논문 2005-42SD-10-8

3D 그래픽 Geometry Engine을 위한 부동소수점 연산기의 설계

(Design of a Floating Point Unit for 3D Graphics Geometry Engine)

김 명 환*, 오 민 석*, 이 광 엽*, 김 원 종**, 조 한 진**

(Myeong Hwan Kim, Min Seok Oh, Kwang Yeob Lee, Won Jong Kim, and Han Jin Cho)

요 약

본 논문에서는 실시간 3D 가속을 효과적으로 하기 위해 기하학 처리 과정에 적합한 부동 소수점 연산기를 설계하였다. 설계한 부동 소수점 연산기는 IEEE-754 단정도 형식을 지원하도록 하여 기하학 처리에 적합하게 하였고 설계한 부동 소수점 연산기는 Xilinx-Vertex2에서 부동소수점 덧셈/곱셈기는 100 MHz, 부동소수점 NR 역수 계산기는 120 MHz, 부동 소수점 멱승기는 200 MHz, 부동 소수점 역 재곱근 연산기는 120 MHz의 동작 주파수를 각각 확인 하였다. 또한 설계된 부동소수점 연산기를 이용해 실제 기하학 프로세서를 구현하여 실제 3D 데이터 처리를 확인하였다.

Abstract

In this paper, we designed floating point units to accelate real-time 3D Graphics for Geometry processing. Designed floating point units support IEEE-754 single precision format and we confirmed 100 MHz performance of floating point add/mul unit, 120 MHz performance of floating point NR inverse division unit, 200 MHz performance of floating point power unit, 120 MHz performance of floating point inverse square root unit at Xilinx-vertex2. Also, using floating point units, designed Geometry processor and confirmed 3D Graphics data processing.

Keywords : Floating Point Unit, 3D Graphics, Geometry, IEEE-754 single precision

I. 서 론

2D 그래픽의 세계에서는 면의 세계에 국한된 표현만이 가능했지만 3D 그래픽의 세계는 면으로 이루어진 공간의 세계를 표현해내기 때문에 그 표현범위가 대폭 넓어지게 되었다. 그 세계는 공간과 거리, 방향, 높이 그리고 깊이의 세계를 나타내는 현실감이 존재하는 세계이다. 때문에 3D 그래픽의 응용분야는 거의 무한정에 가깝다고 볼 수 있다.

가장 흔하게 적용되는 3D 페인팅 분야의 렌더링과 CAD분야를 비롯하여 3D 애니메이션, 가상현실, 3D 디지털 만화영화, 3D 게임, 영화, 3D 입체영상, 의료, 군사 훈련 시뮬레이터, 엔터테인먼트분야 등 다양한 분야에서 쓰여 지고 있다. 예전의 3D 그래픽은 정적인 장면을 만들어내는 용도로 제한적으로 사용되어왔으나 이제는 3D 그래픽이 일반화됨과 동시에 실시간 3D 그래픽 기술의 등장으로 그 활용 분야가 대폭 넓어지게 되었다[1],[2].

3D 그래픽을 표현하기 위해서는 많은 데이터를 연산해야 하며 연산 과정 또한 복잡하다. 초기에는 CPU가 이 모든 연산 및 처리를 담당하여 3D 영상을 표현하였으나 보다 정교하고 현실감 있는 3D 영상을 표현하기 위한 데이터의 양이 기하급수적으로 증가함에 따라 CPU만으로 이를 처리하기에는 큰 부담이 되었다. 따라서 그래픽 처리를 위한 가속 하드웨어를 별도로 설계하거나, 고속의 범용 프로세서를 이에 맞게 확장한 방법이 사용 되어지고 있다[3].

본 논문에서는 휴대 정보기기 시스템에서 더욱 향상된 실시간 3D 가속을 제공하기 위한 효과적인 기하학 처리기(Geometry processor) 구조를 설계하기 위해 이에 적합한 부동소수점 연산기를 연구하였으며, 이를 기

* 정회원, 서경대학교 컴퓨터공학과
(Department of Computer Engineering, SeoKyeong University)
** 정회원 한국전자통신연구원 SoC설계연구부
(SoC Design Team, ETRI)
※ 본연구는 ETRI와 ITSOC 사업단의 지원으로 수행되었으며, IDEC 지원 장비를 활용하였습니다.
접수일자: 2005년3월23일, 수정완료일: 2005년9월24일

반으로 부동소수점 연산기를 설계하였으며 기하학 처리기와 OpenGL-ES 기반에서 성능을 검증하였다.

II. 기하학 처리 과정에 필요한 부동소수점 연산

1. 기하학 처리 과정

기하학 단계는 호스트 즉 어플리케이션 단계에서 처리된 모델 데이터의 정보들을 입력 받음으로서 시작된다. 기하학 단계에서 입력 받는 모델 데이터는 한 좌표의 정보, 좌표의 노말 벡터(Normal Vector), 색깔 정보와 각 단계에서 요구되는 파라메터들이 포함된다[4].

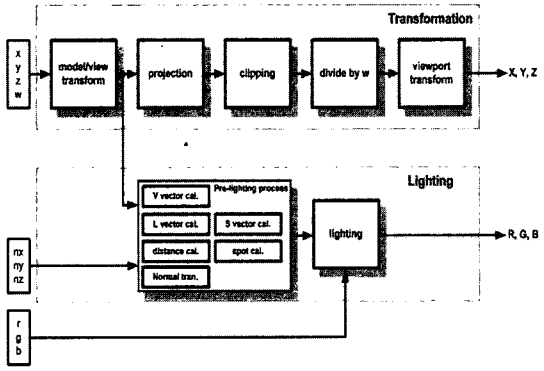


그림 1. 3D 그래픽 Geometry pipeline
Fig. 1. 3D Graphic Geometry pipeline.

기하학 단계는 객체에 대한 공간상의 위치 변환과 빛에 대한 계산이 이뤄지는 과정이다. 이 단계는 그림 1에서 보듯이 크게 변환(Transformation) 부분과 빛 처리(Lighting) 부분으로 분류된다. 변환 부분에서는 모델/시야 변환, 투영, 클리핑, 화면 매핑으로 다시 분류가 되며 빛 처리 부분은 실제 빛 처리를 하기 전에 빛 처리에 필요한 요소들을 미리 계산 하는 부분과 실제 빛 처리 수식을 수행하는 부분으로 분류된다[4],[5].

2. 기하학 처리 과정에 필요한 연산

기하학 단계에서 사용되는 입력 데이터는 IEEE 단정도 부동소수점 형식이며 앞에서 언급한 변환과 빛 처리를 수행하기 위해서는 기본적인 사칙 연산 외에 더 복잡한 연산들을 요구한다.

