저전력 모바일 장치를 위한 완전 프로그램 가능형 쉐이더 프로세서

A Fully Programmable Shader Processor for Low Power Mobile Devices

정 형 기*, 이 주 석**, 박 태 룡***, 이 광 엽*★
Hyung-Ki Jeong*, Joo-Sock Lee**,Tae-Ryong Park*** ,Kwang-Yeob Lee*★

Abstract

In this paper, we propose a novel architecture of a general graphics shader processor without a dedicated hardware. Recently, mobile devices require the high performance graphics processor as well as the small size, low power. The proposed shader processor is a GP-GPU(General-Purpose computing on Graphics Processing Units) to execute the whole OpenGL ES 2.0 graphics pipeline by using shader instructions. It does not require the separate dedicate H/W such as rasterization on this fully programmable capability. The fully programmable 3D graphics shader processor can reduce much of the graphics hardware. The chip size of the designed shader processor is reduced 60% less than the sizes of previous processors.

요 약

본 논문에서는 전용하드웨어를 사용하지 않는 새로운 구조의 범용 그래픽 쉐이더 프로세서를 제안한다. 최근 모바일 기기에서는 고성능을 유지하면서 저전력의 작은 크기를 가지는 그래픽 프로세서를 요구한다. 제안하는 쉐이더 프로세서는 OpenGL ES 2.0 그래픽 파이프라인 전체를 쉐이더 명령어로 실행할 수 있는 GP-GPU 구조를 갖는다. 프로그램을 구현하여 하나의 프로세서로 모든 그래픽 파이프라인 처리가 가능하기 때문에 Rasterization Unit과 같은 별도의 전용 하드웨어를 필요로 하지 않는다. 따라서 쉐이더 프로세서 하나로 Fully Programmable 3D Graphics Engine 구현이 가능하며 기존 쉐이더 프로세서에 비해 하드웨어 크기를 60% 줄였다.

Key words : Shader, graphics, graphics pipeline

Ⅰ. 서론

최근 핸드폰이나 PMPs(Personal Multimedia Players) 같은 모바일 기기에서는 게임이나 GUI를 3차원 그래픽으로 표현하기 위해 그래픽 프로세서가 사용된다. 점점 높은 그래픽 성능이 요구되면서임베디드 그래픽 프로세서도 PC그래픽 프로세서와 유사한 방향으로 발전하였다[2].

그래픽 파이프라인은 [그림 1]과 같이 발전하였다. [그림 1-a] 는 그래픽 파이프라인 과정 중 쉐이더

* 회원, 서경대학교 컴퓨터공학과
** 충북테크노파크 차세대반도체 임베디드 시스템 기술개발지원센터 센터장
*** 서경대학교 컴퓨터공학과 교수
*★ 서경대학교 컴퓨터공학과 교수, 교신저자
* 본 논문은 중소기업청 “기술개발혁신사업” 과 서울특별시 “산학연 협력사업”의 지원으로 작성되었습니다. 설계에는 IDEC의 지원장비로 이루어졌습니다.

단계는 프로세서가 처리하고 복잡한 연산은 전용 하드웨어나 소프트웨어 프로그램으로 처리하는 Partially Programmable Pipeline을 표현 하고 있다. Partially Programmable Pipeline은 두 단계로

발전되어 왔다. 각 정점, 픽셀 쉐이더 단계를 고정된 파이프라인 과정으로 처리하는 고정 프로세서로 구현되었고, 그 다음으로 정점과 픽셀 쉐이더의 기능을 통합쉐이더로 합치면서 쉐이더 프로세서를 효율적으로 사용하게 되었다.

[그림 1-b]는 Fully Programmable Pipeline 구조를 나타낸다. Fully Programmable Pipeline 은 모든 과정을 명령어로 프로그램을 구현하여 Fully Programmable Graphics Processing Unit(FPGPU)으로 처리 하기 때문에 소프트웨어나 다른 하드웨어의 도움이 없이 그래픽 파이프라인 전 과정을 처리할 수 있는 구조로 발전되었다.

