The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 19, No. 4, pp.119-128, Aug. 31, 2019. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2019.19.4.119
JIIBC 2019-4-18

Multi-Access Memory System을 이용한 3D 그래픽 프로세서 제안

Proposal of 3D Graphic Processor Using Multi-Access Memory System

이스라엘[*], 김재희[**], 고경식[***], 박종원[****]

S-Ra-El Lee[*], Jae-Hee Kim[**], Kyung-Sik Ko[***], Jong-Won Park[****]

요 약 3D 그래픽 프로세서의 시스템의 특성상 많은 수학적 계산이 요구되면서 고속처리를 위하여 GPU(Graphics Processing Unit)를 이용한 병렬처리 연구가 많이 진행되고 있다. 본 논문에서는 GPU에서 발생하는 문제점 중 캐시 메모리 미스에 의하여 발생하는 대역폭 증가와 3D 셰이더 처리 속도가 일정하지 않은 문제점을 해결하기 위하여 캐시메모리를 사용하지 않는 병렬처리기인 MAMS를 이용한 3D 그래픽 프로세서를 제안한다. 본 논문에서 제안된 MAMS를 이용한 3D 그래픽 프로세서는 DirectX 명령 분석을 이용해 Vertex shader, Pixel shader와 Tiling 및 Rasterizing 구조를 설계 하였고, MAMS를 위한 FPGA(Xilinx Virtex6@100MHz) 보드를 구성하여, Verilog를 사용하여 설계된 구조를 개발하였다. 개발된 FPGA(100Mhz)와 nVidia GeForce GTX 660(980Mhz)의 처리시간을 확인한 결과 GTX 660를 이용한 처리 시간은 일정하지 않음을 확인하였고, MAMS를 이용한 처리 시간은 일정함을 확인하였다.

Abstract Due to the nature of the 3D graphics processor system, many mathematical calculations are required and parallel processing research using GPU (Graphics Processing Unit) is being performed for high-speed processing.

In this paper, we propose a 3D graphics processor using MAMS, a parallel processor that does not use cache memory, to solve the GPU problem of increasing bandwidth caused by cache memory miss and the problem that 3D shader processing speed is not constant. The 3D graphics processor using MAMS proposed in this paper designed Vertex shader, Pixel shader, Tiling and Rasterizing structure using DirectX command analysis, the FPGA(Xilinx Virtex6@100MHz) board for MAMS was constructed and designed using Verilog. We compared the processing time of the developed FPGA (100Mhz) and nVidia GeForce GTX 660 (980Mhz), the processing time using GTX 660 was not constant and suing MAMS was constant.

Key Words : MAMS, Multi-Access Memory System, 3D Graphics Processing, Shaders

[*]정회원, 충남대학교 정보통신학과
[**]정회원, 충남대학교 정보통신학과
[***]정회원, 충남대학교 정보통신학과
[****]정회원, 충남대학교 정보통신학과(교신저자)
접수일자 2019년 7월 9일, 수정완료 2019년 7월 29일
게재확정일자 2019년 8월 2일

Received: 9 July, 2019 / Revised: 29 July, 2019 /
Accepted: 2 August, 2019
[***]Corresponding Author: jwpark@cnu.ac.kr
Dept of Information Communications Engineering, ChungNam National University, Korea

Ⅰ. 서 론

3D 그래픽 프로세서의 연구가 지속 되면서 칩의 성능과 크기 면에서 다양한 설계가 가능해졌으며, 특히 병렬처리를 이용한 여러 3D 연산구조가 제시되었다.

그 중 Shader는 버텍스 셰이더(Vertex Shader)와 픽셀 셰이더(Pixel Shader)[1] 란 이름으로 서로 비슷한 API로 정의 되며, Shader가 표준화 이전에는 3D 그래픽스에 대한 처리 과정은 광원과 물체의 위치를 계산해 내는 T&L(Transform & Lighting)[1]과 같은 고정 기능 파이프라인(Fixed Function Pipeline)방식으로 처리됨으로써 한계가 존재하기 때문에 이를 해결하고자 많은 연구가 되어져 표준화 되어진 것이 Shader[2] 구조이다.

3D 그래픽 프로세서의 고속 처리를 위해 병렬처리기를 이용한 연구들이 진행되었으며, 대표적인 GPU (Graphics Processing Unit)가 개발 사용되고 있다.

GPU의 병렬처리를 SIMT (Singe Instruction Multlple Threads) 방식으로 구현되기 위하여 Cache를 사용하고, Hit율을 올리기 위한 많은 연구가 이루어져 왔다.

Cache 메모리의 Miss를 줄인다고 여러 알고리즘을 이용하지만, 무조건 발생하게 된다. 그러면, 처리시간이 일정하지 못하게 되는 문제점도 가지고 있다. 이러한 문제점을 해결하기 위해서는 Cache메모리를 사용하지 않는 병렬처리 아키텍처를 필요하다.

본 논문에서는 3D 그래픽스 프로세서의 Cache메모리의 문제점을 해결방안으로 다수의 메모리 모듈을 사용한 다중접근 메모리 시스템(MAMS)[3]을 설계한다.

메모리 시스템에서 효율적으로 메모리 대역폭을 지원하기 위해서는 고속 접근이 가능한 다수의 메모리 모듈이 필요함과 동시에 다수의 PE(Processing Element)들이 동시에 같은 메모리 모듈에 접근함으로써 발생할 수 있는 충돌(conflict)을 방지하여야 한다. 특히 3D 그래픽스 처리는 텍스처 이미지를 사용하여 네 방향의 블록과 행, 열, 대각선, 역대각선 형태의 Pixel에 대해 단순하고 반복적인 연산을 수행하기 때문에 다양한 형태로 다수의 텍스처 데이터에 신속하게 접근할 수 있는 다중접근 메모리 시스템과 이에 알맞은 병렬처리기의 개발이 요구되고 있다. MAMS(Multi Access Memory System)는 영상처리를 위한 병렬 접근 메모리 시스템으로 자료의 행, 열, 대각선, 블록 등을 동시에 접근하는데 적합하도록 설계되고 개선되어 왔다[3][4][5][6].