변환 과정에서는 3D 그래픽 모델의 좌표를 변환하는 과정으로 4 * 4 행렬과 4 * 1 행렬의 곱셈 연산이 주를 이룬다. 이 연산 과정은 곱셈과 덧셈 연산으로 이루어지며, 이 외에 비교 연산과 나눗셈 연산이 요구된다. 곱셈과 덧셈 연산은 변환 과정의 모든 과정에서 요구되며, 비교 연산은 View Volume 안쪽에 있는지의 여부를 검사하는 과정인 클립핑, 나눗셈 연산은 원근 투영 변환 후 모델의 좌표 동차화 과정에서 요구된다.

표 1. 3D 그래픽 기하학 단계에서 요구되는 연산
Table 1. Arithmetic that is required at 3D Geometry Stage.

기하학 처리 요구 연산	사 용 과 정
가산, 감산, 곱셈	Transformation, Lighting 전 과정
비교	Transformation의 절단 과정
나눗셈	Transformation의 1/W 과정 역의 전치 행렬 계산 과정
역 제곱근	Lighting의 vector norm. 과정
멱승	Lighting의 집중 조명 처리 과정 Lighting의 specular 처리 과정

빛 처리 과정은 3D 그래픽 모델의 각 정점에 빛 요소를 적용하여 해당 정점의 색을 구하는 과정으로 일반적으로 변환 과정보다 더 복잡한 연산으로 구성된다. 변환 과정에서와 마찬가지로 곱셈과 덧셈은 음영 처리 전 과정에서도 필요하다. 나눗셈 연산은 모델/시야 행렬의 역의 전치 행렬을 구하는 과정에서 필요하게 된다. 역 제곱근(Square-Root)은 빛 처리에서 계산된 벡터들을 단위 벡터(벡터의 길이가 '1'인 벡터)로 만들어 주는 과정에서 필요하게 된다. 마지막으로 멱승(Power)은 집중 조명광 계산과 정반사 성분 계산에서 필요하게 된다.

이런 기하학 단계에서 필요한 연산을 표 1에 정리하였다. 기하학 연산 과정을 살펴보면, 먼저 곱셈 연산을 한 후 그 결과를 다른 오퍼랜드를 더하는 연산이 주류이다.

가장 먼저 고려될 수 있는 연산기가 MAC(Multiply and Accumulation)이며, 여러 개의 MAC 연산기를 이용하여 행렬의 곱셈과 벡터의 내적(Dot Product) 계산을 가속시킬 수 있다[6].

기하학 각 단계 중 MAC 연산을 사용할 수 있는 부분이 많이 있지만, 이 경우 그 외의 단일 곱셈, 덧셈, 뺄셈, 비교, 나눗셈 연산을 위한 연산기가 별도로 요구된다. 그래서 MAC 연산기와 함께 다른 많은 단일 연산기들의 사용은 연산기의 사용율을 저하시키고 설계 비용과 프로세서의 비용을 증가 시키는 원인이 될 수 있다. 또, 부동소수점 MAC 연산기의 구현을 위해 연산기 내부에 부동소수점 덧셈 연산기를 사용하는데, 지수부에 따른 정렬(Alignment)을 다시 해 주어야 하기 때문에 MAC 연산기의 지연 시간이 커지게 된다. 이러한 문제로 인해 부동소수점 덧셈과 곱셈 연산기를 별도로 분리하여 MAC 연산을 가능하게 하는 방법이 사용된다.

일반적으로 부동소수점 곱셈과 덧셈 연산의 지연 시간은 고정 소수점과 정수 연산의 3배 가량이 길며, 면적 또한 2배 정도 크다. 이는 부동 소수점의 소수 부분(Fraction) 덧셈과 곱셈기를 고속화 시키고, 반올림(Rounding)과 정규화(Normalize) 부분을 병렬 처리함으로서 지연 시간을 줄일 수 있으며, 곱셈 연산기와 곱셈 연산기의 지수 처리부와 반올림, 정규화 처리기를 공유함으로서 면적을 줄일 수 있다. 그리고 뺄셈 및 비교 연산은 큰 의미에서 덧셈 연산과 동일하다. 그러므로 뺄셈 및 비교 연산은 별도의 단일 연산기 구현 없이 덧셈기의 기능을 이용하여 처리할 수 있다.

나눗셈 연산의 경우 기하학 과정에서 빈번하게 이용되지는 않지만, 다른 연산에 비해 상당히 긴 지연 시간을 갖는다. 최근의 마이크로프로세서에 사용된 부동소수점 연산기의 지연시간을 살펴보면, 연산기의 구성 방법에 따라 다르지만, 덧셈과 곱셈은 같은 지연시간을 갖는다. 그러나 나눗셈 연산은 다른 연산에 비해 약 3배 이상의 지연 시간을 갖는다. 그러므로 나눗셈 연산은 기하학 과정에서 빈번하게 이용되지는 않지만, 데이터 의존성과 긴 지연 시간 때문에 전체 기하학 과정을 처리하는데 걸리는 시간의 상당 부분을 차지하게 된다. 나눗셈 연산으로 인한 기하학 처리의 성능 저하는 3D 그래픽 데이터와 처리 과정의 특성을 이용하여 줄일 수 있다. 기하학 처리 과정 중 나눗셈 연산이 집중적으로 요구되는 단계는 표 1.에서와 같이 모델의 좌표 x, y, z, w를 w로 나누는 동차화 단계와 역의 전치 행렬을 구하는 단계이다. 이 과정은 3번 또는 4번의 나눗셈 연산이 연속적으로 요구되며, 이는 지연 시간이 긴 나눗셈 연산의 특성상 기하학 처리 단계 중 가장 임계 경로(Critical Path)이다. 그러나 이 연속적인 나눗셈 연산대신 1/w를 계산하여, 이를 x, y, z, w에 곱하는 방법을 사용함으로서 4번의 나눗셈 연산이 1번의 나눗셈 연산과 4번의 곱셈 연산으로 대처될 수 있다. 이는 지연 시간이 긴 나눗셈 연산의 사용 빈도를 줄이는 방법으로 나눗셈 연산으로 인한 성능 저하를 해결하기 위한 효율적인 방법이다.

다음의 항목은 기하학 과정의 가속화를 위해 적합한 연산기의 특징과 이에 대하여 본 논문에서 부동소수점 연산기 설계 시 채택한 구조를 요약하였다.

