프레임 버퍼 액세스 대역폭 개선에 관한 연구

문 상 호[†] · 강 현 석[††] · 박 길 흠[†††]

요 약

본 논문에서는 프레임 버퍼 액세스 대역폭을 개선하는 두 가지 방안을 제안한다. 첫째 방안은 래스터라이저내에 Span Z Buffer와 Span Z & Color Buffer를 가지는 SBUFRE라 불리어지는 새로운 래스터라이저이고, 두 번째 방안은 DRAM 내부에 Z값 비교기를 갖는 ZDRAM 이다. 이들 방안은 읽기-수정-쓰기 Z 버퍼 비교를 단지 쓰기 동작만으로 바꾸어 주므로 프레임 버퍼 액세스 대역폭을 약 50% 정도 개선한다.

A study to improve the frame buffer access bandwidth

Moon, Sang Ho[†] · Kang, Hyun Syug[††] · Park, Kil Heum[†††]

ABSTRACT

This paper introduces two schemes to improve the frame buffer access bandwidth. The first scheme suggests a rasterizer called SBUFRE that has Span Z Buffer and Span Z & Color Buffer within a rasterizer. The second scheme suggests a ZDRAM that has Z-value comparator within the DRAM. These schemes are to convert read-modify-write Z buffer compare into single write only operation that improves approximately 50% frame buffer access bandwidth.

1. 서 론

컴퓨터 그래픽스 기술은 1960년대부터 General Electric사의 모의 비행 시뮬레이션용 NASA Ⅱ시스템과 Evans & Sutherland사의 분자 모델링용 PS-3000과 같은 시스템에 적용되었지만 극히 제한적인 분야에서 사용되었다. 최근에는 광고, 오락, 가상현실 등의 응용분야에서 무궁무진하게 그 쓰임새를 넓혀가고 있는 추세이다. 이를 지원하는 여러 가지 요소가 있지만 1980년대 초반부터 반도체 메모리와 래스터 연산 출력(Raster Operation Display)를 위한 CRT (Cathode Ray Tube)의 가격저하, VLSI 설계 및 공정 기술의 급속한 발전에 힘입어 3차원 그래픽스를 실시간으로 처리하는 시스템의 개발이 활발히 진행되고 있다.

3차원 컴퓨터 그래픽스는 기하학적 처리(geometry processing)와 래스터라이징(rasterizing)이라는 두 개의 주된 기능을 처리하는 파이프라인 방식으로 구성되어 삼각형, 선 등의 기본요소(primitive)로 나타내어지는 물체를 기하 변환 처리기(geometry processor)와 래스터라이저(rasterizer)를 거치면서 변환된 화소값을 프레임 버퍼에 저장한다. 기하 변환 처리기 부분은 기하학적 변환(geometric transformation), 광원 모델링(light modeling), 클리핑(clipping), 투시 투영(perspective projection) 등을 처리하기 위해 부동 소수점 연산이 수행되므로 일반적으로 부동 소수점 프로세

† 정 회 원:영남전문대학 전자계산기과
†† 종신회원:경상대학교 컴퓨터과학과
††† 정 회 원:경북대학교 전자전기 공학부
논문접수:1995년 11월 13일, 심사완료:1996년 1월 12일

서나 디지털 영상처리 프로세서를 사용하여 구현한다. 반면에 래스터라이저 부분에서는 기본요소의 각 화소에 대해 음영 모델에 의거한 색상값, 은면제거를 위한 Z값 비교 등의 다량의 화소 데이터값을 구하는 계산이 수행된다[1]. 여기서는 비교적 간단하면서 반복적인 연산이 수행되므로 고유의 VLSI 칩을 설계, 제작하여 사용한다[2, 3]. 은면제거를 위한 3차원 그래픽스 처리를 위해서는 프레임 버퍼로부터 매 화소당 읽은 기존의 Z값과 보간기에서 보간법(interpolation)으로 구한 Z값을 비교하여 최종의 색상값과 Z값을 프레임 버퍼에 저장한다. 이 과정에서 래스터라이저와 프레임 버퍼사이에 많은 양의 데이터가 오고 가게 되는데 대부분 그래픽스 시스템의 성능은 여기서 병목현상이 생기게 된다[4, 5]. 따라서 그래픽스 시스템의 성능을 향상시키기 위해 이 부분의 대역폭을 높이는 것이 매우 중요하다.

본 논문에서는 고성능 그래픽스 시스템의 실현을 위해 요구되는 단계별 성능을 Ⅱ절에서 분석하고, Ⅲ절에서는 고성능 그래픽스 시스템 구조를 살펴보고, Ⅳ절에서는 래스터라이저와 프레임 버퍼간의 대역폭을 개선하는 방안 및 실현을 위한 구조를 제안하고자 한다.

