3차원 그래픽을 위한 Geometry 프로세서의 설계

정 철 호[†] · 박 우 찬[††] · 김 신 덕[†††] · 한 탁 돈[††††]

요 약

본 논문에서는 3차원 그래픽의 처리 과정 중 부동 소수점 연산이 많이 소요되는 geometry 프로세싱 처리 방법과 계산량을 단계별로 분석하였다. 그리고, 그래픽 프로세싱의 수행 특성을 추출하여, 이에 맞는 기능 유닛을 설계하고, 데이터 처리 방안과 제안하는 geometry 프로세서의 구조를 설명한 다음, 성능을 분석하였다. 제안하는 geometry 프로세서는 부동 소수점 덧셈, 곱셈, 나눗셈 연산을 동시에 수행 가능하며, geometry 프로세싱 전 단계를 수행하는데 23.5%의 성능 향상이 있었다. 그리고, 나눗셈/제곱근 연산을 위해서 면적대 성능비가 우수한 SRT 나눗셈 연산기를 추가하여 곱셈 연산기를 이용하는 연산기보다 약 23%의 성능 향상을 이루었다.

The Design of Geometry Processor for 3D Graphics

Cheol-Ho Jeong[†] · Woo-Chan Park[††] · Shin-Dug Kim[†††] · Tack-Don Han[††††]

ABSTRACT

In this thesis, the analysis of data processing method and the amount of computation in the whole geometry processing is conducted step by step. Floating-point ALU design is based on the characteristics of geometry processing operation. The performance of the devised ALU fitting with the geometry processing operation is analyzed by simulation after the description of the proposed ALU and geometry processor. The ALU designed in the paper can perform three types of floating-point operation simultaneously-addition/subtraction, multiplication, division. As a result, the 23.5% of improvement is achieved by that floating-point ALU for the whole geometry processing and in the floating-point division and square root operation, there is another 23% of performance gain with adding area-performance efficient SRT divisor.

1. 서 론

3차원 그래픽은 공학용 CAD/CAM, 3차원 애니메이션, 게임 등 그 이용 분야가 멀티미디어의 도입과 함께 크게 성장해 왔다. 특히 게임분야에 있어서 사용자들의 현실감에 대한 꾸준한 요구로 게임기의 성능이 일반적인 PC의 성능을 넘어서고 있다. 이러한 3차원 그래픽의 성장에 힘입어 3차원 그래픽 가속기의 대중화를 앞당기고 있다[1]. 3차원 그래픽 프로세싱의 내재된 특성상 엄청난 양의 계산을 수행하며, 그 데이터형 또한 다양하기 때문에 특별히 제작된 프로세서나 고속의 범용 프로세서를 이에 맞게 확장한 프로세서들이 등장하기에 이르렀다[2]. 현재, 3차원 그래픽 프로세싱을 가속하는 기법으로 앞의 두 가지 프로세서들을 사용하는 방법이 사용되고 있다. 이런 프로세서들의 등장으로 고성능 웍스테이션에서만 가능했던 3차원 그래픽이 PC수준에서도 가능하게 되었다.

※ 본 연구는 1997년도 한국학술진흥재단 대학부설연구소과제 연구비에 의하여 진행되었음.
† 정 회 원 : 연세대학교 대학원 컴퓨터과학과
†† 준 회 원 : 연세대학교 대학원 컴퓨터과학과
††† 정 회 원 : 연세대학교 컴퓨터과학과 교수
†††† 종신회원 : 연세대학교 컴퓨터과학과 교수
논문접수 : 1999년 8월 9일, 심사완료 : 1999년 11월 4일

일반적으로 3차원 그래픽처리는 크게 geometry프로세싱과 rasterization으로 나눌 수 있다. geometry 프로세싱 이전의 단계는 일반적으로 호스트에 의해 처리되어지며, 그 이후 단계를 그래픽 가속기가 담당하게 된다. geometry프로세서는 모델링된 데이터의 형(type)이 실수이기 때문에 부동 소수점 연산을 하게 되며, rasterizer는 화면의 해상도와 관련된 연산을 수행하므로 주로 정수 연산이 이용된다[3]. 일반적인 산술연산은 부동 소수점 연산과 정수 연산으로 나뉘는데 부동 소수점 연산이 정수 연산에 비해 세 배 이상의 계산량을 필요로 하기 때문에 부동 소수점 연산을 빠르게 처리할 수 있는 연산기의 설계가 필수적이다. 또한, 일반적으로 그래픽 데이터들은 병렬성이 높기 때문에 다수의 연산기를 이용한 프로세서를 구성하여 그래픽 가속기를 설계한다. 그러므로, 효율적인 geometry프로세서를 설계하는 방법은 그래픽 데이터에 내재된 병렬성을 이용할 수 있는 프로세서의 구조와 명령어의 병렬성을 이용할 수 있는 연산기를 설계하는 것이다.

본 논문에서는 geometry 프로세싱의 연산에 적합한 연산기와 그 구성에 중점을 두었으며, 효율적인 geometry 프로세서 구조와 성능 향상을 위한 개선안을 제시하였다. 먼저, 3차원 그래픽 처리과정 중 geometry 프로세싱 단계를 중심으로 각 단계별 처리 과정을 규정하고, 각 단계별로 연산의 특성과 계산량을 분석하였다. 분석된 데이터를 바탕으로 부동 소수점 연산기의 설계와 각 연산기의 구성 방법을 제시하고, 이에 적합한 프로세서의 구조와 데이터의 처리 모델을 정립하여 성능 향상을 위한 제안을 하였다.

이 논문은 제2장에서 3차원 그래픽 랜더링의 기본 과정과 geometry 프로세싱의 단계를 기술하고, 각 단계의 처리 방법과 계산량을 분석하였고, 제3장에서 관련 연구를, 제4장에서 설계 대안과 제안하는 geometry 프로세서의 구조를 기술하였다. 그리고, 제5장에서 성능 분석에 대해 기술하였고, 마지막으로 제6장에서 결론을 제시하였다.

2. 3차원그래픽의 랜더링

이 장에서는 3차원 그래픽 랜더링의 기본 과정과 배경 지식을 설명한다. 그리고, 랜더링 과정 중 geometry프로세싱의 단계를 구분하여 각 단계의 데이터 처리 방법과 연산량을 규정하였다.

2.1 그래픽Primitive, 색, 과 빛

그래픽primitive는 한 장면의 물체들을 구성하는 요소로써, 점, 선과 다각형이 해당되며, 모든 그래픽 데이터는 이들의 집합으로써 이루어진다. 예를 들어 곡선은 일련의 직선들로 이루어지며, 복잡한 다각형들은 삼각형의 집합으로 나누어 표현되어진다. 일반적으로 3차원 이미지의 구성에서, 복잡하고 정교한 장면은 아주 많은 수의 그래픽 primitive들로 이루어지며, 한 장면이 실제와 가까우면 가까울수록 더 많은 primitive들로 이루어진다. 또한, 표면을 모델링한 데이터들은 대개 일련의 메쉬 형태로 표현되는데, 이들을 삼각형으로 나누게 되면 각 모서리 부분에서는 6개의 삼각형이 한 모서리를 공유하게 된다. 이런 삼각형 각각을 계산하게 되면 공유되는 모서리 부분에서 중복 계산된다. 이런 중복 계산을 줄이기 위해 인접한 삼각형들의 일련의 줄 모양으로 줄여 나타내는 것을 트라이앵글 스트립(triangle strip)이라 한다. 이외에도 삼각형 대신 사각형을 이용해 복잡한 표면을 표현하는 쿼드메쉬(quad mesh)가 이용된다[5].

물체의 색은 빨강색(R), 녹색(G), 파란색(B)이 세가지 요소로 결정되는데, 비트맵에 R, G, B값이 직접 저장되어 있는 시스템을 트루 컬러(true color) 시스템이라 하고, 색의 인덱스 값이 저장되어 있는 시스템을 의사 컬러(pseudo color)시스템이라 한다. 의사 컬러 시스템은 실제 색을 표시할 때, 색 변환표(color lookup table)에 있는 정보를 이용해 변환하는 처리를 필요로 한다. 이는 색깔 값을 저장하는데 필요한 많은 양의 메모리를 줄일 수 있는 이점이 있다. 또, 색깔의 정보와 함께 물체의 상대적인 위치 즉 깊이를 나타내는 정보인 깊이정보(depth value : Z value)도 포함하여 표현한다.

