효과적인 메모리 구조를 갖는 병렬 렌더링 프로세서 설계

박 우 찬[†] · 윤 덕 기[††] · 김 경 수[†††]

요 약

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 각 래스터라이저마다 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 해결하는 병렬 렌더링 프로세서를 제안한다. 제안하는 구조에서는 픽셀 캐시 미스에 의한 지연(latency)을 감소시켰다. 이러한 2가지 성과를 위하여 현재의 새로운 픽셀 캐시 구조에 효과적인 메모리 구조를 포함시켰다. 실험 결과는 제안하는 구조가 16개 이상의 래스터라이저에서 거의 선형적으로 속도 향상을 가져옴을 보여준다.

키워드 : 컴퓨터 그래픽스, 렌더링 프로세서, 병렬 렌더링, 그래픽 하드웨어

Design of a Parallel Rendering Processor Architecture with Effective Memory System

Park Woo-Chan[†] · Yoon Duk-Ki[††] · Kim Kyoung-Su[†††]

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. For the above two goals, effective memory organizations including a new pixel cache architecture are presented. The experimental results show that the proposed architecture achieves almost linear speedup at best case even in sixteen rasterizers.

Key Words : Computer Graphics, Rendering Processor, Parallel Rendering, Graphic Hardware

1. 서 론

근래 들어 비교적 저렴한 가격의 고성능 그래픽 칩들이 발표되어 대부분의 PC에 장착이 되고 있다. 또한, Sony사의 PlayStation®2[1]나 MS사의 X-BOX 등의 게임 콘솔 부분에서도 전용 3차원 그래픽 프로세서가 장착되고 있다. 그런데, 현재의 대부분의 3차원 그래픽 프로세서는 다수개의 픽셀 파이프라인을 사용하여 한 개의 삼각형을 고속으로 렌더링 처리하는 구조를 가지고 있다. 이러한 성능은 아주 현실감 있는 영상을 생성할만한 규모의 3차원 데이터를 처리하기에는 여전히 한계가 있다.

현재 다수개의 삼각형을 동시에 처리하는 병렬 3차원 그래픽 하드웨어는 여러 개의 3차원 그래픽 프로세서를 보드에 장착하거나[2-4] 다수개의 PC를 클러스터링 하여 구현이 되는 경우가 대부분이다[5]. 그러나 이는 가격적인 면에서 매우 비싸기 때문에 과학적 시각화, 확장형 디스플레이, 대규모 모델 렌더링 등의 일부 특수 응용분야에 사용된다.

최근 반도체 기술의 발전으로 인하여 다수개의 가속기를 내장한 병렬 3차원 그래픽 프로세서의 등장이 가능하게 되었다. [6]에서는 PlayStation®2에 상응하는 Graphics Processing Unit (GPU) 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.

※ 이 논문은 IDEC의 지원에 의해 작성된 논문입니다.
† 정 회 원 : 세종대학교 컴퓨터공학과 조교수
†† 준 회 원 : 세종대학교 컴퓨터공학과 석사과정
††† 정 회 원 : 세종대학교 컴퓨터공학과 석사과정
논문접수 : 2005년 11월 30일, 심사완료 : 2006년 6월 9일

이는 [7]에서 규정된 병렬 렌더링 분류 중 sort-last와 유사하다. 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(consistency) 문제가 발생한다. 이를 해결하기 위해 범용 마이크로프로세서에서의 슈퍼 스칼라 방식을 이용하는 구조가 제시되었다[8]. 이는 [7]의 렌더링 분류 중 sort-middle에 해당된다. 그런데, [8]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[17-19]. 이에 대한 방안으로 모든 가속기가 접근 가능한 전 역적 픽셀 캐시를 사용하는 것을 고려해볼 수 있는데, 이러한 전역 픽셀 캐시는 포트의 수가 가속기의 수에 비례적으로 늘어나기 때문에 가속기의 수가 많아지면 구현이 매우 어렵기 때문에 비용적인 면에서 비현실적이다.

본 논문에서는 다음의 두 가지 특징을 갖는 새로운 병렬 렌더링 프로세서를 제안한다. 첫째, 제안하는 병렬 렌더링 프로세서는 각각의 가속기 당 픽셀 캐시를 가지고 래스터라이제이션을 수행하기 때문에 전역 픽셀 캐시를 사용할 필요가 없다. 이를 위하여, 지역 픽셀 캐시를 사용함으로써 발생하는 픽셀 캐시의 일관성 문제를 효과적으로 해결할 수 있는 방식을 제안하였다. 제안하는 방식은 각각의 픽셀 캐시 일관성 문제를 허락하나 픽셀 캐시 블록이 프레임 버퍼로 쓰여질 경우 consistency-test(C-test)를 수행하여 최종 프레임 버퍼에는 일관성을 유지시켰다. 둘째, 픽셀 캐시 미스 시 발생하는 지연대기시간을 대폭 줄였다. 제안하는 구조는 memory interface unit(MIU)로 픽셀 캐시 미스가 발생된 캐시 블록을 전송한 후에 즉시 래스터라이제이션 파이프라인을 실행시킬 수 있는 방식을 제시하였다. 이로 인하여, 가속기의 수에 비례하여 거의 선형적인 성능 향상을 달성하였다.

본 논문에서는 3종류의 메모리 시스템을 보일 것이다. 첫 번째 종류의 메모리 시스템은 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.

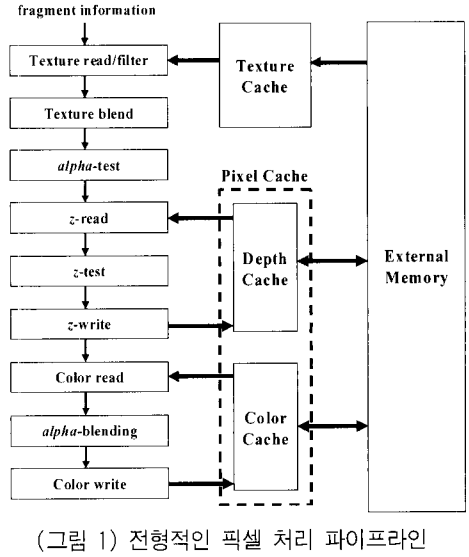
제안하는 구조의 타당성을 위해 3가지 벤치마크로 다양한 시뮬레이션 결과를 제공하였다. 또한, 제안하는 구조를 위해 Trace-driven 시뮬레이터를 만들었다. 처음으로 래스터라이저의 숫자를 증가시키면서 픽셀 캐시 시뮬레이션을 수행하였다. 또한 우리는 래스터라이저의 숫자에 따른 메모리 latency 감소 비율을 계산하였고. 16개의 래스터라이져에서 90%의 지연대기시간 감소를 얻었다. 0%는 모든 지연대기시간이 소요된다는 의미로써 기존의 방식이라고 볼 수 있으며, 100%는 지연대기 시간이 전혀 없다는 의미로써 이때 가속기가 수가 증가함에 따라 완전한 선형으로 성능이 증가한다.

본 논문의 구성은 다음과 같다. 2장에서는 우리는 일반적인 래스터라이저 파이프라인 흐름과, SuperScalar Rendering Processor (SRP), 그리고 3D-RAM에 대해서 살펴볼 것이다. 3장에서 우리의 제안하는 구조를 살펴볼 것이다. 4장에서는 제안하는 구조에서의 3가지 메모리 시스템에 대해서 논의할 것이고 5장에서는 시뮬레이션 결과와 성능 평가가 주어질 것이다. 6장에서는 결론으로써 향후 계획을 논하였다.