1. 3D 그래픽 데이터의 내재된 병렬성을 이용할 수 있도록 다수의 연산기 구성에 용이해야한다. : **부동 소수점 곱셈기와 덧셈기 설계 시 많은 부분을 서로 공유하여 면적을 줄임으로서 다수의 부동소수점 곱셈기와 덧셈기 구성에 이점이 있다.**
2. 연속되는 연산 중 서로간의 데이터 의존성에 의한 성능 저하를 최소화해야한다. : **부동소수점 곱셈기와 덧셈기의 파이프라인 단계를 최적화하여 데이터 의존성에 의한 기하학 과정의 성능 저하를 최소화한다.**
3. 사용 빈도가 낮고, 지연 시간이 긴 나눗셈 연산을 빠르게 처리 할 수 있어야 한다. : **나눗셈 연산기 대신 고속의 역수 연산기를 설계하여, 나눗셈 연산의 사용 빈도를 줄이고, 연속되는 나눗셈 연산으로 인한 긴 지연 시간을 최소화한다.**

Ⅲ. 부동소수점 연산기 설계

1. 부동소수점 덧셈/곱셈 연산기 설계

본 논문에서 설계한 부동소수점 덧셈/곱셈기는 면적의 최소화를 위하여, 지수 처리부와 반올림, 정규화 부분 기능 유닛을 공유하여 설계하였으며, 데이터 의존성으로 인한 성능 저하를 막기 위해 파이프라인 단계를 일반적인 부동 소수점 덧셈기와 곱셈기의 파이프라인 단계인 3단계에서 2단계로 줄였다. 표 2는 일반적인 부동 소수점 덧셈기와 곱셈기의 파이프라인 단계와 제안하는 부동소수점 덧셈/곱셈기의 파이프라인 단계에 대한 설명이다. 표 2에서 각 단계 마다 곱셈 또는 덧셈에 필요한 과정이 괄호로 구분되어 있으며, 제안하는 부동소수점 곱셈/덧셈기 경우 덧셈 연산과, 곱셈 연산이 공

표 2. 부동 소수점 덧셈/곱셈 파이프라인 설명
Table 2. Floating point addition/multiplication pipeline explanation.

3단계 파이프라인		제안하는 2단계 파이프라인	
단계	수행 동작	단계	수행 동작
1	지수 처리(덧셈, 곱셈), 가수의 정렬(덧셈), 크기 비교(덧셈), 위치 교환(덧셈), 스티키 생성(덧셈, 곱셈), 가수의 곱셈(곱셈)	1	지수 처리(공유), 가수의 정렬(덧셈), 크기 비교(덧셈), 위치 교환(덧셈), 스티키 생성(공유), 가수의 곱셈(곱셈)
2	가수의 가감산(덧셈), 선행 '0' 검출(덧셈, 곱셈),	2	가수의 가감산(덧셈), 선행 '0' 개수 검출, 지수 처리(공유), 라운딩(공유), 정규화(공유)
3	지수 처리(덧셈, 곱셈), 라운딩(덧셈, 곱셈), 정규화(덧셈, 곱셈)		

유하는 과정도 괄호로 구분 되어진다[7],[8].

제안하는 부동소수점 곱셈/덧셈기는 3단계 파이프라인 구조와 전체 지연시간의 차이는 없지만 라운딩과 정규화 과정을 가수의 가감산 과정과 같은 파이프라인 단계에서 처리한다. 이는 2, 3단계를 2단계로 통합함으로서 임계 경로에 영향을 주어 연산기의 성능 저하가 발생할 수 있다. 그러나 일반적으로 부동소수점 곱셈기와 덧셈기의 파이프라인 단계 중 여러 과정을 수행하는 1단계가 2단계와 3단계에 비하여 지연 시간이 길며, 반올림과 정규화 과정이 직렬로 형성되는 일반적은 구조와 달리 표 3과 같이 반올림과 정규화 과정의 배타성을 이용하여 두 과정을 병렬로 수행함으로서 2단계와 3단계를 통합한 2단계의 지연시간을 크게 줄일 수 있다. 즉 제안하는 2단계 파이프라인 구조는 임계 경로에 큰 영향을 주지 않아 3단계 파이프라인 구조에 비하여 지연 시간에 대한 성능 저하가 없다[8].

제안하는 부동소수점 곱셈/덧셈기는 그림 2와 같이 파이프라인 1 단계는 덧셈 연산의 경우 두 오퍼랜드의 지수 차를 계산하고, 곱셈 연산의 경우 두 오퍼랜드의 지수의 합을 계산하는 지수 처리 가감산기, 덧셈 연산에서 요구되는 지수의 차를 참조하여 가수를 정렬하는 배럴 쉬프터(Barrel Shifter), 두 오퍼랜드의 가수를 비교하여 항상 큰 오퍼랜드가 왼쪽으로 향하게 하는 비교기와 위치 교환기(Swap Unit)와 반올림을 위해 스티키(Sticky) 비트를 생성하는 스티키 비트 생성기와 곱셈 연산을 처리하기 위해 가수를 곱셈하는 24 * 24 Radix-4 booth 곱셈기로 구성되어있다.

파이프라인 2 단계는 덧셈 연산의 경우 두 오퍼랜드의 가수를 덧셈 하는 BCLA(Block Carry Look-up Adder) 구조의 25-bit 고속 덧셈기와 곱셈 연산과 덧셈 연산이 공유하는 유닛으로서 정규화를 위해 곱셈, 덧셈기 결과의 선행 0의 개수를 검출하는 Zero Counter, 반올림을 수행하기 위한 증가기(Incrementor), 정규화를 수행하기 위한 배럴 쉬프터와 최종 지수를 계산하는 가감산기로 구성된다.

표 3. 반올림과 정규화 과정의 배타성
Table 3. Exclusion of rounding and normalization process.

경우	필요 동작	실제 예
가산, d>1인 감산	라운딩 (추가 좌우 쉬프트 필요)	1.xxxxxxxx +/- 0.000xxxxxGRS 1.xxxxxxxxxxxxxxx or 1x.xxxxxxxxxxxxxxx >> 1 or 0.1xxxxxxxxxxxxxx << 1
d=1인 감산	정규화	1.xxxxxxxx - 0.1xxxxxxxx 0.xxxxxxxxx << 선행 '0' 개수
	라운딩	1.xxxxxxxx - 0.1xxxxxxxx 1.xxxxxxxxx
d=0인 감산	정규화	1.xxxxxxxx - 1.xxxxxxxx 0.xxxxxxxxx << 선행 '0' 개수

2. 부동소수점 나눗셈(역수) 연산기 설계

고속의 부동소수점 나눗셈기를 구현하는 방안은 크게 두 가지 부류로 나뉜다. 첫째로 덧셈/뺄셈과 쉬프트 동작을 조합하여 구현하는 방식으로 복원(restoring) 알고리즘, 비복원(nonrestoring) 알고리즘, SRT 알고리즘이 있다. 둘째는 승산기를 이용하여 나눗셈을 수행하는 방식으로 분모, 분자에 동일한 수를 곱하는 수렴(convergence) 방식과 분모의 역수를 구해 곱하는 NR (newton-raphson) 방식이 있다. 본 논문에서는 3D 그래픽 처리 특성상 나머지 값이 필요하지 않고, 적은 수행 단계의 파이프라인 수행이 가능하며, 역수 계산에 유리한 NR 방식으로 나눗셈(역수) 연산기를 설계하였다[7],[9].

