3차원 그래픽 지오메트리 연산을 위한 벡터 지오메트리 엔진의 설계

김원석° 정철호 이길환 박우찬 한탁돈 이문기*
연세대학교 컴퓨터 과학과 전자공학과*
{wskim, chjeong, kiwh, chan, hantack}@kurene.yonsei.ac.kr, *mklee@spark.yonsei.ac.kr

The Design of VGE(Vector Geometry Engine) for 3D Graphics Geometry Processing

Won-suk Kim° Cheol-ho Jeong Kil-hwan Lee Tack-Don Han Moon-Key Lee*
Dept. of Computer Science, *Dept. of Electrical Engineering Yonsei University

요 약

3차원 그래픽 가속기는 지오메트리 처리(geometry processing)와 레스터라이저(rasterizer)로 구성된다. 본 논문에서는 지오메트리 처리를 고속으로 수행할 수 있는 벡터 형태의 처리 구조(VGE)를 제안하였다. 특히 기존의 부동소수점을 계산할 수 있는 구조에 4개의 FADD, FMUL, 128개의 벡터 레지스터를 추가하여 지오메트리 연산을 가속했으며 VGE와 비슷한 H/W 비용을 갖는 Hitachi의 SH4와 비교했을 때 평균 4.7배의 성능향상을 보였다. 또한 성능 평가를 위해 범용프로세서 시뮬레이터인 Simplescalar를 수정하여 시뮬레이터를 제작했으며 Viewperf Benchmark를 입력으로 사용하였다.

1. 서 론

실감나는 영상을 향한 사용자들의 욕구가 증가함에 따라 처리해야할 3차원 그래픽 데이터가 늘어나고 있다. 그에 따라 애니메이션, 게임, CAD, GIS 등의 각종 어플리케이션에서 요구되는 그래픽 H/W성능이 급격히 증가하였다. 특히 게임분야는 프레임마다 30만개 이상의 정점이 처리되고 있으며 앞으로 점점 늘어날 것으로 예상된다[1].

3차원 그래픽 가속기는 지오메트리 처리(geometry processing)와 레스터라이저(rasterizer)로 구성된다. 1999년대까지, PC에서 사용하는 대부분의 3D 그래픽 가속기는 레스터라이저 단계만 처리했으며 CPU가 지오메트리 처리를 수행하였다.

그러나, 많은 양의 정점들을 처리해야 하는 현재의 어플리케이션 환경에서 CPU 프로세싱 능력만으로 지오메트리 처리를 하기에는 한계에 도달하였다. 또한 H/W 제작 가격의 하락과 지오메트리 엔진의 필요성이 대두되면서 3D 그래픽 가속기에 지오메트리 엔진을 장착하는 것이 필수적으로 되었다. 예를 들어 Geforce 2에 장착한 nVIDIA의 T&L엔진, Radeon에 탑재한 ATI의 Charisma엔진이 있으며, 얼마나 많은 지오메트리 정보를 처리하느냐가 3D 그래픽 가속기의 성능척도가 되고 있다[2].

본 논문에서는 지오메트리 처리를 고속으로 수행할 수 있는 구조를 제안하였다. 이 구조는 기존의 부동소수점을 계산할 수 있는 구조에 4개의 FADD, FMUL, 128개의 벡터레지스터를 이용했으며, 추가된 H/W를 이용하여 벡터 연산을 수행해 지오메트리 연산을 효율적으로 가속할 수 있다. 또한 성능평가를 위해 제안하는 구조와 비슷한 H/W 비용을 갖는 Hitachi의 SH4를 비교 대상으로 하여 시뮬레이션을 수행하였다. 시뮬레이터는 수행구동(execution driven) 시뮬레이터인 Simplescalar의 소스코드를 수정하여 제작했으며, VGE와 SH4의 어셈블리 명령어를 입력으로 파이프라인으로 처리된다. 또한 분석적인 모델을 통해 최적의 성능을 나타내는 VGE의 레지스터 개수와 데이터 메모리 엔트리를 결정하였다.

