2005년도 대한전자공학회 추계종합학술대회 제28권 제2호

Mobile 기기에 적합한 Vertex Shader 의 설계 및 구현

정형기*, 남기훈*, 이광엽*, 허현민**, 이병옥**, 이주석**
*서경대학교 컴퓨터공학과, **엠텍비젼®(주)

A Design of a Vertex Shader for Mobile Devices

HyungKi Jeong*, KiHun Nam*, KwangYeob Lee*, Hyun Min Hur**, ByungOk Lee**, James Lee**
*Department of Computer Engineering Seokyeong University, **MtekVision® Co., Ltd.

E-mail : *ks2ii@hanmail.net

Abstract

In this paper, we designed a vertex shader for mobile devices. Proposed Vertex shader is compatible with the OpenGL ARB & DirectX 8.0 Vertex Shader 1.1 and is organized of modified IEEE-754 24 bits float point SIMD architecture. All float point arithmetic unit process 1 cycle operation with 100Mhz frequency more. We made a vertex shader demo system with Xilinx-Virtex II and get synthesis result that confirm 11M gates size at TSMC 0.13um @ 115MHz.

1. Introduction

3D 가속 칩에 있어서 기존에는 빠른 성능이 주 목표였으나 오늘날에 이르러 칩 제작 공정과 설계 툴의 발달로 인해 빠른 성능은 물론 융통성 있는 3D 칩 설계가 요구되고 있다. 3D 그래픽에서도 그 중 하나인 프로그래밍 가능한 Vertex Shader Core 설계를 연구하였다.

Vertex Shader[1]는 Geometry 연산처리를 프로그래밍 가능한 Core로써 다양한 3D 그래픽 벡터 연산을 위해서 4 개의 float point 연산을 동시에 처리할 수 있도록 하는 SIMD 구조로 이루어져 있다. 현재까지 존재하는 프로그래밍 가능한 3D Geometry 처리 표준은 DirectX의 Vertex Shader와 OpenGL의 Vertex Program이 있어 각각 Macro Assembly 레벨의 표준을 제안하고 있다. 각 제안한 표준에 의해 만들어진 결과물은 기존의 고정된 3D Geometry 처리인 T&L(Transform & Lighting) 보다 훨씬 뛰어난 그래픽 효과를 보이고 있다.

본 논문에서는 설계한 Vertex Shader 의 Software 에 대한 인터페이스 구조와 각 표준(DirectX, OpenGL)에 대한 지원 방법을 설명하고, 또한 향후 minor upgrade 의 가능성에 대하여 나타내었다.

2. Vertex Shader Architecture

본 논문은 DirectX 8.0 Vertex Shader 1.1 과 OpenGL ARB[5]를 기준으로 설계하였다. 두 표준의 기본적인 구조는 vector 연산을 위한 4 float point SIMD를 제안하고 있으며 OpenGL ARB가 Vertex Shader 1.1 을 확장한 표준이므로 OpenGL ARB를 기준으로 설계를 하였다. 단 Mobile 환경을 목표로 삼고 있어 변형된 24bits float point 데이터 처리를 하고 있다. OpenGL ARB에는 총 27 개의 Instruction을 제안하고 있는데 크게 ALU Operation 과 Special Function으로 구별 지을 수 있다.[4]

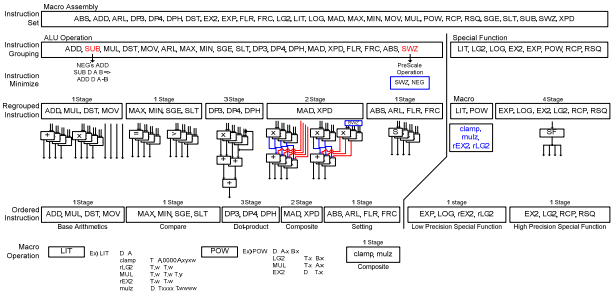

그림 1. Instruction Analysis

그림 1 과 같이 최대 3 단계의 연산과정을 거치며 Macro 명령어 LIT, POW를 제외한 모든 명령어는 1

Instructioin / 1 cycle의 기본적인 연산 처리능력과, Special Fuction 명령어 2^n(EXP, EX2), Log_2N(EX2, LG2), 1/N(RCP), 1/sqrt(N)(RSQ)가 모두 1 cycle에 처리되어 고속의 float point SIMD 연산을 가능하게 한다.[7, 8]