2. 단계별 성능분석

3차원의 물체를 표현하는 방법은 여러 가지가 있지만 다면체(polyhedrons)로 근접시키는 방식을 사용하면 선으로 구성되는 다각형(polygon)은 같은 평면상에 존재하기 때문에 은면제거 알고리즘과 랜더링(rendering) 알고리즘에 선형성을 이용할 수 있어 매우 유용하다. 대화형 3차원 그래픽스 시스템은 계산량을 줄이기 위해 다각형의 가장 간단한 형태인 삼각형으로 물체를 모델링하기도 한다. 현재 가장 빠른 그래픽스 시스템은 초당 100만개의 삼각형을 Gouraud 쉐이딩(shading)으로 처리할 수 있다[6]. 이를 위해 요구되는 계산량과 대역폭을 단계별로 고찰한다.

2.1 기하학적 처리를 위한 계산량

한 삼각형을 물체 좌표계(object coordinate)에서 화면 좌표계(screen coordinate)로 변환하고 클리핑, 광원 모델링, 투시 투영 등의 기하학적 변환을 수행하는데 부동 소수점 연산이 약 200번 필요하다. 삼각형이 메쉬 형태로 주어질 경우에는 이 계산량은 반으로 줄어들어 상기의 성능으로 처리하기 위해서 기하 변환 처리기는 100 MFLOPS 가량의 성능이 필요하다.

2.2 기하학적 처리를 위해 요구되는 대역폭

하나의 삼각형은 기하학적 값을 나타내는 데이터값(3개의 꼭지점 좌표값을 나타내는 좌표계 데이터와 표면 정규값, 삼각형 자체의 표면 정규값), 물체의 색상을 나타내는 물체의 색상 데이터값, 물체의 광학특성을 나타내는 물체 표면 매개변수(확산계수 Kd, 투명계수 Kt, 반사계수 Ks 및 반짝임 n), 그리고 주어진 장면에 대한 총체적인 정보를 나타내는 장면 데이터(광원관련 데이터, 뷰 피라미드등)를 감안하면 대략 40~200 바이트 정도로서 표현할 수 있다. 이런 분량의 데이터는 물체 공간에도 필요하고 화면 공간에서도 비슷하게 필요하므로 기하 변환 처리기의 입력과 출력 대역폭은 대략 초당 40~200 메가 바이트가 요구된다.

2.3 래스터라이저 계산량

일반적으로 기본요소는 많은 화소로 구성되므로 엄청난 양의 데이터가 래스터라이저 내에서 처리된다. Z 버퍼 알고리즘을 사용하는 그래픽스 시스템은 모든 화소당 단순한 고정 소수점 덧셈으로 진행되는 전향편차(forward difference)를 사용한다. 또한 한 화소마다 해당 색상성분을 구하는데 한 번의 고정 소수점 덧셈을 필요로 하는 전향 편차를 이용한 선형적인 보간법 즉, Gouraud 쉐이딩을 수행한다. 삼각형이 평균 50개의 화소를 가진다고 가정할 때 래스터라이저는 상기의 성능을 위해 초당 20,000~25,000만 정도의 고정 소수점 덧셈을 수행해야 한다.

2.4 래스터라이저와 프레임 버퍼간의 대역폭

기본요소내의 모든 화소에 대해 그래픽스 시스템은 프레임 버퍼에서 Z값과 비교된다. 만일 보여지는 화소이면 새로운 Z값과 색상값을 다시 프레임 버퍼에 써야 한다. 기본요소를 삼각형으로 간주할때, 삼각형의 평균 화소는 50개이며 이 중 3/4 만이 보여진다고 가정하면 초당 11,250만번 정도의 프레임 버퍼 액세스가 요구된다.

3. 기존의 고성능 그래픽스 시스템의 구조

초당 100만개의 삼각형을 처리할 수 있는 일반적인 고성능 그래픽스 시스템은 그림 1과 같은 MIMD (multiple instruction multiple data)구조를 갖는다[7]. 기하학적 처리를 하여 화면 좌표계로 변환된 삼각형이나 스팬 데이타는 고속의 대역폭을 갖는 글로벌 네트워크, 크로스바 네트워크, 링 네트워크를 통해서 래스터라이저로 전달된다. 래스터라이저도 MIMD 형태의 병렬로 구성하여 요구되는 성능을 처리한다. 프레임 버퍼는 DRAM이나 VRAM으로 구현하고 메모리 사이클당 여러 화소를 동시에 액세스 할 수 있게 여러 개의 뱅크로 구성하여 래스터라이저와 프레임 버퍼간의 대역폭을 만족시킨다.

화소 병렬(pixel-parallel) 래스터라이저 설계에서 프레임 버퍼 메모리는 각각이 n개의 주사선과 m개의 화소에 해당하는 m × n 크기의 영역으로 나누어지고 래스터라이저가 각 영역을 담당하는 인터레이스(interlace) 프레임 버퍼 방식이 가장 많이 사용된다. 일반적으로 m × n의 크기는 다각형내에 화소수 보다 더 작기 때문에 각각의 래스터라이저는 기본요소내에서 여러 개의 화소를 담당하게 되므로, 래스터라이저 내부에 FIFO와 같은 버퍼링을 가진 MIMD 구조는 모든 래스터라이저에 부하를 균등하게 분배한다.

