3D 그래픽 프로세서에서 효율적인 명령어를 위한 가변길이 명령어 설계

*김우영, *이보행, *이광엽, 곽재창

*서경대학교 컴퓨터 공학과, 서경대학교 컴퓨터 과학과

Design of a Variable-Length Instruction for the Effective Usability Instruction in 3D Graphics Processor

Woo-Young Kim1, Bo-Haeng Lee1, Kwang-Yeob Lee1, Jae-Chang Kwak2

1Dept. of Computer Engineering Seokyeong University,

2Dept. of Computer Science Seokyeong University

요 약

최근 OpenGL ES 2.0이 개정됨에 따라 모바일 기기에 Shader 3.0모델을 지원 가능한 프로세서가 요구된다. 이 쉐이더 3.0 모델의 지원과 관련하여 명령어의 길이의 증가가 필요하고, 이는 메모리 용량의 증가를 초래한다. 본 논문에서는 가변길이 구조와 유닛구조를 채택한 새로운 명령어 구조를 제안한다. 이 명령어 구조는 쉐이더 3.0 모델을 지원하고 명령어 필드 낭비를 줄일 수 있도록 최대 4개의 32비트 유닛 명령어가 가변적으로 조합되어 수행된다.

ABSTRACT

Recently, Khronos institude OpenGL ES 2.0 API for support Shader 3.0 model that can possible variable graphic processing. For this reason, the mobile device have need of supporting processor for a shader 3.0 model. We should extend instruction's length to support OpenGL ES 2.0 API, so we need more memory size. In this paper, we propose a new instruction form that adopted variable length and unit instruction architecture. This proposed instruction architecture that support to Shader 3.0 model has consist of 32bit unit instructions up to 4 which can be combined for embellishing each other. Therefore, it can execute flexible instruction combination and reduce waste of instruction fields.

키워드

Variable Length, Instruction Words, Shader Instructions

Ⅰ. 서 론

프로세서의 발전과 그래픽 기술의 발전에 따라 더욱더 사용자는 현실적이고 화려한 효과를 원하기 시작했다. 이 욕구를 충족하기 Programble한 쉐이더를 사용하기 시작했다. 하지만 복잡한 구조의 쉐이더를 사용하기 위해서 기존의 명령어 방식인 SIMD 명령어 체제로 사용하게 된다. 일반적인 SIMD 프로세서는 하나의 명령어 필드에 연산기와 레지스터들의 사용에 대한 설명으로 기술되어 있으며, 빠른 명령어 디코딩을 위해서 대부분의 각 필드의 영역은 고정되어 있다. 이 OpenGL ES 2.0 이나 DirectX Shader 3.0 버전부터는 기존의 기능에서 Dynamic Branch / Looping 을 포함한 다양한 기능이 추가됨에 따라서 명령어에 표현되어야 할 정보들이 많아지게 되고, 이는 명령어 길이의 증가를 초래한다. 이로 인하여 기존의 고정된 명령어를 사용 하게 되면 하나의 Instruction은 쉐이더를 만족하기 위해 VLIW(Very Long Instruction Words) 방식을 사용해야 한다.

VLIW 형식을 사용하게 되면 고정되어 있는 명령어 형식상 많은 필드들이 사용되지 않는 상태로 있는 경우가 생기게 된다. 이것은 명령어 필드의 낭비뿐만 아니라 메모리의 낭비나 패치로 인한 속도 저하도 야기 시킬 수 있다.

본 논문에서는 위와 같은 명령어 필드 낭비를 줄일 수 있도록 4개의 32bit 유닛 명령어가 가변적으로 조합되어 효과적으로 수행되는 Variable-Length Instruction 명령어 구조를 제안한다.

II.기존 방식의 OpenGL ES 2.0 명령어

일반적인 OpenGL의 명령어는 고정된 SIMD 64비트 RISC타입의 명령어 구조였다. 이러한 고정된 명령어는 단순하고 설계가 쉽다는 장점이 있으나 명령어 표현상에 제약이 많고 향후 명령어 추가에 대하여 취약하다는 문제점이 있었다.

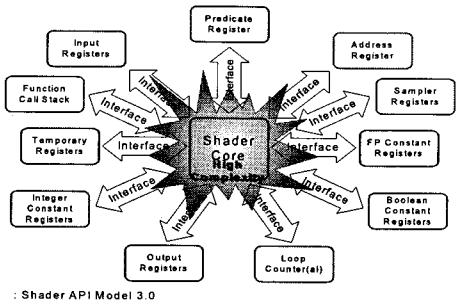

그림 1. 쉐이더 모델 3.0의 레지스터 복잡성

OpenGL ES 2.0 의 Shader 3.0 모델로 가게 되면서 [그림 1]과 같이 Dynamic-branch, Looping 과 같은 기능이 추가되어서 그에 따른 많은 레지스터가 추가적으로 사용되었다. 이러한 구조에서 기존의 방식인 고정된 명령어 형식을 유지하고 Shader 3.0의 모든 기능을 사용하려면 [그림 2]와 같이 최소 109비트로 확장된 구조를 가지게 된다. 이러한 확장된 구조는 실제 사용되는 명령어 필드에 비하여 낭비되는 필드가 더 많아지며 그만큼 명령어 표현과 구동 효율이 떨어지게 된다.