병렬처리 시스템은 일반적으로 다수의 PE와 장치들을 제어하기 위한 제어장치, 명령어(instruction)들을 저장하는 메모리, 그리고 여러 개의 데이터들 에게 동시에 접근이 가능한 다중접근 메모리 시스템 등으로 이루어진 SIMD(Single Instruction Multiple Data stream) 처리기로 구성된다.

본 논문의 2장에서는 3D 그래픽스 프로세서를 위한 기존 Shader 대하여 설명하고, 3장 에서는 MAMS를 이용한 3D 그래픽스 프로세서 구조를 설계 하였고, 4장에서는 MAMS를 이용한 3D 프로세서를 구현하였다 마지막으로 5장에서는 결론을 기술한다.

Ⅱ. 기존 연구

1. 3D 그래픽 Shader의 처리

3D 그래픽 처리는 다중의 연속적인 3D 그래픽스 데이터를 처리해야 한다. 데이터의 종류는 크게 Vertex와 Pixel로 나누어진다. Vertex는 폴리곤(Polygon)을 이루는 단위로써 일반적으로 3개의 Vertex가 하나의 삼각형 폴리곤을 이루어 그 면적에 해당하는 부분을 화면에 Pixel 단위로 출력한다. 그림 1은 3D 그래픽 처리과정을 보여 주고 있다.

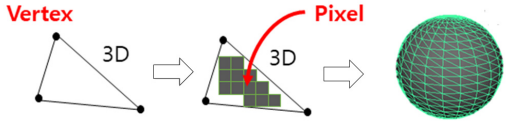


그림 1. 3D 그래픽 처리
Fig. 1. 3D graphics processing

Shader의 그래픽스 렌더링방식은 Vertex Shader, Rasterizer, Pixel Shader등으로 그림 2의는 타일기반의 렌더링 기본의 방식이며 Rasterizer를 구성하기 전에 Tiling을 처리하여 전체 화면의 Pixel 정보를 3D 처리를 하지 않고 탐색하는 Polygon을 줄여 Rasterizer 와 Pixel Shader를 처리하여 처리 속도를 향상시키는 기법이다.

본 논문에서 구성하고자 하는 타일기반의 렌더링에서는 Tiled Primitive List와 Transformed Vertex Data 외부의 시스템 메모리를 사용하지 않고 MAMS 내부 메모리슬 사용한다.

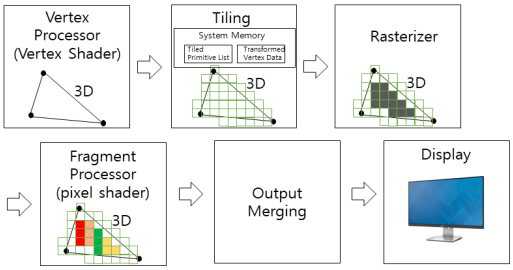


그림 2. 3D 그래픽 랜더링 방법
Fig. 2. 3D graphics rendering method

Vertex Shader는 Vertex 단위로 입력 받아 연산을 수행한다. Vertex에는 Vertex의 위치, 색상, 노말, 텍스쳐 좌표정보 등 폴리곤을 구성하기 위한 많은 정보가 담겨 있다.

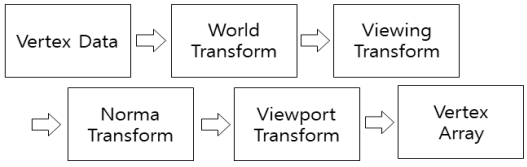


그림 3. Vertex shader 처리과정
Fig. 3. Vertex shader processing

본 논문의 Vertex Shader는 앞에서 설명한 처리과정을 모두 부합하도록 설계 하였으며 Shader 명령어에 특화된 아키텍처이므로 DirectX의 Vertex Shader 모델 1.0 명령어를 이용하여 설계되어 있다.

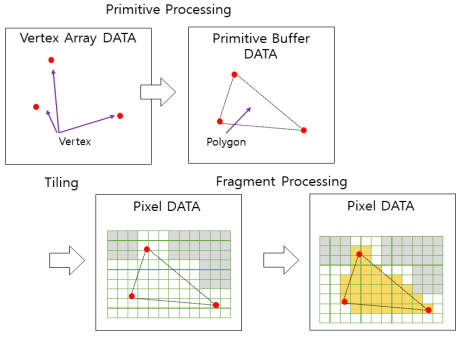


그림 4. 타일 기반 렌더링 방법의 타일링 및 래스터라이저 처리과정
Fig. 4. Tiling and rasterizer processing of tile-based rendering method

타일 기반 렌더링 방식에서의 Tiling 및 Rasterizer 처리는 어레이 데이터를 Primitive 처리에 의해 Polygon으로 구성하고, Tiling 처리를 통해 호스트 시스템 메모리에 Tiled Primitive List와 Transformed vertex 데이터를 저장한다. Tiled Primitive List에 있는 타일만을 선택하여 Fragment 처리를 통해 각 Pixel의 색상데이터(R, G, B, 깊이를 표현하는 Z, 투명도를 표현하는 α)를 구한다.

Pixel shader는 Rasterizer로부터 Pixel Data 단위로 입력 받아 연산을 수행한다.

