3D 그래픽 가속기를 위한 검증시스템의 설계 및 구현

송인석, 하진석, 김명환, 이광엽*, 조태헌**
서경대학교 컴퓨터공학과*
C&S technology**

A Design of a Verification System for a 3D Graphic Geometry Engine

InSeok Song, JinSeok Ha, Myung hwan Kim, KwangYeob Lee*, TaeHyun Jo**
Department of Computer Engineering
Seokyeong University*
C&S technology**
E-mail : *inseok007@skuniv.ac.kr

Abstract

The geometry stage, which performs the transformation and lighting operations of vertices, became the critical part in 3D graphics pipeline. In this paper, we have planned and designed the Geometry Processor for the better and more efficient way to process the real-time 3D using the floating point unit. We also designed a verification system for Geometry engine. It is implemented with Xilinx-Virtex2 and Visual C++.NET. In the Synopsis, we confirmed 100 MHz performance and 137107 cell area of Geometry Engine.

I. Introduction

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. 현재는 3D 그래픽이 일반화됨과 동시에 실시간 3D 그래픽 기술의 등장으로 그 활용 분야가 대폭 넓어지게 되었다.[1],[2].

본 논문에서는 Mobile 시스템에서 더욱 향상된 실시간 3D 가속을 제공하기 위한 효과적인 Geometry Engine 구조를 설계하고, 이를 검증하기 위한 시스템을 구축하였다.

II. Geometry Engine

2-1. Geometry 처리 과정

Geometry Engine 은 점에 대한 좌표의 정보, 좌표의 Normal Vector, 색깔 정보와 연산을 위한 파라메터들을 입력 받으면서 시작된다. Geometry Engine 은 이들 입력 데이터들을 이용하여 객체들에 대한 계산을 해주는 Engine 이다. Geometry Engine 은 그림 1 에서 보여지는 것과 같이 크게 Transformation 부분과 Lighting 부분으로 분류할 수 있다. Transformation 부분에서는 모델/시야 변환, 투영, 클리핑, 화면 매핑으로 다시 분류가 되며 Lighting 부분은 실제 빛 처리를 하기 전에 빛 처리에 필요한 요소들을 미리 계산 하는 부분과 실제 빛 처리 수식을 수행하는 부분으로 분류된다.[3],[4]

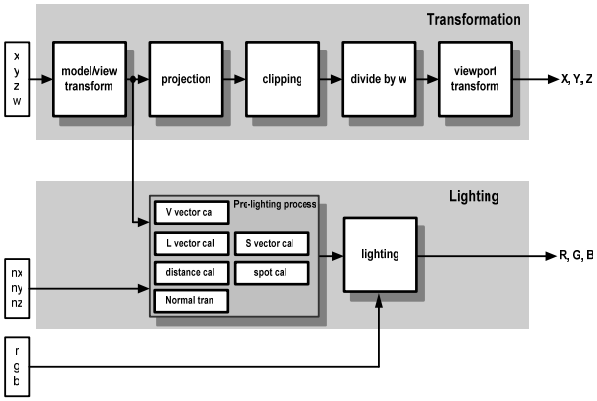

그림 1 3D Graphic Geometry 처리과정

2-2. Geometry 처리에 필요한 연산기

Geometry 단계에서 사용되는 입력데이터는 대부분이 Floating Point 형식이며 Translation 과 Lighting 을 처리하기 위해서는 기본적인 사칙 연산 외에 더 복잡한 연산들이 요구된다.

Transformation 과정에서는 모델의 좌표를 변환하는 과정으로서 4*4 행렬과 4*1 행렬의 곱셈 연산이 주를 이루게 된다. 행렬의 곱셈 과정은 덧셈기와 곱셈기로 이루어지고, 그 외에 View Volume 안에 있는지의 여부를 검사하는 비교연산, perspective projection 시 모델의 좌표 동차화 과정에서 역수기가 필요하게 된다. Lighting 과정은 Transformation 과정보다 더 복잡한 연산을 요구한다. Transformation 과정과 마찬가지로 덧셈, 곱셈기가 주를 이루며, Distance 과정에서 역수기가 추가로 필요하다. 역제곱근은 빛 처리에서 계산된 벡터들을 단위벡터로 만들어주는 과정에서 필요하고, 마지막으로 멱승기는 집중 조명광 계산과 정반사 성분 계산에 필요하게 된다. .[5],[6] ,[7],[8],[9]

본 논문에서는 Mobile 환경을 목표로 하고 있기 때문에 IEEE-754 표준에 32 bit floating point format 을 24 bit 형식으로 수정하였고, Lighting 과정에서 쓰이는 0~1 사이의 값으로만 이루어지는 일부 데이터들은 16 bit Fixed Point Format 으로 구성하였다.