II.기존 방식의 OpenGL ES 2.0 명령어

1. 가변 길이 명령어 형식

가변 길이 명령어는 메모리의 효율을 높이고, 연산에 필요한 명령어만을 해독하여 효과적으로 쓸 수 있는 명령어 형식이다.

본 논문에서 제안하는 VL-IW(Variable Length - Instruction Words)방식의 가변 길이 명령어는 [그림 3]과 같이 유닛 명령어(Unit Instruction)를 최대 4개까지 조합할 수 있는 구조를 제안 한다. 기존에 있던 쓸모없이 구현되어 있던 명령어 필드를 각각 유닛 명령어로 바꾸어 구현되는 방식을 사용 한다.

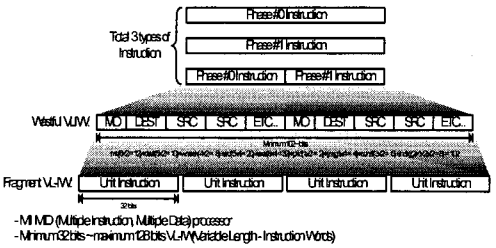

그림 3. 가변 길이 명령어 방식

[그림 3]에서 사용되어 지는 하나의 유닛 명령어는 [그림 4]와 같은 4가지 종류의 32비트 형식으로 되어있다. 명령어 종류에 따라 각 유닛 명령어도 필드의 내용이 달라지는 것을 볼 수 있다.

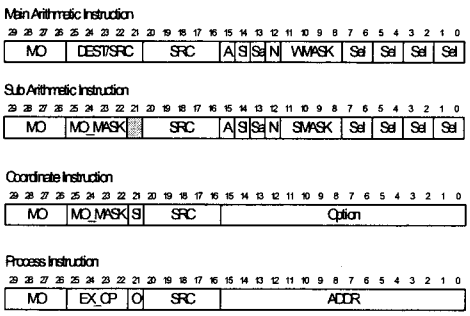

그림 4. 유닛 명령어 형식

2. 제안하는 명령어 셋

제안하는 명령어는 [그림 5]와 같은 명령어 구성을 가지고 있으며, 명령에는 크게 연산(Arithmetic) / 수식(Coordinate) / 프로세스(Process) 명령어그룹으로 나누어진다. 이러한 명령어들은 OpenGL ES 2.0과 DirectX Shader 3.0을 지원하게끔 만들어진 명령어이다.

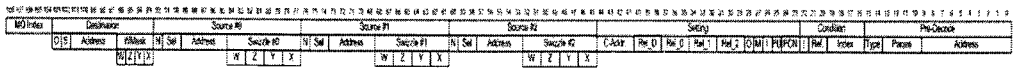

그림 2. 기존 방식의 쉐이더 3.0 모델 지원 명령어

연산 명령어의 경우 특별히 기존에 있던 기본적인 명령어 외에 유닛 구조를 효과적으로 지원하는 Additional Istruction을 명령어를 만들어 사용한다. MOV, MVS 명령을 제외한 나머지 연산 명령어에 대하여 동일한 scalar 위치의 연산을 사용하지 않는 범위에서 중복 선언이 허용된다.

Instruction Type		Phase #0		Phase #1	
		Canonical	Additional	Canonical	Additional
Arithmetic	Move	MOV (move)	MVS(move status register)	MOV (move)	MVS(move status register)
	Base Operation	ADD (add) MUL (multiply)	reserved reserved	ADD (add) MUL (multiply)	reserved reserved
	Compare	CMP (compare)	reserved	CMP (compare)	reserved
	Special Function	RCP (Reciprocal) MAN (Mantissa)	RSQ (Reprocal squared) EXP (Exponent)	RCP (Reciprocal) MAN (Mantissa)	RSQ(Reprocal squared) EXP (Exponent)
	Setting	FLR (Floor) CONV (data type conversion)	FRC (Fraction) reserved	FLR (Floor) CONV (data type conversion)	FRC (Fraction) reserved
	Logical	AND (bit logic and) XOR (bit logic exclusive)	OR (bit logic or) SHF (logical or arithmetic shift)	AND (bit logic and) XOR (bit logic exclusive)	OR (bit logic or) SHF (logical or arithmetic shift)
	reserved	reserved reserved		reserved reserved	
Coordinate		PRED (predicate coordinate) ADDR (address coordinate)		reserved reserved	
Process		reserved reserved		BRC (branch) MEM (memory operation)	