전용 그래픽 하드웨어를 갖지 않는 구조 이기 때문에 모바일 기기에 적합한 적은 규모의 하드웨어로 구현이 가능한 방법이다. 그러나, 모든 파이프라인 단계를 명령어로 처리하기 때문에 성능에 많은 문제점이 있다. 따라서 FPGPU를 적용하기 위해서는 성능을 유지하면서 크기를 줄이는 방법이 필요하다.

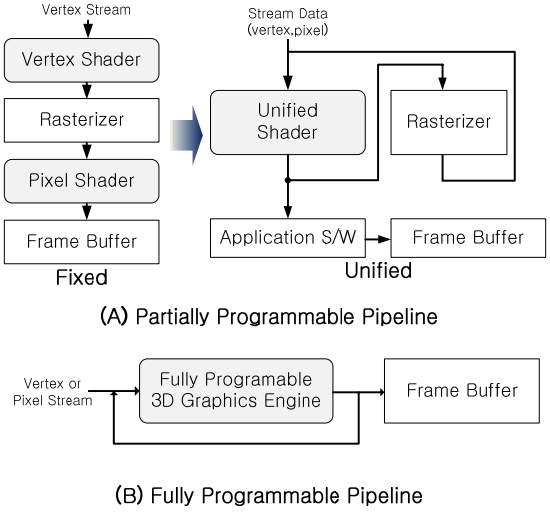


Figure 1. Pipeline of 3D Graphics pipeline

II. 본론

1. 아키텍처 기본 구조

A. 기존 방식의 GPU구조

[그림 2] 는 기존 방식의 FPGPU 구조를 표현한 그림이다[4]. 이 구조는 통합 쉐이더 외에 그래픽 파이프라인 처리에 필요한 H/W인 Matrix / quaternion Vector Generator (MG), Vertex Generator (VG), Fragment Generator (FG), Pixel Generator (PG), Matrix/Quaternion Vector Generator (MG) and graphics caches 이 포함되어 있다. 즉 이 구조는 GPU안에 연산이 많은 부분이나 통합 쉐이더가 처리하지 못하는 단계는 별도의 전용 하드웨어로 처리한다. 전용 하드웨어를 사용하게 되면 명령어의 파이프라인 처리 과정 중 동시에 처리할 수 있어서 빠른 처리가 가능하지만 그래픽 프로세서는 복잡하고 크기가 커지게 된다.

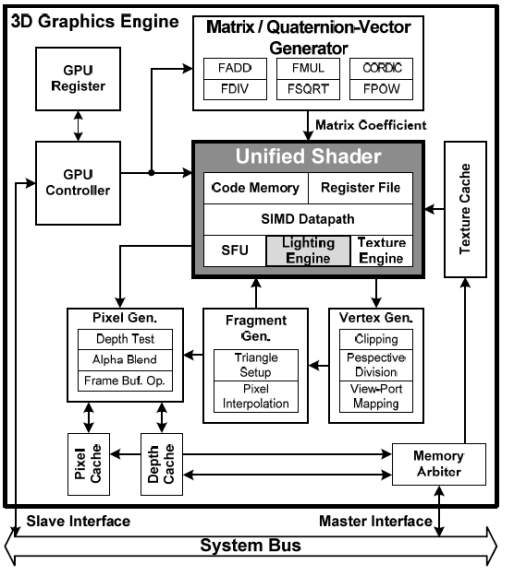


Figure 2. Previous Programmable 3D Graphics Engine

B. 제안하는 GPU구조

이와 같은 구조는 모바일을 대상으로 하는 그래픽 프로세서에서는 적용하기 어렵기 때문에 전용 하드웨어 없이 명령어 파이프라인만으로 수행 가능한 구조가 필요하다.

본 연구는 프로세서의 크기를 작게 하기 위해 전용 하드웨어를 사용하지 않는 구조를 제안한다. 전용 하드웨어를 없애기 위해서 쉐이더 프로세서는 모든 그래픽 파이프라인 과정을 처리 할 수 있어야 한다. 제안하는 Programmable Shader Core(PSC)는 GPGPU (General-Purpose computing on Graphics Processing Units) 같이 그래픽에 관련된 스트림 데이터뿐만 아니라 일반 응용 프로그램도 처리 할 수 있는 프로세서이다[1]. 따라서 FPGPU에서 전용 하드웨어로 처리되던 그래픽 파이프라인 단계를 SPC에서 명령어로 구현하여 처리가 가능한 것을 알 수 있다. 따라서 제안하는 FPGPU는 전용 하드웨어를 없애고 기존 연구에 비해 프로세서 사이즈를 60% 줄일 수 있었다.