2. 배경 및 관련 연구

2.1 전통적인 렌더링 파이프라인

렌더링 처리 과정은 크게 기하학 처리와 래스터라이즈 과정으로 나뉜다. 일반적인 래스터라이즈 과정은 크게 삼각형 셋업(triangle setup)[11], 변 처리(edge processing), 픽셀 래스터라이제이션(pixel rasterization)으로 나뉜다. 전형적인 픽셀 처리 파이프라인은 (그림 1)과 같다[15-17]. 처음 두 개의 파이프라인 단계에서는 해당 텍스처 좌표에 대하여 4개 혹은 8개의 텍셀(texel)을 텍스처 캐시[12, 13]로부터 읽기 연산을 하고 필터링을 수행하여 한 개의 텍셀을 생성하고, 이를 입력되는 프래그먼트 정보의 일부인 색깔 값과 혼합한다. 다음으로 프래그먼트 정보의 알파(alpha) 값과 필터링 된 텍셀의 알파 값을 검사한다. 여기서 알파 값은 투명도를 나타낸다. 평가의 결과가 성공이면 다음 파이프라인으로 계속 진행이 되고 실패이면 다음 파이프라인으로 진행되지 않고 현재의 프래그먼트 정보는 버려진다. 다음의 두 개의 파이프라인 단계에서는 픽셀캐시로부터 깊이 값에 대한



(그림 1) 전형적인 픽셀 처리 파이프라인

읽기 연산을 수행하고 현재의 프래그먼트에 대한 깊이 값과 비교한다. 만약 검사의 결과가 실패이면, 즉 현재의 프래그먼트가 이전에 처리된 영상으로 인하여 안보이면, 현재의 프래그먼트는 파이프라인에서 버려진다. 만약 성공이면 현재의 프래그먼트의 깊이 값이 픽셀 캐시에 쓰인다. 마지막 세 개의 단계에서는 픽셀캐시로부터 색깔 값에 대한 읽기 연산이 수행되고, 이 값과 파이프라인에서 현재까지 처리된 색깔 값에 대하여 알파 블렌딩이 수행되고, 최종적인 색깔 값이 픽셀 캐시에 쓰인다.

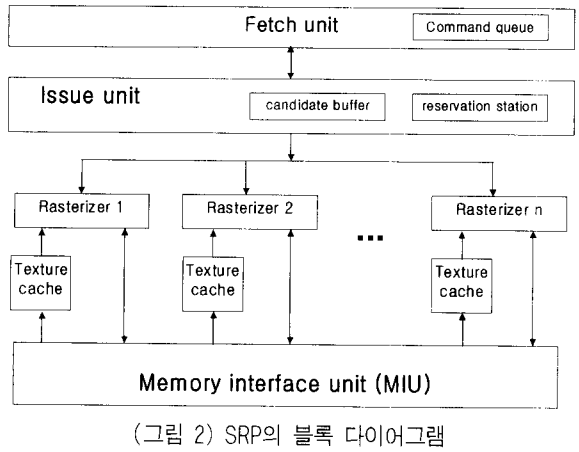
(그림 1)의 파이프라인은 현재 렌더링 프로세서의 능력인 Fragment Shader를 이용한 효과, 환경 매핑, 스텐실 효과 그리고 많은 다른 그래픽 효과들을 처리하기에는 부족하다. 우리는 (그림 1)에 여러 단계를 추가하여 이러한 효과를 처리하도록 할 것이다.

2.2 SRP의 overview

(그림 2)는 [8]에서 제안된 SuperScalar Rendering Processor (SRP)의 블록 도이다. 이는 다수개의 병렬 래스터라이저가 겹쳐지는 않은 삼각형들만을 병렬로 처리하여 일관성이 유지 되도록 슈퍼 스칼라 기법을 사용하였다. (그림 2)에서 issue 부내의 candidate buffer와 reservation station은 슈퍼 스칼라 기법을 위한 별도의 하드웨어이다.

먼저, 기하학 처리 연산이 수행된 3차원 모델 데이터가 fetch 부를 통하여 그래픽 프로세서로 입력된다. Issue 부는 한 사이클에 일관성 문제가 없는 한 개의 삼각형을 한 개의 래스터라이저로 issue할 수 있다. Issue 부의 reservation station은 해당 래스터라이저에 issue되어 현재 수행중인 삼각형에 대한 정보를 가지고 있으며, candidate buffer에는 issue되지 않은 삼각형의 정보 및 issue시 제약 점에 대한 정보를 가지고 있다. Reservation station이 가용하면 삼각형이 issue 부에 입력되어 의존 관계를 검사하고 scheduling을 한다. 각각의 삼각형이 수행이 끝나면 래스터라이저는 issue unit에 이를 알려주고 해당 reservation station의 사용을 끝낸다.

Issue 부에서 수행되는 의존 관계 검사는 래스터라이저에서 수행되는 삼각형과 입력되는 삼각형 간에 겹쳐지는 지의

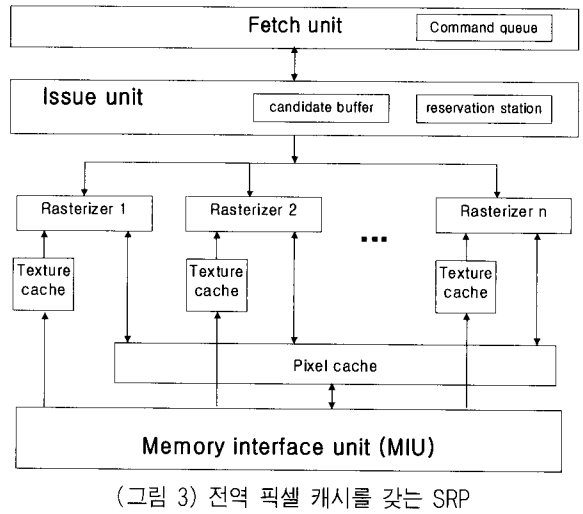


(그림 2) SRP의 블록 다이어그램

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 스트립[20, 21]으로 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 슬라이스로 나누어서 처리한다. 이러한 기법을 사용하면 삼각형 레벨에서의 병렬성이 95% 정도가 되는 것을 실험을 통하여 보여주었다.

2.3 전역 픽셀 캐시를 갖는 SRP

(그림 2)의 구조에서 픽셀 캐시를 장착할 수 있는 경우가 두 가지 있다. 첫 번째는 (그림 3)과 같이 한 개의 중앙 집중적 전역 픽셀 캐시를 장착하는 것이다. 이러한 전역 픽셀 캐시는 포트의 수가 가속기의 수에 비례적으로 늘어나기 때문에 가속기의 수가 많아지면 구현이 매우 어렵기 때문에 비용적인 면에서 비현실적이다. 두 번째는 각각의 래스터라이저 당 픽셀 캐시를 지역적으로 두는 방안이다. 그런데, 두 번째 경우는 각각의 픽셀 캐시와 프레임 메모리 간에 일관성 문제가 발생하기 때문에 병렬 컴퓨터에서 사용되는 snoopy 버스 프로토콜과 같은 캐시 일관성 프로토콜[19]을 사용해야 된다. 그러나 이는 매우 복잡하고 병렬 컴퓨터에 적합하도록 되어있기 때문에 본 구조에서는 적합하지 못하다.