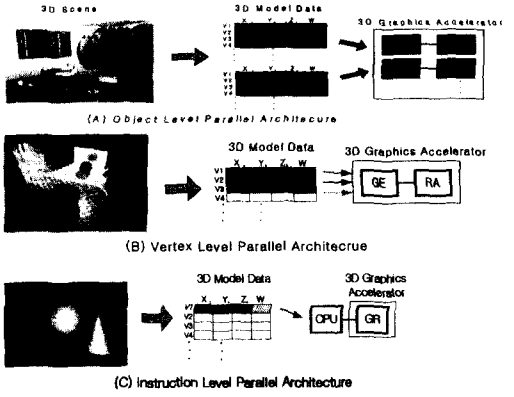
본 논문의 구성은 다음과 같다. 2절에서는 관련연구를 알아보고, 3절에서는 제안하는 VGE 구조와 특징을 살펴본다. 4절에서는 시뮬레이션 환경을 설명하며, 5절에서는 시뮬레이션의 결과 및 평과를 알아보며, 6장에서는 결론을 논한다.

이 논문은 과학기술부에서 지원하는 국가지정연구실 사업과제 (과제번호 2000-N-NL-01-C-133)연구비에 의해 수행됨.

2. 관련연구

지오메트리 처리는 좌표단위의 행렬곱셈과 Lighting 계산이 대부분이다[3]. 그러므로 행렬곱셈과 Lighting 계산을 빠르게 가속하는 것이 중요하다.

일반적으로 그래픽 데이터들은 병렬성이 높기 때문에 어떻게 H/W를 구성하느냐에 따라 효율적인 지오메트리 엔진을 설계할 수 있다. 즉, 물체 단위로 병렬성을 이용하거나(a), 정점 간의 병렬성을 이용하거나(b), 정점을 처리하는 명령어들 간의 병렬성(c)을 이용하는 방법이 있다.



[그림 1] 3D 그래픽 데이터의 병렬성을 이용한 가속방법들

[그림 1]에서 (a)의 경우는 한 장면을 구성하는 물체단위로 가속하므로 고성능이지만 많은 비용이 든다. 대표적으로 Sony의 Emotion Engine, SGI의 Infinite Reality Engine을 들 수 있다[4]. (c)의 경우는 기존 범용프로세서를 확장하여 새로운 명령어를 추가하는 방식이 사용되기 때문에 비용은 저렴하지만 지오메트리 연산을 처리하기에는 성능이 떨어진다. 예를 들어, AMD의 3DNOW!, Intel의 SSE가 대표적이다[5].

(b)의 방식은 물체를 구성하는 정점단위로 정점의 좌표값(x, y, z, w)들을 병렬적으로 계산하므로 (a)와 (c)의 경우를 절충한 방법이다. 특히 지오메트리 계산은 정점을 기준으로 처리되기 때문에 이 방식은 병렬성이 비교적 크며, 저렴한 가격으로 H/W를 구성할 수 있다. 대표적으로 Hitachi의 SH4를 들 수 있다. 기존의 부동소수점 연산기에 Inner-Product H/W를 추가하여 다음의 [식 1]의 연산을 한 개의 명령어로 처리하며 주로 변환연산을 가속한다[6].

m0*x + m4*y + m8*z + m12*w ---- [식 1]

또한 SH4는 32개의 부동소수점 레지스터를 갖고 있으며 4개의 부동소수점 레지스터가 한 개의 벡터 레지스터로 리맵핑(Remapping)되어 [식 1]의 연산자로 활용된다.

3. 제안하는 VGE 구조

이 절에서는 제안하는 VGE 구조의 H/W 구성과 특징을 살펴보겠다. 또한 VGE의 지오메트리 연산을 가속하는 방식을 알아보겠다.

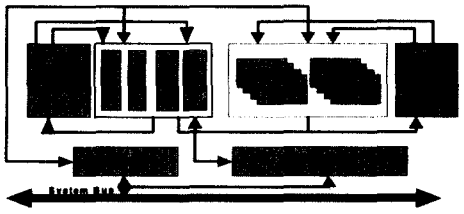
3.1 VGE 구조의 특징

지오메트리 연산에 있어서 대부분을 차지하는 것은 변환연산과 Lighting 연산이다. 그런데 변환연산은 (x, y, z, w)의 성분을 갖는 정점단위로 이루어지고 Lighting 계산은 색의 성분인 RGB단위로 각각 이루어진다. 특히 이 연산들은 정점과 색의 곱셈과 덧셈이 대부분이기 때문에 곱셈과 덧셈을 벡터로 처리했을 때 효율적이다.