[그림 3]은 제안하는 FPGPU 구조를 나타낸 그림이다. PSC는 [그림 3]에서 GPR과 Texture Sampler, Global 레지스터와 더불어 3D 그래픽엔진을 구성하는 코어 모듈이며 내부 구조는 [그림 5]에 나타내고 있다. 이 구조에서 사용되는 PSC는 모든 과정을 프로그램으로 구현하여 처리 가능하기 때문에 별도의 전용 하드웨어를 포함하고 있지 않고 오로지 데이터 저장공간으로 사용되는 General Purpose Registers(GPRs), Global Register(Global) and Texture Sampler(Texture)로 구성되어 있다. PSC GPRs는 Input Registers, Output Registers, Temporary Registers 등 쉐이더 프로세서에서 필요로 하는 모든 레지스터의 기능을 합쳐놓은 저장공간이다. PSC에서 쓰이는 모든 데이터 처리과정은 GPRs를 이용해서 처리한다. Global은 그래픽에서 사용되는 Marix data 같은 공통적으로 사용되는 데이터들을 저장해 놓은 공간이며 Texture는 texel data(texture의 한 점)를 저장해두는 공간이다.

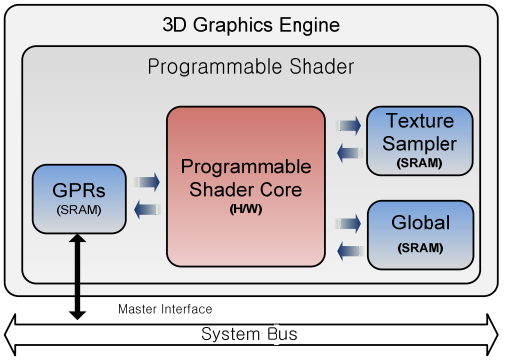


Figure 3. Proposed Programmable 3D Graphics Engine

2. 쉐이더 구조 및 성능 개선 방법

제안하는 FPGPU는 전용 하드웨어를 제거하여 프로세서의 크기가 60% 줄어들었지만 전용 하드웨어를 사용하여 병렬처리 되던 단계를 PSC를 하나만 사용하여 순차적으로 처리해야 하기 때문에 전체적인 속도면에서 성능이 저하된다. 이러한 단점을 개선하기 위해 PSC성능 자체를 올리는 방법과 명령어 프로그램의 알고리즘을 개선하는 방법이 필요하다.

A. Multi-Thread Architecture

OpenGL ES 2.0 를 지원하는 FPGPU를 구현하기 위해서는 흐름 제어(Flow Control) 명령어가 필요하다. 흐름제어 명령어를 지원할 때에 가장 문제가 되는 것은 Branch 명령어로 야기되는 파이프라인 지속성의 결여이다. Branch 명령어가 실행되어 다음 PC (Program Counter) 값이 결정될 때 상위 파이프라인상에서는 이미 다음 명령어가 미리 디코딩된 상태이기 때문에 만약 다른 PC값이 다를 경우 상위 파이프라인의 명령어들은 모두 무효 처리되어야 한다. 이 때문에 일어나는 클럭 손실은 Branch 명령어 개수만큼 비례한다. 또한 그래픽 프로세서 특성상 많은 수의 정점(Vertex) 또는 픽셀(Pixel) 스트림 데이터를 처리하기 때문에 1 클럭만 손실이 있어도 전체 클럭 손실에 많은 비중을 차지하게 된다.