액세스 시간이 60ns인 현재의 DRAM과 VRAM은 Z 버퍼를 읽기-수정-쓰기(read-modify-write) 사이클로 초당 600만번 액세스할 수 있으므로 100만개의 삼각형을 처리하기 위해서는 최소한 20개의 분할이 필요하다[8].

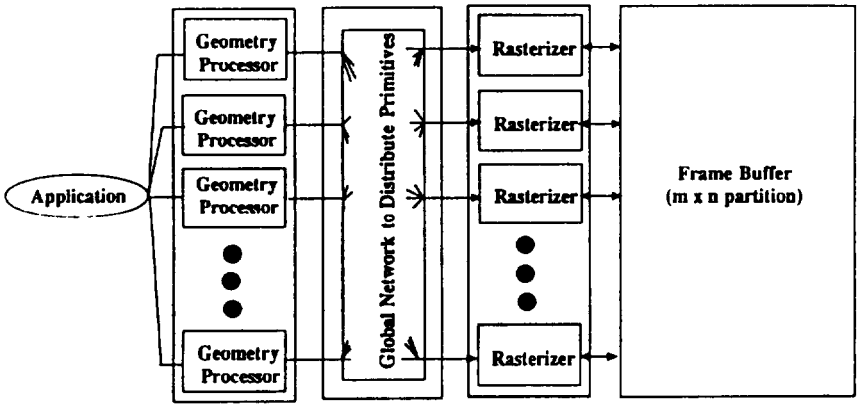
4. 프레임 버퍼 대역폭 개선방안

초당 100만개의 삼각형을 처리하는 3차원 그래픽스 시스템은 요구되는 많은 계산량과 메모리 액세스 대역폭 문제를 해결하기 위해 기하 변환 처리기와 래스터라이저를 병렬처리 시스템으로, 프레임 버퍼는 인터레이스 방식으로 구성하고 있다. 그러나 최근의 반도체 메모리의 고집적화가 급속하게 진행되는데 반해 액세스 속도는 그에 미치지 못하므로 효율적인 프레임 버퍼를 구현하는데 장애 요소가 되고 있다.

본 논문에서는 구체적인 프레임 버퍼 병목현상의 원인을 고찰하고 고성능 그래픽스 시스템에서 요구되는 래스터라이저와 프레임 버퍼간의 대역폭을 개선하는 두 가지 방안과 구조를 제안한다.

4.1 프레임 버퍼 액세스 병목 원인 분석

그래픽스 시스템의 프레임 버퍼는 계산된 최종 색상값을 저장하는 색상 버퍼(Color-buffer), Z값을 저장하는 Z 버퍼, 오버레이 버퍼(overlay-buffer) 등으로 구성된다. 초창기 그래픽스 시스템은 프레임 버퍼가



(그림 1) 고성능 그래픽스 시스템의 일반적인 구조
(Fig. 1) A general architecture for high performance graphics system

DRAM으로 구성되어 CRT의 특성상 요구되는 화면 재생 동작 때문에 래스터라이저가 비록 빠른 속도로 이미지를 생성하더라도 이미지를 그리는데는 약 25% 정도의 시간만 프레임 버퍼 액세스에 할당되므로 고속으로 이미지를 생성하는데 상당한 부담이 있었다. 그러나 1984년 Texas Instruments사에서 최초로 다중 포트 액세스 기능을 갖는 Video DRAM(VRAM)이 제공되면서 래스터라이저는 DRAM이 제공하는 메모리 액세스율로 프레임 버퍼 액세스가 가능하게 되었다. 그러나 VRAM의 다중 포트에서 SAM(Serial Access Memory) 포트는 래스터라이저가 생성한 이미지를 화면에 그리기 위해 프레임 버퍼의 데이터 전송을 위한 액세스율은 충분히 제공하지만, RAM 포트의 경우는 DRAM의 일반적인 속성이자 중요한 특징인 많은 용량을 제공하는 대신 프로세서에서 요구되는 빠른 액세스율을 만족시키지 못하는 문제점이 있다.