그림 5. 명령어 구성

3. 가변 길이 명령어 구현 방법

[그림 6]에서 보듯이 한 명령어는 1개에서 최대 4개까지의 유닛 명령어로 구성된다. 유닛 명령어의 최상위 비트 'E'(End of Instruction) 필드는 한 명령어가 이루는 유닛 명령어의 개수를 정한다. 'P'(Phase) 필드는 유닛 명령어가 해석될 Phase를 의미한다. 명령어 조합은 [그림 4]에서와 같이 최대 8가지 종류의 유닛 명령어 조합이 이루어 질 수 있다. 또한 반드시 Phase 1의 유닛 명령어는 Phase 0의 명령어 뒤에 존재하여야 하며 각 Phase는 최대 2개의 유닛 명령어를 선언할 수 있다.명령어 형식은 [그림 6]에서와 같이 Arithmetic, Coordinate, Process 명령어 타입에 따라 다른 구조로 설계하였다. [그림 6]에서 보듯이 Arithmetic Instruction 구조는 Main 과 Sub로 나뉘는 것을 알 수 있다. 이는 유닛 명령어를 정렬하는 순서에 따라 Main인지 Sub인지 구분을 한다.

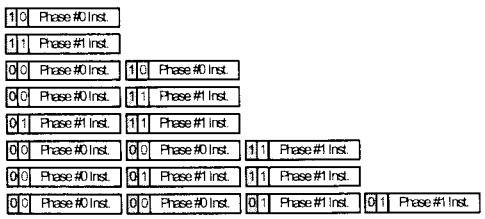

그림 6. 유닛 명령어의 기본 조합

IV. 가변 길이 명령어 처리 데이터 패스

1. 데이터 구조

본 논문에서 제안하는 가변 길이 명령어를 처리하기 위한 쉐이더 프로세서의 데이터 패스 구조를 [그림 7]과 같이 나타낼 수 있다.

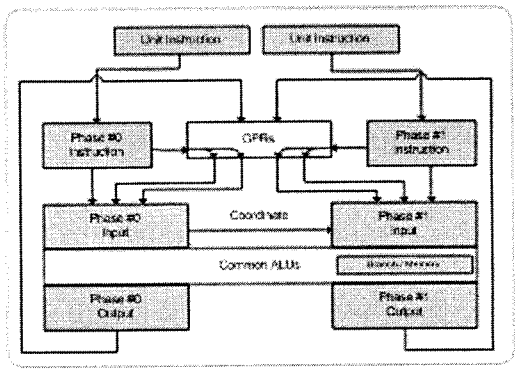

그림 7. 코어의 기본 구조

[그림 7]에서 보이는 바와 같이 쉐이더의 데이터 패스 구조는 2 Phase 구조로 되어있다. 각 Phase마다 유닛명령어를 지정할 수 있으며, 하나의 Phase에 Main 유닛 명령어와 Sub 유닛 명령어로 2개의 유닛 명령어를 처리가 가능하며, 이로 인하여 하나의 명령어 조합에 최대 4개까지의 유닛 명령어 조합이 가능하다.

그러나 모바일 기기 기반의 쉐이더 설계를 위하여 면적을 고려하여 데이터 패스에서 가장 큰 면적을 차지하는 ALU는 하나의 ALU만을 가지도록 설계하여 두 개의 Phase가 공용으로 사용하도록 설계 되어있다. 그러므로 각 Phase에서는 다른 Phase와 연산이 중복되어서는 안 된다. 또한 GPRs(General Purpose Registers) 하나를 사용하여 각 Phase에서 연산에 필요한 다양한 레지스터나 연산 결과가 저장되는 레지스터 역시 공용으로 사용한다.

Phase #0과 Phase #1은 모두 연산 명령어 수행이 가능하고, Branch와 같은 흐름제어 명령어와 Memory 명령어는 Phase #1에서 전담하도록 되어있다. 이러한 Phase #1의 흐름제어 명령어나 메모리 명령어의 보다 다양하고 복잡한 표현을 위하여 Phase #0에서는 Phase #1의 명령어를 수식 가능하도록 되어있다. 이로써 Indirect Address나 Dynamic Branch, Looping과 같은 복잡한 명령어도 사용이 가능하도록 되어있다.