이를 해결하는 방법으로 [그림 4]에서 보여주듯이 여러 개의 스레드(Thread)를 한 개의 프로세서에서

순차적으로 실행하는 방법을 사용하였다. 즉 멀티 스레드로 파이프라인을 실행하는 방법이다. 이 방법은 하나의 스레드에서 명령어가 처리된 후에 다음 명령어 파이프라인이 처리

되기까지 여유 클럭을 가지게 되어서 Branch hazard가 발생하지 않게 할 수 있는 이점이 있다. 또한 각 스레드는 다른 스레드의 영향을 받지 않고 독립적으로 수행이 가능하기 때문에 스레드마다 데이터 의존성(Data Dependency)을 가지지 않는다.

B. *Dual-Phase Architecture*

전용 하드웨어를 사용하지 않아 저하된 프로세서의 성능을 높이기 위해 PSC내부에 연산기를 필요한 만큼 추가하여 여러 연산을 한 파이프라인 단계에 처리하는 방법이 있다. 그러나 이러한 방법은 추가되는 연산기 종류와 크기에 따라 프로세서의 크기가 증가된다. 이러한 방법은 프로세서의 크기를

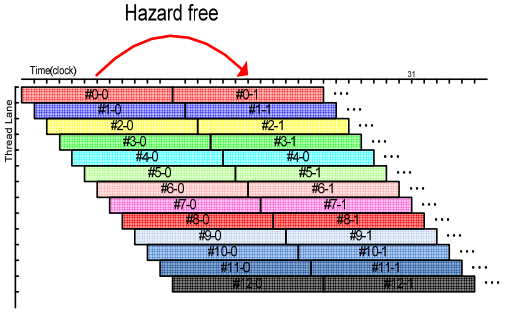


Figure 4. Multi Thread Method

줄이기 위해 전용 하드웨어를 제거한 의도와 맞지 않는다.

프로세서의 크기는 유지하면서 성능을 향상 시키기 위해서 [그림 5]과 같은 듀얼 페이지 구조를 제안한다. 이 듀얼 페이지 구조는 사용되는 명령어는 기존 VLIW(Very Long Instruction Word)[5]에서 사용 되었던 방식과 유사하게 두 개의 MO(micro-operation)를 가지고 있다. 각 MO는 Phase #0 과 Phase #1로 나뉘어 처리된다. 기존 방식은 동시에 두 개의 MO를 처리하기 위해 두 개의 ALU를 가지고 처리하지만 제안하는 프로세서는 하나의 ALU를 두 개의 Phase에서 공유하여 사용한다. 따라서 두 개의 Phase가 서로 다른 연산을 동시에 처리 할 수 있는 구조이다. 같은 연산을 동시에 사용하지 못하지만 4x4 Matrix 곱셈 같은 그래픽 프로세서에서 자주 사용되는 기능을 구현해본 결과 제안하는 프로세서는 평균적으로 40% 성능 향상되는 것을 입증하였다.

C. *프로그램으로 구현된 그래픽 파이프라인*

그래픽 파이프라인 처리과정 중에 프로세서에서 가장 지연이 많은 단계가 외부 메모리와 데이터를 주고 받는 단계이다. 특히 그래픽 프로세서는 많은 수의 정점 또는 픽셀 데이터를 외부 메모리에서 받아 처리하기 때문에 이 과정을 줄이는 방법이 중요하다. 기본적인 그래픽 파이프라인은 [그림 1]의 고정파이프라인 같이 정점 처리 후 Rasterization 단계를 거쳐 픽셀처리를 한다. 이 과정은 각 단계마다 외부에서 정점이나 픽셀데이터를 읽어와야 한다. 외부 Data 전송을 줄이기 위해 제안하는 파이프라인은 Rasterization 단계와 픽셀 쉐이더 단계를 합쳐 구현하였다.