ALU와 Special Function은 동시 처리 할 수 있으며, Core가 사용하는 Register는 표 1 과 같이 정의 한다.[6]

Formal Registers

Spec.	OpenGL 2.0 & DirectX 8.1(vertex shader 1.1)
Max. Number of Inputs	16 vertex streams
Max. Program Length	128 Instruction slots (64 bits)
Temporary Registers	12 4D vectors
Constant Registers	96 4D vectors
Address Registers	1 scalar (x component of 4D vector)
Output Registers	13 vectors

Informal Registers

Spec.	Extended Extra
Temporary Registers	+4 4D vectors
Micro-Operation Look-Up Table	64 Micro Operations (32 bits)

표 1. Registers

레지스터는 표준[1]을 그대로 지원하고 있으며, Core의 융통성 위하여 내부 Temporary Register 개수가 4 개 추가되며 별도의 MO-LUT(Micro-Operation Look-Up Table)을 가진다. MO-LUT는 총 64 개의 LUT로 구성되어 있는데 각 레지스터는 명령어당 세부 동작을 정의하는 Micro-Operation으로써 소프트웨어에서 설정하게 된다.

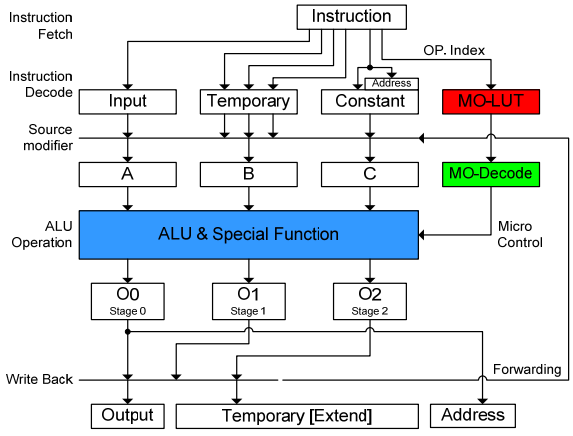

그림 2. Micro-Operation Flow

이 LUT 의 내용에 따라 Vertex Shader 의 구체적인 동작 방식을 정의하게 된다. 이렇게 별도로 변경 가능하도록 하게 한 것은 ALU 와 Special Function 의 연산 방법이 ALU 가 각 xyzw 마다 17 개, Speical Function 의 6 개이므로 동시에 처리한다고 할 때, 이를 조합할 수 있는 연산 방법은 총 1 만 개가 넘는다. 이것을 고정 시키는 것은 다양한 연산 방법을 제약하게 된다. 또 이 내용을 Instruction bit field 에 모두 포함 시키기엔 비트 수가 많이 요구되므로 Instruction bits field 의 6 비트만을 차지하고 MO-LUT 에서 이것을 Index 로 입력하여 다시 Micro-Operation 을 얻어낸 후 ALU 와 Special Function 에 전달하여 준다.

이 방법은 명령어가 다양한 연산 조합이 이루어지는 Core 에 적합하며, 융통성 있고, 향후 소프트웨어에 의한 Core 기능 추가를 할 수 있으나 Core 가 초기에 명령어 설정을 위해 다른 CPU 의 도움을 받아야 하므로 Stand Alone CPU 의 경우 부적합하다. 본 논문의 Core 는 CPU 의 그래픽 처리 능력을 향상시키는 보조기능을 하는 GPU 이므로 적합한 적용 방식이다.

MO-LUT 에 담겨질 명령어 내용을 표준에 호환하도록 정의하였고, 이를 표 2 와 같이 정리 할 수 있다.