그래서 VGE는 정점을 구성하는 네 개의 성분(x, y, z, w)과 색을 구성하는 RGB를 벡터 단위로 처리할 수 있도록 네 개의 부동소수점을 저장하는 128비트 벡터 레지스터를 지원한고, 데이터 버스 폭 또한 128비트로 하였다. 또한 곱셈과 덧셈에 벡터연산을 지원한기 위해 네 개의 FALU와 FMUL을 갖는다.

또한 변환연산에는 Load/Store가 정점단위로 이루어지고 Lighting 연산에는 RGB의 색단위로 이루어진다. 그래서 범용프로세서의 경우 정점과 RGB성분 각각을 Load/Store 해야 한다. 하지만 VGE에서는 벡터레지스터 단위로 Load/Store가 가능하므로 한번의 Load/Store로 처리될 수 있다.

또한 변환연산과 Lighting 연산은 모든 정점에 대해 동일한 연산을 수행하므로 루프(loop)가 많이 발생한다. 그래서 VGE는 128개의 벡터 레지스터를 이용하여 루프를 최대한 언롤(unroll)하여 분기로 인한 지연시간을 최소화하였다.



[그림 2] VGE Block Diagram

3.2 VGE에서 지원하는 그래픽 가속연산

[그림 3]에서 보는 것 처럼 행렬 변환의 식을 범용 프로세서로 처리하기 위해서는 16번의 곱셈과 12번의 덧셈이 필요하다. 하지만 VGE는 [m0, m1, m2, m3]벡터와 x 스칼라 형태의 벡터 곱셈이 네 번 실행되며 이러한 곱셈의 결과를 더하는 벡터 덧셈이 세 번 실행된다.

$$\begin{bmatrix} m0 & m1 & m2 & m3 \\ m4 & m5 & m6 & m7 \\ m8 & m9 & m10 & m11 \\ m12 & m13 & m14 & m15 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

```
vEye[i][0] = m0 * x + m4 * y + m8  * z + m12 * w;
vEye[i][1] = m1 * x + m5 * y + m9  * z + m13 * w;
vEye[i][2] = m2 * x + m6 * y + m10 * z + m14 * w;
vEye[i][3] = m3 * x + m7 * y + m11 * z + m15 * w;
             V1     V5.x     V2     V5.y
```

[그림 3]벡터 연산을 이용한 행렬 변환 연산

그리고 Lighting 계산을 위한 물체의 색을 결정하기 위해서는 물체 고유의 색에 Ambient, Diffuse, Specular 특성을 곱하여 각각을 더해줘야 한다[7]. 또한 이러한 과정을 RGB 성분마다 계산하므로 세 번 반복해야 한다. 하지만 VGE는 RGB성분을 하나의 벡터로 처리하기 때문에 곱셈과 덧셈 연산에 대해서는 3배의 성능향상을 기대할 수 있다.

3.3 메모리 크기변화에 따른 면적증가와 SH4와 면적 비교

지오메트리 엔진의 비용은 H/W 점유 면적에 비례한다. 그런데 VGE는 다른 지오메트리 엔진에 비해서 많은 벡터레지스터를 갖는다. 그래서 메모리 면적만을 놓고 보았을 때 VGE와 비슷한 H/W구성을 갖는 SH4보다 면적이 더 필요하다.