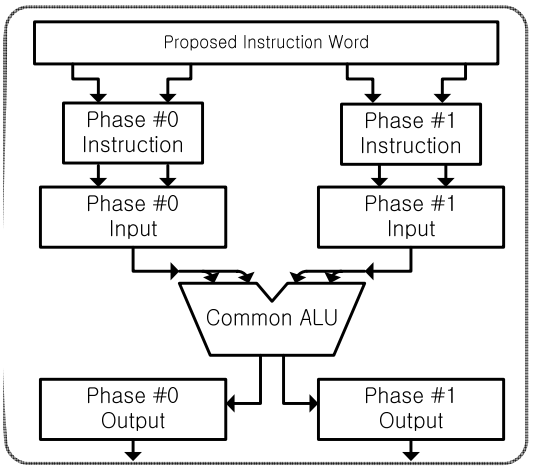


Figure 5. Dual-Phase Architecture

[그림 6]은 명령어로 구현한 그래픽 파이프라인

프로그램을 단계별로 표현한 그림이다. Stage #0 단계에서는 Index Data를 읽어와 폴리곤(Polygon) 단위로 처리를 한다. 즉 3개의 정점 데이터를 읽어와 Culling 과 Clipping을 포함한 정점 쉐이더 기능을 수행 한다. Stage #0의 수행 결과로 만들어진 정점 데이터를 이용하여 Stage #1의 Rasteraztion 단계에서 사용할 데이터

를 미리 연산하여 최종적으로 정점 쉐이더 처리와 Pre-calculation 결과를 Memory에 저장한다.

Stage #1 단계에서는 Stage #0 단계에서 처리된 varying value 및 primitive를 읽어와서 rasterization 연산을 하고 픽셀 쉐이더 처리를 한 후에 Alpha test와 Blending 단계를 처리한 결과를 최종적으로 FrameBuffer에 저장한다. Rasterization 과 픽셀 쉐이더 처리를 Stage #1에서 한번에 구현 함으로써 전송 지연이 심한 외부 메모리전송 횟수를 줄일 수 있어 성능이 개선 된다.

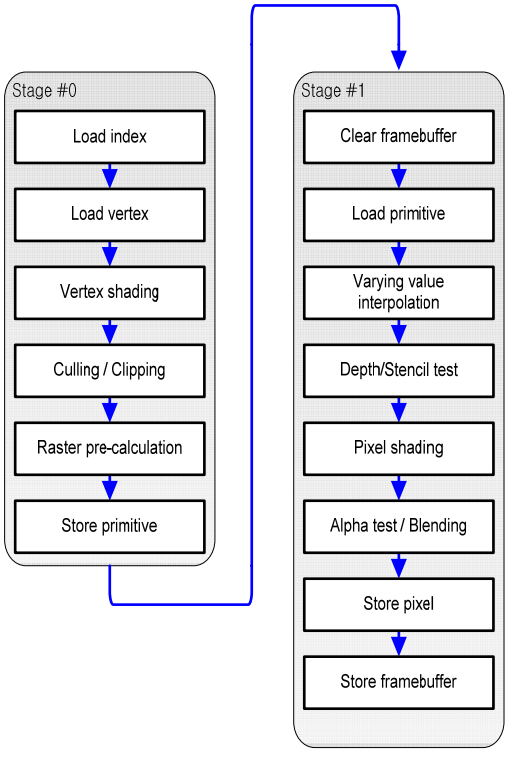


Figure 6. 제안하는 Graphics pipeline algorithm

D. 래스터 알고리즘 개선

모든 과정을 프로그램으로 구현하기 때문에 그래픽 파이프라인 에서 처리되는 알고리즘 개선도 성능에 많은 영향을 끼친다. 특히 Rasterization 알고리즘은 많은 연산을 포함 하기 때문에 어떤 알고리즘을 사용하느냐에 따라 연산량의 차이가 많이 난다. 특히 제안하는 PSC는 멀티 스레드 구조를 사용하기 때문에 멀티 스레드에서 효과적인 알고리즘을 선택해야 한다.

제안하는 Vector기반 Rasterization 알고리즘은 하나의 폴리곤 내에서 픽셀 데이터를 독립적으로 처리하여 Multi-Thread 기반의 병렬 처리 구조에서 효율적인 연산이 가능하다.