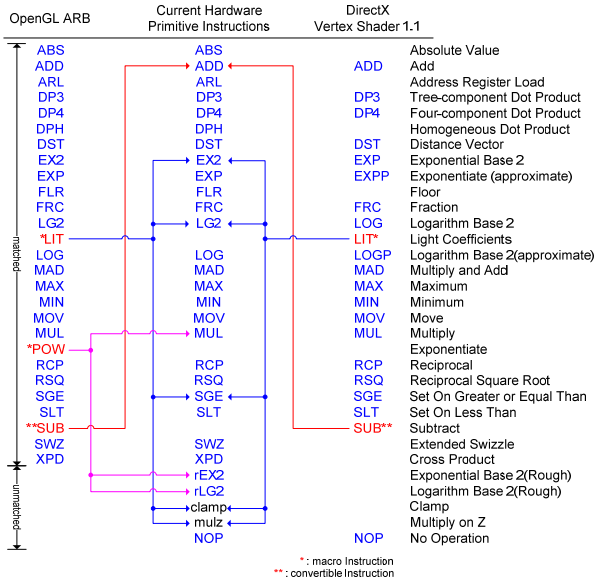

표 2. Primitive Assembly Level Compatibility

MO-LUT 는 최대 64 개까지 확장 가능하므로 S/W 에 의해 다양한 명령어가 향후 추가하여 업그레이드 할 수 있다.

Vertex Shader의 동작 구조는 n개의 점을 반복해서 주어진 명령어에 따라 수행하여 결과를 다음 단계에 넘겨주게 되는데, 이 때 각 점의 프로세스를 끝내기 전에 다음 점의 프로세스의 시작이 동시에 이루어지는 Process Convolution이 발생한다. 이를 피하기 위해 delay를 둔다면 단지 몇 cycle을 소요하는 것처럼 보이지만 수 만개의 점을 처리하는 Vertex Shader에서는 1 클럭만 delay 시킨다고 해도 점의 개수만큼 cycle이 더 소요되는 낭비가 발생한다. 본 논문의 Vertex Shader Core는 n개

의 점에 대하여 연속적으로 process를 처리하는 n-finite 기술을 구현하고 있다.[2, 3]

또 하나의 문제점은 설계된 Vertex Shader Core 의 적용에 있어서 상당히 많은 레지스터들을 포함하고 있는데 설계상 전부 Core 에 포함시키느냐 하는 문제이다. 여기에서는 Temporary / Address Register 만을 Core 설계에 포함하고 나머지 Input / Output / Constant Register, MO-LUT 는 외부의 인터페이스를 두어 설계한다. 이것은 메모리로 사용될 가능성이 있는 레지스터에 대하여 GPU 통합 개발자가 복잡한 Core 내부 설계를 변경하지 않고 Core Interface 만을 가지고 설계 가능하게 하기 위함이다.

통합 개발자는 Vertex Shader Core 내부의 구조를 이해할 필요가 없고 그림 3 의 인터페이스 구조만을 이해하면 된다.

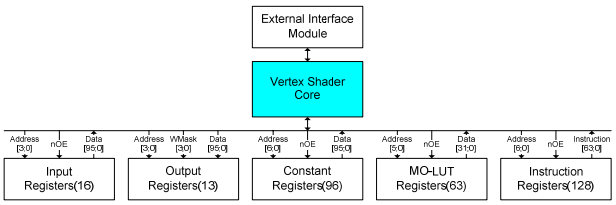

그림 3. Vertex Shader Core Register Interface

3. Implementation

제안하는 Vertex Shader Core 는 Verilog 로 기술하였으며, 파라메터화 된 Verilog 모델을 선언하여 내부 구조를 쉽게 변경 가능하도록 설계하였다.

디자인된 모델을 PC 기반의 FPGA 모듈을 응용하여 검증 데모 시스템을 제작하였고, Synopsis 툴을 사용하여 게이트 면적과 동작속도를 확인하였다.

