58
논문번호11-03-09

멀티코어 GP-GPU 기반의 OpenVG 가속기 구현

Implementation of OpenVG Accelerator based on Multi-Core GP-GPU

이 광 엽*, 박 종 일*, 이 찬 호**
Kwang-Yeob Lee*, Jong-Il Park*, Chan-Ho Lee**

Abstract

Recently, processing burden of CPU is growing because of graphical user interface according to enhance the performance of mobile devices and various graphical effects and creation of contents with 3D graphical effect or Flash animation. Therefore, the GPU are introduced to mobile device for support to variety contents. In this paper, OpenVG accelerator was implemented based on multi-core GP-GPU. OpenVG accelerator is verified using the sample image provided by Khronos group, and overall function is processed by only instruction set without dedicate hardware. The performance of processing the Tiger Image was 2 frames/sec.

요 약

최근 모바일 환경에서도 GUI(Graphic User Interface)나 3D 컨텐츠, Flash 등 다양한 그래픽 효과를 이용한 멀티미디어 컨텐츠들이 요구 된다. 이러한 컨텐츠들을 지원하 위하여 모바일 기기에도 GPU (Graphic Processing Unit)의 탑재가 필요조건이 되었다. 본 논문에서는 모바일 환경에 적합하도록 설계된 GP-GPU를 이용하여 OpenVG 가속기를 구현하였다. OpenVG 가속기는 크로노스 그룹에서 제공하는 샘플 이미지들을 사용하여 검증하였으며, OpenVG에서 제공해야 하는 동작 및 기능들이 정상 동작함을 검증하였다. 본 논문에서 구현한 가속기는 Tiger Image 렌더링시 초당 2프레임의 성능을 가진다.

Key words : OpenVG, GP-GPU

Ⅰ. 서론

과거에 흑백 액정과 텍스트를 기반으로 정보를 표현했던 모바일 기기 및 임베디드 시스템은 최근 급격한 발전을 거듭해왔다. 고 해상도의 컬러 액정이 탑재되고, 그래픽 기반의 유저 인터페이스로 변화하였다. 또한, PC 환경에서의 그래픽 효과를 경험한 사용자들의 요구에 맞추어 모바일 기기에서도 3D나 고수준의 그래픽 처리를 필요로 하는 응용 프로그램들이 탑재되고 있다. 이러한 그래픽 처리는 CPU만으로 수행하기에는 부담이 크기 때문에 그래픽 처리 장치인 GPU의 탑재가 불가피하다. 최근에는 빠르게 발전하고 있는 GPU의 연산 능력을 3D 그래픽스 이외의 다른 분야에도 활용하기 위한 연구가 진행되고 있는데, 이러한 형태의 GPU를 GP-GPU(General-Purpose computing on Graphics Processing Units)[1]라 한다. 기존의 GPU는 컴퓨터 그래픽스를 위한 연산만을 다루었지만, GP-GPU 구조에서는 부동소수점 연산과 병렬처리에 취약한 CPU를 대신하여, GPU가 해당 연산을 수행함으로써 처리속도를 높일 수 있다. 따라서, PC에 비하여 상대적으로 성능이 낮은 모바일 기기에서 유용한 기술이라 할 수 있다.

본 논문에서는 GP-GPU의 명령어를 통하여

* 서경대학교 컴퓨터공학과
** 숭실대학교 정보통신전자공학부
※ 감사의 글 (Acknowledgment)
본 논문은 ETRI 시스템반도체진흥센터에서 수행한 시스템반도체 융복합형 설계인재양성사업의 연구결과입니다.
接受日:2011年 08月 16日, 修正完了日: 2011年 08月 29日
掲載確定日: 2011年 09月 04日

OpenVG 가속기를 구현하였다. OpenVG의 데이터 처리과정 중에서 병렬처리가 가능한 부분을 멀티 코어, 멀티 스레드 구조를 통해 수행함으로써 처리 속도를 높이고, 외부 시스템 메모리와의 데이터 전송을 최소화하여 메모리 전송 시 발생하는 속도 저하를 줄였다.