2-3. Geometry Engine Architecture

Geometry Engine 은 Hardwired decoder 구조로 설계되었다. Hardwired 구조의 특징은 명령어 없이 입력이 들어가면 바로 출력이 나오는 구조이다. 명령어 중심의 프로세서보다는 유연성이 떨어지지만 명령어 구조에서 빈번하게 일어나는 Load/Store 과정이 없게 되고 중간 저장공간을 따로 둘 필요가 없다. 상대적으로 단순한 로직만을 실행할 수밖에 없지만 빠르고 메모리 용량을 줄일 수 있다는 장점을 가지고 있다. Geometry Engine 에서 Transformation 부분이나 Lighting 부분이나 똑같은 패턴의 연산이 계속된다는 것을 이용해 Hardwired 구조를 채택할 수 있었다. 다음 그림 2 는 Hardwired 구조로 만들어진 Geometry Engine 의 데이터 흐름도이다.

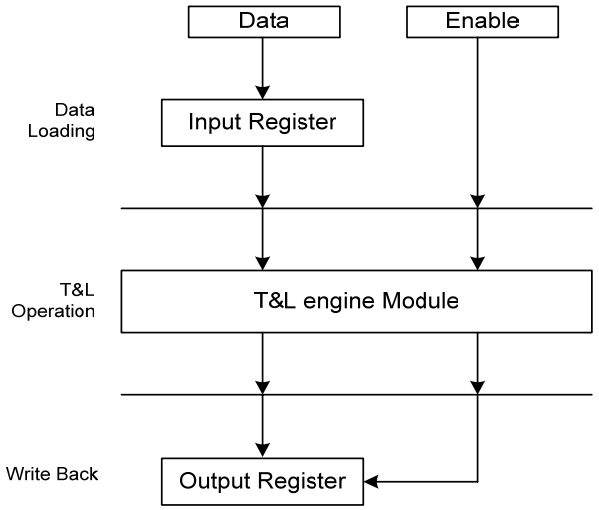

그림 2 Data Flow of Geometry Processor

Geometry Engine 은 Enable 신호와 함께 Input Register 에 담겨 있던 데이터들을 받아들이면서 동작하게 된다. Pipeline 구조에 따라 올바른 Output 데이터들이 나오는 시점에 Enable 신호도 함께 나오게 되어 Enable 신호가 Output Register 에 Write 신호가 된다.

2-4. Synthesis Result of Geometry Engine

설계된 Geometry Engine 의 Size 를 알기 위해 Magnachips 0.35 공정에서 합성하였다. Clock 주기는 100 MHz 로 설정하였다. 아래 표 1 은 Geometry Engine 에 사용된 연산기들과 Geometry Engine 의 전체 구조에 대한 총 Total Area 의 크기이다.

표 1 Synthesis Result of Geometry Engine

Unit	Pipeline	Total Area
Adder/Subtractor	1 Stage	2809.108
Multiplier	1 Stage	2527.327
Riciprocal	3 Stage	9335.437
Riciprocal square root	4 Stage	6040.750
Geometry Engine	8 Stage	137106.969

III. Verification System

3-1. Verification System Environment

본 논문에서는 Geometry Engine 을 검증할 수 있는 Verification System 를 설계하였다. 설계한 Geometry Engine 은 verilog HDL 을 이용하여 설계하고, Mentor 사의 ModelSim 을 이용하여 회로기능을 확인한 후, Dynalith 의 iProve Vertex2 Xilinx 300 만 게이트에 porting 한 후 Visual C++.NET 을 이용하여 전체적인 GUI 시스템을 구축하여 검증하였다.

다음 그림 3 는 설계한 Verification 의 기본 환경이다.