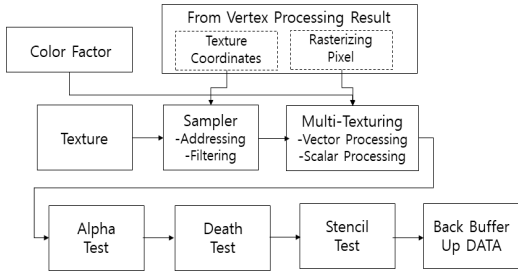


그림 5. 픽셀 셰이더 처리과정
Fig. 5. Pixel Shader processing

2. 기존 3D 그래픽 Shader 연구

3D 그래픽 Shader의 구조에서 타일 기반 렌더링 방식의 연구도 Cache의 Hit율을 높이기 위한 연구들[7][8]이 많이 진행되어왔고, 그중 멀티 코어 GPU를 위한 타일 기반 렌더링 설계를 2011년 서경대학교에서 발표하였다. 전체 타일링에서 탐색하는 Ploygon을 줄이기 위해 계층적 타일링을 이용한 성능향상을 연구하였다. 이는 타일 단위로 분할하는 과정에서 타일과 타일 사이에 존재하는 데이터는 두 개의 타일에 중복되어 primitive 데이터가 충돌하는 문제점은 타일링 단계에서 메모리 중복 접근에 의해 충돌한다.[8] 이를 해결하기 위해서 polygon을 미리 순회 하며 해당 타일에 포함됨을 체크하는 방식으로 계층적 타일링 구조를 제안하였고, 생성된 타일 리스트는 데이터들이 Locality를 가지게 되고, 각 타일에

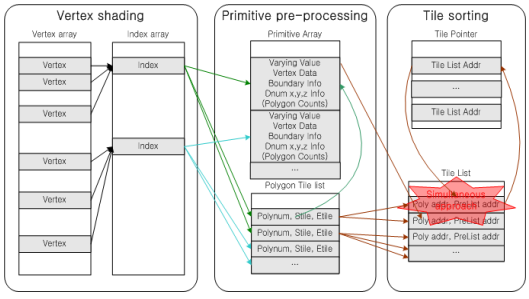


그림 6. 타일링 단계에서 메모리 충돌
Fig. 6. Memory conflict in tiling stage

포함 되는 데이터의 개수를 파악할 수 있도록 구성 되었지만, 스크래치 카우터와 멀티 코어, 멀티 스레드가 동시에 접근 가능한 병렬처리 아키텍처가 필요함을 보여주고 있다.

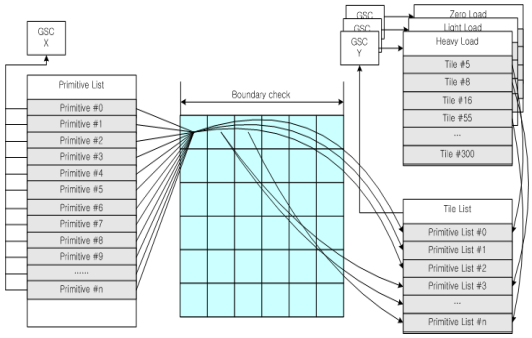


그림 7. 계측적 타일링에 대한 목록 구조
Fig. 7. List structure for hierarchical tiling

Ⅲ. MAMS를 이용한 셰이더 설계

MAMS를 이용한 3D 그래픽 렌더링 시스템을 구성하기 위해서 Vertex 와 Pixel Shader의 연산기 및 레지스터 및 타일 기반의 Rasterizer를 설계 하였으며, 그림 8은 MAMS를 이용한 타일기반의 3D 그래픽렌더링 시스템의 흐름도를 보여주고 있다.

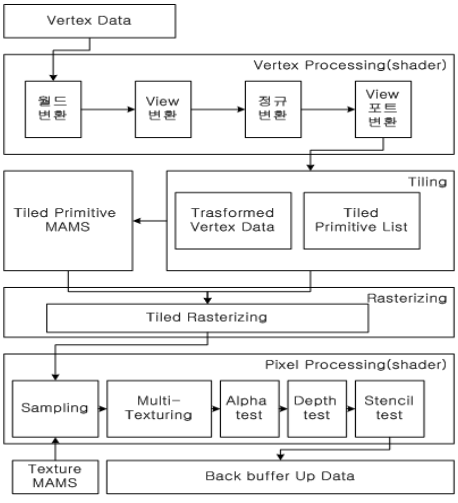


그림 8. MAMS를 이용한 3D 그래픽 렌더링 시스템 흐름도
Fig. 8. 3D graphics rendering system flow diagram using MAMS

그림 9는 Host Computer에 attached하게 연결된 구조를 보여 주고 있으며, SIMD Command Memory에서는 Vertex array, Index array, Rasterizing Command 등이 포함 되어 있으며, 4개의 PE는 Vertex Shader와 Pixel Shader의 연산을 하고, Tiling & Rasterizing은 Triangle Setup, Edge Walk, Span Processing하여 타일 내부의 삼각형 Pixel 값을 처리하도록 구성하였다.

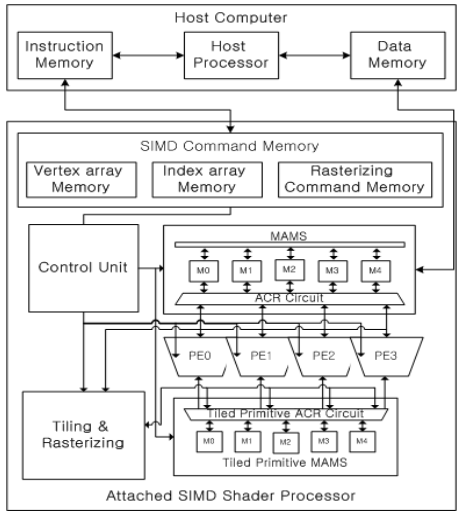


그림 9. MAMS를 이용한 3D 그래픽 렌더링 시스템 블록 다이어그램
Fig. 9. 3D graphics rendering system Block Diagram using MAMS

1. Vertex Shader 와 Pixel Shader의 연산기 설계

앞에서 확인한 Shader는 3D 프로세서를 대상으로 한 명령어들의 집합 렌더링 파이프라인에 대한 프로세서들을 뜻하며, 고정 함수 파이프라인의 정점(vertex)을 T&L (Transform and Lighting) Fog등을 처리하는 정점 Shader(Vertex shader) 와 Pixel 의 Sampling (Filtering, Addressing), Texturing(Blending)등을 처리하는 Pixel Shader(Pixel Shader)로 나누어진다.