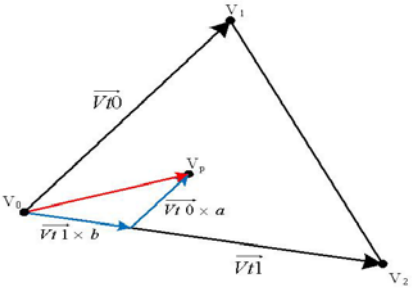


Figure 7. Pixel Position using Vector Algorithm

그림 1은 세 개의 정점 V0(x0, y0), V1(x1, y1), V2(x2, y2)로 구성된 하나의 폴리곤을 나타내며, Vp(xp, yp)는 이 폴리곤 내의 임의의 한 픽셀을 나타내고 있다. 이 폴리곤에서 V0와 V1의 두 정점을 이용하여 하나의 Vector $\overrightarrow{Vt0}$ 를 구성하고 V0와 V2를 이용하여 또 다른 하나의 Vector $\overrightarrow{Vt1}$를 구성한다면, 폴리곤 내의 한 픽셀 Vp는 $\overrightarrow{Vt0}$ 와 $\overrightarrow{Vt1}$ 두 Vector를 통하여 표현이 가능하다.

[그림 7]과 같이 세 개의 정점 V0(x0, y0), V1(x1, y1), V2(x2, y2)로 구성된 하나의 폴리곤 내의 한 픽셀 Vp는 2개의 Vector $\overrightarrow{Vt0}$, $\overrightarrow{Vt1}$ 를 각각 a배, b배로 축소하여 표현할 수 있다. 즉 $\overrightarrow{Vt0}$ 을 a배로

축소한 벡터와 $\overrightarrow{Vt1}$ 을 b배로 축소한 두 벡터의 합은 픽셀 Vp를 나타내는 것이다. a와 b는 폴리곤 내의 모든 픽셀이 다른 값을 가지며, 이 a, b값을 통하여 해당 픽셀의 Color나 u,v Texture 좌표, Normal등 다른 속성 정보들의 interpolation이 가능하다.

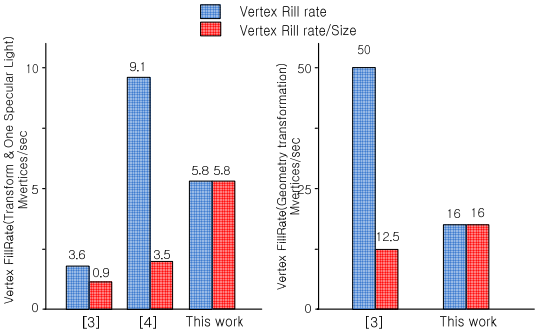
Rasterization 과

정에서 10000개의 픽셀 데이터를 처리 한다고 할 때 Vector기반 Rasterization 알고리즘 (606255개의 연산량)은 Scan-line (111835150개의 연산량) 알고리즘에 비해 연산이 50% 줄어든 것을 검증하였다. 폴리곤당 50%의 연산이 줄어들어 전체적인 그래픽 파이프라인 처리 속도가 줄어들었다..

Ⅲ. 성능

제안하는 FPGPU는 CHARTERED 0.18μm 공정에서 구현하였을 때 GPRs 포함하여 0.52M logic gates크기를 가진다.

[그림 8]은 기존 연구결과와 비교한 결과를 나타내고 있다. 왼쪽 그래프를 보면 lighting 과정을 포함한 Vertex Fill rate 수치를 나타내고 오른쪽 그래프는 lighting 과정을 포함하지 않은 Transformation 처리한 수치를 비교한 그림이다. 기존 프로세서와 Transistor Counts를 비교하면 기존 연구에 비해 크기가 60% 이상 줄어든 것을 확인 할 수 있다. 그러나 앞서 기술한 바와 같이 전용 하드웨어를 제거하여 크기가 줄어든 반면에 프로세서의 전체적인 성능이 줄어들었다(파란색 그래프). Transistor Counts 측면에서는 크기대비 성능(빨간색 그래프)으로 비교 하면 기존 연구에 비해 성능이 더 좋아지는 것을 확인 할 수 있다.



	[3]	[4]	This Work
Operation Frequency (MHz)	200	100	100
Vertex Fill Rate (Mvertices/s)	3.6	9.1	5.8
Transistor Counts (M logic gate)	2	1.3	0.5
Programmability	Geometry Only	Geometry Randering	Geometry Rasterization Randering

Figure 8. Performance & Size Comparison

Ⅳ. 결론

기존 쉐이더 프로세서는 쉐이더 기능만 처리할 수 있는 구조를 가지고 있다. 따라서 쉐이더 프로세서에서 할 수 없는 기능은 소프트웨어의 도움을 받거나 별도의 전용 하드웨어가 필요하다. 기존 쉐이더 프로세서는 전용 하드웨어의 도움으로 그래픽 파이프라인의 처리 성능이 좋지만 추가된 전용 하드웨어만큼 프로세스의 크기는 증가된다.