물체의 밝기는 물체 표면에 도달하는 주요한 광원, 물체의 반사도, 물체의 색깔에 따라 달라진다. 또한, 광원의 세기는 물체와의 거리에 따라 강도가 결정된다. 그래서, 광원의 특성 모델이 이러한 요소들을 고려하여 실제와 거의 완전히 같아야 실제와 같은 질감을 표현할 수 있다. 실례로 OpenGL에 사용된 광원의 모델은 emissive, ambient, diffuse, specular의 4가지 요소로 나뉘어 구성되며, 각 요소들이 독립적으로 계산된 후 전체의 합으로 한 정점의 색깔을 결정한다[6]. Ambient 요인은 모든 주변 환경으로 퍼져 나가는 광원을 말하며, 광선이 물체에 부딪치면 모든 방향으로 골고루 퍼져가는 특성을 가진 광원의 모델이다. Diffuse 요인은 한 방향으로 유입되는 광원으

로, 빛의 위치와 표면의 반사각에 의해 빛의 세기가 결정되는 광원의 모델이다. 즉, 유입된 빛이 표면에 부딪쳐 직각으로 반사되는 부분은 하일라이트되어 반짝이는 효과가 있는 곳이고, 이 부분에서 멀어질수록 빛의 세기가 약해진다. Specular 요인은 어느 특정한 방향에서 유입되어 특정한 방향으로 흩어지는 광원의 모델을 말하며, 물체의 반사율에 의해 빛의 세기가 결정된다. 즉, 반사율이 100%인 곳은 거울과 같은 효과를 만들어 낼 수 있다.

한 물체의 색깔은 물체의 반사율, 빛의 특성에 의해서 결정된다. 즉, 유입되는 빛과 표면에서 반사되는 백분율에 의해 결정된다. 빛의 특성과 같이, 물체에도 반사율을 결정하는 ambient, diffuse, specular 특성이 모델링되어 표현된다. 그리고, ambient, diffuse, specular 반사율은 유입되는 빛의 요인들과 함께 계산된다. 또, 여기에 물체 고유의 특성을 고려한 emissive color도 포함되기도 한다.

2.2 Geometry 프로세싱의 전체 과정

Geometry 프로세싱은 호스트에 의해 처리된 모델 데이터의 정보들을 입력 받음으로써 시작된다. 모델 데이터는 한 정점의 정보, 각 정점의 법선 벡터, 색깔 정보와 각 단계에서 요구되는 파라메터의 정보들이 포함된다. Geometry 프로세싱을 다음과 같이 7단계로 세분할 수 있다.[4]

1) 정점 변환(Vertex transformation)
2) 법선 벡터 변환(Normal vector transformation)
3) 광원 계산(Light calculation)
4) 클리핑(Clipping)
5) 투영(Projection)
6) Viewport 변환(Viewport transformation)
7) Edge-slope 계산(Edge slope calculation)

1)~6)단계의 데이터 처리는 한 정점에 대한 부동소수점 연산이며, 7)단계는 그래픽 primitive에 대한 처리이므로, 각 primitive의 조합에 대한 정보를 포함하고 있어야 한다.

2.3절에서 각 단계별 수행연산의 특성을 기술하였다. 각 단계의 계산량은 덧셈, 뺄셈, 비교 연산은 수행 사이클이 같으므로 전부를 덧셈으로 취급하였고, 제곱근, 곱셈, 나눗셈의 수를 나누어 <표 1>에 제시하였다

〈표 1〉 Geometry 프로세싱 전 단계의 계산량

Geometry 프로세싱 단계	계 산 량
정점 변환	12 mul + 12 add
법선 벡터 변환	9 mul + 6 add
광원 계산	3×(2 sqrt + 3 div + 28 mul + 24 add)
클리핑(Clipping)	6×(1 add)
투 영	16 mul + 12 add
Viewport 변환	1 div + 19 mul + 12 add
Edge slope 계산	5×(1 div + 2 add)

정점 변환 단계는 모델 데이터에 대한 좌표계 변환 단계로, 전환(translation), 스케일링(scaling), 회전(rotation) 변환이 수행된다. 또, 그래픽 primitive들의 좌표로써 x, y, z값을 사용하는데, homogeneous좌표계 변환을 위해서 스케일 벡터(scale vector : w)를 추가하여 사용한다. 이 변환에는 4×4행렬의 곱셈을 수행하며, 각 서브 오퍼레이션마다 다른 행렬을 곱하여야 한다. 전체 계산량은 16 곱셈과 12 덧셈이나, 그래픽 primitive의 좌표값에 포함된 w값은 대개 1이므로, 4번의 곱셈을 줄일 수 있다. 법선 벡터 변환 단계는 한 정점의 법선 벡터를 변환하는 단계이다. 법선 벡터는 한 정점에 직각인 단위 벡터에 대한 정보로써 광원 계산에 이용되며, 3×3행렬 곱셈으로 광원의 법선 벡터를 변환한다. 계산량은 9 곱셈과 6 덧셈이며, 만약 단위 벡터가 아니면, 단위 벡터로 바꾸는 연산이 더 필요하다. 즉, 법선의 길이(n)로 각 x, y, z값을 나누어 주어야 한다. 이때, 1/n을 계산하여 각각에 곱하면, 나눗셈 연산이 곱셈연산에 비해 수행 사이클이 현저히 길기 때문에 전체의 수행 사이클을 줄일 수 있다.

광원 계산 단계는 한 정점의 색을 결정하는 단계이다. OpenGL의 기본적인 랜더링 알고리즘으로 사용되는 Gouraud Shading은 광원 계산을 하는데 빛의 방향, 법선 벡터, 시점, 다각형의 한 정점의 값이 필요하며, 반사방향과 범위만큼이 하일라이트 되는 것을 나타낸다[5]. 한 정점의 색 R, G, B에 대하여, 물체의 emission, 전역 ambient 광원과 물체의 ambient 요인의 곱, 감소요인이 포함된 모든 광원에 대한 ambient, diffuse, specular 요인의 합을 계산한다. 즉,

- 한 정점의 색 = 한 정점의 material emission + 한 정점의 전역 ambient 광원 × 물체의 ambient 특성 + 빛의 감소요인이 고려된 모든 광원의 ambient, diffuse, specular 요인

광원 계산을 위해 위와 같은 연산을 각각 R, G, B값에 대하여 연산을 세번 반복해야 한다. 클리핑(clipping) 단계는 전체 화면의 크기에 따라 보여지는 부분과 보이지 않는 부분을 구분하여 계산되는 데이터의 양을 줄이는 방법이다. 즉, 사용자의 시점에 의한 view-volume 클리핑과 프로그램에 따라 추가적인 클리핑을 하는 것과 물체의 뒷면을 제거하는 뒷면 컬링(back-face culling)이 포함될 수 있다. 다각형 클리핑 방법으로는 Sutherland-Hodgman 알고리즘을 이용하며, 각 클리핑면에 대하여 한번의 부동 소수점 비교연산이 필요하다. 많은 연산이 필요한 곳은 실제 클리핑이 이루어지는 부분이며, 빈번히 발생되는 것이 아니다. 투영 단계는 물체에 원근감을 주기 위해 원근투영을 사용하며, CAD/CAM도면과 같은 효과를 위해 직교투영을 사용한다. 이런 투영은 4×4 행렬의 곱셈으로 수행할 수 있다. Viewport 변환단계는 투영된 모델 데이터를 실제 화면상에 나타내기 위해 각 x, y, z값을 w값으로 나누어 정규화한 후 윈도우 좌표계로 변환을 한다. 역시 1/w를 먼저 계산한 후에 각각에 곱셈을 하는 방법으로 계산시간을 단축 시킬 수 있으며, 4×4 행렬의 곱셈으로 좌표계 변환을 한다. edge-slope 계산단계는 스캔변환에 필요한 한 모서리의 기울기 값과 각 요소의 증가량을 그래픽 primitive에 대해, x, z, R, G, B에 대하여 계산한다[5].

2.3 Geometry 연산의 특성

Geometry 프로세싱 단계 중, 가장 시간이 많이 걸리는 단계는 광원 계산과 edge-slope 계산이다. 광원 계산은 광원의 위치, 시점, 정점에 대한 단위 벡터를 계산하는 부분이 계산량이 많고, edge-slope계산은 기울기를 계산하는데 필요한 나눗셈 연산의 지연시간 때문에 계산시간이 많이 걸리는 것을 알 수 있다. 또, viewport 변환 단계에서는 w 값으로 각 정점의 x, y, z 값을 나누는 연산이 필요로 한다.