본 논문에서 나눗셈 연산 설계 시 사용한 NR 알고리즘은 식 (1)과 같은 테일러 급수 확장식을 사용하였다

$$\frac{X}{Y}=\frac{X}{Y_h+Y_l}=\frac{X}{Y_h}\left(1-\frac{Y_l}{Y_h}+\left(\frac{Y_l}{Y_h}\right)^2-\cdots\right) \quad (1)$$

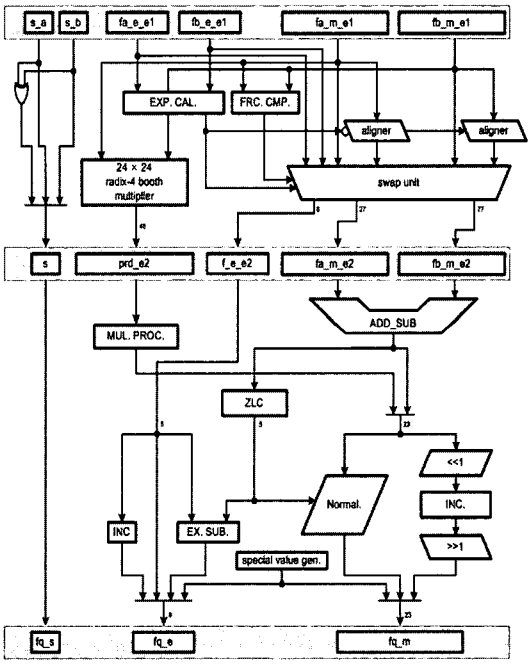


그림 2. 부동소수점 덧셈/곱셈기 구조
Fig. 2. Floating point addition/multiplication architecture.

$Y_h = 2^0 y_0 + 2^{-1} y_1 + 2^{-2} y_2 + \cdots + 2^{-(p-1)} y_{p-1}$, $Y_l = Y - Y_h$이며, 따라서 $Y_h \gg Y_l$이고 Y_l/Y_h는 '0'에 가깝게 된다. X와 Y는 고정 소수점 수로 일반화하기 하기 때문에 x_0, y_0는 항상 '1' 이므로 제수와 피제수의 범위는 $1 \le X, Y < 2$로 제한한다.

또한 식 (1)의 각 요소들은 $1 \le Y_h < 2 - 2^{-(p-1)}$, $0 \le Y_l < 2^{-(p-1)}$의 범위를 가지게 된다. 식 (2)는 식 (1)의 첫 부분의 두 개항을 사용한 테일러 급수의 확장식이 된다.

$$\frac{X}{Y} \simeq \frac{X(Y_h - Y_l)}{Y_h^2} \tag{2}$$

NR방식의 나눗셈기는 일반적으로 파이프라인이 가능한 장점을 가지고 있지만 매우 큰 룩업 테이블(Look-Up table)의 크기가 사용된다. 따라서 룩업 테이블을 최소화할 수 있는 알고리즘을 사용하여 전체 나눗셈기의 크기를 줄일 수 있다. 사용된 알고리즘은 식 수식 (3)과 같다. 우선 식 (2)에서 Y_h의 비트열의 크기를 줄이고 1차 근사 몫 Q'를 구한다.

$$Q' = \frac{X(Y_h - Y_l)}{Y_h^2} \tag{3}$$

그리고 1차 근사 몫 Q'를 이용하여 중간 피제수를 식 (4)와 같이 만들어 낸다.

$$\begin{aligned} X' &\simeq X - Y \cdot Q \\ &= X - Y \cdot \frac{X(Y_h - Y_l)}{Y_h^2} \\ &= X(1 - \frac{(Y_h + Y_l)(Y_h - Y_l)}{Y_h^2}) \\ &= X(1 - \frac{Y_h^2 - Y_l^2}{Y_h^2}) = \frac{X \cdot Y_l^2}{Y_h^2} \end{aligned} \tag{4}$$

식 (4)에서 피제수 X에 중간 피제수 X'를 대입하여 2차 근사 몫 Q''을 생성하고, 이와 같은 두 개의 몫 Q', Q''이 더해진다면 최종적인 몫은 식 (5)와 같이 계산된다.

$$Q'' = \frac{X'(Y_h - Y_l)}{Y_h^2} \tag{5}$$

$$\frac{X}{Y} \simeq Q' + Q''$$

위의 식은 다음과 같이 재계산할 수 있다[7],[9].

$$\begin{aligned} &= \frac{(X + X')(Y_h - Y_l)}{Y_h^2} \\ &= (X + X')A, \; A = \frac{(Y_h - Y_l)}{Y_h^2} \\ &= (2X + YQ')A \\ &= (2X - AYX)A \\ &= (2 - AY)AX \end{aligned} \tag{6}$$

그림 3은 식 (6)를 구현하기 위해서 4 단계 과정으로 설계된 부동소수점 나눗셈기이다.

첫 단계에서 $1/Y_h^2$은 주소 값이 8비트인 룩업 테이블에 의해 구해지며, 룩업 테이블은 엔트리의 수가 128개이고. 13비트의 데이터로 구성되어 총 208Byte의 크기를 갖는다. 두 번째 단계에서 A는 $Y_h - Y_l$과 $1/Y_h^2$을 곱하여 구해진다. 곱셈기 M1의 크기는 24 * 13이 되며 결과 값인 A는 MSB를 제외한 상위 15비트로 구성된다. 세 번째 단계에서 AY가 계산되며, 곱셈기 M2의 크기는 28 * 15가 되며 결과 값인 AY는 MSB를 제외한 상위 28비트로 구성된다. 여기서 $2 - AY$는 AY의 bit inversion에 의해서 구한다. 마지막으로 최종 몫 $AX(2 - AY)$는 A와 $2 - AY$의 곱셈으로 구해진다. 곱셈기 M4의 크기는 15 * 28이 되며 결과 값인 Q는 MSB를 제외한 상위 24비트로 구성된다.

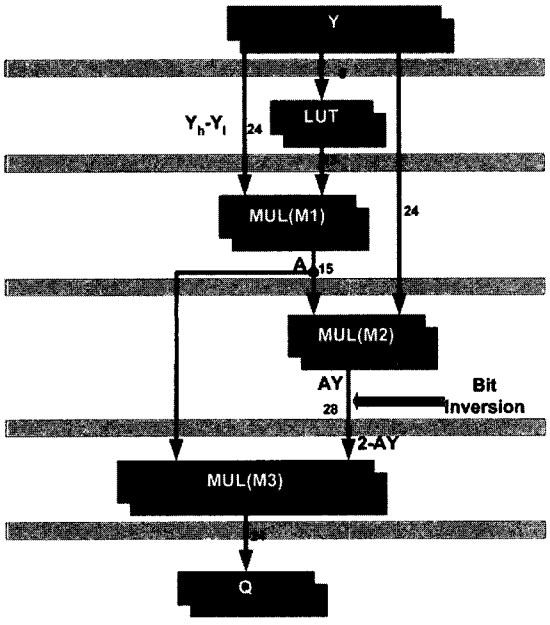


그림 3. 부동소수점 NR 역수 계산기
Fig. 3. Floating point NR converse unit

3. 부동소수점 멱승 연산기 설계

멱승 연산은 음영 처리과정 중 정반사 성분 $(n \cdot h)^{m_{shi}}$와 집중 조명광 $(-l \cdot s_{dir})^{s_{exp}}$를 연산하기 위해 사용된다. 일반적으로 멱승 연산을 구하는 방법은 여러 가지이나 대부분의 경우 곱셈, 나눗셈 연산이 필요하여, 속도면에서 크게 떨어진다. 그러므로 본 논문에서는 그림 4와 같이 음영 처리 프로세서 구조에 적합하고, 속도 면에서 가장 좋은 롬 테이블 방식을 사용했다[10],[11].