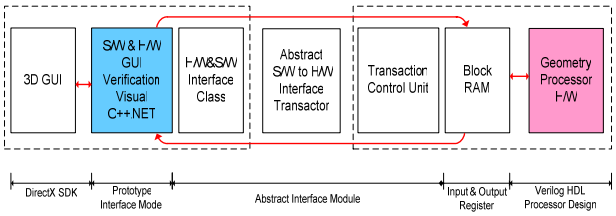

그림 3 Verification System Environment

3-2 Transactor

Visual C++.NET 과 Geometry Engine 을 연결하는 Transactor 모듈은 iProve 를 이용하여 설계하는 파트이다. Visual C++.NET 에서 float 변수는 32 bit format 을 가진다. 이에 반해 Geometry Engine 에는 24 bit Floating Point format 과 16 bit Fixed Point format 을 가지고 있어 이를 Transactor 에서 변환을 해주는 코드를 사용하였다.

Transactor Module 은 3 개의 port 와 8K Block RAM 을 가지고 있다. 이를 이용하여 Geometry Engine 에 필요한 Input 들을 Read Port 를 통해 Block RAM 에 저장하고 연산을 통해서 나온 Geometry Engine 의 Output 들을 다시 Block RAM 에 저장하여 Write Port 를 통해 S/W 로 값을 전달할 수 있게 된다. Command Port 는 Read 나 Write 를 할 Block RAM 의 주소값을 제공해주는 Port 로 사용하였다. Transactor 의 Block RAM 이 Geometry Engine 의 Data Flow 에서 Input Register 와 Output Reigister 의 역할을 하게 된다.

다음 그림 4 은 Transactor 모듈이다.

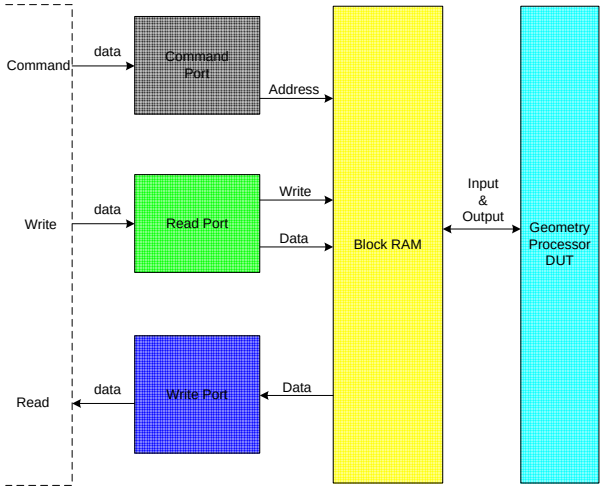

그림 4 Transactor Module

3-3 GUI System

그래픽의 특성상 파형을 통한 검증이 아니라 실제 눈으로 보고 Transformation 과정과 Lighting 과정이 현실세계처럼 자연스러운지를 확인해야하기 때문에 직접 눈으로 확인할 수 있는 GUI system 의 구축이 필요하게 된다.

Testbench 의 역할을 하게 되는 Visual C++.NET 은 전체적인 GUI System 구축과 Transactor 의 3 가지 Port 를 컨트롤을 하게 된다.. 다음 함수들이 Transactor 의 3 가지 Port 를 컨트롤하는 명령어이다.

```
m_iProve.SendCommand(&command,1);
        //use command port
m_iProve.Write((float*)matrix16,4);
        //use write port
m_iProve.Read((float*)matrix16,4);
.       //use read port
```

32 Bit 의 크기를 가지는 commad 변수는 Block RAM 의 address 및 Transactor 의 Mode 를 설정해준다. Write 나 Read 에는 실제 Write 나 Read 를 할 데이터와 보내줄 데이터의 개수를 Parameter 로 전달하게 된다. 이 3 가지 명령어들을 조합하여 Geometry Engine 의 Input 과 Output 을 컨트롤 하게 된다.

Visual C++.NET 에서 전체적인 GUI System 과 Display 하게 될 Model 을 생성한다 이 생성된 Model 에는 Model 의 좌표값과 RGB 값들이 배열로 정리되어 있다. 이 좌표값들과 RGB 값들을 Transactor 로 넘겨주고

Geometry Engine 에서 수행되어 Transactor 로 전해져 오는 좌표값들과 RGB 값들을 받아들여 이 값들을 GUI System 으로 적용해주는 구조를 갖고 있다.