Geometry 프로세싱에 사용된 주요 연산으로는 덧셈, 곱셈, 나눗셈, 제곱근이 사용된다. 곱셈과 덧셈이 다른 연산에 비해 빈도수가 현저히 높으며, 덧셈이나 곱셈을 한 결과에 나눗셈과 제곱근 연산이 수행되는 특성이 있다. 서로 독립적인 오퍼랜드에 대한 곱셈과 덧셈이 균형적인 수로 수행되며, 이 계산 결과에 나눗셈이나, 제곱근이 취해지므로, 덧셈과 곱셈연산의 병렬성을 최대한으로 활용함으로써 성능향상을 기대할 수 있다. 그러나 광원 계산은 단위 벡터 계산에 필요한 제곱근 연산과 나눗셈 연산의 데이터 의존성(data dependency)이 크고, viewport변환 단계에서도 먼저 나눗셈 연산이 모든 정점에 대해서 이루어지고 난 후 시작되기 때문에, 고속의 나눗셈/제곱근기를 이용하는 것이 더 효과적이다. 또, 부동 소수점 연산시 덧셈/뺄셈, 비교, 곱셈연산은 같은 수행 사이클이 소요되지만, 나눗셈과 제곱근 연산은 구현 알고리즘의 특성상 반복연산을 하기 때문에 연산의 주요한 병목지점이 된다.

3. 관련 연구

앞 장에서 살펴본 바와 같이 geometry 프로세싱의 가속은 부동 소수점 연산의 처리방법과 프로세서의 데이터 처리방법에 달려있다. 먼저 모델 데이터의 처리방법에 따라 파이프라인, SIMD, MIMD, Hybrid 등으로 나눌 수 있다. 본 장에서는 각 구조의 모델을 설명하고, geometry프로세서의 구조를 중심으로 기술하였고, 각 구조의 장점과 단점을 나열하였다.

3.1 파이프라인 구조

3D Lab.에서 발표한 GLINT Gamma 구조가 파이프라인 구조를 이용한 것이다[7]. 즉, geometry 프로세싱은 일련의 단계를 거쳐야하므로, 각 단계별로 기능 유닛을 두어 파이프 라인을 형성시켜 가속하는 방법이다. 일반적인 처리에 맞는 기능 블록을 설계하여, 이를 각 단계마다 사용하여 설계 비용을 줄일 수 있다. 이 구조는 단순히 각 단계에 맞는 연산들에 따라 기능 유닛을 구성한 것이기 때문에 연산기와 레지스터 이외의 부가적인 유닛을(sequencer, micro-code store 등) 필요치 않는다. 또, 각 기능 유닛에 데이터를 분배하는 과정이 필요치 않으며, 프로세서의 제어가 다른 구조에 비해 훨씬 간단하다.

반면에, 파이프라인 구조는 시간이 가장 많이 걸리는 단계에 의해 전체 성능이 좌우되며, 앞장에서 본 바와 같이 각 단계들의 연산양이 균형을 유지하지 못하기 때문에 전체 성능에 제한이 있게 된다. 또, 새로운 처리 알고리즘의 출현으로, 기능 유닛을 추가 하려면 새로운 코드의 작성, 최적화, 이전 파이프라인 단계와의 조절 등의 제한이 따르게 된다.

3.2 SIMD 구조

SIMD형태의 대표적인 가속기로 SGI사에서 개발된 Indigo 2 EXTREME의 구조이다[8]. 여러 개의 geometry

프로세서를 두어 일정 시간동안 전체가 같은 명령어로 각각 다른 자신의 그래픽 primitive를 처리하는 방법이며, 계산 시간을 줄이기 보다는 데이터의 병렬성을 이용하여 시스템의 throughput을 높이는 방법이다. 전체의 프로세서가 같은 동작을 하기 때문에 구현이 복잡하지 않으며, 부가적인 장치들을 각 프로세서들이 공유할 수 있다. 또한 그래픽 데이터의 특성상, 풍부한 병렬성 때문에 프로세서를 더 추가할수록, 선형적인 성능의 향상을 기대할 수 있다. 그러나, 각 프로세서에 데이터를 분배하는 과정이 필요하며, 분배되는 데이터의 종류가 같은 형태가 아니면, 프로세서들이 각각 다른 종류의 데이터들을 처리하는 메커니즘이 없기 때문에 한 개의 프로세서를 사용할 때 보다 성능이 더 저하된다.

이 프로세서는 멀티포트 레지스터파일의 점유면적 증가와 복잡한 버스중재에 의한 복잡도 증가를 초래하였으며, 별도의 나눗셈 연산기를 이용하지 않아 나눗셈 연산이 필요한 단계에서는 곱셈 연산기를 이용한 나눗셈 연산을 하기 때문에 나눗셈 연산을 하는 동안에 다른 곱셈 명령을 수행할 수 없다.

3.3 MIMD 구조

MIMD 구조는 SIMD 방법과 매우 유사한 구조를 지내고 SIMD구조보다 더 많은 작업 메모리와 연산기를 가지고 있어 한 단계의 연산을 수행하고 있는 도중에도 다른 단계의 연산을 할 수 있다. 즉, 한 정점을 단위로 하는 처리(per-vertex 오퍼레이션)와 그래픽 primitive를 단위로 하는 처리(edge-slope계산) 등을 독립적으로 수행할 수 있다. 즉, 별도의 연산기를 이용해 viewport 단계에서 정규화하는데 필요한 나눗셈과 정점 단위의 연산을 동시에 할 수 있는 것이다. 또, 레지스터 파일 포트의 증가에 따른 면적과 복잡도 증가를 줄이기 위해 레지스터 파일을 네 부분으로 나누어 사용하며, 레지스터의 수도 충분히 크게 하였다. 또한 부동 소수점 덧셈과 곱셈 연산기를 별도로 두었으며, 곱셈의 결과 바로 덧셈 연산기의 입력이 되게 하여 행렬 곱셈을 빠르게 할 수 있다. 나눗셈 연산 중 특정한 값, Y로 각 요소들을 나누는 과정이 많으므로 X/Y를 매번 계산하지 않고, 1/Y를 계산하는 방법을 이용하기 위해 곱셈 연산기를 이용하는 나눗셈 연산을 하는 방법인 Newton-Raphson 알고리즘을 이용하여 구현하였다.

MIMD구조 역시 SIMD와 마찬가지로 데이터를 분배하는 과정이 필요하며, SIMD와는 달리 분배된 데이터가 각각 다른 형태의 것이라도 성능의 저하를 발생시키지 않으며, 프로세서의 수에 따라 선형적인 성능 향상을 기대할 수 있다. 하지만, 부가적인 유닛들의 중복과 각 프로세서들의 데이터 처리에 따른 조정과 프로세서의 사용율을 높이기 위한 관리 등의 구현부담(implementation overhead)이 따르게 된다. 그리고, 각 프로세서의 설계에 드는 비용의 증가로 가속기 전체 비용의 증가를 초래한다.

3.4 Hybrid 구조

Hybrid구조로써 SGI사에서 가장 최근에 개발된 그래픽 가속기에 사용된 구조로써, SIMD나 MIMD구조와 큰 차이는 보이지 않으나, 한 프로세서 내에 다수의 기능 유닛을 두어 각 모델데이터의 처리를 프로세서 내에서 하고, 각 프로세서들은 MIMD방식으로 동작하는 구조를 지니고 있다. 그래픽 primitive중 가장 큰 단위가 삼각형이고, rasterizer의 처리 단위 역시 삼각형인 점에 착안하여, 각 프로세서들이 각각의 정점들을 처리하여 rasterizer에 보내기 위해 다시 조합하지 않고 한 프로세서 내에서 삼각형을 전부 처리하여 rasterizer로 보내는 SIMD형태의 단일 프로세서를 MIMD형태의 보드로 구성한 것이다[11]. 이는 MIMD구조의 장점을 그대로 수용하면서 rasterizer에서 처리하는 단위로 재구성하는 단계를 줄일 수 있는 이점이 있다. 또, 한 프로세서가 정점 단위로 처리하지 않고 그래픽 primitive 단위로 처리하기 때문에 시스템의 throughput도 함께 향상된다.

한 프로세서내에 여러 개의 기능 유닛이 포함되어야 하기 때문에 설계 비용과 복잡도가 증가하게 되어 전체 비용이 증가 하게 된다. SIMD나 MIMD구조에 비해 복잡하지는 않지만 역시 데이터를 분배하는 과정이 필요하다. 또, 각 그래픽 primitive에 대한 모든 정보를 필요로하므로, 이전의 구조에서 보다 상대적으로 많은 작업 메모리를 필요로 하게 된다.

4. 연산기 설계와 제안하는 Geometry 프로세서의 구조

앞 장에서 살펴본 바와 같이, 데이터의 병렬성을 최대한으로 이용할 수 있는 구조를 채택하고, 오퍼랜드의 병렬성 이용과 부동 소수점 연산을 빠르게 수행할

수 있는 연산기를 geometry 프로세싱 연산에 맞게 구성하는 것이 프로세서 설계의 목표이다. 이 장에서는 각 단계별로 분석된 geometry 연산의 특성에 적합한 연산기를 선택 장/단점을 비교하였고, 이러한 연산기로 단일 기능 유닛을 구성하고 다양한 구성 방법을 적용해 최적의 단일 기능 유닛을 제안한다. 그리고, 이 단일 기능 유닛을 이용해 geometry 프로세서의 구조를 제안하고 그 특징을 설명한다.