식(1)로 부터 계산된 화소 갱신하는데에 필요한 시간인 T_{pixel}은 표 1에서 보여주듯이 화면이 복잡하고 고속의 프레임을 생성할 때 나노초 미만이 요구됨을 알 수 있다[9].

$$T_{pixel} = Frame\_Time / Pixels\_In\_Frame \quad (1)$$

〈표 1〉 프레임과 화소당 요구되는 화소 갱신 시간
〈Table 1〉 Requested pixel update time according to frames and pixels

Frame / Pixels	1 sec	1/10 sec	1/30 sec	1/60 sec
1 M	1,000 ns	100 ns	33 ns	17 ns
5 M	200 ns	20 ns	6.7 ns	3.3 ns
10 M	100 ns	10 ns	3.3 ns	1.7 ns
25 M	40 ns	4 ns	1.3 ns	0.7 ns
50 M	20 ns	2 ns	0.7 ns	0.3 ns

메모리 칩의 종류에 따라 제공할 수 있는 화소 갱신 시간을 식(2)로 부터 구할 수 있다.

$$T_{pixel} = (Memory\_Cycle\_Time \times Memory\_Chip\_Size) / (Pixels\_On\_Screen \times Memory\_Bit\_Width) \quad (2)$$

〈표 2〉 메모리 유형별 가능한 화소 갱신 시간
〈Table 2〉 Available memory pixel update time

Density / Width	256K	1M	4M	16M
×16	-	10ns	40ns	128ns
× 8	-	20ns	80ns	256ns
× 4	10ns	40ns	160ns	0.5μs
× 1	40ns	160ns	640ns	2.56μs

예를 들어 액세스 시간이 60ns인 VRAM을 사용한다고 가정할 때 기존의 시스템에서 3차원 그래픽스 처리를 위해서 Z 버퍼에서 Z값을 읽고 비교해서 다시 쓰기 위해 메모리의 읽기-수정-쓰기 사이클 시간이 160ns가 소요되므로, 메모리 칩의 종류에 따라서는 요구되는 화소 갱신율을 만족할 수 없음을 알 수 있다. 그러므로 고성능 그래픽스 시스템을 구현하기 위해서는 저집적 메모리칩을 계속 사용하여야 한다. 그러나 최근의 고집적 메모리칩을 사용하여 요구되는 대역폭을 만족시키기 위해 프레임 버퍼를 매우 많이 분할 함으로써 요구되는 대역폭은 만족시킬 수 있지만 이 경우 메모리의 사용 효율이 떨어져 저집적 메모리 칩을 사용하는 것과 같은 결과를 초래한다. 따라서 이와 같은 문제를 개선하고 고집적 메모리 칩의 사용 효과를 얻기 위해서는 프레임 버퍼 액세스 대역폭을 향상시키는 방안이 강구되어야 함을 알 수 있다.

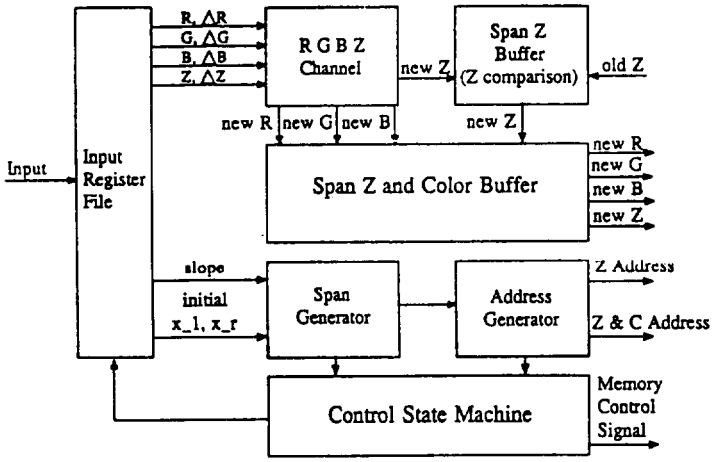
4.2 래스터라이저 구조 개선 방안

일반적인 래스터라이저의 내부 구조는 매 메모리 사이클마다 한 화소에 해당하는 Z값을 프레임 버퍼에서 읽어와서 새로 계산된 Z값과 색상값을 다시 프레임 버퍼에 씀으로써 기본요소의 화소값을 구하는 방식을 사용한다[10]. 그러나 SBUFRE(Span BUFfer Raster Engine)라 불려지는 새로운 래스터라이저는 입력 레지스터 화일(Input Register File), RGBZ 채널, XY 주소 생성기 및 제어 상태 장치(Control State Machine)으로 구성되는 래스터라이저 내부에 Span Z Buffer와 Span Z & Color Buffer를 가지고 있어 스팬별로 Z값을 읽고 계산된 Z값과 색상값을 동시에 쓸 수 있게 하여 프레임 버퍼 액세스 대역폭을 개선한다.

