIC Design

논문투고일 - 2005년 8월 17일
심사완료일 - 2005년 9월 15일