4.1 기존의 고성능 부동 소수점 연산 알고리즘

이 절에서는 제안하는 geometry 프로세서에 채택될 기존의 고성능 부동 소수점 덧셈/뺄셈, 곱셈, 나눗셈 연산 알고리즘에 대해서 기술하였다.

일반적으로 부동 소수점 덧셈/뺄셈 연산은 지수정렬, 분수부 덧셈/뺄셈 연산, 정규화, 반올림의 4단계로 이루어진다. 이러한 구조의 연산기에서는 반올림 처리를 위해 별도의 증가기나 덧셈 연산기를 사용하거나, 피드백(feedback)을 통해 덧셈/뺄셈에 사용되는 가산기를 중복 이용한다. 그러면, 전자의 경우에는 면적이 커지고 후자의 경우는 지연시간이 길어지는 단점이 있다. 그리고, 반올림 단계에서는 반올림할 때 발생하는 오버플로우(overflow)로 인한 재정규화가 발생하게 된다. 이러한 점을 극복하기 위해서 분수부 덧셈과 반올림을 병렬로 수행하는 하드웨어 모델을 사용하였다. 이 연산기는 반올림이 분수부 덧셈/뺄셈과 병렬로 수행되기 때문에 기존의 부동 소수점 덧셈/뺄셈 연산기에서 반올림 단계에 사용되는 덧셈 연산기와 재정규화의 처리를 위한 별도의 하드웨어가 필요 없다. 그리고, 부동 소수점 덧셈/뺄셈 연산을 지수정렬, 분수부 덧셈/뺄셈, 정규화의 세 단계로 수행하여 덧셈/뺄셈 연산기의 지연시간을 줄였다[16].

부동 소수점 곱셈 연산은 분수부의 곱셈, 곱셈 과정에서 생성된 합과 올림수의 덧셈, 반올림, 정규화의 네 단계로 이루어진다. 이 역시 반올림 처리를 위한 고속의 증가기나 덧셈 연산기를 필요로 함으로, 점유면적과 처리 시간이 길어진다. 덧셈/뺄셈 연산기와 마찬가지로 반올림과 덧셈을 병렬로 수행하여 전체 처리 단계를 세단계로 줄여, 반올림 처리를 위한 덧셈 연산기를 제거하였다. 이로 인하여 곱셈 연산기의 전체 점유면적과 지연시간을 줄여 성능을 높였다. 곱셈 연산기는 booth 알고리즘과 부분 합을 계산하는 wallace 트리를 이용해 분수부의 곱셈을 하는 부분과 덧셈과 반올림을 동시에 하는 부분으로 나누어 구현된다. 즉, 캐리 셀렉트 덧셈 연산기(carry select adder)를 이용하여 덧셈의 결과를 구한 후, 반올림 처리에 의해 이 두개의 값 중에서 반올림이 된 값을 선택하여 재정규화를 거친 다음에 곱셈의 결과 값을 구한다[17].

부동 소수점 나눗셈 연산기는 앞 절에서 설명한 것처럼 SRT 나눗셈 알고리즘을 이용하며, 곱셈 연산기와 같은 반올림 처리 및 재정규화를 위한 유닛을 가진다. 나눗셈 연산은 정수 나눗셈 연산, 몫변환, 반올림, 정규화의 단계를 거치나 곱셈 연산기의 반올림 유닛을 이용하여 몫변환과 반올림 처리를 동시에 수행한다. 대부분의 마이크로 프로세서는 분수부의 나눗셈처리를 위해 SRT 알고리즘을 이용하여 결과로 몫과 나머지를 중복된 형태로 생성된다. 즉, 몫은 양수부분과 음수부분으로 생성되며, 나머지가 합과 올림수의 형태로 생성된다. 몫변환에서는 중복된 형태를 갖는 몫을 일반적인 형태로 바꿔야 하는데, 이때 고성능 덧셈 연산기가 이용된다. 그리고, 반올림 처리는 몫변환에서 발생된 몫을 이용하여 반올림 연산을 수행하므로, 약간의 추가적인 로직으로 몫변환과 반올림을 동시에 수행이 가능하다[18].

고속 SRT나눗셈 연산기를 구현하는 방법으로 radix를 높이는 방법, 시스템 클럭을 나누어 사용하여 한 클럭에 두 번의 반복연산을 하는 방법, 시스템 클럭과 독립적으로 클럭을 사용하는 self-timed 나눗셈 연산기를 구현하는 방법이 있다[15]. 그러나, 나눗셈 연산기의 radix를 높일수록 몫 선택표(quotient selection table)의 크기가 급격히 증가하므로 radix-8 이상은 대개 구현에 이용되지 않으며, radix-4 나눗셈 연산기를 중복시켜 radix-16 나눗셈 연산기를 구현하는 것이 일반적이다. 그리고, 나눗셈 연산기의 최장경로(critical path)가 다른 연산기에 비해 현저히 짧고, 구현하기 쉽기 때문에 1/2클럭을 사용하는 방법을 사용한다.

4.2 Geometry 프로세싱을 위한 부동 소수점 연산기의 구성에 대한 고려

Geometry연산을 살펴보면, 먼저 곱셈을 한 후 그 결과에 다른 오퍼랜드를 더하는 연산이 주류이다. 가장 먼저 고려될 수 있는 연산기가 MAC이며, 또 여러 개의 MAC연산기를 이용하여 행렬의 곱셈과 벡터의 dot product 계산을 가속시킬 수 있다.

각 단계 중 MAC연산을 사용할 수 있는 부분이 많

이 있지만, 그 외의 뺄셈, 비교, 나눗셈 연산을 위한 연산기가 별도로 요구된다. 그래서, MAC연산기와 함께 다른 많은 연산기들의 사용은 연산기의 사용율을 저하시키고 설계비용과 프로세서의 비용을 증가시키는 원인이 된다. 또, 부동 소수점 MAC연산기의 구현을 위해 연산기 내부에 부동 소수점 덧셈 연산기를 사용하는데, 지수부에 따른 지수정렬을 다시 해주어야 하기 때문에 MAC연산기의 지연시간이 길어지게 된다. 그래서, 부동 소수점 덧셈과 곱셈 연산기를 별도로 분리하여 MAC연산을 가능하게 하는 방법이 사용된다. 또한, 덧셈과 곱셈을 동시에 수행함으로써 파이프라인처럼 이용할 수 있어 데이터의 처리능력을 향상시킬 수 있다. 그리고, 새로운 알고리즘에 의한 고속의 곱셈 연산기를 이용해 곱셈 연산기의 지연시간을 단축시킬 수 있다.

레지스터의 포트 수에 따라 동시에 수행될 수 있는 연산이 제한된다. 예를 들어, 다섯 가지의 다른 연산을 동시에 수행하려면 10개의 read 포트와 5개의 write 포트가 요구되는데, 이러한 포트의 증가는 레지스터의 면적과 설계 복잡도의 증가를 초래하게 된다. 또, 프로그램 수행시 레지스터의 할당과 데이터 의존성을 고려해야하며, 컴파일러 설계를 어렵게 하는 요인이 된다. 이 문제를 해결하기 위해, 연산기와 레지스터 사이에 크로스 바와 같은 스위칭 네트웍을 두거나 레지스터를 나누어 사용하는 방법이 이용된다.

나눗셈과 제곱근 연산은 빈번하게 이용되지 않지만, 다른 연산에 비해 상당히 긴 지연시간 때문에 고속의 나눗셈 연산기가 필요하다. 최근의 마이크로 프로세서에 사용된 부동 소수점 연산기의 지연시간을 살펴보면, 덧셈과 곱셈은 같은 지연시간을 가지고, 나눗셈과 제곱근 연산은 다른 연산에 비해 약 3배 이상의 지연시간을 갖는다. 그러므로, 전체 geometry 프로세싱 단계를 처리하는데 나눗셈이나 제곱근 연산이 자주 이용되지는 않지만 데이터 의존성과 지연시간 때문에 처리시간의 상당부분을 차지하게 된다. 그러나, 이런 지연시간을 프로그램 최적화로 줄일 수 있고, 고속의 나눗셈 연산기를 이용하여 연산기의 성능을 높일 수 있다. 즉, 광원 계산과 나눗셈을 위해서는 1/Y를 계산하는 연산기가 요구되며, edge-slope 계산을 위해서는 X/Y를 계산해야 한다. 하지만, 매번 X/Y를 계산하면 지연시간이 상당히 길어지므로, 1/Y을 먼저 계산하고, 그 결과에 X를 곱하여 지연시간을 단축하는 방법이 이 단계에도 적용 되어질 수 있다.