Shader를 구현하기 위해 SIMD 명령어(Instruction) 단위 지원을 목표로 설계를 시작하였고, API 명령어를 확인해 볼 필요가 있으며 그림 10과 같이 명령어들을 구분할 수 있다.

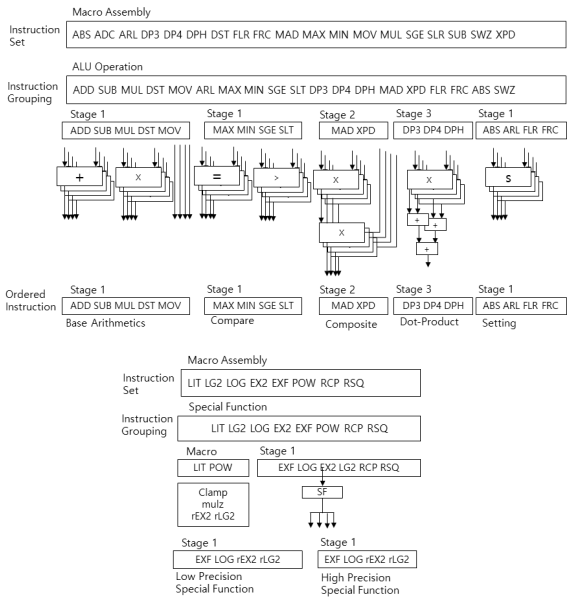


그림 10. 지시 분석 과정
Fig. 10. Instruction analysis process

2. PE를 위한 ALU 설계

앞에서의 명령어 분석에 따라 연산기의 활용에 대한 방법을 정하였고, 내적 연산을 위하여 최소 3 스테이지 연산을 하여야 하며, 크로스-프로덕트(Cross-product) 연산을 위해 3개의 소스 입력을 받아 연산하여야 하는데, 크로스-프로덕트는 3D 벡터로 한정되어 있다. 또한 MAD(Multiply and Add) 명령어를 지원하기 위해서는 곱셈기와 덧셈기가 직렬로 4개가 존재하여 한다[9].

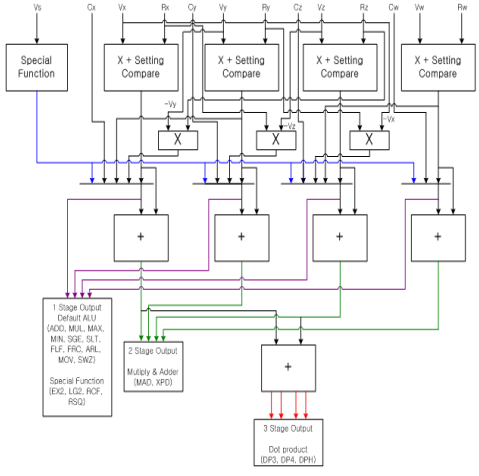


그림 11. PE 용 ALU
Fig. 11. ALU for PE

그림 11의 첫 번째 스테이지에서 기본적인 비교기, 실수 연산기, 변환기, 특수 함수 연산기들이 존재한다. 두 번째 스테이지에서는 MAD와 XPD 명령을 지원하고 세 번째 스테이지에서는 내적 연산 명령어를 지원한다.

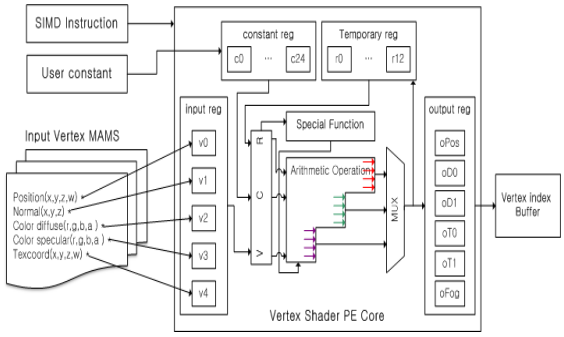


그림 12. 버텍스 셰이더 및 ALU 연결도
Fig. 12. Connectivity diagram for vertex shader registers and ALUs

그림 12는 Vertex Shader와 관련된 레지스터와 PE의 연결도를 도식화 하고 실제 설계할 코어를 정의하며,

그림 13은 Pixel Shader와 관련된 레지스터와 PE의 연결도를 도식화 하고 실제 설계할 코어를 정의한다.

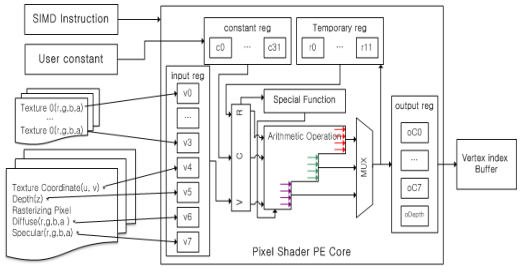


그림 13. 픽셀 셰이더 레지스터 및 ALU연결 다이어그램
Fig. 13. Connectivity diagram for pixel shader registers and ALUs

3. Tiling 및 Rasterizing 설계

2장에서 정의한 타일 기반 렌더링 방식의 Tiling 및 Rasterizer를 설계하기 위해서 Vertex Data를 Primitive처리 하여 Polygon으로 구성하여 Primitive index buffer에 저장 하여야 한다. 그리고 Fragment 처리를 통해 각 Pixel의 색상 데이터(r, g, b, 깊이를 표현하는 z, 투명도를 표현한 α)를 구한다. 사용되는 타일을 정의하여 Tilde Primitive Buffer에 저장 시키고, 각 Pixel의 깊이 값(z)을 이용하여 최종적으로 표현되는 Pixel의 값을 선정하고 Pixel MAMS에 저장한다.