(그림 3) 전역 픽셀 캐시를 갖는 SRP

2.4 3D RAM vs 현재 DRAM 기술

SUN과 Mitsubishi가 개발한 3D RAM은 픽셀 당 깊이 비교(z-test) 및 알파 블렌딩을 프레임 메모리에서 별도의 하드웨어로 수행함으로써, 읽기-수정-변경(read-modify-write)

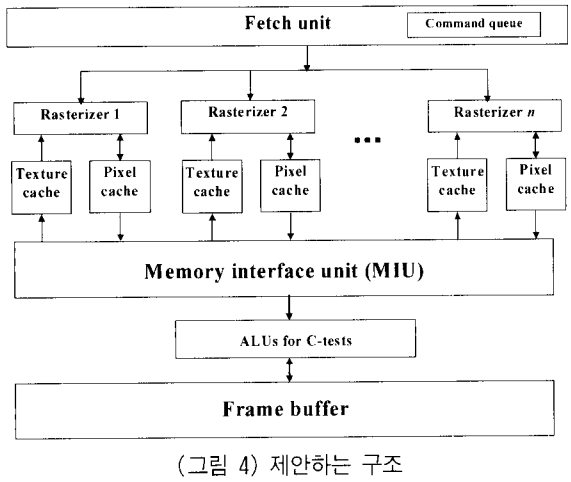
연산을 쓰기 전용(write-only) 연산으로 바꾸었다. 3D RAM에서는 깊이 비교 및 알파 블렌딩을 수행하는 픽셀 ALU와 DRAM 참조간의 처리 속도를 맞추어 주기 위하여 2K 비트의 내부 캐시를 장착하였다. 이러한 특징들로 인하여 전형적인 DRAM 기술에 비하여 좋은 성능을 얻는다고 주장하고 있다.

그러나 [14]에 언급했듯이 현재 DRAM기술과 일치하지 않는 것이 있다. 3D RAM이 발표될 당시에 비해 현재의 메모리 기술은 집적도나 속도 면에서 월등히 발전하였다. [9]에서 제시한 프레임 메모리 구성은 12개의 3D RAM을 사용하여 four-way interleaving 시켰다. 12개중 8개는 색깔 값에 대한 더블 버퍼에 사용이 되고 4개는 깊이 값에 사용된다. 그리고 four-way interleaving 되었기 때문에 한 번에 색깔 값에 해당하는 4개의 3D RAM과 깊이 값에 해당하는 4개의 3D RAM을 합쳐서 총 8개의 3D RAM에 데이터를 보낼 수가 있다. 즉, 256 비트의 데이터 버스를 사용한다는 것이다. 그런데, 최근의 메모리 집적도는 12개의 3D RAM을 한 두 개의 칩에 모두 집적이 가능할 정도에 이르렀다. 따라서 현재의 메모리 기술로 3D RAM의 기능을 하는 메모리를 만드는 경우 프레임 메모리와 텍스처 메모리를 같은 칩 내에 저장할 수 있는 구조가 되어야 한다. 그러나 3D RAM의 동작은 텍스처 메모리의 동작과 매우 상이하기 때문에 효과적인 구조를 구성하기가 매우 어렵다.

3. 제안하는 구조

(그림 4)는 제안하는 병렬 3차원 그래픽 프로세서의 전체 블록 도이다. SRP 방식의 (그림 2)에 비하여 issue 부가 제거 되었고 MIU와 프레임 버퍼 사이에 C-test를 위한 ALU가 삽입 되었다. 그리고 각각의 래스터라이저마다 지역적 픽셀캐시가 장착되었다.

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

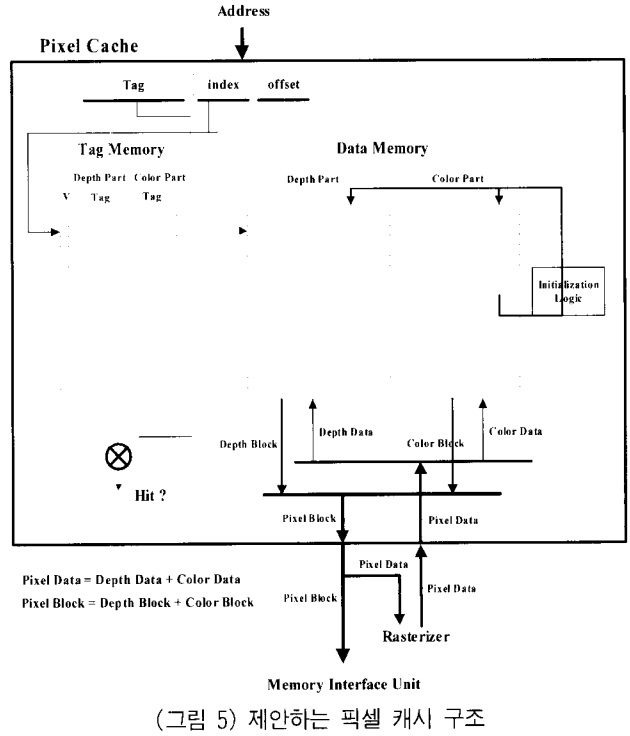


(그림 4) 제안하는 구조

이 논문의 가장 주된 아이디어는 각각의 픽셀 캐시에서는 일관성 문제가 발생하는 것을 허용하나 프레임 버퍼에서는 일관성을 유지하는 것이다. 제안하는 구조의 프레임 버퍼 일관성 문제는 픽셀캐시에 있는 캐시 블록이 프레임 버퍼로 전송될 때 추가적인 C-test를 수행함으로 유지한다. 제안하는 구조의 다른 아이디어로는 캐시 블록을 전송한 후에 즉시 래스터라이제이션 단계를 수행하는 것 이다. 그러므로 프레임 버퍼로부터 픽셀 캐시로 전송될 응답 블록의 시간을 포함한 캐시 미스에 의한 latency가 감소하게 된다. 더 나아가 래스터라이제이션 파이프라인과 C-test는 독립적으로 수행된다.

3.1 제안하는 픽셀 캐시 구조

(그림 5)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 쓰일 주소를 저장하기 위하여 사용된다.



(그림 5) 제안하는 픽셀 캐시 구조

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 래스터라이저 사이에 발생하는 읽기와 쓰기 연산이다. 두 번째는 픽셀 캐시에 미스가 발생하여 해당 캐시 블록을 외부 프레임 메모리로 보내는 연산이다. 일반적인 픽셀 캐시 구조와는 달리 픽셀 캐시와 프레임 메모리 사이에 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.

위의 경우의 전체 지연 시간은 태그 비교 연산, 캐시 블록을 메모리 인터페이스 부로 보내는 연산, 캐시 블록의 모든 비트를 1의 값으로 초기화 하는 연산으로 구성된다. 여기서 두 번째 것과 세 번째 것은 동시에 수행될 수 있다. 따라서 전체 지연 시간은 태그 비교 연산과 캐시 블록 쓰기 연산으로 볼 수 있다. 그런데, 이것은 일반적인 캐시에서의 쓰기 히트와 유사하며 일반적으로 쓰기 히트는 한 사이클에 처리된다고 본다. 따라서 위의 두 번째 경우는 한 사이클에 수행이 된다고 볼 수 있다.

3.2 제안하는 구조의 처리 과정

새로운 프레임이 시작되면 픽셀 캐시의 valid bit는 invalid를 나타내는 0으로 초기화 되고 깊이 값 및 색깔 값을 저장하고 있는 데이터 메모리가 1로 초기화 되는데, 이때 깊이 값은 시점과 가장 먼 깊이 값을 나타내며 색깔 값은 검은 배경색을 의미한다. 이로 인하여 3차원 모델 데이터가 스크린을 전부 가린다고 가정할 때, 별도의 프레임 메모리 초기화를 수행하지 않아도 된다. 그 후 래스터라이저로 부터 깊이 비교를 위한 참조가 발생하면 valid bit를 1로 바꾸고 태그 메모리에 해당 참조 주소 값을 저장한다. 이는 일반적인 캐시에서 캐시를 처음 참조할 때 발생하는 cold-start 미스에 대한 벌칙(penalty)이 없다는 것을 의미한다.