나눗셈 연산을 하는 방법으로는 곱셈 연산기를 이용한 나눗셈 알고리즘과 뺄셈 연산을 이용한 알고리즘이 있다. 먼저 곱셈 연산기를 이용해 나눗셈을 하는 알고리즘은 Newton-Raphson 반복 알고리즘을 이용해 1/Y를 계산하고 X/Y를 계산하기 때문에, 곱셈 연산기를 공유하여 설계할 수 있으며, 1/Y연산이 X/Y연산보다 적은 지연시간을 갖는다. 또, 제곱근 연산 역시 나눗셈 연산기를 이용하여 계산하기 때문에 같은 방법이 적용될 수 있다. 즉, 하나의 나눗셈 연산기로 세가지 연산을 할 수 있다[15].

그래픽 연산의 특성상 나머지를 구할 필요가 없기 때문에 SRT나눗셈 연산기 보다는 곱셈 연산기를 이용하여 나눗셈을 하는 고속의 reciprocal 나눗셈 연산기를 사용한다. 그러나, 나눗셈을 위해 곱셈 연산기를 공유할 경우, 반복적인 곱셈 연산에 의한 긴 지연시간으로 다른 연산을 계속 수행하지 못하기 때문에 성능 저하의 요인이 된다. 그러므로, 별도의 나눗셈 연산기를 두어 덧셈, 곱셈과 동시에 나눗셈을 수행할 수 있게 하는 것이 바람직하다. 하지만, 이에 따른 레지스터 포트, 설계의 복잡도, 점유면적의 증가와 나눗셈 연산기의 사용율이 저하되는 것을 피할 수 없다. 또, 독립적인 reciprocal 나눗셈 연산기를 이용하면, 나눗셈 연산기 면적이 다른 연산기에 비해 상당히 크고, 낮은 사용빈도 때문에 덧셈, 곱셈, 나눗셈 연산을 동시에 하는 이점이 사라지게 된다. 그러므로, 상대적으로 차지하는 면적이 작은 SRT나눗셈 연산기를 이용하여, 점유면적을 감소시킬 수 있다. 또한, 1/Y연산과 곱셈으로 X/Y연산을 하므로, SRT나눗셈 연산기의 ROM lookup table을 최적화 할 수 있게 되어 비교적 고속의 나눗셈 연산기를 구현할 수 있다.

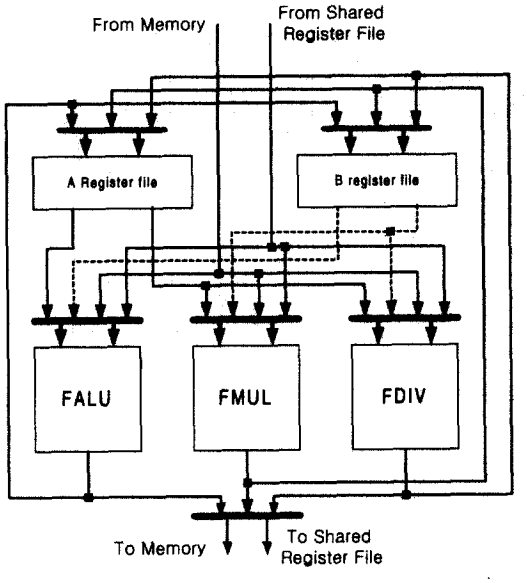
요약하면, geometry 오퍼레이션을 빠르게 처리할 수 있는 프로세서 다음과 같은 특성을 지녀야 한다.

- 데이터의 의존성이 적은 덧셈/뺄셈, 곱셈 연산을 병렬로 처리하거나, 곱셈 결과를 다음 덧셈에 바로 이용할 수 있어야 한다.
- 빈도는 높지 않고 데이터의 의존성이 큰, 나눗셈/제곱근 연산을 빠르게 처리 할 수 있어야 한다.
- 그래픽 데이터의 내재된 병렬성을 이용할 수 있도록 다수의 연산기 구성이 용이해야 한다.

4.3 단일 기능 유닛 구성과 프로세서 구조의 제안

(그림 1)과 같이 세 개의 독립적인 연산기를 두어 덧셈, 곱셈, 나눗셈 연산을 동시에 수행할 수 있도록 하였다. 또한 이들 명령어의 독립적인 수행을 위해 필요한 멀티포트 레지스터 파일을 A와 B 레지스터로 나누어 포트 수를 2 read, 2 write로 줄여 점유면적을 크게 줄였다. 그 크기는 각 32비트이며, 32개가 사용된다. 계산결과는 A, B 레지스터, 공유 레지스터로 동시에 기록이 가능하다. 그러므로, 오퍼랜드의 fetch와 결과의 기록은 A와 B 레지스터, 공유 레지스터, 입력/출력 메모리에서 할 수 있다.

나눗셈 연산은 빈번히 사용되지 않고, 긴 지연시간 때문에 나눗셈 연산의 오퍼랜드 fetch를 위해 레지스터 read포트를 곱셈 연산기와 공유하게 설계하였다. 이 때, 오퍼랜드를 fetch하는 동안만, A, B 레지스터에 대해 곱셈 연산을 할 수 없고, 다시 새로운 나눗셈 연산을 하기 전까지는 세가지 연산을 동시에 수행할 수 있다. 그리고, 세가지 연산들이 오퍼랜드를 각각 다른 레지스터나 메모리에서 가져 온다면, 세가지 연산을 동시에 수행 가능하다.

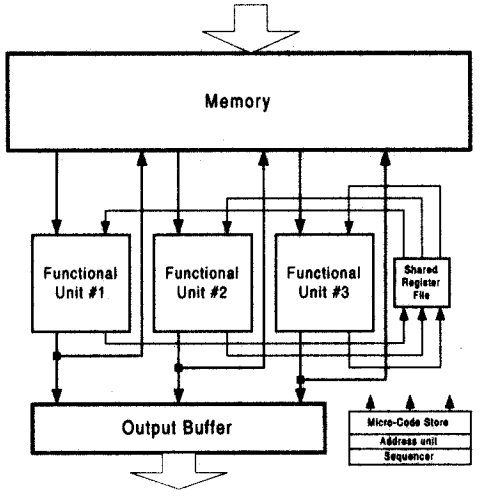


(그림 1) 제안하는 단일 기능 유닛

데이터를 분배하는 장치에 의해 모델 데이터를 입력 메모리로 입력 받은 후, 프로세서의 geometry 프로세싱이 시작되고, 전 단계가 모두 수행된 후 출력버퍼에 저장되어 있는 데이터가 rasterizer로 보내지게 된다. 이 프로세서는 두개의 부동 소수점 연산을 동시에 시작할 수 있고, 나눗셈 연산이 수행되는 동안 다른 덧셈과 곱셈 연산을 수행할 수 있다. 프로세서내의 기능 유닛들은 SIMD방식으로 동작하며, 공유 레지스터로 각 기능 유닛간에 데이터의 교환을 가능하게 한다. 또, 앞장에서 살펴 본 바와 같이 geometry 프로세싱은 특정 파라메터와 모델 데이터의 연산이므로, 이 파라메터들을(예, 정점 변환행렬, 광원특성 모델 파라메터) 공유 레지스터 파일에 공유시킴으로써, 작업 메모리의 참조횟수와 읽기(read) 포트의 수를 줄일 수 있다. 그리고, geometry 프로세싱의 각 단계에 맞는 명령어들을 저장하는 micro-code store, 메모리의 주소를 계산하는 address unit, sequencer, 공유 레지스터로 구성되어 있다. 입력메모리의 포트 수는 3 read, 3 write로 비교적 많으나, 입력된 모델 데이터가 서로 독립적이고, 각 기능 유닛이 SIMD방식으로 동작하기 때문에, 입력 메모리를 3개의 뱅크로 나누어 각각의 뱅크를 1 read, 1 write로 하여 포트증가에 따른 메모리의 점유면적과 설계복잡도를 줄였다.

3차원 그래픽 처리의 가장 큰 그래픽 primitive의 단위가 삼각형이므로, 한 프로세서 내에서 이 세 정점 모두를 처리하도록 세 개의 기능 유닛을 두면, 시스템의 throughput을 높일 수 있다. 또, geometry프로세서가 edge-slope 계산단계를 포함하므로, 삼각형의 조합에 관한 정보가 포함되어 있기 때문에 edge-slope계산에서 삼각형의 재조합 단계가 필요치 않으며, 별도의 기능 유닛이 필요치 않는다. 또, 다른 연산기에 비해 빈번하지 않는 나눗셈 연산기의 사용율을 edge-slope 계산을 geometry 프로세서가 수행함으로써 증가시킬 수 있으며, 전체적인 프로세서의 사용율도 증가 시킬 수 있다.