		Width(um)	Height(um)	Area(um^ 2)
VGE	Inst Memory(4KB)	985.54	375.44	2027541.5
	Data Memory(8KB)	1127.29	401.16	
	Register(128x128bit)	2389.68	504.38	
SH4	Inst Memory(8KB)	1127.29	401.16	1523707.6
	Data Memory(16KB)	1525.03	508.64	
	Register(32x32bit)	289.55	1021.56	

[표 1]VGE, SH4의 Memory, Register 구성의 따른 크기

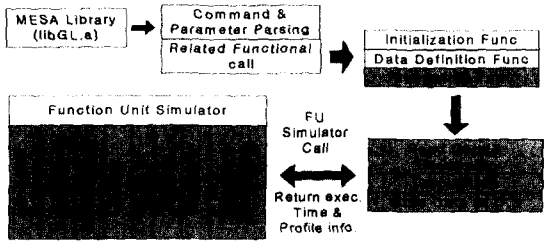
0.18um 삼성 표준 셀 라이브러리(samsung standard cell library)를 참고하여 VGE와 SH4의 레지스터와 메모리 크기를 추정하였다. [표 1]에서 보는 것과 같이 레지스터와 메모리를 합한 면적이 약 30% 증가한다.

4. 시뮬레이션 환경

실험 환경은 대표적인 그래픽 라이브러리인 OpenGL의 기능을 구현한 Mesa 라이브러리의 공개된 코드를 이용하였다. 또한 수행사이클을 계산하기 위해서 지오메트리 연산을 수행하는 부동소수점 연산에 관련된 코드마다 대상 구조의 어셈블리 명령어를 삽입하였다. 그리고 Mesa 라이브러리를 호출할 때마다 시뮬레이터도 같이 호출하는 방식으로 수행사이클 수를 계산하였다.

시뮬레이터는 범용프로세서 시뮬레이터인 Simplescalar의 소스코드를 수정하여 제작했으며, 어셈블리 명령어를 입력으로 파이프라인으로 처리된다. 또한 데이터 의존성과 어셈블리 명령어의 Throughput과 Latency를 고려하여 수행 사이클과 프로파일링 된 정보를 반환한다.

비교 대상은 제안하는 VGE 구조와 비슷한 H/W 비용을 갖는 SH4를 채택하였다. 또한 시뮬레이터 입력으로 SPEC View perf의 벤치 마크(Awadvs-04, DRV-07, DX-06, Light-04,



[그림 4] 전체적인 시뮬레이터 실험환경

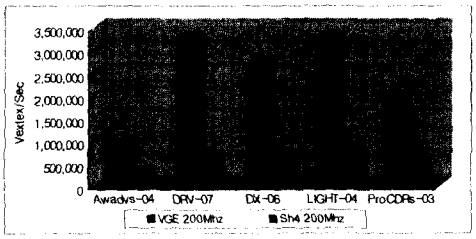
ProCDRs-03)들을 사용하였다[8].

5. 실험결과 및 분석

[그림 5]에서 보는 것과 같이 Lighting계산이 비교적 적은 DRV, DX, LIGHT, ProCDRs에서는 평균 네 배 이상의 성능 향상이 있었다. 그리고 Lighting 계산이 많은 Awadvs에서는 약 세 배정도의 성능 향상이 있었다.

SH4는 벡터레지스터의 8개이므로 루프 언롤링(Loop Unrolling)을 사용할 수 없었다. 하지만 VGE는 128개의 충분한 벡터 레지스터를 이용하여 루프 횟수를 변환연산의 경우 최대 15배 줄일 수 있었다. 즉, 변환할 행렬을 Load 하는데 4개가 쓰이고 변환할 정점을 Load 하는데 15개가 쓰인다. 그리고 한 정점을 행렬변환하는데 7개의 임시레지스터가 필요하므로 최소 124(=15*7+ 19)개의 레지스터가 필요하였다.

또한 벡터 단위의 Load/Store 연산은 데이터의 입출력이 자주 발생하는 변환연산에서 SH4보다 더 많은 성능향상이 있었다. Lighting 연산에도 SH4의 Inner-Product 연산보다는 벡터 단위의 덧셈과 곱셈이 빈번하므로 VGE가 더 효율적이었다.



[그림 5] SH4와 VGE의 Vertice/Sec 비교(200Mhz)

5.1 VGE의 메모리 계층구조

[식 2]와 같은 분석적인 모델을 통해 n개의 정점을 변환할 때 걸리는 실행시간을 예측하였다. 또한 데이터 메모리와 메인 메모리를 접근하는 시간은 삼성의 표준 셀 라이브러리를 참고했으며 레지스터 개수(32~512)와 메모리 엔트리(256~32768)를 변화시켜 [그림 6]과 같은 결과를 얻었다.