롬 테이블 방식의 멱승 연산기는 $(n \cdot h)$값 또는 $(-l \cdot s_{dir})$의 값과 지수인 m_{shi} 또는 s_{exp} 값을 입력으로 받아 양자화된 지수승 결과 값이 저장되어 있는 롬 주소를 발생시켜 해당하는 $(n \cdot h)^{m_{shi}}$ 또는 $(-l \cdot s_{dir})^{s_{exp}}$를 얻는 방식이다.

롬 테이블 방식의 멱승기는 지수승을 표현하는 롬 테이블의 사이즈가 커지고, 롬 테이블의 주소를 발생하는 방법이 실제적으로 하드웨어 구현이 용이하지 못한다는 단점이 있다. 그러나 롬 테이블의 크기를 줄이기 위한 방안을 채용하고, 롬 테이블 값의 규칙적인 특징을 이용함으로써 주소를 간단하게 발생시킬 수 있다.

멱승 연산 결과 값인 $(n \cdot h)^{m_{shi}}$와 $(-l \cdot s_{dir})^{s_{exp}}$는 (0 ~ 1)의 값을 갖는 부동소수점 표현 값이다. n 벡터와 h 벡터의 내적을 취한 값 $(n \cdot h)$와 $(-l \cdot s_{dir})$는 (0 ~ 1)의 값을 갖는 부동소수점 표현 값이며, 지수인 m_{shi}와 s_{exp}는 (1 ~ 128)을 갖는 부동소수점 표현 값이다. 롬 테이블은 $(n \cdot h)$, $(-l \cdot s_{dir})$와 m_{shi}, s_{exp} 값을 양자화 시켜, 이 두 값의 실제 멱승 연산 값을 저장하고 있다. 그러므로 $(n \cdot h)$, $(-l \cdot s_{dir})$와 m_{shi}, s_{exp}의 양자화 값이 작을수록 롬 테이블은 실제 멱승 연산과의 오차가 적어지는 반면 롬 테이블 크기는 커지며, 양자화 값이 클수록 롬 테이블은 실제 멱승 연산과의 오차가 커지는 반면 롬 테이블 크기는 작아진다. 본 설계에서는 작은 롬 테이블 크기를 갖으면서도, 오차가 작은 롬 테이블을 얻기 위하여 다음과 같은 특성을 이용하였다.

- 일정한 m_{shi}, s_{exp}값에 대해서 $(n \cdot h)$, $(-l \cdot s_{dir})$값이 클 때의 $(n \cdot h)$, $(-l \cdot s_{dir})$의 양자화에 의한 $(n \cdot h)^{m_{shi}}$, $(-l \cdot s_{dir})^{s_{exp}}$의 오차가 $(n \cdot h)$, $(-l \cdot s_{dir})$의 값이 작을 때의 $(n \cdot h)^{m_{shi}}$, $(-l \cdot s_{dir})^{s_{exp}}$의 오차보다 크다.
- m_{shi}, s_{exp}값이 증가함에 따라 $(n \cdot h)$, $(-l \cdot s_{dir})$의 값이 클 때의 $(n \cdot h)$, $(-l \cdot s_{dir})$의 양자화에 의한 $(n \cdot h)^{m_{shi}}$, $(-l \cdot s_{dir})^{s_{exp}}$의 오차가 더 커진다.
- m_{shi}, s_{exp}값이 증가함에 따라 $(n \cdot h)^{m_{shi}}$, $(-l \cdot s_{dir})^{s_{exp}}$을 '0'이 아닌 값으로 만드는 $(n \cdot h)$, $(-l \cdot s_{dir})$의 최소 값이 커지고 $(n \cdot h)^{m_{shi}}$, $(-l \cdot s_{dir})^{s_{exp}}$의 오차를 크게 하는 $(n \cdot h)$, $(-l \cdot s_{dir})$의 값도 커진다.

위의 3가지 특성을 이용하여 m_{shi}, s_{exp}와 $(n \cdot h)$, $(-l \cdot s_{dir})$ 값을 비균등 양자화 하여 표 4, 표 5와 멱승 연산 롬 테이블을 생성하였다[10],[11].

표 4와 표 5의 n은 각 구간의 양자화 값을 결정하는 중요한 변수로 n의 값은 멱승 연산 롬 테이블의 크기와 정밀도와 직결된다. 표 6은 양자화 값 n에 따른 멱승 연산 롬 테이블의 크기를 나타내며, 그림 5는 양자화 값 n에 따른 멱승 연산 롬 테이블을 생성하여, OpenGL의 오픈 소스인 MESA 6.0.1 library[12]를 이용하여, $(n \cdot h)^{m_{shi}}$와 $(-l \cdot s_{dir})^{s_{exp}}$ 연산하는 부분을 멱승 연산 롬 테이블로 대체하여 실험한 결과이다. 그림 5의 실험 결과에서 보면 n이 3인 (d)을 제외하고는 육안으로 식별하기 힘든 오차를 갖는다. (d)의 경우 큰 오차로 인해 마하 밴드 효과를 갖는다. 작은 롬 테이블 크기를

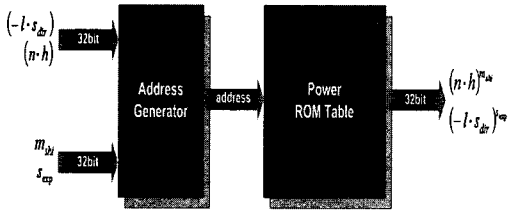


그림 4. Rom Table방식의 부동소수점 멱승기
Fig. 4. Floating point power unit of Rom Table method.