행렬 곱셈에 사용되는 4×4 행렬이나, 3×3 행렬 등은 세 개의 기능 유닛들에게 공통적으로 이용될 수 있기 때문에, 이 들을 공유 레지스터에 저장하여 각 기능 유닛들끼리 공유함으로써, 기능 유닛 내의 레지스터들을 충분히 사용할 수 있다. 또한, 메모리에서 데이터를 읽어 오는 횟수를 감소시켜 프로세서의 전체 수행시간의 단축을 기대할 수 있다. 그러나, 공유 레지스터 파일은 각 기능 유닛들이 공유해야 하므로 3 read, 3 write의 포트를 필요로 하다.



(그림 2) 제안하는 프로세서의 구조

5. 성능분석

제안하는 프로세서의 성능분석을 분석하기 위하여 먼저 단일 기능 유닛의 연산 성능을 분석하였으며, 프로세서의 성능을 geometry 프로세싱의 각 단계를 수행하는데 소요되는 시간을 연산사이클로 표시하였다. 또, 연산기의 성능을 측정하고 비교하기 위해 실제 모델 데이터를 이용해 연산기의 구성 방법과 지연시간에 따라 모의실험을 하였으며, 성능비교를 위해 InfiniteReality의 연산기를 모델링하여 사용하였다.

5.1 모의실험

모의실험은 OpenGL 벤치마크에 포함된 Viewset 중 Advanced Visualizer Viewset(Awadvs-01)를 이용하였다. 이 모델 데이터는 14874개의 정점 좌표, 각 정점의 색깔정보와 법선 벡터의 좌표로 구성되어 있으며, 11943개의 다각형 정보를 이용하였으며, 모든 다각형에 대해 단일한 물체특성과 하나의 광원을 가정하였다.

먼저 연산기의 구성에 의한 성능을 측정하기 위해 부동 소수점 연산기의 구성방법에 따라 명령어를 한번에 하나씩 수행하는 모델(single issue)과 한번에 두개의 명령어를 수행하는 모델(dual issue)을 가정하였다. 일반적인 프로세서의 배정밀도(double precision) 부동소수점 연산의 지연시간을 <표 2>에 제시하였다[15]. 일반적으로 부동 소수점 덧셈/뺄셈과 곱셈 연산은 단정밀도 연산과 배정밀도 연산의 지연시간이 같지만, 부동 소수점 나눗셈과 제곱근 연산의 경우는 구현에 사용되는 알고리즘과 데이터 형에 따라 달라지게 된다. 그래서, <표 2>에 근거하여 제시하고자 하는 나눗셈/제곱근 연산기와 같은 형태로 구성된 프로세서의 데이터를 가지고 단 정밀도 부동 소수점 나눗셈과 제곱근 연산을 하는데 걸리는 수행 사이클을 계산하여 사용하였다.

〈표 2〉 고성능 마이크로프로세서의 부동 소수점 연산 지연시간(배정밀도)

Design	Cycle time Latency/throughput (cycles/cycles)				
	(Ns)	+	×	/	SQRT
HP PA7200	7.14	2/1	2/1	15/15	15/15
HP PA8000	5	3/1	3/1	31/31	31/31
IBM RS/6000 Power2	13.99	2/1	2/1	16-19	25/24
Intel Pentium	5.0	3/1	3/2	39/39	70/70
Intel Pentium Pro	7.52	3/1	5/2	30/30	53/53
Mips R8000	13.33	4/1	4/1	20/17	23/20
Mips R10000	3.64	2/1	2/1	18/18	32/32
PowerPC 620	7.5	3/1	3/1	18/18	22/22
Sun SuperSparc	16.67	3/1	3/1	9/7	12/10
Sun UltraSparc	4	3/1	3/1	22/22	22/22

부동 소수점 덧셈과 곱셈 연산의 지연시간이 Intel의 Pentium Pro를 제외한 모든 프로세서에서 동일하며, 이는 단정도 연산일 때도 역시 같다는 것을 알 수 있다. 그러나, 나눗셈과 제곱근 연산의 지연시간은 SRT 나눗셈 알고리즘을 사용하는 프로세서에서 그 지연시간이 일치하고, 곱셈 연산기를 이용하여 알고리즘을 사용하는 프로세서는 일치하지 않음을 알 수 있다. 하지만, 인텔의 프로세서처럼 SRT 알고리즘을 이용하였을지라도, 회로의 최적화 기술과 구현방법에 따라 다른 지연시간을 가진 프로세서도 있다. 예를 들면, HP의 PA7200프로세서는 SRT 알고리즘을 사용하여 나눗셈과 제곱근 연산의 지연시간이 같지만, IBM의 RS6000 Power2 프로세서의 경우 곱셈 연산기를 이용한 알고리즘을 사용하기 때문에 각각의 지연시간이 현저하게 차이를 보인다.

연산기의 구성을 살펴보면, HP의 PA7200, Mips의 R10000, Sun의 UltraSparc는 덧셈과 곱셈 연산기가 독립적으로 구성되고, SRT 알고리즘을 이용해 나눗셈과 제곱근 연산을 수행하는 프로세서로 제안하는 geometry 프로세서와 연산기의 구성이 비슷한 프로세서들이다. Mips의 R8000과 IBM의 RS6000 Power2 프로세서의 경우, MAC연산기를 이용하며 나눗셈이나 제곱근 연산을

곱셈 연산기를 이용해서 수행하므로, SUN사의 LeoFloat와 연산기의 구성이 비슷하다. 또, SGI의 InfiniteReality의 Geometry프로세서와 연산기의 구성이 비슷한 프로세서는 Sun의 SuperSparc 프로세서와 비슷하며, 덧셈 연산기와 곱셈 연산기가 독립적으로 구성되었으며, 곱셈 연산기를 이용하여 나눗셈과 제곱근 연산을 수행한다.

그러나, 단순히 수행 사이클만으로 지연시간을 가정하여 각 프로세서들을 비교하는 것은 <표 2>에 제시된 프로세서들의 동작 클럭이 다르므로 불가능하다. 즉, SuperSparc 프로세서의 나눗셈 지연시간은 9사이클이며, 펜티엄 프로세서는 39사이클로 단순히 지연시간만으로는 SuperSparc프로세서가 약 5배 가량 빠르다. 그러나, 한번의 나눗셈 연산을 수행하는데 걸리는 시간(ns)을 계산해 보면, 펜티엄은 39×5ns = 195ns이고, SuperSparc은 9×16.67ns = 150.03ns로 SuperSparc프로세서가 펜티엄에 비해 동작 클럭은 느리지만, 나눗셈 연산시간은 약 23% 정도 빠르다. 즉, 구현된 프로세서의 동작 클럭에 따라 연산 알고리즘과 구현방법이 연산기의 성능차이를 가져온다. 그러므로, 성능을 비교하기 위해서는 제안하는 프로세서와 가장 비슷한 연산기의 구성과 연산 알고리즘을 사용하는 프로세서의 모델을 채택하여 단정밀도 연산을 수행할 때의 지연시간을 계산해서 모의실험에 사용하였다.

〈표 3〉 제안하는 프로세서의 나눗셈 연산 지연시간

	1 clock	1/2 clock
Radix 4	14	7
Radix 8	10	5
Radix 16	8	4

제안하는 프로세서는 HP의 PA 7200과 유사한 연산기의 구성과 같은 나눗셈 연산 알고리즘을 사용한다. 덧셈과 곱셈 연산은 현재의 프로세서 중에서 제안하는 방법을 사용하는 것은 없지만, 같은 지연시간을 같게 구현한다고 가정해도, 빠른 클럭을 사용할 수 있기 때문에 전체 수행시간을 단축시킬 수 있다. 그리고, 덧셈과 곱셈 연산의 지연시간이 같고 geometry 프로세서가 부동 소수점 연산만을 수행하기 때문에, 각각의 지연시간을 1 사이클로 가정하였다. 이렇게 함으로써 성능비교를 하는데 동작 클럭의 영향을 줄일 수 있다.

제안하는 프로세서는 별도의 SRT 나눗셈 연산기를 이용하기 때문에, 그 구현방법에 따라 지연시간이 각각 달라지므로 HP의 PA7200프로세서의 지연시간을 모델로 하여 단정밀도 계산에 소요되는 지연시간을 계산하여 <표 4>와 같이 세가지 구성으로 가정하였다. 그리고, PA7200 프로세서의 나눗셈 연산은 1/2 클럭을 사용하여 동작하므로, 한 사이클 동안에 두 번의 반복을 하여 나눗셈과 제곱근 연산의 지연시간을 현저히 낮춘 것이다. 그래서, 실제 모의실험에서는 1/2 클럭을 가정한 모델만을 사용하였다.