3.1 Vertex Shader Evaluation

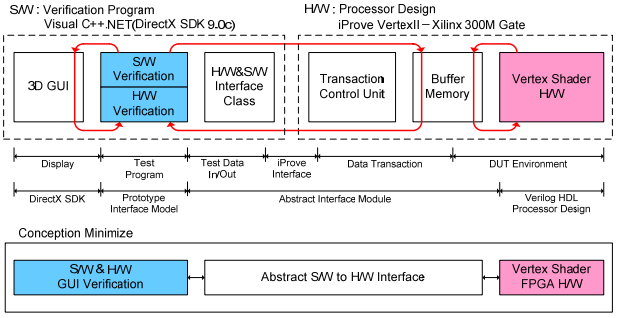

그림 4. Evaluation Process

본 논문에서 Core 동작을 검증하기 위하여 Dynalith 사의 iProve 를 이용한다. iProve 에는 Xilinx Virtex2 300M Gate 가 장착되어 있으며 설계한 Vertex Shader 를 FPGA 레벨에서 검증을 하였다.

FPGA 에서 얻은 결과를 화면에 DirectX 를 이용하여 출력하여 검증하는데, 설계된 H/W 와 똑같이 24bits float point 연산을 수행하는 S/W 에뮬레이션 모듈과 결과를 비교할 수 있도록 하였다. 그림 5 는 각 S/W, H/W 모듈과 DirectX 의 Rendering Unit 과 어떻게 연결되는지 보인다.

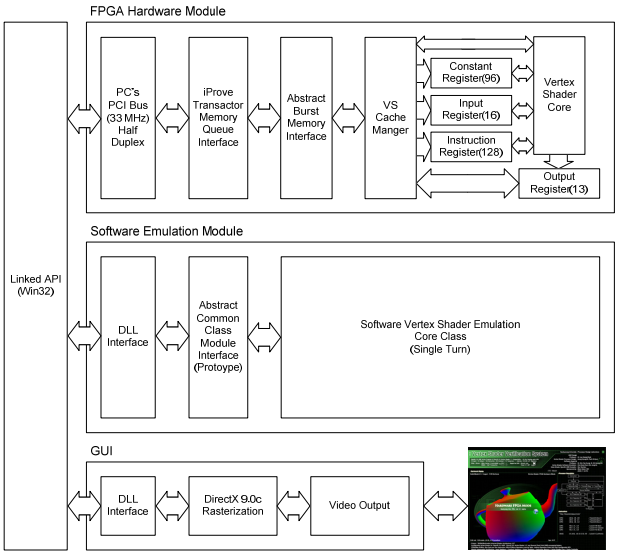

그림 5. Software /Hardware Module Interface

그림 5 의 설계 구조와 같이 PC 기반의 FPGA 검증 시스템을 제작하여 Vertex Shader 명령어를 활용한 검증 결과의 일부를 그림 6, 7 과 같이 확인하였다.

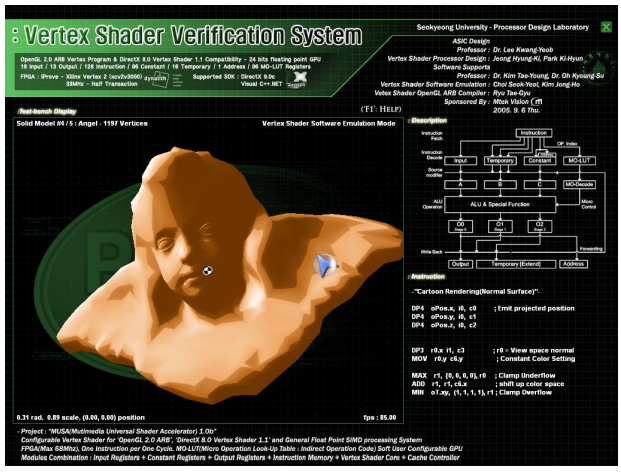

그림 6. Normal Cartoon Rendering

[Normal Cartoon Rendering]

```
DP4 oPos.x, i0, c0      ; Emit projected position
DP4 oPos.y, i0, c1
DP4 oPos.z, i0, c2

DP3 r0.x, i1, c3        ; r0 = View space normal
MOV r0.y c0.y           ; Constant Color Setting

MAX r1, {0, 0, 0, 0}, r0; Clamp Upper Clipping
ADD r1, r1, c6.x                  ; shift up color space
MIN oT.xy, {1, 1, 1, 1}, r1       ; Clamp Overflow
```