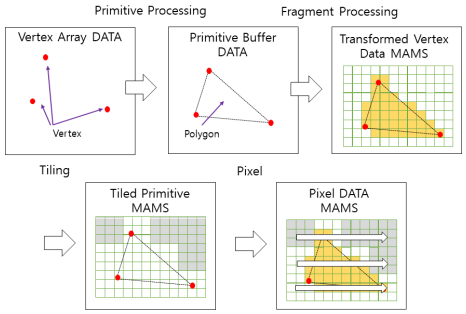


그림 14. 제안된 타일링 및 래스터 라이저 처리
Fig. 14. Proposed tiling and rasterizer processing

마지막으로 Pixel MAMS를 loop 하면서 선정된 타일만 Pixel Shader처리를 위해 Tiled Primitive MAMS에 사용이 허가된 타일에서의 Color register, Texture Register, Sampler Register를 입력 받아 연산한다.

앞에서 설명한 그림 5의 방법은 Tiled Primitive List와 Transformed Vertex Data를 외부 메모리에서 사용하지만, 그림 14는 Tiled Primitive List와 Transformed Vertex Data를 내부 MAMS 메모리에 저장하도록 구성하였기 때문에 순서를 바꾸어 처리하였다.

가. Primitive 처리

화면에서 출력되는 화소 정보를 구하기 위해 수행되는 여러 과정은 모두 폴리곤 단위로 연산을 진행해야 한다. Vertex Shading의 결과로 출력된 정점 단위의 정보를 모아 폴리곤으로 구성하는 Primitive assembly 과정을 수행 하며, 폴리곤을 구성하는 최소의 단위는 정점 정보 3개를 이용해 만드는 삼각형(Triangle)으로 이를 기준으로 차후 모든 Rasterizer의 연산을 수행한다. 그림 15은 Primitive 처리 과정을 보여주고 있다.

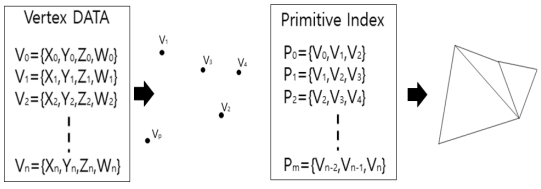


그림 15. 프리미티브 처리
Fig. 15. Primitive processing

나. Fragment 처리

Trainagle Setup은 Primitive index의 데이터인 V0, V1, V2가 들어오면 Y값을 중심으로 비교하여 큰 순서대로 정렬된 세 정점은 이후 변 처리기에서도 계속 사용하며 X축, Y축의 좌표를 나타내는 위치 데이터와 색깔을 표현하는 R, G, B, 깊이를 표현하는 Z, 투명도를 표현하는 α등의 색상 데이터를 가진다.

Edge Walk & Span 처리는 특정 Pixel이 삼각형 외 내부를 찾아서 처리한다. 본 논문에서느 Center-Line 트래버설 알고리즘[10]을 이용하여 처리한다. 그리고 Center -Line 트래버설 알고리즘 처리는 8단계로 구성되며 그림 16과 같이 하드웨어 구조를 설계하였다.

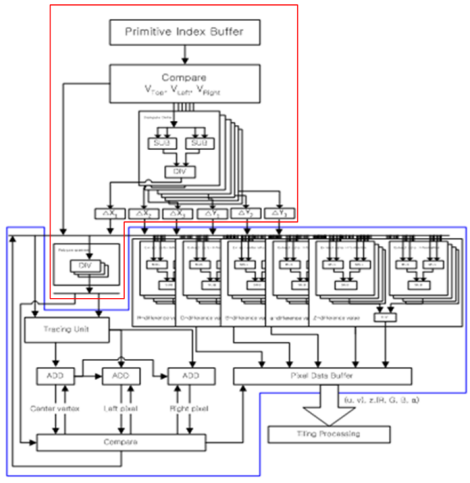


그림 16. 프래그먼트 처리를 위한 다이어그램
Fig. 16. Configuration diagram for Fragment processing

다. Tiled 처리

일반적인 타일기반 렌더링 기법에는 Non-tiling 기법과 Tiling 기법등으로 나눈다 Non-tiling 기법은 모든 object들을 렌더링하고 사용자에게 제공 하고 object들 배치하여 화면에 출력하므로 Object 모든 정점과 Pixel들을 처리되어 대역폭이 증가하고, 그림 17의 (a)와 같다.

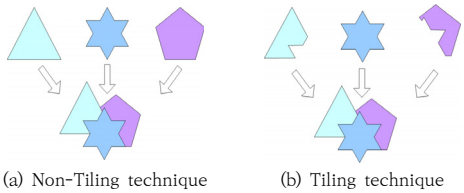


그림 17. 타일 기술
Fig. 17. Tiled technique

Tiling 기법은 화면에 보여주는 부분을 분리하고 화면에 보여 주는 부분만 렌더링하여 처리되어 대역폭이 감

소하고, 그림 17의 (b)와 같다.

본 논문에서는 그림 17의 (b) Tiling 기법을 MAMS를 이용한 하드웨어로 그림 18과 같이 구성된다.