V. 기능 검증 및 결과

설계된 유닛 명령어의 기능을 검증하기 위해서 OpenGL ES 2.0 API에서 요구되는 명령어를 사용하였다. [그림 8]에서는 유닛 명령어를 조합해서 다양한 연산을 수행하는 과정을 보인다. 이 중 다섯 번째 조합의 예는은 MUL, MOV, ADD, RCP 네 개의 명령어가 각 Phase에 Main 과 Sub 로 나뉘어서 하나의 명령어로 조합이 가능함을 보여주고 있다.

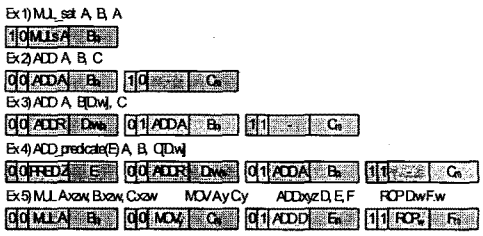

그림 8. 유닛 명령어의 조합

또한 [그림 7]에서 본 코어 구조에서 하나의 GPR(General Purpose Register)을 사용함에도 불구하고 Phase#0에서 Phase #1을 수식할 수 있는 처리방식 때문에 Dynamic/nested Branch 명령어를 단일 명령어로 사용하는 방법을 쓸 수 있다는 것을 [그림 9]에서 나타내고 있다.

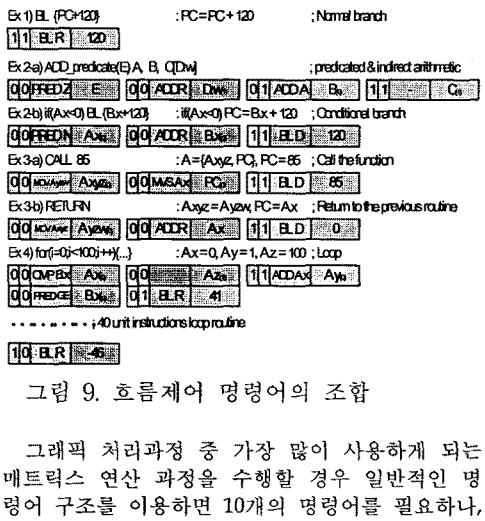

그림 9. 흐름제어 명령어의 조합

그래픽 처리과정 중 가장 많이 사용하게 되는 매트릭스 연산 과정을 수행할 경우 일반적인 명령어 구조를 이용하면 10개의 명령어를 필요하나, 가변길이 명령어를 이용하면 [그림 10]과 같이 6개의 명령어 조합만을 사용하여도 수행이 완료되었다.

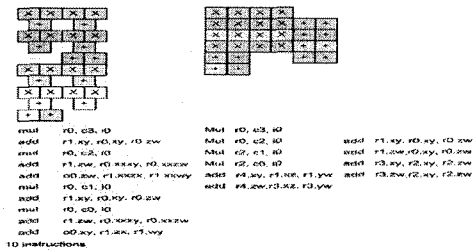

그림 10.4X4 매트릭스 연산의 명령어 차이

IV. 결 론

제안하는 가변 길이 명령어는 잘 알려진 SIMD와 VLIW(Very Long Instruction Words)의 문제점을 해결하고 수식 가능한 유닛 명령어 구조를 통하여 개별적으로는 단순한 구조를 제시하며 전체적으로 진보된 프로세서 구조를 제시한다.

제안하는 명령어에서는 여려 방법을 사용 가능하게 유닛 명령어로 나누어 놓고 그것을 어떻게 조합하느냐에 따라 여러 명령어를 효율적으로 구현할 수 있다. 또한 간단한 연산 명령어는 유닛 명령어 하나 만으로도 구현이 가능하게 되었다. 이 방법을 사용하여 기존에 사용하던 문제점을 없애고 조합방식에 따라 명령어도 크게 줄일 수 있음을 나타냈다.

* 본 논문은 지식경제부가 지원하는 국가 반도체 연구개발사업인 "시스템집적반도체기반기술개발사업 (시스템 IC 2010)"을 통해 개발된 결과이며 설계에는 IDEC 지원 장비를 사용했음을 밝힙니다.

참고문헌

[1] http://msdn.microsoft.com Microsoft Shader3.0,
[2] H.K. Jeong, "Design of 3D Graphics Geometry Accelerator using the Programmable Vertex Shader" ITC-CSCC 2006.
[3] Mauricio Breternitz, Jr., "Compilation, Architectural Support, and Evaluation of SIMD Graphics Pipeline Programs on a General-Purpose CPU" Proceedings of the 12th international conference on parallel architectures and compilation techniques.
[4] James C. Lelterman, "Learn Vertex and Pixel Shader Programming with DirectX9" Wordware Publishing, Inc. 2004.
[5] Liza Fireman, "The Complexity of SIMD Alignment" Technion - Computer Science Department - M.Sc. Thesis MSC - 2006.