표 4. m_{shi}, s_{exp}의 비균등 양자화 방법
Table 4. Non-uniform quantization method of m_{shi}, s_{exp}.

m_{shi}, s_{exp}	양자화 간격	m_{shi}, s_{exp}	양자화 간격
1 ~ 2	$2^{-(n-3)}$	2 ~ 4	$2^{-(n-4)}$
4 ~ 8	$2^{-(n-5)}$	8 ~ 16	$2^{-(n-6)}$
16 ~ 32	$2^{-(n-7)}$	34 ~ 64	$2^{-(n-8)}$
64 ~ 128	$2^{-(n-9)}$		

표 5. $(n \cdot h)$, $(-l \cdot s_{dir})$의 비균등 양자화 방법

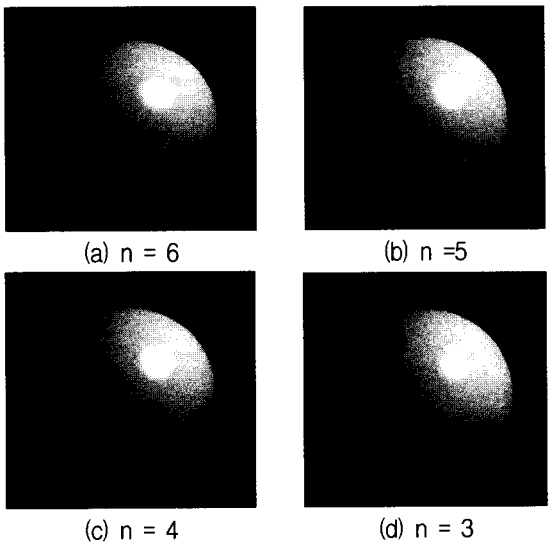
Table 5. Non-uniform quantization method of $(n \cdot h)$, $(-l \cdot s_{dir})$.

$(n \cdot h)$ / $(-l \cdot s_{dir})$ shi	양자화 구간	첫 번째 구간 범위	양자화 간격	두 번째 구간 범위	양자화 간격	세 번째 구간 범위	양자화 간격
1~11	0~1	0~0.5	$2^{-(n-1)}$	0.5~0.75	2^{-n}	0.75~1	$2^{-(n+1)}$
12~23	0.5~1	0.5~0.75	$2^{-1} * 2^{-(n-1)}$	0.75~0.875	$2^{-1} * 2^{-n}$	0.875~1	$2^{-1} * 2^{-(n+1)}$
24~47	0.75~1	0.75~0.875	$2^{-2} * 2^{-(n-1)}$	0.875~0.9375	$2^{-2} * 2^{-n}$	0.9375~1	$2^{-2} * 2^{-(n+1)}$
48~95	0.875~1	0.875~0.9375	$2^{-3} * 2^{-(n-1)}$	0.9375~0.96785	$2^{-3} * 2^{-n}$	0.96785~1	$2^{-3} * 2^{-(n+1)}$
96~128	0.9375~1	0.9375~0.96785	$2^{-4} * 2^{-(n-1)}$	0.96785~0.984375	$2^{-4} * 2^{-n}$	0.984375~1	$2^{-4} * 2^{-(n+1)}$

표 6. n에 따른 롬 테이블 크기

Table 6. Rom table size by n.

n의 값	ROM 테이블의 크기
8	256Kbyte
7	64Kbyte
6	16Kbyte
5	4Kbyte
4	1Kbyte
3	250byte



(a) n = 6 (b) n =5

(c) n = 4 (d) n = 3

그림 5. n에 따른 롬 테이블의 MESA library 실험 결과

Fig. 5. MESA library experiment result of Rom table by n.

갖으면서도, 오차가 작은 롬 테이블을 얻는 것이 본 설계의 목표이기 때문에 n의 값을 4로 설정하여 위와 같은 방법으로 롬 테이블을 생성하여 부동소수점 멱승 연산기를 설계 하였다.

4. 부동소수점 역 제곱근 연산기 설계

부동소수점 역 제곱근기는 벡터의 정규화 과정에서 필요하다. 3D 그래픽 분야에서의 필요성 때문에 실제로 역 제곱근에 대한 특수 명령어들이 모토롤라의 AntiVec에 추가되기도 했다. 역 제곱근을 계산하기 위한 여러 알고리듬들이 있지만 일반적으로 긴 레이턴시나 큰 용량의 메모리의 사용을 요구한다. 설계된 부동소수점 역 제곱근 연산기에서는 변형된 피 연산자과 승산을 이용한 룩업 테이블을 생성해 초기 근사화를 사용하였다. 이 초기 근사화 과정 후에 변형된 뉴튼-랩슨(Newton-Raphson) 반복법은 더 정확한 역 제곱근을 생성하게 된다. 이 초기 근사화와 변형된 뉴튼-랩슨 반복법은 하드웨어의 딜레이, 면적 그리고 전력 소모를 줄일 수 있다[13].

부동소수점 역 제곱근 연산기 알고리듬은 식 (7)과 같은 변형된 뉴튼-랩슨 반복법을 사용한다. 식 (7)에서 R_i는 i번째 근사값을 나타내고 R_{i+1}은 $i+1$번째의 조금 더 정확한 근사값이 되며, j는 정수값이다. 또한 R_0은 임의의 초기 근사값이고 $1 \le X < 2$로 제한한다.

$$R_{i+1} = \frac{R_i(3 - XR_i^2)}{2} \quad (i = 0,1,..,j) \tag{7}$$

임의의 초기 근사값 $R_0 = \frac{1}{\sqrt{X}}$를 구하기 위해서 $X = [1.x_1x_2...x_n]$ $(x_i \in 0,1)$라고 가정한다. 이 방법을 이용하여 X를 $X_1 = [1.x_1x_2...x_m]$과 $X_2 = [1.x_{m+1}x_{m+2}...x_m] \times 2^{-m}$의 두 부분으로 나눈다. 그리고 초기 근사값은 식 (8)과 같이 계산된다.

$$R_0 = X' \cdot C \tag{8}$$

식 (8)에서 $X' = [1.x_1x_2...x_mx_{m+1}'x_{m+1}x_{m+2}'...x_k]$, k는 X'의 소수부분의 비트수이고 ,

$C = X_1^{-\frac{3}{2}} - 3 \cdot 2^{-m-2}X_1^{-\frac{5}{2}} + 33 \cdot 2^{-2m-6}X_1^{-\frac{7}{2}}$이다. X'은 X의 $k-m$ 비트를 보수화 시킨 값이다. C의 값은 미리 계산되어 $[x_1x_2...x_m]$ 비트를 주소화해서 램이나 롬 테이블로 저장한다. 초기 근사값 R_0가 계산된 후에 뉴튼-랩슨 역 제곱근 알고리즘의 변형된 반복법이 근사

값의 정확도를 높이기 위해 사용된다. 식 (7)은 다음과 같은 식으로 표현할 수 있다.

$$W = R^2 \tag{8}$$

$$D = 1 - WX \tag{9}$$

$$Y = R + \frac{RD}{2} \tag{10}$$

이 변형된 식들은 3번의 승산, 한번의 감산, 그리고 한 비트의 오른쪽 쉬프트로써 구현될 수 있다. 식 (8)은 하나의 승산기로써 계산되고, 식 (9)는 WX의 보수를 취함으로써 계산되어지며, 식 (10)은 RD를 한 비트 오른쪽 쉬프트하여 계산한다[13],[14]. 그림 6은 설계된 부동소수점 역 제곱근기 연산기이다.