구성된 Tiling Processor는 Pixel Data Buffer에 임시 저장된 데이터 U, V, Z, R, G, B, α 등을 입력 받아 Z-Compare에서 깊이 비교를 한다. 깊이 비교는 입력받은 U, V 좌표에서 Pixel MAMS에 저장된 데이터가 있는 지를 확인하고, 데이터가 존재하지 않으면, Pixel MAMS에 입력 된 Pixel 값을 저장하고, Tilde Primitive MAMS에 1을 저장한다. 만약에 데이터가 존재하면, Pixel MAMS에 저장되어 있는 깊이 값(Zold)과 입력된 깊이 값(ZNew)을 비교하여 깊이 값이 작은 값을 저장하고, Tilde Primitive MAMS에 1을 저장한다. 그리고 Tilde Primitive MAMS는 초기 상태 확인 시 모든 값에 0을 입력하고, 처리하는 모든 과정에 사용되는 사이클은 27+5+27Clk 이다.

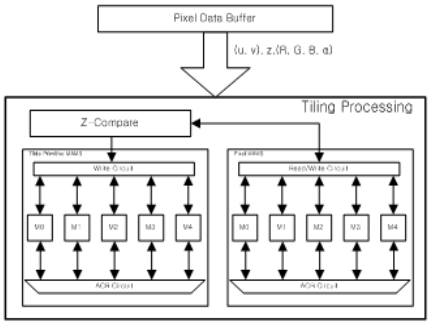


그림 18. MAMS를 이용한 타일링 프로세서
Fig. 18. Tiling Processor using MAMS

Ⅳ. MAMS를 이용한 3D그래픽스 구현

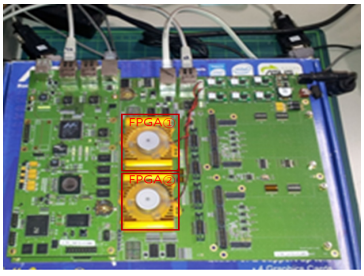


그림 19. FPGA 보드 구성
Fig. 19. Configuration of board by FPGA

본 장에서 새로이 제안한 Vertex Shader, Pixel Shader와 Tiling & Rasterizing을 nVidia와 비교하기 위하여 그림 19와 같이 2개의 FPGA(Xilinx Virtex6 @100MHz) 보드를 구성하였고, Verilog를 사용하여 FPGA①에는 기존 구조를 FPGA②에는 제안한 구조를 구성하였다.

Table 1은 nVidia Processor, 제안한 MAMS Processor 처리 기반을 보여주고, nVidia Processor는 nVidia GeForce GTX660@ 980 MHz이고 기존 방식과 제안한 MAMS Processor의 클럭 속도는 100 MHz이다.

표 1. 영상처리와 프로세서 처리의 상관관계
Table 1. The processing base of correlation image algorithm.

Classification	Existing Processor	Proposed Processor
Clock speed	nVidia GeForce GTX660@980MHz	FPGA Vertex6 @100MHz
Shader unit	960	4
Operating system	Window7	Linux
Program	Visual C++, DirectX(API사용)	Verilog(FPGA구성), C++(FPGA 실행)

MAMS를 이용한 3D 그래픽스 렌더링 구현을 위해 Verilog로 Vertex Shader, Pixel Shader와 Tiling & Rasterizing을 그림 20과 같이 하였으며, 파란색 내부는 Vertex Shader가 처리 시 사용되는 영역을 의미하며, 붉은색 내부는 Pixel Shader가 처리 시 사용되는 영역을 의미한다. 그리고 녹색 내부는 Tiling & Rasterizing이 처리 시 사용되는 영역을 의미한다.

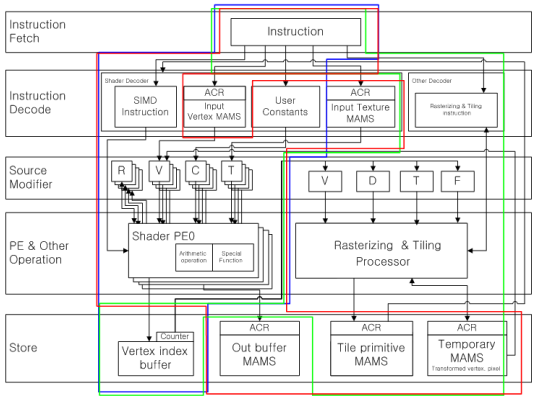


그림 20. MAMS를 이용한 3D 그래픽 렌더링을 위한 FPGA 내부 구조
Fig. 20. Internal structure of FPGA for 3D graphics rendering using MAMS

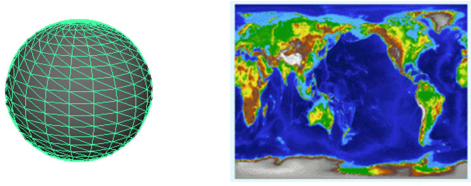
MAMS를 이용한 3D 그래픽스 렌더링을 위한 FPGA 내부 구성은 총 Instruction Fetch, Instruction Decode, Soure Modifier, PE & Other Operation, Store등으로 5개의 Pipelining으로 구조로 구현하였다. 그리고 Pipelining 구조의 순서대로 하나의 명령어를 처리하는데 Table 2와 같이 구현하였으며, Vertex Shader는 99clk이 사용되고, Tiling & Rasterizing는 73clk, 마지막으로 Pixel Shader 는 117clk 사용하도록 구현되었다.