기본적으로 타이머를 이용해 자동으로 Scaling 과 Rotation 을 반복하도록 설계했고, Translation 의 경우 사물의 중점에서 보이는 아이콘을 이동시켜주면 물체도 따라서 이동하도록 설계하였다. 이런 과정들 중에서도 Lighting 과정의 증명을 위해 광원을 한 개 두어 Transformation 하는 과정에 빛의 명암효과를 보여주도록 설계하였다.

다음 그림 5,6 은 Visual C++.NET 을 통해 만든 Demo 화면이다. 그림 5 는 Translation 만 된 상태를 캡쳐하였고, 그림 6 에서는 Scaling 값이 크고 Rotation 된 상태의 캡쳐 화면이다.

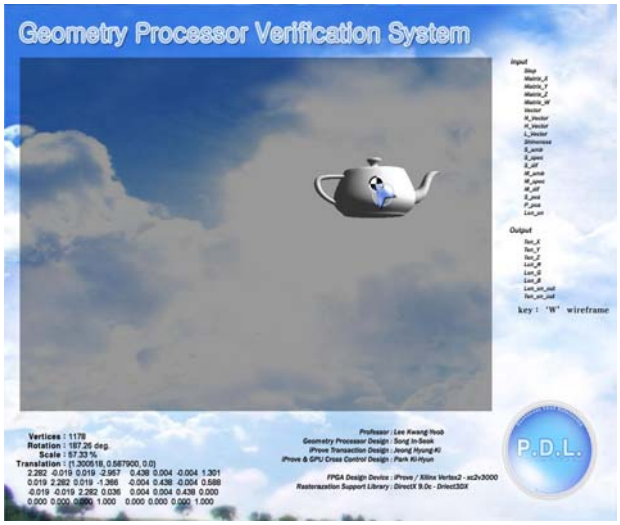

그림 5 결과 화면(1)

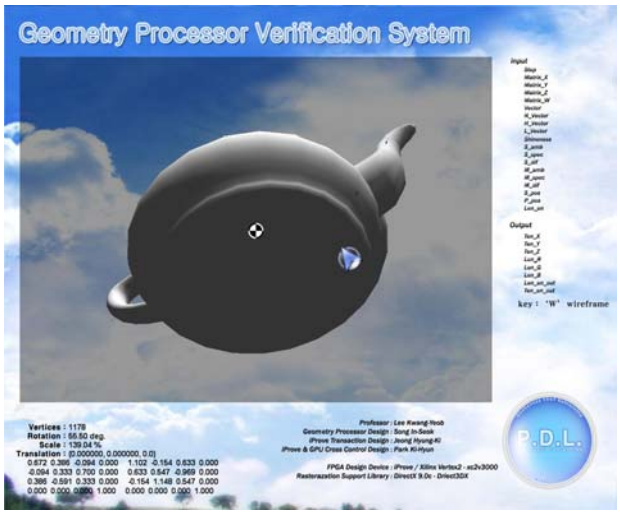

그림 6 결과화면(2)

IV. Conclusion

본 논문에서는 설계된 Geometry Engine 을 검증하기 위한 Verification System 을 구축하였다. 검증하기 위한 Geometry Engine 은 부동소수점 기반 24 비트 구조로 설계하였으며 효과적인 제어를 위해 Hardwired 구조를 채택하였다. 제작된 검증시스템은 Visual C++.NET 환경으로 설계된 Geometry Engine 을 효과적으로 검증하였다.

* 본 논문은 IT-SOC 사업단의 지원과 IDEC 장비를 활용하였습니다.

참고문헌

[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] Foley, Van Dam, Feiner, Hughes, “Computer Graphics Principle and Practice," Addison & Wesley, June 1996.

[4] Tomas Akenine-Moller, Eric Haines, “Real-Time Rendering,” AK Peters, Dec 2002.

[5] Behrooz Parhami, "Computer Arithmetic : Algorithms and Hardware Design," Oxford University Press, pp.128-211, 2000.

[6] Jeong, Woo Kyeong "A SIMD-DSP/FPU for High-Performance Embedded Microprocessors" Master's Theis, 2002.

[7] 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.

[8] Hyun-Chul Shin, Jin-Aeon Lee, Lee-Sup Kim, “A Hardware Cost Minimized Fast Phong Shader," IEEE Transaction, pp297-304, 2001.

[9] Hyun-Chul Shin, “A Hardware Implementation of fast Phong Shading using Taylor series approximation", Master Thesis, KAIST, Dec, 1997.