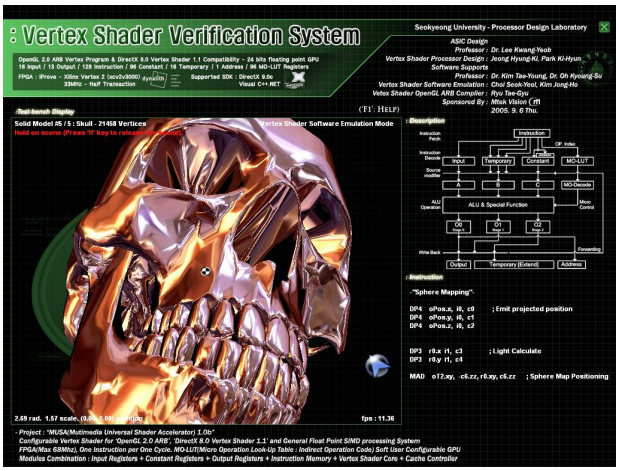

그림 7. Sphere Mapping

[Sphere Mapping]

```
DP4 oPos.x, i0, c0      ; Emit projected position
DP4 oPos.y, i0, c1
DP4 oPos.z, i0, c2

DP3 r0.x, i1, c3        ; Light Calculate
DP3 r0.y, i1, c4

; Sphere Map Positioning
MAD oT2.xy, -c6.zz, r0.xy, c6.zz
```

3.2. Synthesis results

VS_Core	Description
Source	Source_Modifier_O_A_88 (FF)
Destination	WReg_Output<94> (PAD)
Source Clock	MCLK falling
Combinational area	467848.406250
Noncombinational area	89363.171875
Total cell area	557137.625000
Approximate gate count	111427.525000

표 3. Synthesis result : Maximum Path & Area Report

표 3 과 같이 Synopsys 의 Synthesis 결과를 보면 TSMC 0.13um 공정에 115Mhz 동작 스피드 설정에서 약 11 만 게이트의 적은 면적을 차지함을 알 수 있었다.

4. Conclusion

설계한 Vertex Shader 는 프로그램이 용이한 범용 SIMD Float Point Processor 의 특징을 가지고 있다. 특히 MO-LUT 의 설정이 3D Geometry 처리에 알맞게 구성되어 있어 컨텐츠 개발에 유용하다. 설계된 프로세서는 24 bits Float Point SIMD 구조로 체택하여 모바일폰에서 초당 15 프레임 이상의 Shading 효과를 제공하는 것이 입증되었다.

Acknowledgements

본 논문은 IT-SoC 사업단의 지원과 IDEC 의 지원장비로 작성하였음.

References

[1] Microsoft MSDN Vertex Shader 1.1 (http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c/directx/graphics/reference/Shaders/VertexShader1_1/vertex.asp)

[2] Ujval J. Kapasi, William J. Dally, Scott Rixner, John D. Owens, and Brucek Khailany. The Imagine Stream Processor. In *Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors* (ICCD'02).

[3] Erik Lindholm, Mark J Kilgard, Henry Moreton. A User-Programmable Vertex Engine. In *ACM SIGGRAPH 2001, 12-17 August 2001*.

[4] James C. Lelterman. Learn Vertex and Pixel Shader Programming with DirectX® 9. *Wordware Publishing, Inc*.

[5] Mark Segal, Kurt Akeley. The OpenGL Graphics System : A Specification (*Version 2.0 - October 22, 2004*). (http://www.opengl.org/)

[6] Michael Doggett*. Programmability Features of Graphics Hardware (*ATI April 23, 2002*).

[7] 김재우. 모바일 3D 그래픽을 위한 음영처리 프로세서 설계 (*서경대학교 대학원 2004 년 12 월*).

[8] 박용진. 역함수를 이용한 작은 룩업테이블을 갖는 조명처리 지수 연산 유닛. (*연세대학교 대학원 2004 년 12 월*).