래스터라이저로 부터 픽셀 캐시 참조 시 히트 되면 파이프라인을 계속하여 진행한다. 픽셀 캐시 참조가 미스일 경우 캐시 교환(cache replacement) 정책에 의하여 프레임 버퍼로 보내어질 대체 블록이 정해지고 이를 메모리 인터페이스 부로 보낸다. 그 후, 캐시 블록에서 사용될 부분이 초기화 되고 메모리 인터페이스 부로 보낼 때 까지 교체된 캐시 블록이 래스터라이제이션 결과를 가지고 있다. 따라서 픽셀 캐시는 캐시 본연의 역할과 버퍼의 역할을 동시에 수행한다고 볼 수 있다.

교체된 캐시 블록은 C-test로 보내지고 C-test후 최종 캐시 블록 결과가 프레임 버퍼에 쓰인다. 3차원 모델 데이터에 대한 레시터라이제이션 결과가 모두 끝나면 픽셀 캐시에 내용을 프레임버퍼에 전송해야 하기 때문에, 각각의 픽셀 캐시 블록이 교체된 캐시블록과 똑같은 수행을 한다. 메모리 시스템 구조와 관련된 자세한 내용은 다음 장에서 나타내겠다.

3.3 픽셀 캐시 연결에 관련된 주요 문제와 해결방안

3.1에서 언급하였듯이, 깊이 캐시와 색깔 캐시가 쌍으로 되어있고 깊이 캐시에서 깊이 값의 히트 혹은 미스를 결정하기 위한 태그 비교를 수행한다. 그 후 깊이 읽기로부터 색깔 쓰기(color-write)까지 많은 파이프라인 단계를 수행한다. 그래서 깊이 읽기와 색깔 쓰기 파이프라인 단계 사이의 일관성 문제가 발생할 수 있다. 이 일관성 문제는 [14]에서의 유지문제의 해결책으로 해결되었다.

[14]에는 픽셀 캐시를 가지고 있지 않다. 한 개씩 프래그먼트를 처리함으로 생기는 과부하를 줄이기 위해서 [14]에서는 8개의 프래그먼트를 묶음 처리한다. 이러한 묶음 처리는 읽기 / 쓰기 일관성 문제를 발생한다. 만약 같은 픽셀 주소에 두 개의 프래그먼트가 있으면 첫 번째 프래그먼트가 수행이 완료된 후에 두 번째 프래그먼트의 깊이 읽기가 수행된다. 그렇지 않으면 두 번째 픽셀의 깊이 읽기는 우회하게 된다. 연합된 8개의 overlap detector는 일관성 문제의 조건을 탐지한다.

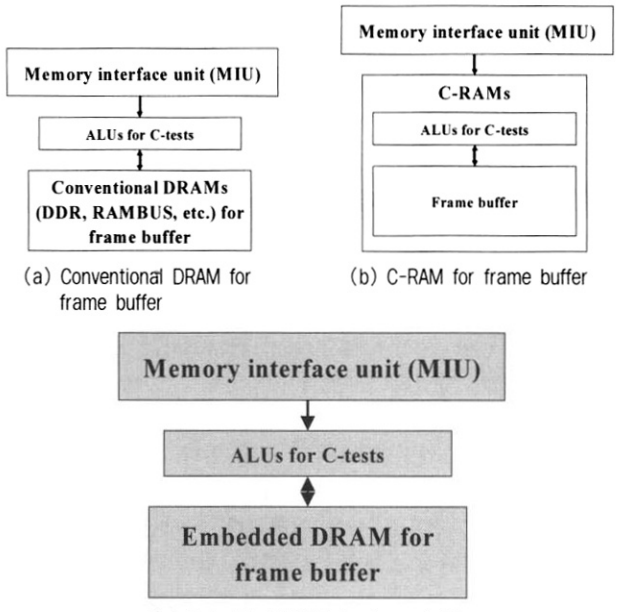
제안하는 구조는 유지문제를 탐지하기 위해서 overlap detector와 유사하게 깊이 비교와 색깔 쓰기 픽셀 파이프라인 단계 사이에서 교체된 캐시 블록 주소와 프래그먼트 주소를 비교한다. 유지 문제 발생 비율은 5.2절에서 볼 것이며 그 비율은 매우 낮기 때문에 우리는 유지 문제로 인한 성능 감소를 무시할 것이다.

4. 제안하는 메모리 시스템 구조

단일 메모리 시스템은 현재의 렌더링 프로세서에 널리 채택되고 있다. [14]에서 언급했듯이, 단일 그래픽 메모리 시스템의 가장 큰 장점은 메모리 대역폭을 동적으로 재 할당 할 수 있는 것이다. 현재 렌더링 프로세서의 외부 버스 폭은 128비트이거나 256비트이고 다음 세대의 렌더링 프로세서는 더 넓은 버스를 사용할 것으로 기대된다. 픽셀 캐시와 텍스처 캐시는 본질적으로 렌더링 프로세서에 넓은 외부 버스를 효과적으로 사용하기 위해 포함되고 래스터라이즈 파이프라인을 높은 비율로 사용 가능하게 한다.

전통적인 MIU에서 몇몇 큐들은 프로세서와 외부 메모리 사이에 데이터를 전송하기 위해 포함된다. 예를 들어 [14]에서 각각의 메모리 컨트롤러는 5개의 요청 큐를 가지고 있다. 픽셀 캐시에서 보내진 대체 캐시 블록은 MIU내의 픽셀 출력 큐에 입력으로 들어가고, C-test(깊이 비교, 알파 비교)를 거친 후 프레임 버퍼에 쓰인다. 효과적인 메모리 시스템에서 MIU의 입력 비율은 MIU의 출력 비율과 잘 조화된다.

(그림 6)은 제안하는 3가지 메모리 시스템을 보여주고 있다. 회색 블록은 렌더링 프로세서 칩 안에 위치하고 있고, 흰색 블록은 칩과 분리되어 있다. (그림 6)의 3가지 경우는



(그림 6) 3가지 메모리 시스템

메모리 시스템의 성능에 따라 배열 하였다. (그림 6) (a)가 가장 낮고 (그림 6) (c)가 가장 높다.

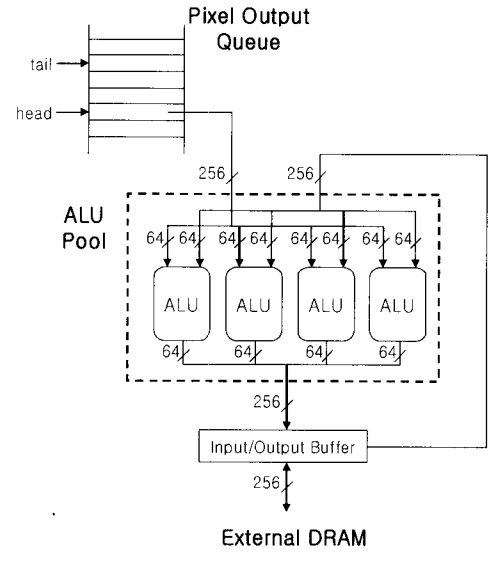
(그림 6) (a), 프레임 버퍼에 일반적인 DRAM이 사용되었고, 렌더링 프로세서 안에 C-test를 위한 ALU가 포함되어 있다. 다른 (그림 6) (b)는 C-RAM이 프레임 버퍼에 사용되었고 C-test를 위한 ALU가 포함되어 있다. (그림 6) (a)에서 프로세서와 프레임버퍼 사이의 관계는 read-modify-write 이고 (그림 6) (b)은 쓰기 전용이다. 그래서 래스터라이제이션 단계에서 프레임 버퍼에 접근 시 (그림 6) (b) 구조는 (그림 6) (a)보다 절반의 메모리 대역폭을 요구한다. 그러나 (그림 6) (a)은 비용 면에서 압도적으로 유리하다. 왜냐하면 (그림 6) (b)의 C-RAM과 같은 새로운 형태의 메모리는 개발비용이 많이 들기 때문이다.

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 6) (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. Sony사의 PlayStation®2 [1]와 GScube 는 렌더링 프로세서의 프레임 버퍼를 Embedded DRAM을 사용하였다.