4.2.1 SBUFRE 구조

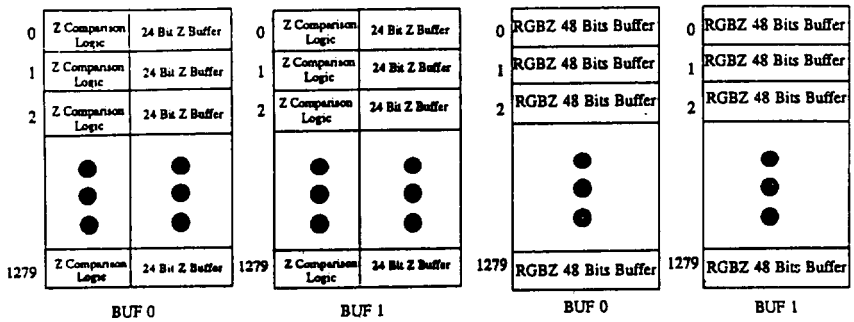
SBUFRE는 그림 2와 같이 7개의 블록으로 구성되어 있다. 입력 명령을 받아들이는 입력 레지스터 화일, 보간법을 이용하여 색상값 계산과 은면제거를 위한 Z값을 계산하는 RGBZ 채널, 스팬의 좌표값을 구하는 스팬 생성기와 주소 생성기가 있다.

제어 상태 장치는 SBUFRE 내부를 제어하는 신호와 프레임 버퍼를 제어하는 신호를 생성한다. Span Z Buffer는 (그림 3(a)참조) 1280 화소 × 24 비트에 해당하는 Z값을 저장하는 버퍼와 Z값 비교 로직으로 구성되어 읽어온 스팬의 Z값과 Z 채널에서 계산된 새로운 Z값과의 비교연산을 수행한다. Span Z & Color Buffer(그림 3(b)참조)는 1280 화소 x 48 비트의 크기를 갖는 단순한 버퍼로 RGBZ 채널에서 구한 색상값과 Span Z Buffer에서 비교된 Z값을 프레임 버퍼로 쓰기 전에 스팬 단위로 일시적으로 저장하는데 사용된다.



(그림 2) Span Z Buffer와 Span Z & Color Buffer를 가진 SBUFRE 블록도

(Fig. 2) SBUFRE block diagram with Span Z Buffer and Span Z & Color Buffer



(a) Span Buffer (b) Span Z and Color Buffer

(그림 3) 스팬 Z Buffer와 Span Z & Color Buffer

(Fig. 3) Block diagram of Span Z Buffer and Span Z & Color Buffer

4.2.2 SBUFRE 동작 원리

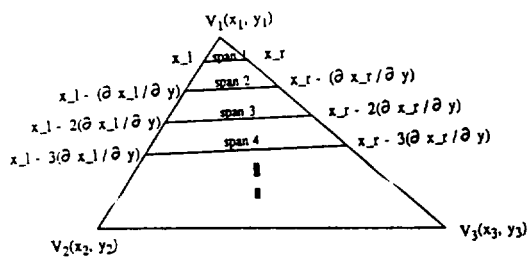
래스터라이징 처리를 위해 삼각형의 스팬에 해당하는 Z값만 Span Z Buffer에 미리 읽어와서 래스터라이저에서 화소당 보간기에서 구한 새로운 Z값과 비교연산을 수행한다. 스팬의 Z값을 프레임 버퍼에서 읽어 매 화소당 계산된 새로운 Z값을 Span Z Buffer에서 비교 연산을 수행하여 RGBZ 채널에서 계산된 색상값과 함께 Span Z & Color Buffer에 쓴다. 계산된 Z와 색상값을 매 화소마다 프레임 버퍼로 직접 쓰지 않고 Span Z & Color Buffer에 일시적으로 저장했다가 스팬 단위로 프레임 버퍼에 씀으로써 일반적인 그래픽스 시스템에서 사용되는 읽기-수정-쓰기 사이클을 쓰기 사이클로 대체할 수 있는 이점이 있다. 그림 4에서 삼각형내의 스팬 1에 대한 스팬을 프레임 버퍼로부터 읽어 각 화소에 대한 색상 및 Z값은 보간법으로 구하여 Span Z Buffer의 BUF 0에서 Z값 비교연산을 수행할 동안에 스팬 생성기는 x_l, x_r 초기값에서 $\partial x\_l/\partial y$, $\partial x_r/\partial y$값을 빼므로써 스팬 2의 좌표값을 구하여 프레임 버퍼로 부터 스팬 2의 Z값을 Span Z Buffer의 BUF 1에 미리 읽어 놓는다. 이 동작은 RGBZ 보간기와는 별도로 수행할 수 있어 연속되는 스팬에 대해 BUF 0과 BUF 1을 번갈아 가면서 Z값을 Span Z Buffer에 미리 읽어 놓는다. 즉 삼각형내의 스팬들간에는 응집성(coherence)이 있으므로 하나의 스팬이 처리될 동안에 인접한 다음의 스팬을 미리 읽어올 수(prefetch) 있다. Span Z & Color Buffer도 이 중으로 구성하여 래스터라이저와 프레임 버퍼사이의 액세스 대역폭을 최대로 활용하게 한다. 프레임 버퍼는 기본적으로 2 개의 메모리 뱅크 접근 방식 구조를 가지게 함으로써 한 메모리 뱅크에 Z값을 Span Z Buffer로 미리 읽어 놓는 동안에 다른 메모리 뱅크에 Span Z & Color Buffer의 Z값과 색상값 저장시 메모리 사이클의 충돌 없이 수행되게 한다.