표 2. Pipelining 명령어 처리 시간
Table 2. Pipelining command processing time

	Vertex shader	Tiling & Rasterizing	Pixel Shader
Instruction Fetch	6clk	6clk	6clk
Index	34clk	12clk	34clk
Source Modifier	3clk	3clk	3clk
PE Operation	38clk	-	38clk
Other Operation	-	25clk	-
Store	18clk	27clk	36clk
Sum	99clk	73clk	117clk

V. 실험 및 결과

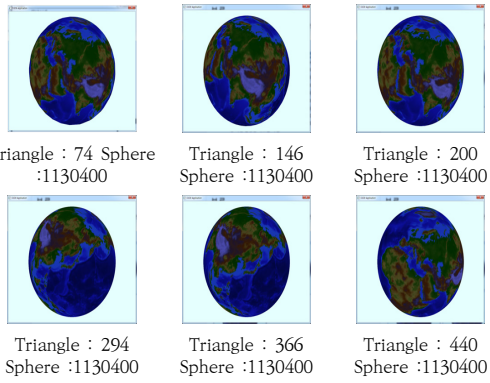
3D 그래픽스 렌더링 구현을 위해 구체(sphere)에 대한 링 그룹을 생성하여 Triangle 생성하고 pixel의 Texture 값을 입력하여 그림 21와 같이 3D 지구본을 구성하기 위한 구체와 Texture image를 보여 주고 있다.



(a) Sphere for Triangle (b) Texture image(800×600)

그림 21. 3D 그래픽 렌더링 구성
Fig. 21. Configuration for 3D graphics rendering performance

실험 방법은 Triangle 개수를 74, 146, 200, 294, 366, 440 등을 사용하였고, Sphere 전체 면적($4\pi r^2$) (r=300) 1130400 크기를 사용한 결과는 그림 22와 같은 구현 결과를 보여주고 있다.



Triangle : 74 Sphere :1130400 / Triangle : 146 Sphere :1130400 / Triangle : 200 Sphere :1130400
Triangle : 294 Sphere :1130400 / Triangle : 366 Sphere :1130400 / Triangle : 440 Sphere :1130400

그림 22. 3D 그래픽 렌더링 구현 결과
Fig. 22. 3D graphics rendering implementation result

표 3. nVidia 프로세서의 처리 결과
Table 3. Processing results of nVidia Processor

	Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Ex 6
Triangle	74	147	220	294	366	440
Sphere	300 × 300					
1th	0.018793	0.018829	0.018842	0.018742	0.018857	0.018861
2th	0.018810	0.018790	0.018793	0.018830	0.018846	0.018861
3th	0.018801	0.018790	0.018792	0.018843	0.018808	0.018813
4th	0.018840	0.018790	0.018843	0.018849	0.018808	0.018811
5th	0.018723	0.018801	0.018831	0.018800	0.018808	0.018812
6th	0.018811	0.018818	0.018793	0.018802	0.01885	0.018812
7th	0.018824	0.018809	0.018851	0.018800	0.018857	0.018812
8th	0.018831	0.018848	0.018734	0.018850	0.018807	0.018861
9th	0.018781	0.018731	0.018822	0.018850	0.018827	0.01885
10th	0.018782	0.018819	0.018835	0.018801	0.018866	0.018812
11th	0.018781	0.018832	0.018842	0.018801	0.018749	0.018811
12th	0.018831	0.018839	0.018792	0.018812	0.018837	0.018861
13th	0.018831	0.018789	0.018793	0.018829	0.018808	0.018811
14th	0.018782	0.018790	0.018792	0.018820	0.018807	0.018841
15th	0.018781	0.018789	0.018842	0.018859	0.018857	0.018854
16th	0.018831	0.018839	0.018804	0.018801	0.018857	0.01887
17th	0.018820	0.018839	0.018821	0.018810	0.018808	0.018753
18th	0.018782	0.018790	0.018812	0.018850	0.018808	0.01884
19th	0.018782	0.018789	0.018793	0.018830	0.018819	0.018823
20th	0.018782	0.018839	0.018793	0.018801	0.018836	0.018831
Average	0.018800	0.018808	0.018811	0.018819	0.018826	0.01883

Table 3는 nVidia Processor의 20회 처리 결과를 보여주고 있고, 20회 처리 결과 처리시간이 일정하지 않음을 확인 할 수 있다.

표 4. 제안된 MAMS 프로세서의 처리 결과
Table 4. Processing results of proposed MAMS Processor

		Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Ex 6
Triangle		74	147	220	294	366	440
Sphere		300 × 300					
Vertex	1th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	2th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	3th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	4th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	5th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	6th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	7th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	8th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	9th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	10th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	11th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	12th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	13th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	14th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	15th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	16th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	17th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	18th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	19th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	20th	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	Average	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
Pixel	1th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	2th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	3th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	4th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	5th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	6th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	7th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	8th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	9th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	10th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	11th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	12th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	13th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	14th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	15th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	16th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	17th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	18th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	19th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	20th	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	Average	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
Tiling and Rasterizing		1.5323	1.5325	1.5327	1.5331	1.5334	1.5337
Total		4.974	4.976	4.977	4.979	4.981	4.982

Table 4는 제안한 MAMS Processor의 20회 처리 결과를 보여 주고 있고, MAMS를 이용한 Vertex Shader와 Pixel Shader는 처리시간이 일정함을 확인하였다. 그리고 Table 6은 nVidia Processor와 제안한 MAMS Processor의 전체 처리 결과와 Ratio를 보여 준다.

Table 5와 같이 Triangle의 개수에 따라서 처리 시간이 증가함을 확인할 수 있었고 nVidia Processor와 제안한 MAMS Processor ratio는 0.0037이며, 동일한 클럭 사이클에 의한 ratio는 0.37이므로 제안한 MAMS Processor가 현재 FPGA를 이용해서는 nVidia Processor를 극복할 수 없었다. 하지만 3D 렌더링 처리에 성공적인 결과를 확인할 수 있었고, Cache의 Miss로 인하여 대역폭이 증가 및 처리시간이 일정함을 확인하였다.