첫번째 실험에서는 명령어의 병렬성을 이용한 성능향상의 정도를 측정하였다. 즉, 한번에 한 개의 명령어를 수행할 수 있는 연산기와 한번에 두 개의 명령어를 수행할 수 있는 연산기의 성능을 전체 수행 사이클에 대해 비교하였다. 연산기의 구성은 곱셈 연산기를 이용한 모델과 radix-4 SRT 나눗셈 연산기를 이용한 모델을 이용하였다.

두번째 모의 실험은 나눗셈 연산기의 영향을 측정하기 위해 나눗셈 연산기의 지연시간을 이용하였다. 즉, SRT 나눗셈 연산기를 이용하는 모델과 곱셈 연산기를 이용하여 나눗셈을 수행하는 연산기의 성능을 측정하였다. 성능비교를 위해 곱셈 연산기로 나눗셈 연산을 하는 InfiniteReality의 연산기를 모델링 하였다. 이 프로세서는 별도의 나눗셈 연산기가 사용되지 않았기 때문에 곱셈 연산기를 이용해 나눗셈과 제곱근 연산을 하게 되며, 이 연산이 수행되는 동안에 다른 연산(곱셈)을 할 수 없도록 모델링 하였다. 덧셈과 곱셈 연산기의 지연시간을 1 사이클로 하여 다른 연산의 지연시간은 <표 4>와 같이 계산되었다.

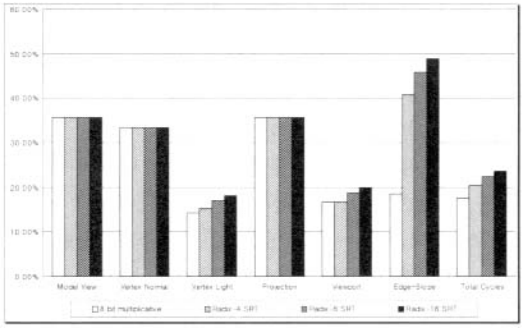
〈표 4〉 부동 소수점 연산의 지연시간

	+/-	×	/	Sqrt
InfiniteReality	1	1	7	10
제안하는 프로세서	1	1	표3	표3

5.2 모의실험 결과

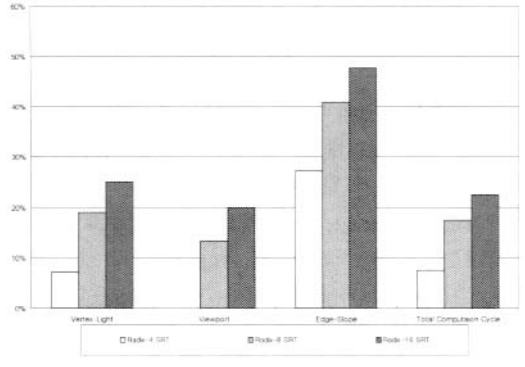
(그림 3)은 첫번째 모의실험의 결과를 나타낸 것이다. 실험결과를 각 단계별로 한번에 한 개의 명령어를 수행하는 모델(single issue FPU)을 기준으로 두개의 명령어를 수행하는 모델(dual issue FPU)의 성능 향상치를 백분율로 나타낸 것이다. 나눗셈 연산을 하지 않는 단계에서는 명령어의 dual issue에 대해 성능이 약 35%, 33%, 35.7%의 향상을 보였으며, 이 단계들에서는 나눗셈 연산기의 성능에 영향을 받지않고 단지 issue

율에 의해서만 영향을 받는다. 그러나, 나눗셈 연산을 하는 단계에서는 나눗셈 연산기의 성능에 따라 각 단계의 성능향상이 뚜렷하다. 즉, edge-slope계산단계에서는 radix-16 single issue에 비해 radix-16 dual issue연산기가 약 49%의 성능 개선이 있었고, 전 단계를 수행하는 데는 23.5%의 성능향상을 보였다. 또, 곱셈 연산기를 이용하는 multiplicative 방법과 지연시간이 같은 radix-4 SRT 나눗셈 연산기의 경우에서도 약 40%의 향상을 보이는데, 이는 1/Y 계산에 의한 나눗셈 연산의 횟수가 줄어들어 성능의 향상을 볼 수 있었다. 즉, 별도의 나눗셈 연산기를 이용해 나눗셈 연산을 곱셈 연산으로 바꾸어, 명령어의 병렬성을 최대한으로 이용한 것이다. 그러나, viewport 변환단계와 광원 계산단계에서는 명령어의 의존성이 높기 때문에 성능의 개선이 크지 않다.



(그림 3) Dual Issue연산기에 의한 성능향상

(그림 4)는 두 번째 모의 실험의 결과를 보인 것으로, 추가된 나눗셈 연산기에 의한 성능의 향상치를 나타낸 것이다. 나눗셈 연산기에 의한 성능의 개선이 있는 단계 즉, 광원 계산, viewport 변환과 edge-slope 계산 단계의 성능 개선을 표시하였다. 곱셈 연산기를 이용하여 나눗셈을 하는 연산기를 기준으로 radix-4, radix-8, radix-16의 SRT나눗셈 연산기를 이용하였을 때의 성능 개선을 백분율로 나타낸 것이다.



(그림 4) 고성능 나눗셈 연산기에 의한 성능 개선

결과를 살펴보면, viewport변환 단계의 경우, perspective 나눗셈 연산을 전 데이터에 대해서 수행해야 하므로, 지연시간이 같은 radix-4 나눗셈 연산기의 경우 성능이 전혀 없다. 그러나, 광원 계산 단계는 제곱근 연산의 지연시간 개선과 나눗셈 연산기를 공유하지 않음으로써 생기는 다른 명령어의 병렬성 이용이 가능함으로 고속 나눗셈 연산기에 의한 성능 향상을 얻을 수 있었다. 또, edge-slope단계에서 1/Y을 계산하여 그 결과에 X를 곱하는 방법을 사용하여 나눗셈 연산의 수를 줄여 처리하기 때문에 곱셈 연산기를 이용하지 않는 고성능 SRT나눗셈 연산기에 의해 큰 성능 향상을 보였으며, geometry 프로세싱 전 단계를 수행하는데 소요되는 계산 사이클은 약 23%의 개선이 있었다. 지연시간이 제일 짧은 radix-16 SRT의 경우, 각각 광원 계산시 25%, Viewport 변환시 20%, edge-slope계산시 48%의 성능 개선이 있었다. 그러므로, geometry프로세서의 연산기의 성능을 높이려면 고속의 나눗셈 연산기의 설계와 각 연산기의 독립적인 구성으로 명령어issue의 높이고, 데이터의 병렬성을 이용할 수 있게 다수의 기능 유닛을 사용할 수 있도록 해야 한다.

5.3 면적증가

곱셈 연산기를 이용해 나눗셈 연산을 하는 방법이 나눗셈 연산에 필요한 로직의 추가를 최소화한다. 즉, 초기 추측치를 저장하고 있는 ROM에 의한 증가면적이 주요한데, 8 비트 크기의 추측치를 이용하는 알고리즘으로 구현 되었을 때의 면적증가를 1로 정규화하여 상대적인 SRT나눗셈 연산기를 이용하였을 때의 면적 증가요인을 살펴보면 <표 5>와 같다.

〈표 5〉 나눗셈 연산기의 면적증가

나눗셈 연산기의 유형	면적증가요인
8비트 초기 추측치	1.0
16비트 초기 추측치	22
Radix-4 SRT	1.5
Radix-16 SRT	2.2

radix-4로 구현할 경우 면적 차이가 거의 없으며, radix-16인 경우는 2.2배 정도로 증가한다. 그러나, 곱셈 연산기를 이용하는 방법의 경우 지연시간을 줄이기 위해 16비트로 늘릴 경우 22배로 늘어나 나눗셈 연산기의 업그레이드 비용이 크게 증가한다[15]. 그러므로, SRT나눗셈 연산기가 점유면적과 업그레이드 비용면에서 이점이 있으며, 알고리즘의 특성상 나눗셈과 제곱근연산에 대해 같은 지연시간을 갖는다.

5.4 프로세서의 성능

Geometry 프로세싱의 각 단계별 계산량을 수행 사이클로 계산하면 <표 6>과 같다. 광원 계산은 제곱근 연산과 나눗셈 연산의 지연시간 때문에 수행시간이 현저히 길다. 특히, 광원의 위치, 시점에 따라 변하는 한 정점과의 단위 벡터의 계산을 위한 나눗셈과 제곱근 연산의 긴 수행 사이클로 인해 전체 수행시간의 절반 이상이 소모된다.

나눗셈 연산기를 고속 나눗셈 연산기로 대체하면, 나눗셈 연산의 지연시간이 절반정도로 줄어들기 때문에 광원 계산에 소용되는 수행 사이클을 68사이클로 낮출 수 있다. 하지만, 비교적 큰 점유면적과 낮은 사용율로 인한 프로세서의 비용의 증가를 일으키게 된다.