4.2.3 개선 효과

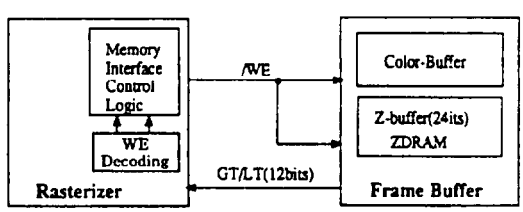
기존의 래스터라이저는 3차원 그래픽스 처리를 위해 읽기-수정-쓰기 메모리 사이클로 프레임 버퍼를 액세스하지만 SBUFRE는 2개의 메모리 뱅크 접근 방식(2-way interleaving) 구조를 갖는 프레임 버퍼에서 Span Z Buffer로 스팬의 Z값을 미리 읽어 놓고 Span Z & Color Buffer의 Z값과 색상값을 쓰기 위해 동시에 프레임 버퍼를 액세스 하므로 단지 읽기나 쓰기 메모리 사이클로만 수행할 수 있다. 그러므로 액세스 시간이 60ns인 메모리를 기준으로 할 때 정규 모드에서 읽기나 쓰기 사이클 시간이 115ns 인데 비해 읽기-수정-쓰기 사이클 시간은 160ns이므로 약 30%의 메모리 액세스 시간을 개선하고, 페이지 모드에서 읽기나 쓰기 사이클 시간은 45ns 이지만, 읽기-수정-쓰기 사이클 시간은 90ns이므로 약 50%의 프레임 버퍼 액세스 대역폭을 개선할 수 있다.

4.3 Z값 비교기를 내장한 메모리 방식

3차원 그래픽스 처리에 필요한 Z값 연산을 메모리 칩 내부에서 수행하는 ZDRAM(Z값 비교기를 내장한 DRAM)을 이용하여 프레임 버퍼 대역폭을 개선한다. ZDRAM을 이용한 시스템 구성은 그림 5에 나타나 있다. 프레임 버퍼는 일반적인 DRAM이나 VRAM으로 구성되는 색상 버퍼와 ZDRAM을 사용하는 Z 버퍼로 구성되어 있다. Z 버퍼의 ZDRAM은 새로 계산된 Z값이 기존의 Z값과 비교 결과 더 크면 GT 신호를 발생하고 더 작으면 LT 신호를 발생시켜



(그림 4) 삼각형내에서 스팬 생성
(Fig. 4) Span generation in the triangle



(그림 5) ZDRAM을 사용한 래스터라이저와 프레임 버퍼
(Fig. 5) Rasterizer and frame buffer block diagram using ZDRAM

래스터라이저로 전달하면 래스터라이저는 이 신호를 이용하여 Z 버퍼 내용을 그대로 유지할 것인지 혹은 갱신할 것인지를 결정하는 $\overline{WE}$ 신호를 WE 디코딩 로직에서 발생한다.

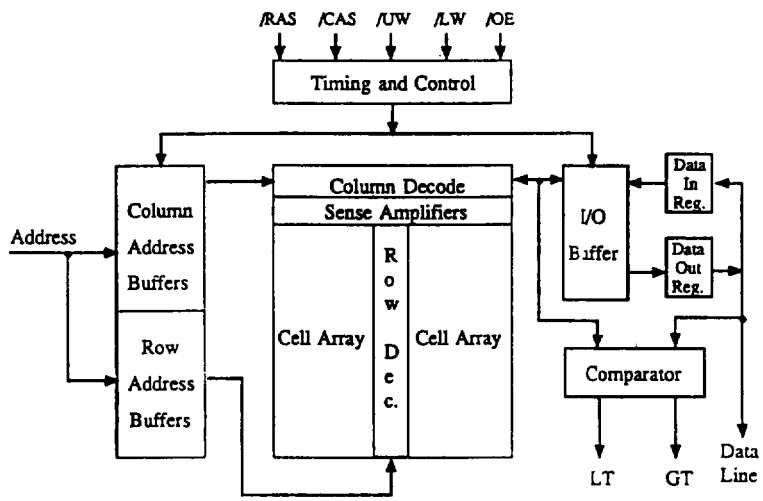
4.3.1 ZDRAM 구조

ZDRAM은 그림 6과 같이 기존 DRAM의 구조 및 액세스 사이클은 큰 변화가 없도록 하고 DRAM 내부에 단위 메모리의 입출력의 비트(데이터 라인)수에 해당하는 비교기와 비교결과를 외부의 제어 로직으로 전달할 수 있는 외부핀 2개, 그리고 이 핀으로 부터 비교결과 정보를 받아 메모리의 쓰기 신호를 디코딩해주는 외부 제어 로직으로 구성된다.