4.1 메모리 시스템의 내부 구조

(그림 7)은 (그림 6) (a) 메모리 시스템의 블록다이어그램이다 (그림 7)의 Pixel out queue는 픽셀 캐시에서 보내진 대체 캐시 블록 들을 저장하는 큐이다. ALU pool에서는



(그림 7) 메모리 인터페이스 부의 구성도

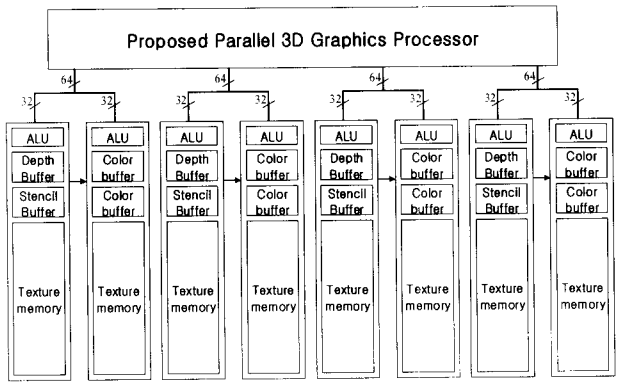
C-test(깊이 비교 및 알파 블렌딩)를 수행하는 ALU가 다수개 존재한다.

(그림 7)에서 외부 DRAM과의 데이터 버스는 256 비트라고 하고 픽셀 캐시 블록의 크기는 n×256이라고 가정하면 (그림 7)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 출력 버퍼(output buffer)를 통하여 외부 메모리에 쓰인다. 위의 과정을 n번 수행하면 한 개의 대체 캐시 블록에 대한 연산이 종료된다.

C-test에서 발생되는 n번의 읽기와 쓰기는 burst하게 수행될 수 있다. 만약 ALU의 파이프라인이 (n+1) 단계로 구성되어 있다고 가정한다. 또한, Burst 읽기 연산과 burst 쓰기 연산 사이에 setup latency가 요구되는데 이 latency를 1이라고 가정한다. 먼저 프레임 버퍼로부터 256비트의 읽기 연산이 n번 수행됨과 동시에 ALU 파이프라인이 실행된다. 그 후 C-test를 수행한 후 처음 256비트는 (n+1) 사이클 이후에 출력 버퍼에 나온다. 따라서 마지막 256비트가 나오는데 까지는 (2n+1) 사이클이 걸린다.

4.2 C-RAM의 메모리 구조

(그림 8)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 같이 있다. 3D-RAM에서 색깔 C-RAM과 깊이 C-RAM에 색깔 자료와 깊이 자료가 분리돼서 저장되어 있다. 그래서 (그림 7)이 (그림 8)보다 ALU의 숫자가 절반으로 작지만 (그림 8) 구조의 모든 ALU 성능은 (그림 7)의 성



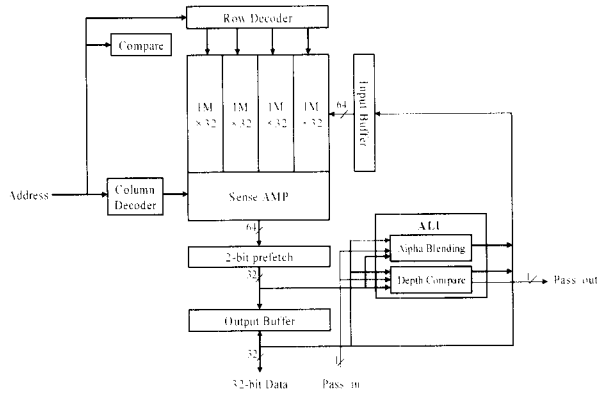
(그림 8) 256 비트 버스 크기를 갖는 경우에 메모리 구성도

능과 같다. 색깔 C-RAM에서 색깔 버퍼는 더블 버퍼 링(Double buffering)을 지원해주기 위하여 두 개가 존재한다. 그 외에 스텐실(stencil) 효과를 주기 위한 스텐실 버퍼가 깊이 C-RAM에 존재한다.

3D-RAM과 동일한 방식으로 4개의 32비트 깊이 자료와 4개의 32비트 색깔 자료는 렌더링 프로세서가 4개의 깊이 C-RAM과 색깔 C-RAM으로 전송됨과 동시에 들어온다. C-RAM으로 전송된 다음 256비트 자료는 다음 사이클에 수행되고 이러한 전송은 보든 교체될 캐시 블락이 완전히 전송될 때까지 반복된다. 그러나 이런 종류의 실행은 3D-RAM의 상황에는 보증하지 못한다.

C-test동안 깊이 비교 연산은 깊이 C-RAM의 ALU에서 수행된다. 그 후 각각 깊이 비교 연산 결과가 별도의 패스를 경유해서 ALU의 연결된 색깔 C-RAM으로 연결된다. 그 후 색깔 블렌딩 연산은 ALU의 색깔 C RAM에서 수행된다. 상세한 C-RAM 구조를 다음 단락에서 예로 들겠다.

4.3 상세한 C-RAM의 예



(그림 9) C-RAM의 상세한 블록 다이어그램의 예

(그림 9)는 C-RAM 칩의 블록 다이어그램의 예이다. 현재 렌더링 프로세서에서 널리 쓰이는 128M DDR 그래픽 DRAM으로 변경돼서 디자인 되었다. 내부 버스의 폭은 64 비트 이므로 32 비트 데이터는 출력 버퍼로 한 번에 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 블록이 MIU로부터 전송되고 C-RAM이 C-test를 수행하게 변경된다. 이 경우에, 파이프라인 구성과 ALU의 실행 방향은 3D-RAM과 유사하다.

ALU는 깊이 비교와 알파 블렌딩 파이프라인을 포함한다. 각각의 파이프라인은 여러 단계로 되어있다. 예를 들어 [10]에서 깊이 비교는 2단, 알파 블렌딩의 3단 파이프라인이다. 처음에는 MIU와 프레임 버퍼의 깊이 값을 각각의 C-RAM 깊이 비교 파이프라인에서 비교한다. 만약 깊이 비교 시 성공이면 깊이 값을 입력 버퍼(input buffer)에 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.

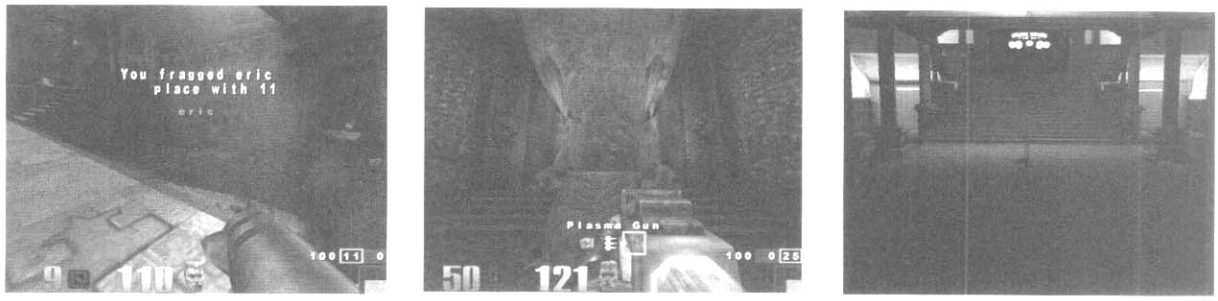
4.4 세번째 메모리 시스템의 대기시간

기존 2개의 메모리 시스템과 비교해보면 3번째 메모리 시스템은 픽셀 캐시 블록에 적합한 C-test 수행해서 지연시간을 줄일 수 있다. 내부 버스 폭은 픽셀 캐시 블록의 크기와 같고 ALU의 수는 C-test를 한 사이클에 수행할 수 있을 정도라고 가정하였다. 이때, 픽셀 캐시의 C-test를 끝내기 위한 지연 대기기간은 프레임 버퍼로부터 읽고 쓰기 위한 대기시간과 C-test를 수행하는 한 사이클과 같다.