본 논문에서는 저전력을 목표로 하는 모바일 기기에 적합한 적은 크기의 프로세스를 개발하기 위하여 그래픽 파이프라인 전체를 프로그램으로 구현 가능한 GP-GPU 구조를 가지는 PSC를 구현하였다. 전용 하드웨어 없이 그래픽 파이프라인의 처리가 가능하기 때문에 적은 크기의 프로세서를 만들 수 있다. 파이프라인의 성능저하 문제는 벡터 기반의 래스터 알고리즘과 멀티 스레드 구조로 개선하였다.

제안하는 PSC는 Fully Programmable로 처리 할 수 있기 때문에 Rasterization 뿐만 아니라 다른 단계에서도 알고리즘의 개선으로 성능을 향상 시킬 수 있다. 또한 PCS의 크기가 다른 그래픽 프로세서에 비해 40% 정도를 차지 하기 때문에 멀티 코어로 구현하여 성능을 향상시킬 수 있는 가능성이 있다.

참고문헌

[1] Aaaftab Munshi,et al. “OpenGL ES 2.0 Programming Guide”

[2] Enhua Wu; Youquan Liu, Emerging technology about GPGPU, APCCAS 2008. IEEE Asia Pacific Conference on Volume , Issue , Nov. 30 2008-Dec. 3 2008 Page(s):618 – 622

[3] J.H Sohn, et al., “A 155-mW 50-Mvertices/s Graphics Processor With Fixed-Point Programmable Vertex shader for Mobile Applications”, IEEE Journal of Solid State Circuits, Vol 41, No. 5, pp.1081-1091, May, 2006

[4] Jeong-Ho Woo, et al. “A 195mW, 9.1MVertices/s fully programmable 3D graphics processor for low power mobile device “, Solid-State Circuits Conference, 2007

[5] Philips Semiconductors. “Very-Long Instruction Word(VLIW) Computer Architecture” ,Philips. http://www.nxp.com/acrobat_download/other/vliw-wp.pdf

저 자 소 개

정 형 기 (학생회원)

2005년:서경대학교 컴퓨터공학과 졸업 (공학학사)
2007년: 서경대학교 대학원 컴퓨터공학과 (공학석사)
2007년~현재: 서경대학교 컴퓨터공학과 공학박사과정
<주관심분야> 마이크로 프로세서, 멀티 스레드 프로세서, 3D Graphics System

이 주 석 (정회원)

1983년 : 서강대학교 전자공학과 졸업 (공학학사)
1985년 : 고려대학교 대학원 전자공학과 (공학석사)
1999년 : 고려대학교 대학원 전자공학과 (공학박사)
1984~1995년 : LG전자 선임연구원
1997~2000년 : 용인송담대학 전자과 조교수
2004~2006년 : 엠텍비젼㈜ 연구소장
2007년~현재 : 충북테크노파크 차세대반도체 임베디드시스템기술개발지원센터 센터장
<주관심분야> SoC 설계, 이미지 프로세싱, Embedded System

박 태 룡 (비회원)

1985년 : 한양대학교 수학과 졸업 (이학사)
1987년 : 한양대학교 수학과 졸업 (이학석사)
1995년 : 한양대학교 수학과 졸업 (이학박사)
1994년~현재 : 서경대학교 컴퓨터공학부 부교수
<주관심분야>암호알고리즘 및 정보보안, 컴퓨터 연산

이 광 엽 (정회원)

1985년 : 서강대학교 전자공학과 졸업 (공학학사)
1987년 : 연세대학교 대학원 전자공학과 (공학석사)
1994년 : 연세대학교 대학원 전자공학과 (공학박사)
1989~1995년 : 현대전자 선임연구원
1995년~현재 : 서경대학교 컴퓨터공학부 부교수
<주관심분야> 마이크로 프로세서, Embedded System, 3D Graphics System