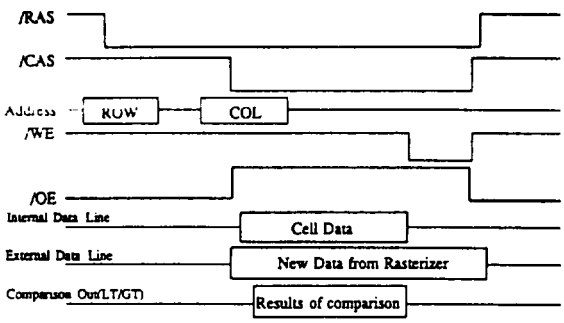
4.3.2 ZDRAM 동작 원리

ZDRAM의 비교 동작 사이클은 기존 DRAM에서 제공하는 쓰기 사이클을 변형하여 구현한다. DRAM의 늦게 쓰기 사이클(late write cycle)은 출력 인에이블로 조정되는 쓰기 사이클(Output Enable(OE) controlled write cycle)이라고도 한다. 이 방법에 의한 메모리 쓰기 사이클은 열 주소를 스트로우브함과 동시에 데이터를 쓰는 조기 쓰기 사이클(early write cycle) 동작과는 달리 주소지정에 의한 메모리칩 내의 셀 지정이 끝난 후에 쓰는 동작이 이루어진다. DRAM은 메모리셀 어레이의 주소가 지정되면 그 주소에 해당하는 메모리 셀 어레이에 저장되어 있던 데이터는 출력버퍼 전단까지 전달되는 특성 때문에 늦게 쓰기 사이클에서는 출력 인에이블(OE) 신호를 이용하여 메모리셀로 부터 나온 데이터가 버퍼를 통하여 칩 외부로 출력되는 것을 막아 주도록 되어 있다. ZDRAM은 이러한 DRAM의 늦게 쓰기 사이클 특성을 이용하여 그림 7과 같이 $\overline{CAS}$ 신호가 인에이블되면 주소가 지정된 셀의 데이터는 출력 버퍼 전단까지 전달되어 진다.

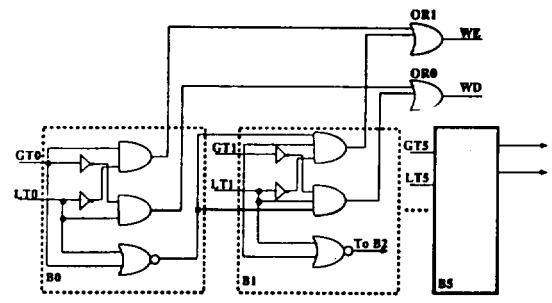
한편 새로운 데이터도 $\overline{CAS}$ 신호가 인에이블되는 시점과 동시에 메모리로 전달된다. 입출력 데이터 라인은 칩 내부 비교기의 입력단에 직접 연결되어서 셀로 부터의 데이터와 비교하여 그 결과를 래스터라이저내의 메모리 제어 로직으로 출력한다. 각 메모리의 단위 칩으로부터 래스터라이저로 입력된 비교결과는 래스터라이저내의 로직으로 입력되어 $\overline{WE}$ 신호를 발생시킬 것인지의 여부를 결정하는 메모리 제어신호 발생 로직으로 전달한다. 그림 8은 4 비트 데이터 라인을 갖는 메모리로 24 비트 Z 버퍼를 구성할 때의 로직을 보여준다.



(그림 6) Z값 비교기가 내장된 ZDRAM 블록도
(Fig. 6) Block diagram of ZDRAM with Z comparison



(그림 7) ZDRAM의 Z값 비교를 위한 타이밍도
(Fig. 7) Timing diagram for Z comparison of ZDRAM



(그림 8) Z값 비교에 따른 메모리 쓰기 인에이블 디코딩 로직
(Fig. 8) Memory write enable decoding logic according to Z value comparison

4.3.3 ZDRAM의 효과

ZDRAM은 3차원 그래픽스 시스템의 Z 버퍼 구현에 사용하면 매우 효과적이다. 일반적인 그래픽스 시스템에서 3차원 그래픽스 연산을 위한 Z값 비교동작은 읽기-수정-쓰기 메모리 사이클을 사용하지만 ZDRAM은 이러한 읽기-수정-쓰기 사이클에서 수정 사이클은 메모리 내부에서 수행되게 함으로써 읽기-수정-쓰기 사이클에 의한 Z값 비교 동작은 래스터라이저에서 소비되는 비교 동작 시간을 제외하고 약 160ns가 소요되지만 ZDRAM의 쓰기 사이클에 의한 Z값 비교 동작은 115ns가 소요된다. 페이지 모드 동작시도 90ns를 45ns로 대치할 수 있으므로 ZDRAM을 사용하여 3차원 그래픽스 시스템을 구현하면 프레임 버퍼 액세스 대역폭을 약 50% 개선할 수 있다.