5. 실험 시뮬레이션 결과

제안하는 구조의 타당성을 위해서, 다양한 시뮬레이션 결과를 수행하였다. 이를 위하여 Trace-driven 시뮬레이터를 사용하였다. 1600x1200 해상도에서 Mesa Opengl 호환하는 API로 Trace를 3가지 벤치마크(Quake3 demo I, Quake3 demo II, Lightscape)를 사용하여 만들었다. 각각의 벤치마크에서 100 프레임이 각각의 trace를 생성하기 위해 사용되었다. 각각 벤치마크의 모델 데이터는 라운드 로빈 방식으로 래스터라이저에게 분배 되었다. 이러한 trace로 픽셀 캐시 시뮬레이션은 잘 알려진 Dinero III 캐시 시뮬레이터[20]를 변경하여 수행하였다. 유지 문제 발생 비율은 5.2 단락에서, 메모리 대기 감소 비율은 각각의 trace를 계산해서 5.3 단락에서 보일 것이다.

(그림 10)은 벤치마크를 캡처한 것을 보여준다. Quake3는 현재 비디오 게임 중에 하나이고 시뮬레이션에서 벤치마크를 위해서 사용된다. Lightscape는 SPECviewperfTM 의 작



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

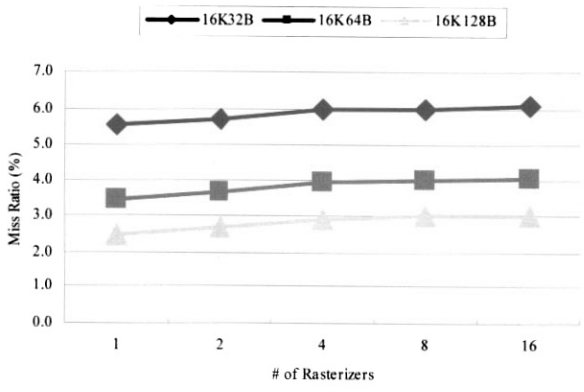
(그림 10) 3가지 벤치마크

품으로 OpenGl 환경 아래서의 3D 렌더링 시스템 벤치마크 표준으로 사용된다. Lightscape는 [16]에서도 언급했듯이 다른 SPECviewperfTM 벤치마크에 비해서 높은 장면 복잡도와 독특한 픽셀캐시 미스 분포를 보이기 때문에 사용되었다.

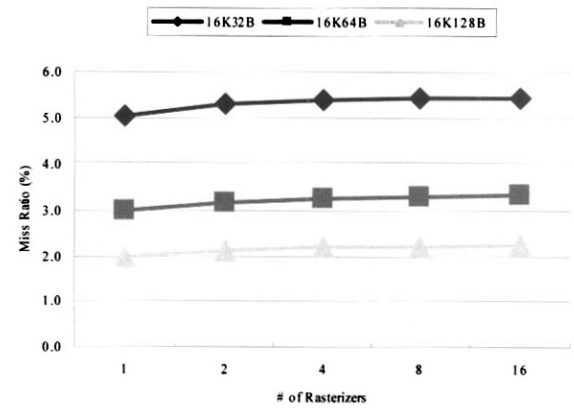
5.1 픽셀 캐시 시뮬레이션

[16]에서는 기존 픽셀 캐시 구조에서 캐시 크기, 블록 크기, set associativity에 따른 캐시 미스 비율을 제공하였다. 시뮬레이션 결과는 캐시 사이즈나 associative 사이즈가 아닌 블록 사이즈에 따른 미스 비율을 보여준다. 또한, 블록 사이즈가 커짐에 따라 미스 비율이 감소하는 것을 보여준다. [9]에서 말한 것처럼, 이러한 결과는 그래픽 하드웨어 렌더링은 시간적 지역성(temporal locality)이 아닌 공간적 지역성(spatial locality)에 그대로 영향을 받는다. 그러므로 우리는 다양한 블록 사이즈로 픽셀 캐시 시뮬레이션 하였다.

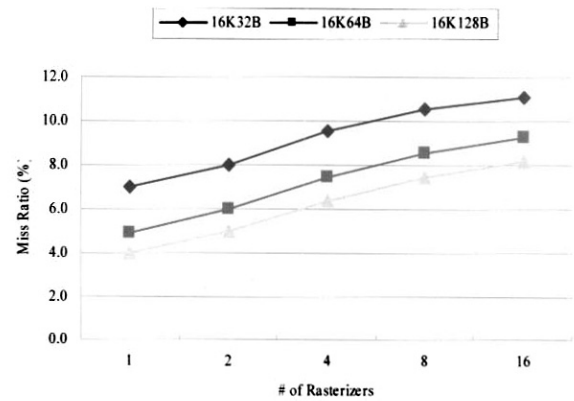
(그림 11)에서 direct-mapped 방식의 16 Kbytes의 깊이 캐시와 1개에서 16개 까지 래스터라이저를 증가 시키면서 3가지 다른 블록 사이즈에서 픽셀 캐시 미스 비율을 보여준다. 왜냐하면 삼각형은 라운드 로빈 방식으로 많은 래스터라이저에 나누어져 있기 때문이고, 많은 래스터라이저의 각 픽셀 캐시의 locality는 1개의 래스터라이저 locality 보다 감소하기 때문이다. 시뮬레이션 결과는 미스 비율이 (그림 11) (a)와 (그림 11) (b)에서 래스터라이저 숫자가 증가하는 것만큼 상당히 느리게 증가한다. 그러나 (그림 11) (c)의 미스 비율은 이전 2가지 경우와 비교하여 빠르게 증가한다.



(a) Quake3 I



(b) Quake3 II

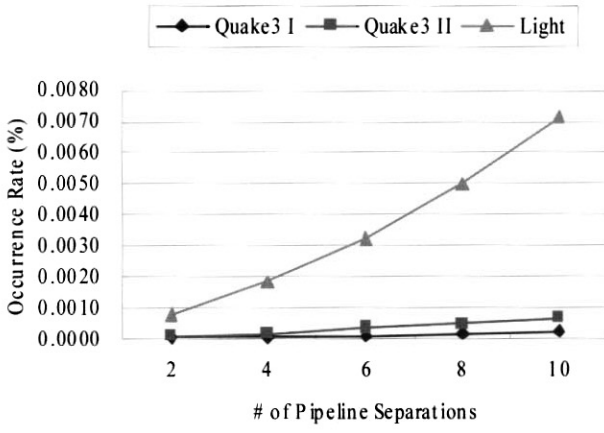


(c) Lightscape

(그림 11) 픽셀 캐시 미스 비율

5.2 유지(Maintenance) 문제 발생 비율

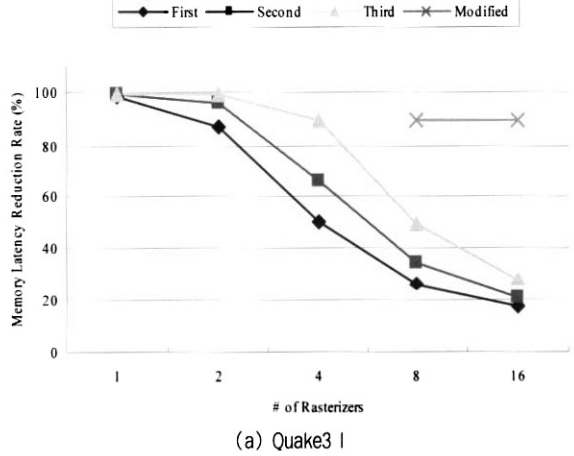
(그림 12)는 깊이 비교 와 색깔 쓰기 사이 단계가 2, 4, 6, 8, 10개 로 나뉨에 따라 maintenance 문제가 발생하는 비율을 보여준다. [16]에서와 마찬가지로 우리는 기하학 처리 단계 유닛, 프레임 메모리 시스템, 텍스처 매핑 유닛이 완벽하다고 가정한다. 그것은 각각의 스테이지의 대기시간이 0임을 가정한다. 그러므로 각각 래스터라이저의 한 개의 픽셀 파이프라인이 한 사이클 동안 1개의 픽셀을 발생시킨다. (그림 12)에서 보는 바와 같이 maintenance 문제의 발생은 매우 느물다 따라서, 이 문제에 따른 성능 감소는 무시할 것이다.