For n vertices,

$$Calc.time = \frac{M}{R}(4 \times R + 5) \times Clock\ cycle\ time$$

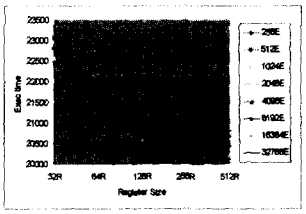
$$Exec.time = Calc.time + (M \times D.Macc\_time) + (\frac{M}{D} \times M.Macc\_time)$$

, where # of Input / output · register Size R, # of data memory entry D, # of memory reference M ($\approx 2n$)

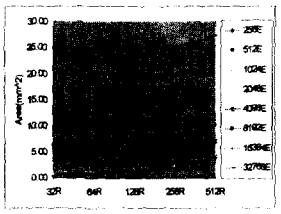
[식 2] n개의 정점을 변환하는데 걸리는 시간

계산 결과, [그림 6]에서 보는 것처럼 레지스터의 개수는 많을수록(512), 데이터 메모리 엔트리는 512일 때 성능이 가장 좋았다. 레지스터 개수는 많을수록 한 번에 많은 데이터를 Load 할 수 있으므로 성능이 좋아진다. 그러나 데이터 메모리 엔트리의 경우, 8KB(512 엔트리)보다 커지면, 한 번에 많은 데이터를 메인 메모리로부터 가져올 수 있으므로 메인 메모리 접근횟수는 줄어들지만, 상대적으로 데이터 메모리를 접근하는 시간(D.Macc_time)이 커지므로 오히려 성능은 떨어진다.

또한, 레지스터의 개수가 커지면 [그림 7]에서 보는 것처럼 면적이 늘어난다. 512 엔트리의 경우, 128개의 레지스터를 갖는 모델과 비교했을 때 면적은 2배 이상 커지지만 실행시간은 2% 밖에 좋아지지 않으므로 비용을 고려한다면 128개의 레지스터인 경우가 가장 최적의 선택이다.



[그림 6]레지스터 개수와 메모리 크기에 따른 실행시간변화



[그림 7] 레지스터 개수와 메모리 크기에 따른 면적의 변화

6. 결론

VGE는 4개의 FADD, FMUL, 128개의 128비트 레지스터로 구성되며, 부동소수점 덧셈과 곱셈의 벡터연산, 벡터 단위의 Load/Store, 루프 언롤링을 이용하여 지오메트리 연산을 가속할 수 있다.

비용을 고려할 때, VGE는 SH4와 비교해서 데이터 메모리와 레지스터를 합친 면적에 30%가 늘어났다 그러나 Viewperf 벤치마크 시뮬레이션 결과 SH4와 비교하여 평균 4.7배까지 성능 향상을 보였다. VGE는 저가형의 가격대비 성능이 뛰어난 지오메트리 구조이다.

참고 문헌

[1]"Balancing Bottlenecks: The Partnership Between the CPU and GPU", nVIDIA Geforce 2 GTS/Pro Technical Briefs

[2]George Lawton, " The Wild World of 3D Graphics Chips," Computer, Vol. 33, No.9, pp. 12~16, Sep. 2000

[3]C. Yang, B. Sano, and A. R.Lebeck, "Exploiting instruction level parallelism in Geometry processing for three dimensional graphics applications," Micro International Symposium on Micro-Architecture, 1998

[4]M.Oka, and M.Suzuoki, "Designing and Programming the Emotion Engine", Micro, Nov. 1999

[5]Stuart.Oberman, Greg Favor, and Fred Weber," AMD 3DNow! Technology: Architecture and Implementation", IEEE Micro, mar-apr 1999,pp 37~48

[6]Fumio Arakawa and Osamu Nishii, "SH4 RISC Multimedia Microprocessor", IEEE MICRO, pp 26~34, 1998

[7]Neider, Mason and Tom Davis, "OpenGL Programming Guide", pp 173~211, 1997

[8] SPECviewperf™ 6.1.2
http://www.specbench.org/gpc/opc.static/opcview.htm