5. 결 론

본 논문에서 제안하는 첫 번째 방안인 SBUFRE 구조는 내장된 Span Z Buffer와 Span Z & Color Buffer를 이용하여 계산된 최종의 색상값을 기존의 읽기-수정-쓰기 사이클에서 쓰기 사이클로 대치하여 스팬 단위로 프레임 버퍼에 저장하게 하므로 프레임 버퍼 액세스 대역폭을 50% 가량 개선할 수 있어 최근의 고집적화 추세의 반도체 메모리를 더욱 효율적으로 사용할 수 있다.

또한, 이 구조는 최근 개발된 몇 가지 새로운 DRAM의 구조에 더욱 적합하다. Rambus 표준을 채택한 RDRAM과 Synchronous DRAM, 칩 내부에 적은 양의 케쉬 메모리를 갖는 CDRAM은 신속한 케쉬 로딩에 적합한 특징이 있어 Span Z Buffer와 Span Z & Color Buffer 단위로 액세스가 가능하므로 기존의 시스템에서 화소 단위로 메모리를 액세스할 수 있는 구조보다 더욱 효율적으로 적용할 수 있는 구조이다[1].

두 번째 제안된 ZDRAM구조는 일반적인 DRAM과 VRAM의 구조 및 액세스 사이클은 큰 변화가 없도록 하고 단지 Z값 비교기를 내장하므로 Z 버퍼 알고리즘을 이용한 3차원 그래픽스 처리시 읽기-수정-쓰기 사이클을 쓰기 사이클로 대치하므로 정규 모드의 경우 약 30%, 페이지 모드에서는 래스터라이저와 프레임 버퍼 사이의 액세스 대역폭 약 50% 정도의 메모리 액세스 속도를 개선할 수 있다.

참 고 문 헌

[1] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hugher, "Computer Graphics Principles and Practice", 2nd Edition, Addison-Wesley, 1990, pp. 855-922

[2] Kurt Akeley, T. Jermoluk, "High-Performance Polygon Rendering", Computer Graphics, August 1988, pp. 229-246

[3] D. Kirt and D. Voorhies, "The Rendering Architecture of the DN10000VS", Computer Graphics, August 1990, pp. 299-307

[4] Kurt Akeley, "The Silicon Graphics 4D/240GTX Superworkstation" CG & A, July 1989, pp. 71-83

[5] B. Apgar, B. Bersack, A. Mammen, "A Display System for the Stellar Graphics Supercomputer Model GS1000", August 1988, pp. 255-262

[6] Kurt Akeley, "Reality Engine Graphics, Computer Graphics", 1993, pp. 109-116

[7] Evans & Sutherland Co., October 1992, "Technical Report"

[8] Samsung Electronics Co., 1994, "CMOS Memory Data Book"

[9] N. Gharachorloo and S. Gupta, E. Hokenek, "Subnanosecond Pixel Rendering with Million Transistor Chip", Computer Graphics, August 1988, pp. 41-49

[10] 최 상길, 최 병균, 문 상호, 위 영철, "고성능 그래픽스 전용 프로세서 개발", 전자공 학회지 Vol. 20, NO. 7, 1993. 7, pp. 115-123

[11] Don Tuite, "Cache Architectures Under Pressure to Match CPU Performance", March 1993, Computer Design, pp. 91-97

문 상 호

1982년 2월 경북대학교 전자공학과 공학사

1984년 2월 경북대학원 전자공학과 공학석사

1994년 3월~현재 경상대학교 전자계산학과 박사과정 재학중

1983년 10월~1986년 10월 삼성전자 시스템 개발실 주임 연구원

1986년 11월~1995년 1월 삼성종합기술원 기반기술연구소 그래픽스팀 선임연구원

1989년 10월~1991년 11월 미국 워싱턴 주립대학 전기 및 전자공학과 교환연구원

1995년 3~현재 영남전문대학 전자계산기과 전임강사

관심분야: 컴퓨터 그래픽스, 컴퓨터 구조, 멀티미디어

강 현 석

1981년 2월 동국대학교 전자계산학과 졸업

1983년 2월 서울대학교 계산통계학과 이학석사(전산학)

1989년 서울대학교 전자계산학과 이학박사(전산학)

1981년~1984년 한국전자통신연구소 연구원

1984년~1992년 전북대학교 전자계산학과 부교수

1992년~현재 경상대학교 컴퓨터과학과 부교수

관심분야: 객체 지향 데이터베이스, 컴퓨터 그래픽스, 멀티미디어

박 길 흠

1982년 2월: 경북대학교 전자공학과 공학사

1984년 2월: 한국과학기술원 전기 및 전자공학과 공학석사

1990년 2월: 한국과학기술원 전기 및 전자공학과 공학박사

1984년 3월~현재 경북대학교 전자전기 공학부 부교수

관심분야: 영상신호처리, 컴퓨터 그래픽스