표 5. nVidia 프로세서 및 제안 된 MAMS 프로세서의 처리 결과
Table 5. Processing results of nVidia Processor and proposed MAMS Processor

		Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Ex 6
Num. of Triangle		74	147	220	294	366	440
Sphere radius		300 × 300					
nVidia Result	Average	0.018800	0.018808	0.018811	0.018819	0.018826	0.01883
MAMS Result	Vertex Shader	0.0014	0.0027	0.0041	0.0054	0.0068	0.0081
	Pixel Shader	3.4410	3.4410	3.4410	3.4410	3.4410	3.4410
	Tiling and Rasterizing	1.5323	1.5325	1.5327	1.5331	1.5334	1.5337
	Total	4.974	4.976	4.977	4.979	4.981	4.982
Ratio of nVidia vs MAMS		0.0037	0.0037	0.0037	0.0037	0.0037	0.0037
Ratio at the same clock cycle		0.370	0.370	0.370	0.370	0.370	0.370

Ⅵ. 결 론

본 논문에는 다중접근기억장치(MAMS: Multi-Access Memory System)를 영상처리시스템 보드가 실험용 FPGA(100Mhz)를 사용하여 설계하였고, 비교를 위하여 nVidia GeForce GTX 660을 사용하는 비교 알고리즘을 개발하였다. GTX 660을 사용한 3D 렌더링 결과 처리시간이 일정하지 않았지만, MAMS Processor의 처리시간이 일정함을 확인하였다.

nVidia Processor와 제안한 MAMS Processor는 동일한 Cycle time만 예상하여 단순 속도 비교에서는 nVidia Processor가 처리 속도가 빠름을 확인하였지만, 단순 속도 비교가 아니라 Vertex 와 Pixel Shader의 연산기들의 개수들을 이용하여 비교 하면 정확한 성능을 확인할 수 있다. 향후 ASIC을 위해 SIMD Pipeline구조인 병렬처리 Processor의 성능과 Vertex 와 Pixel Shader의 연산기들의 개수를 미리 예측할 수 있는 성능 측정 방법을 이용하여 Cycle time과 처리기의 개수 정량적으로 비교할 수 있도록 연구가 진행되어야 할 것이다.

References

[1] Jae-Woo kim, “A Design of lighting processor for mobile 3D graphics”,Seokyoung University, 2005.

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

[3] Joung-Won Park, "An Efficient Memory System for Image Processing,“ IEEE Trans. Computers, Vol. C-35, No. 7, pp. 669-674, Jul. 1986. DOI:10.1109/TC.1986.1676813

[4] D.C. Van Voorhis, T.H. Morrin, "Memory System for Image Processing," IEEE Trans. on Computers, Vol. C-27, No. 1, pp. 1145-1155, Dec. 1975.

[5] Joung-Won Park, "Multi Access Memory System for Attached SIMD Computer," IEEE Trans. on Computers, Vol. 53, No. 3, pp. 439-452, Apr. 2004. DOI:10.1109/TC.2004.1268401

[6] Joung-Won Park, "An efficient buffer memory system for subarray access," IEEE Trans. on Parallel and Distributed Systems, Vol. 12, No. 3, pp. 316-335, Mar. 2001. DOI:10.1109/71.914779

[7] Seung-Hyun Cho, “An Efficient Texture cache for Programmable Vertex Shaders,” Korea Advanced Institute of Science and Technology, 2006.

[8] Jun-se Kim, “A Design of Tile based rendering for a Multi-Core GPU,” Seokyoung University, 2012.

[9] Hyeong-Gi Jeong, “A Design of a 3D Graphics Programmable Unified Shader for Mobile Devices”, Seokyoung University, 2007.

[10] B. Kelleher , "Pixel Vision Architecture," Technical Not 1998- 013, System Research Center , Compaq Computer Corporation , Oct . 1998

[11] Yun-Ok Park, Joung-Won Park, “A study on the 3 Dimension Graphics Accelerator for Phong Shading Algorithm,” Journal of The Institute of Internet, Broadcasting and Communication (JIIBC) No, 5, pp. 97-103, Oct. 2010.

[12] Seok-Hyun Kim, “Rebdering States Changing Costs Reducing Technique for Real-time 3D Graphics,” Journal of the Korea Academia-Industrial cooperation Society(JKAIS), Vol. 10, No.8, pp. 1843-1849, 2009. DOI:10.5762/KAIS.2009.10.8.1843

[13] Bong-Soo Sohn, “An Efficient Contour Propagation Algorithm Based on GPU for 3D Volume Visualization,” The Journal of KIIT, Vol. 9, No.11, pp. 219-229, 2011.

저 자 소 개

이스라엘(정회원)

- 2005년 영상대학교 컴퓨터공학과 학사졸업.
- 2011년 충남대학교 정보통신공학과 석사졸업.
- 2011년 충남대학교 정보통신학과 박사과정.

- 주관신분야 : 병렬처리, 영상처리

김 재 희(정회원)

- 2005년 영산대학교 컴퓨터공학과 학사졸업.
- 2009년 울산대학교 교육대학원 전자계산학과 석사졸업.
- 2017년 충남대학교 정보통신공학과 박사졸업

- 주관신분야 : 병렬처리, 영상처리

고 경 식(정회원)

- 1986년 동국대학교 전자계산학과 학사졸업.
- 2006년 충남대학교 정보통신공학과 석사졸업.
- 2006년 충남대학교 정보통신학과 박사과정.

- 주관신분야 : 병렬처리, 영상처리

박 종 원(정회원)

- 1979년 충남대학교 전자공학과 학사졸업.
- 1981년 한국과학기술원 전산학과 석사졸업.
- 1991년 한국과학기술원 전산학과 박사졸업.

- 1983~1993년 충남대학교 전산학과 부교수
- 1994년 ~ 현재 충남대학교 정보통신학과 교수
- 주관신분야 : 병렬처리, 영상처리, 의학영상처리

※ 이 연구는 충남대학교 학술연구비에 의해 지원되었음.