첫 단계에서 R_0는 128개의 엔트리를 가지는 룩업 테이블의 16비트 데이터 C와 14비트의 X'를 승산기 M1을 통해 계산한다. 룩업 테이블은 총 400비트의 크기를 가지며, M1의 크기는 14*16가 된다. 두 번째 단계에서 16비트의 R_0를 승산기의 양 입력으로 하여 W의 상위 25비트를 결과 값으로 하고, 승산기 M2의 크기는 16*16가 된다. 세 번째 단계에서 WX는 25*23의 승산기 M3를 통해 계산되어지며 WX의 상위 14비트는 bit inversion을 통해 14비트의 D를 구하며 마지막 단계에서 Y는 D와 두 번째 단계에서 구한 R_0를 승산기 M4에서 계산한 후 이 값을 한 비트 오른쪽 쉬프트하여 R_0와 더하여 구하게 된다. Y의 상위 23비트가 최종 역 제곱근의 값이 된다.

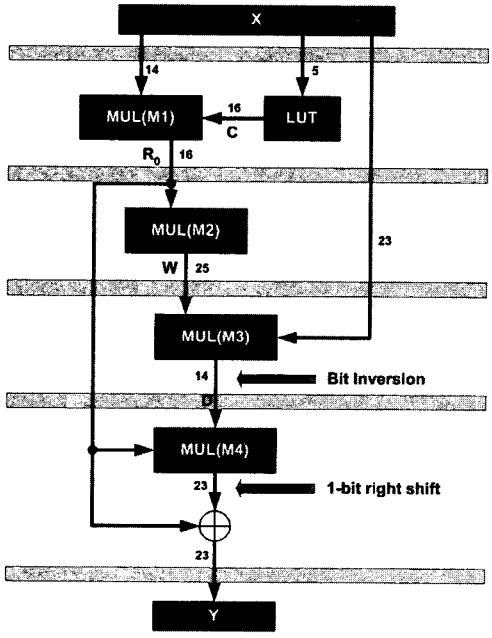


그림 6. 부동소수점 역 제곱근 연산기
Fig. 6. Floating point inverse square-root unit.

Ⅳ. 실험 결과 및 성능 측정

3D 영상 데이터는 3D 그래픽 가속기에서 기하학 처리 및 Rasterization 처리를 거친 후 메모리에 저장되고 이를 화면으로 보내어 원하는 영상을 표현한다. 그래서 3D 영상 데이터를 받아 위치 변환 및 빛에 대한 연산을 수행하는 Geometry 연산 처리기와 삼각형 데이터를 받아 각 픽셀의 색상 정보를 구하는 Rasterization 처리기, 픽셀들의 색상 정보를 모아 한 화면의 영상으로 구성하는 디스플레이 장치가 필요하다[15].

본 논문에서는 설계된 부동소수점 연산기를 검증하기 위해 3D Geometry 프로세서를 FPGA에 구현하여 그림 7과 같은 별도의 기하학 프로세서를 제어하는 호스트 프로세서와 Rasterization 처리기, 디스플레이의

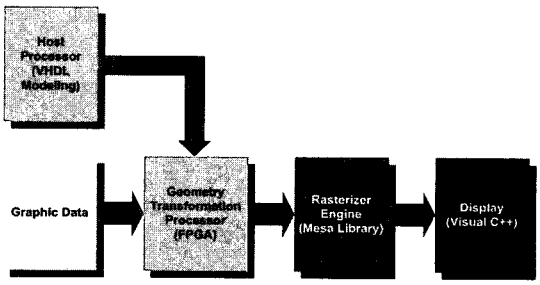


그림 7. 기하학 프로세서 검증 환경
Fig. 7. Geometry processor verification environment.

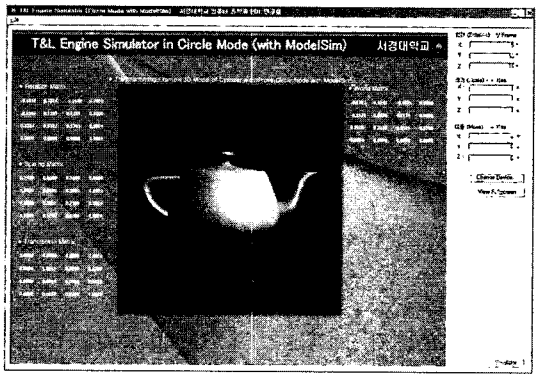


그림 8. 기하학 프로세서 실험 결과
Fig. 8. Geometry processor experiment result.

표 7. 기하학 프로세서 성능
Table 7. Geometry Processor Performance.

	Transform	Directional Light	Positional Light	Spot Light
초당 Polygon 처리 개수	4 M	1.6 M	1.14 M	1 M

표 8. 부동소수점 연산기 합성 결과
Table 8. Floating point unit synthesis result.

부동소수점 연산기	수행 속도
덧셈/곱셈기	100 MHz
나눗셈기(역수 계산기)	120 MHz
멱승기	200 MHz
역 제곱근기	120 MHz

지원을 구성하였다.

그림 7의 실험 환경에서 호스트 프로세서는 VHDL을 이용한 동작 모델링을 통해 PCI 버스 타입의 FPGA에 구현된 기하학 프로세서에게 기하학 과정의 각 단계에 해당하는 동작을 수행하는 SIMD 명령어를 전달한다. Rasterization 처리기는 Mesa 3D 라이브러리를 기반으로 3D 그래픽 파이프라인을 C 프로그램으로 구현한 것으로, Mesa 3D 라이브러리는 OpenGL 그래픽 라이브러리와 아주 유사한 API 3D 그래픽 라이브러리이다. Mesa 라이브러리는 OpenGL을 실제로 구현한 것은 아니지만 OpenGL API를 사용하고 구문을 모방하여 OpenGL에서 실행되는 어플리케이션을 코드 변화 없이 실행 시킬 수 있다. 디스플레이 부분은 Visual C++의 그래픽 라이브러리를 이용하여 구현하였다.

그림 8은 실험 데이터인 Teapot 메쉬에 대하여 Geometry 과정을 수행한 실험 영상이다. 투영 변환은 원근 투영 효과를 사용하였으며, Viewport 변환은 실험 GUI(Graphic User Interface)의 크기에 맞게 고정시켜 수행하였다. Teapot데이터는 폴리곤의 개수가 2256개로 구성되어 있으며 원본 데이터는 빛 요소가 하나도 없는 상태이다.