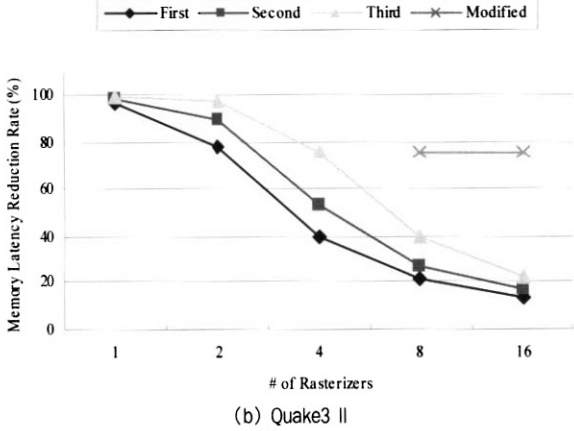
(그림 12) Maintenance 문제 발생 비율

5.3 메모리 latency 감소 비율

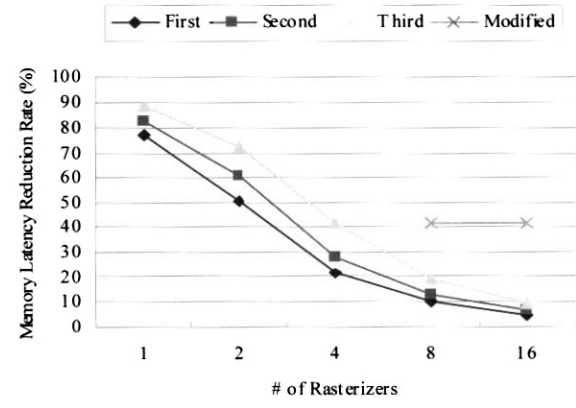
픽셀 캐시로부터 반복 되는 캐시 블록은 테일 포인터가 가리키는 전체 출력 큐의 끝에 저장된다. 헤드 포인터가 가리키는 처음에 도착했을 때 그것은 프레임 버퍼로 저장된다. 전체 파이프라인은 픽셀 출력 큐가 채워지지 않을 때까지 멈추지 않는다. 그러므로 무한 버퍼 크기의 경우에는 제한하는 구조에서는 완벽한 zero-latency 메모리 시스템을 성취할 수 있다.



(a) Quake3 I



(b) Quake3 II



(c) Lightscape

(그림 13) 메모리 latency 감소 비율

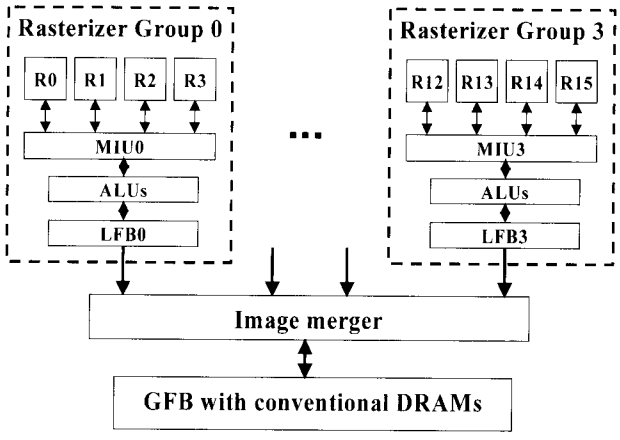
(그림 13)에서는 다음 단락에서의 보일 래스터라이저 숫자에 따른 3개 메모리 시스템과 변경된 구조에서의 메모리 지연시간 감소 비율을 보여준다. 100%에서 감소비율은 zero-latency 메모리 시스템을 나타낸다. 0% 감소 비율의 경우에는 모든 메모리 latency가 픽셀 캐시 미스에 요구되어 진다. 우리는 3가지 메모리 시스템에서 완전한 C-test를 하기 위한 사이클 수가 각각 16, 12, 8 이라고 가정하였다. 이는 픽셀 캐시 불락 사이즈, ALU의 개수, DRAM의 성능 등에 의하여 결정된다, 우리는 또한 전체 픽셀 출력 큐의 개체(entity)수는 128로 고정되었다고 가정하였다.

시뮬레이션 결과는 1개의 래스터라이저와 2개의 래스터라이저의 2번째와 3번째 메모리 시스템에서 거의 zero-latency 메모리 시스템을 만들 수 있다고 보인다. 4개의 래스터라이저에서의 경우에도 상당한 감소 비율을 얻었다. 래스터라이저가 증가할수록 동시에 MIU로 입력되어 교체되어진 블록의 수가 비례적으로 증가하기 때문에 감소 비율은 감소한다. 래스터라이저 수가 8개 혹은 16개일 경우 감소 비율은 현저하게 떨어진다.

5.4 8개 이상의 래스터라이저에서의 성능 향상

(그림 4)에서 제안한 구조의 성능은 sort-last 렌더링 시스템에서 확장성(scalability)을 사용해서 성능 향상 시킬 수 있다. (그림 14)에서 16개의 래스터라이저를 같은 변형된 구조를 보여준다. 래스터라이저가 4개 까지는 효과적인 메모리 시스템을 유지하기 때문에, 4개의 래스터라이저를 1개의 그룹으로 통합하였다. 각각의 그룹이 4개의 래스터라이저를 가지는 제안된 구조는 (그림 14)와 같다.

(그림 14)의 전체 구조와 실행 플로우는 sort-last 렌더링 머신과 유사하다. 각각의 래스터라이저 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 래스터라이제이션 과 이미지 합성 단계는 중첩되게 실행된다.



(그림 14) 16개의 래스터라이저를 가지는 변경된 구조

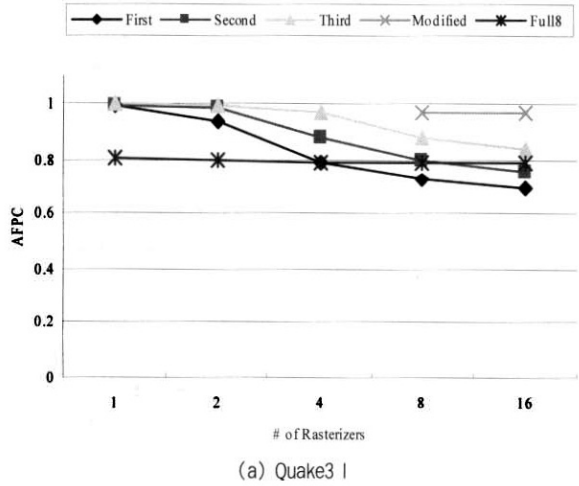
각각의 래스터라이저 그룹은 독립적으로 실행되기 때문에, 8개와 16개의 래스터라이저에서 memory latency reduction은 4개의 래스터라이저와 같다. 하지만 LFBs(2개의 8개 래스터라이저 와 4개의 16개 래스터라이저)와 image merger는 렌더링 프로세서에 내장되어야 한다.