〈표 6〉 Geometry 프로세신의 각 단계별 수행 사이클

Geometry 프로세싱 단계	수행 사이클
정점 변환	12 cycle
법선 벡터 변환	9 cycle
광원 계산	68 cycle
Clipping	6 cycle
투 영	16 cycle
Viewport 변환	27 cycle
Edge-slope 계산	18 cycle

Geometry 프로세싱의 전 단계를 수행하는 데, 하나의 기능 유닛 당 156 사이클이 소요된다.

6. 결 론

3차원 그래픽처리는 크게 geometry 프로세싱과 rasterization으로 나누는데, 본 논문에서는 geometry 프로세싱에 적합한 프로세서의 설계와 부동 소수점 연산기를 설계하였다. 일반적으로 부동 소수점 연산의 지연시간이 정수 연산의 세배이기 때문에 부동 소수점연산을 빠르게 처리할 수 있는 연산기의 설계가 필수적이며, 그래픽 데이터들의 병렬성을 활용할 수 있는 프로세서의 구조와 연산기의 구성이 geometry 프로세서의 성능에 영향을 준다.

많은 부동 소수점 연산을 처리하는 geometry 프로세서는 그 부동 소수점 연산기의 성능과 프로세서의 구조와 데이터의 처리 방식에 따라 성능이 좌우된다. 즉, geometry 프로세서는 부동 소수점 연산기의 성능에 따라 성능의 차이를 갖는다. 특히, 부동 소수점 연산 중 나눗셈과 제곱근 연산의 횟수가 다른 연산에 비해 현저히 적음에도 프로세서의 성능에 큰 영향을 주었다. 그리고, geometry 프로세싱은 단계별로 많은 연산과정을 거치므로, 프로세서의 구조는 명령어의 병렬성을 이용할 수 있도록 연산기를 구성하여야 한다.

제안하는 geometry 프로세서는 부동 소수점 덧셈, 곱셈, 나눗셈 연산을 동시에 수행가능하며, 명령어의 병렬성을 높여 geometry 프로세싱 전 단계를 수행하는데 23.5%의 성능향상이 있었다. 그리고, 지연시간이 긴 나눗셈/제곱근 연산을 빠르게 수행하기 위해서 면적대 성능비가 우수한 SRT 나눗셈 연산기를 추가하여 곱셈 연산기를 이용하는 연산기보다 약 23%의 성능향상을 이루었다.

제안하는 프로세서의 데이터 처리모델은 Hybrid 방식으로 SIMD 형태의 단일 프로세서를 MIMD 형태의 보드로 구성하여 사용할 수 있다. 앞으로, 더 자세한 프로세서의 성능분석을 하기 위해서 작업 메모리의 크기, 메모리 대역폭, 공유 레지스터 파일의 크기 등을 고려한 모의실험이 더 이루어져야 할 것이며, MIMD 형태로 구성한 보드수준에서의 모의실험이 이루어져야 할 것이다. 향후 가격대 성능비가 우수한 rasterizer에 대한 설계가 포함되어, geometry 프로세서와 rasterizer를 포함한 그래픽 가속기에 전체에 대한 연구가 진행되어야 할 것이다.

참 고 문 헌

[1] Satya Simha, "Super Mario Chip," BYTE, pp.59-60, Dec. 1996.

[2] Tom R. Halfhill "Beyond MMX," BYTE, pp.87-92, Dec. 1997

[3] John G. Torborg, "A Parallel Processor Architecture for Graphics Arithmetic Operations," In

Proceeding of SIGGRAPH '87, pp.197-204, 1987.

[4] Kurt Akeley, Tom Jermoluk, "High Performance Polygon Rendering," In Proceeding of SIGGRAPH '88, pp.239-246, 1988.

[5] Alan Watt, "3D Computer Graphics," Addison & Wesley, 1993.

[6] Neider, Mason and Tom Davis, "OpenGL Programming Guide," Addison & Wesley, 1997.

[7] Neil Trevett "GLINT Gamma : A 3D Geometry and Lighting Processor for the PC," In Proceeding Notebook for HOT Chips IX, pp.235-246, August 1997.

[8] Chaadell B. Harell, Farhad Fouladi, "Graphics Rendering Architecture for a High Performance Desktop Workstation," In Proceeding of SIGGRAPH '93, pp.93-99, 1993.

[9] Kurt Akeley, "RealityEngine Graphics," In Proceeding of SIGGRAPH '93, pp.109-116, 1993.

[10] Michael F. Deering, Scott R Nelson, "Leo : A System for Cost Effective 3D Shaded Graphics," In Proceeding of SIGGRAPH '93, pp.101-108, 1993.

[11] John S. Montrym, Daniel R. Baum, David L. Dignaum and Christopher J. Migdal "InfiniteReality : A Real-Time Graphics System," In Proceeding of SIGGRAPH '97, pp.293-302, 1997.

[12] Israel Koren, "Computer Arithmetic Algorithms," John Wiley & Sons, 1993.

[13] D. Goldberg "Appendix A, Computer Arithmetic," in J.L. Hennessy and D.A. Patterson, Computer Architecture : A quantitative Approach, Morgan Kaufman Publisher, 1990.

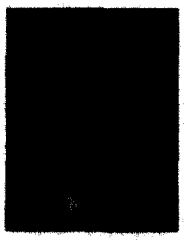
[14] Milos D. Ercegovac and Tomas Lang, "Division and Square Root : Digit-Recurrence Algorithms and Implementations," Kluwer Academic Press, 1994.

[15] Peter Soderquist and Miriam Leeser, "An Area/Performance Comparison of subtractive and Multiplicative Divide/Square Root Implementations," In the Proceedings 12th IEEE Symp. on Computer Arithmetic, IEEE Computer Society press, Jul. 1995.

[16] Woo-Chan Park, Shi-Wha Lee, Oh-Young Kwon and Tack-Don Han, "Floating point Adder/Subtractor Performing IEEE Rounding and Addition/Subtraction in Parallel," IEICE Trans. Inf.&Syst., Vol.E79-D, No.4, Apr. 1996.

[17] 박우찬, 정철호, 양진기, 한탁돈, "IEEE 반올림과 덧셈을 동시에 수행하는 부동 소수점 곱셈 연산기 설계", 전자공학회 논문지 제34권 C편 제11호, pp.897-904, 1997.

[18] 박우찬, 한탁돈, "고성능 부동 소수점 연산기에 대한 연구", 한국정보처리학회 논문지 제4권 제11호, pp.2861-2873, 1997.



정 철 호

e-mail : chjeong@kurene.yonsei.ac.kr

1996년 숭실대학교 소프트웨어공학과 졸업(학사)

1998년 연세대학교 대학원 컴퓨터과학과 졸업(공학석사)

1998년~현재 연세대학교 대학원 컴퓨터과학과 박사과정 재학중

관심분야 : 고성능 컴퓨터구조, 3차원 그래픽 가속기, Computer Arithmetic, ASIC 설계

박 우 찬

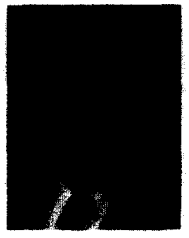
e-mail : chan@kurene.yonsei.ac.kr

1993년 연세대학교 이과대학 전산과학과 졸업(학사)

1995년 연세대학교 대학원 전산과학과 (이학석사)

1995년~현재 연세대학교 공과대학 컴퓨터과학과 박사 재학중

관심분야 : 3차원 그래픽 가속기, ASIC 설계, Computer arithmetic, 고성능 컴퓨터 구조



김 신 덕

e-mail : sdkim@kuerene.yonsei.ac.kr

1982년 연세대학교 공과대학 전자공학과(학사)

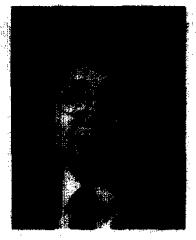
1987년 University of Oklahoma 전기공학(석사)

1991년 Purdue University 전기 및 컴퓨터공학(박사)

1993년~1995년 광운대학교 컴퓨터공학과 조교수

1995년~현재 연세대학교 공과대학 컴퓨터과학과 부교수

관심분야 : Parallel Processing, Advanced Computer Architecture, Internet Computing Applications, Heterogeneous Web Computing. 3D Graphics



한 탁 돈

e-mail : hantack@kurene.yonsei.ac.kr

1978년 연세대학교 공과대학 전자공학과 졸업 (학사)

1983년 Wayne State University 컴퓨터공학 (공학석사)

1987년 University of Massachusetts 컴퓨터공학 (공학박사)

1987년~1989년 Cleveland 주립대학 조교수

1989년~현재 연세대학교 공과대학 컴퓨터과학과 교수

관심분야 : Wearable computer, HCI, ASIC 설계, 고성능 컴퓨터구조