설계된 부동소수점 연산기를 이용한 Geometry 프로세서의 성능은 표 7에 나타내었다. 또한 설계된 부동소수점 연산기의 FPGA(Xilinx Vertex2)의 합성결과를 표 8에 나타내었다.

V. 결 론

본 논문에서는 실시간 3D 가속을 효과적으로 하기 위해 기하학 처리 과정에 적합한 부동 소수점 연산기를 설계하였다. 설계한 부동 소수점 연산기는 IEEE-754 단정도 형식을 지원하도록 하여 기하학 처리에 적합하게 하였고 설계한 부동 소수점 연산기는 Xilinx-Vertex2에서 부동소수점 덧셈/곱셈기는 100 MHz, 부동소수점 NR 역수 계산기는 120 MHz, 부동소수점 멱승기는 200 MHz, 부동 소수점 역 제곱근 연산기는 120 MHz의 동작 주파수를 각각 확인 하였다.

표 9. 제안한 음영 처리 프로세서 성능 비교
Table 9. Performance comparison of a proposed lighting processor

Processor	Lighting Performance
SH4(Hitachi)[16]	500K vertices/sec @200MHz
SATINE (KAIST)[17]	400K vertices/sec @200MHz
Z3D(Mitsubishi)[18]	250K vertices/sec @30MHz
Proposed Geometry Lighting Processor	1M vertices/sec @80MHz

또한 설계된 부동소수점 연산기를 이용하여 기하학 프로세서를 구현하여 실제 3D 데이터를 처리하도록 하여 설계된 부동소수점 연산기의 기능을 확인하였다. 본 논문에서 설계 부동소수점 연산기를 지오메트리 프로세서의 음영처리에 적용하였을 경우 타 프로세서와 성능을 비교하면 표 9와 같다.

참 고 문 헌

[1] David H. Eberly, "3D Game Engine Design," Morgan Kaufmann, May, 2001.
[2] L. Garber, "The wild world of 3D graphics chips," IEEE Computer, vol. 33, no. 9, pp. 12-16, Sep. 2000.
[3] Udo Flohr, "3-D for Everyone," Byte, pp.76-88, Oct. 1996.
[4] Foley, Van Dam, Feiner, Hughes, "Computer Graphics Principle and Practice," Addison & Wesley, June 1996.
[5] Tomas Akenine-Moller, Eric Haines, "Real-Time Rendering," AK Peters, Dec 2002.
[6] J. G. Torborg, "A Parallel Processor Architecture for Graphics Arithmetic Operations," Proceeding of SIGGRAPH '87, pp.197-204, 1987.
[7] Behrooz Parhami, "Computer Arithmetic : Algorithms and Hardware Design," Oxford University Press, pp.128-211, 2000.
[8] Jeong, Woo Kyeong "A SIMD-DSP/FPU for High-Performance Embedded Microprocessors" Master's Theis, 2002.
[9] Jong-Chul Jeong, Woo-Chan Park, Woong Jeong, Tack-Don Han, Moon-Key Lee " A Cost-Effective Pipelined Divider with a Small Lookup Table" IEEE Transaction, pp489-495,

2004.
[10] Hyun-Chul Shin, Jin-Aeon Lee, Lee-Sup Kim, "A Hardware Cost Minimized Fast Phong Shader," IEEE Transaction, pp297-304, 2001.
[11] Hyun-Chul Shin, "A Hardware Implementation of fast Phong Shading using Taylor series approximation", Master Thesis, KAIST, Dec, 1997.
[12] http://www.ssec.wisc.edu/~brianp/ Mesa.html.
[13] Michael J, Schulte, Kent E, "High-Speed Inverse Square Roots," 14th IEEE Symposium, pp124-131, 1999.
[14] R. W. Stewart, R. Chapman, and T. Durrani, "The Square Root in Signal Processing," in Proceedings of Real-Time Signal Processing, pp.89-100, 1989.
[15] Cheol-Ho Jeong, "Design of an Effective Control and Execution Method for Geometry Engines and Rasterizers within Embedded 3D Graphics Accelerators," Phd Thesis, Yonsei University, Dec, 2003.
[16] F. Arakawa, O. Nishii, K. Uchiyama, and N. Nakayama, "SH4 RISC multimedia microprocessor," *IEEE Micro,* vol. 18, no. 2, pp.26-34, April 1998.
[17] Ju-ho Sohn, "Design and Optimization of Geometry Acceleration for Portable 3D Graphics," Master Thesis, KAIST, Dec, 2002.
[18] Masatoshi Kameyama, Yoshiyuki Kato, Hitoshi Fujimoto, Hiroyasu Negishi, Yukio Kodama, Yoshitsugu Inoue, Hiroyuki Kawai, "3D graphics LSI core for mobile phone Z3D," *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware,* pp.60-67, 2003.

저 자 소 개

김 명 환(정회원)
2000년 연세대학교 전자공학과 석사
2002년~현재 서경대학교 컴퓨터공학과 박사과정
1977년~1978년 한국화약 고려시스템(주)
1978년~1983년 KORVAC(한국 유니백)(주)
1883년~2004년 (주)청호컴넷 상무이사
2005년~현재 H&C Technology 부사장
2005년~현재 (주)인트정보시스템 기술고문
2002년~현재 서경대학교 전자공학과 겸임교수
<주관심 분야> Computer Architecture/Analog & Digital Circuit Design/Radio Communication

이 광 엽(정회원)
1985년 8월 서강대학교 전자공학과 학사.
1987년 8월 연세대학교 전자공학과 석사.
1994년 2월 연세대학교 전자공학과 박사.
1989~1995년 현대전자 선임연구원
1995년~현재 서경대학교 컴퓨터공학과 부교수.
<주관심분야 : 마이크로프로세서, 암호프로세서, Embedded System, 3D Graphics System>

조 한 진(정회원)
1982년 한양대학교 전자공학과 졸업.
1987년 New Jersey Institute of Technology 전기공학과 공학석사.
1992년 University of Florida 전기공학과 공학박사.
1992년~현재 한국전자통신연구원 기반기술연구소 SoC설계연구부 팀장
<주관심분야 : 통신, 컴퓨터, 신호처리, 반도체>

오 민 석(정회원)
2003년 서경대학교 컴퓨터공학과 학사 졸업
2005년 서경대학교 컴퓨터공학과 석사 졸업.
2005년~현재 엠텍비젼(주)연구원
<주관심분야 : 저전력 마이크로프로세서, Computer arithmetic, 3차원 그래픽 가속기>

김 원 종(정회원)
1989년 2월 전남대학교 전자공학과 공학사
1982년 2월 한양대학교 전자공학과 석사
1999년 2월 한양대학교 전자공학과 박사
1999년 3월~2000년 3월 한양대학교 공학기술 연구소 선임연구원
2000년 4월~현재 한국전자통신연구원 기반기술 연구소 SoC설계연구부 선임연구원
<주관심분야 : VLSI CAD, Embedded System 설계, SoC 설계 등>