5.5 성능 평가

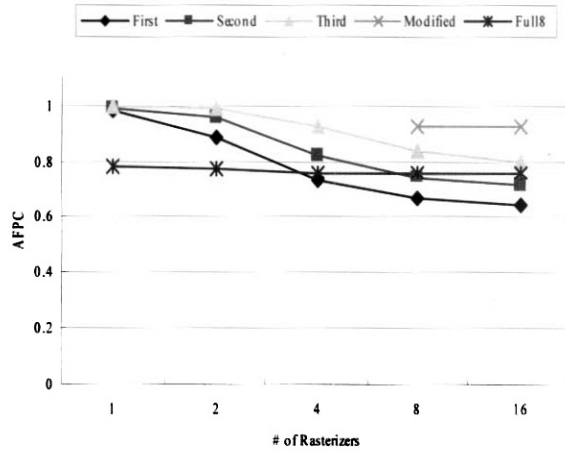
분석적인 성능 평가를 위해서 우리는 래스터라이저에서 사이클 당 평균 프레그먼트 사이클(AFPC)을 계산하였다. [16]에서는 픽셀 캐시와 텍스처 캐시의 miss penalty가 성능의 감소를 가져온다고 가정하였다. 본 논문에서는 픽셀 캐시에 의한 메모리 latency가 성능 감소를 가져온다고 가정한다. 그래서, AFPC는 다음과 같이 계산된다.

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

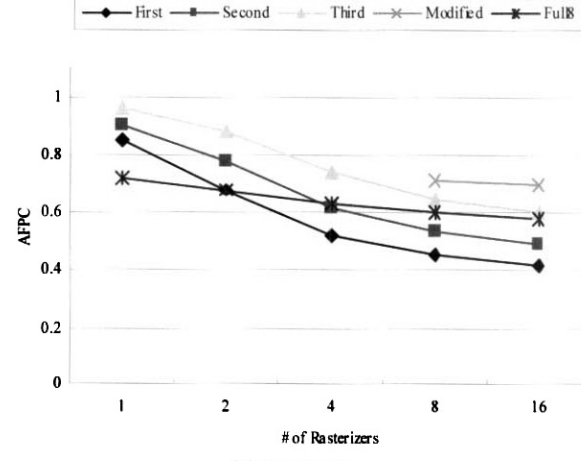
Miss Rate 은 픽셀 캐시 미스 비율이고, Latency는 픽셀 캐시 미스에 따른 메모리 대기 시간 사이클 타임이다. 그리고 reduction 은 (그림 13)에서 보이는 감소 비율을 나타낸다. 이 방정식의 분모는 래스터라이저에서 사이클 당 평균 프레그먼트를 나타낸다.



(a) Quake3 I



(b) Quake3 II



(c) Lightscape

(그림 15) 제안하는 구조의 AFPCs

(그림 15)는 래스터라이저수와 5개의 다른 메모리 구성에 따른 AFPC를 나타낸다. 0%의 감소비율을 가지고 있는 3번째 메모리 시스템의 AFPC는 픽셀 캐시 미스 발생시 8 사이클이 소용되기 때문에 full8으로 나타내었다. AFPC의 full8은 다른 4개의 제안된 구성과 비교하도록 제공된다. 예를 들어 (그림 15) (a)의 4개의 래스터라이저를 보면AFPC의 full8은 처음 메모리 시스템과 같다. n 개의 래스터라이저에서의 성능 증가는 n 과 AFPC을 곱하면 쉽게 계산된다. 그리하여 (그림 15) (a)와 같이 제안된 구조의 AFPC는 16개의 래스터라이저에서 선형으로 증가함을 보인다.

6. 결 론

이 논문에서 제안하는 새로운 병렬 처리 프로세서 구조는 픽셀 캐시와 픽셀 캐시 미스로 인한 메모리 latency 감소의 일관성 문제를 해결할 수 있다. 우리는 제안하는 구조의 trace-driven 시뮬레이터를 만들었다. 실험 시뮬레이션은 제안하는 구조가 16개의 래스터라이저에서 15.5배의 속도 향상을 가져온다. 향후 제안하는 구조의 프로토타입을 만들 계획이다.

참 고 문 헌

[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] K. Akeley, "RealityEngine graphics," In Proceedings of SIGGRAPH '93, pp.109-116, Aug., 1993.

[3] J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal, "InfinityReality: A real time graphics system," Proceedings of SIGGRAPH '97, pp.293 - 302, Aug., 1997.

[4] M. Deering and D. Naegle, "The SAGE Architecture," In Proceeddings of SIGGRAPH 2002, pp.683-692, July. 2002.

[5] G. Humphreys, M.Eldridge, I. Buck, G. Stoll, M. Everett and P. Hanrahan, "WireGL: A Scalable graphics system for clusters," In Proceedings of SIGGRAPH 2001, pp.129-140, Aug., 2001.

[6] 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.

[7] S. Molnar, M. Cox, M. Ellsworth, and H. Fuchs, "A sorting classification ofparallel rendering," IEEE Computer Graphics and Applications, Vol.14, No.4, pp.23-32, July, 1994.

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

[9] F. D. Michael, A. S. Stephen, and G. L. Michael, "FBRAM: A new form memory optimized for 3D Graphics," In Proceedings of SIGGRAPH '94, pp.167-174, 1994.

[10] 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.

[11] A. Kugler, "The setup for triangle rasterization," 11th Eurographics Workshop on Computer Graphics Hardware, pp.49-58, Aug., 1996.

[12] Z. S. Hakura and A. Gupta, "The design and analysis of a cache architecture for texture mapping," In Proceedings of the 24thInternational Symposium on Computer Architecture, pp.108-120, June, 1997.

[13] H. Igehy, M. Eldridge, and K. Proudfoot, "Prefetching in a texture cache architecture," In Proceedings of 1998 SIGGRAPH/Eurographics Workshop on Graphics Hardware, pp.133-142, August, 1998.

[14] 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).

[15] L. Garber, "The wild world of 3D graphics chips," IEEE Computer, Vol.33, No.9, pp.12 - 16, Sept., 2000.

[16] Woo-Chan Park, Kil-Whan Lee, Il-San Kim, Tack-Don Han, and Sung-Bong Yang, "An Effective Pixel Rasterization Pipeline Architecture for 3D Rendering Processors," IEEE Transactions on Computers, Vol.52, No.11, pp.1501-1508, Nov., 2003.

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

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

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

[20] 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.

[21] D. A. Patterson and J. L. Hennessy, Computer organization & design: The hardware/software interface, Morgan Kaufmann Publisher Inc., Second edition, 1998.

[22] http://www.spec.org/gpc/opc.static/opcview70.html.

박 우 찬

e-mail : pwchan@sejong.ac.kr

1993년 연세대학교 전산과학과(학사)

1995년 연세대학교 전산과학과(이학석사)

2000년 연세대학교 컴퓨터과학과(공학박사)

2000년~2001년 연세대학교 아식설계공동연구소 연구원

2001년~2003년 연세대학교 연구교수

2003년~현재 세종대학교 컴퓨터공학과 조교수

관심분야 : 3차원 그래픽 가속기, 실시간 렌더링 프로세서, Computer arithmetic, 고성능 컴퓨터 구조.

윤 덕 기

e-mail : dkyoon@rayman.sejong.ac.kr

2005년 세종대학교 전자정보대학 컴퓨터공학과(학사)

2005년~현재 세종대학교 컴퓨터공학과 석사과정중

관심분야 : 3차원 그래픽 가속기, 고성능 컴퓨터 구조

김 경 수

e-mail : ksKim@rayman.sejong.ac.kr

2005년 세종대학교 전자정보대학 컴퓨터공학과(학사)

2005년~현재 세종대학교 컴퓨터공학과 석사과정중

관심분야 : 3차원 그래픽 가속기, 고성능 컴퓨터 구조