II. 본론

1. GP-GPU의 구조

실험에 사용된 GPU는 명령어 기반의 GP-GPU로 멀티 코어, 멀티 스레드 프로세서이며 하나의 코어에 12개의 스레드가 동작한다.[2] 코어의 개수는 최소 1개부터 최대 16개 까지 늘이고 줄일 수 있는 스케일어블(Scalable) 멀티 코어 프로세서로 능동적으로 작업량을 할당 해주는 스크래치 카운터를 이용하여 코어의 개수가 변하더라도 추가적인 코드 수정이 필요치 않도록 S/W를 작성하는 것이 가능하다.

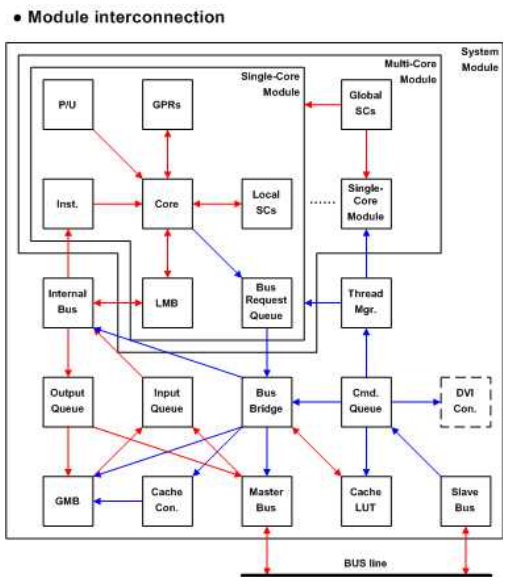

Fig. 1. System interconnection
그림 1. 시스템 구성도

그림 1에서 보여지는 것과 같이 GP-GPU는 각각 스레드마다 32비트 길이의 4D벡터로 구성된 128개의 GPR(General Purpose Registers)이 존재하고, 마찬가지로 코어마다 스레드가 공유하는 4D 벡터로 이루어진 2048개의 LMB(Local Memory Block)와 또한 모든 코어가 공유하는 GMB(Global Memory Block)가 존재한다. 이러한 계층적 메모리 구조는 외부 메모리 접근을 최소화 하며, OpenVG 가속기를 구현할 때 시스템 메모리 대신 LMB를 사용함으로써 메모리 접근을 상당히 줄일 수 있는 구조를 가지고 있다.

2. OpenVG 가속기 구현

본 논문의 OpenVG 가속기는 그림 2와 같이 6단계의 파이프라인으로 구성된다. Transformation 단계까지는 호스트 CPU를 통해 수행되며 그 이후의 정점 데이터를 멀티 코어 GP-GPU로 입력하여 나머지 Stage를 처리하게 된다.

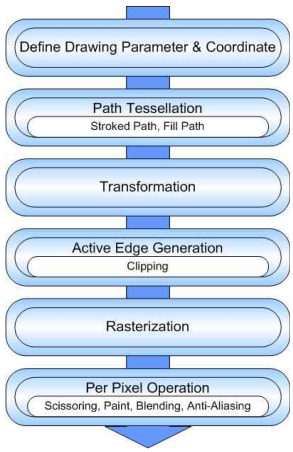

Fig. 2. Pipeline of OpenVG accelerator
그림 2. OpenVG 가속기 파이프라인

가. Active Edge Generation

본 논문에서 구현한 OpenVG 가속기는 Super Sampling을 포함하는 Scan-line Edge Flag 알고리즘[3][4]을 사용한다. Scan-line 단위로 렌더링을 수행하기 위해서는 전체 Edge 중 어느 Edge들이 현재 스캔라인에 영향을 미치는 가를 파악해야 하는데, 이 Edge를 Active Edge라고 한다. Active Edge를 생성하기 위해서 이전 단계인 Transformation을 마친 정점들을 연결하여 Edge를 생성하게 되며 생성된 Edeg들에 대해 각 Scan-line에서 렌더링에 실제 사용되는 Active Edge를 생성하는 과정을 수행한다.

Active Edge를 생성하는 과정에서 실제 렌더링 영역을 벗어나는 Edge들을 검출하여 제거하는 Clipping 과정도 같이 수행한다. 이는 Active Edge를 생성하는 과정과 Clipping 과정 모두 생성된 모든 Edge 데이터에 대해 연산을 수행하여야 하기 때문에 2 과정을 한 번에 수행하여 Edge 데이터에 접근하는 메모리 엑세스 횟수를 줄이기 위해서이다.

Active Edge들은 각 스캔라인 별로 분류하여 저장되는데 이 저장 공간을 액티브 엣지 테이블(Active

Edge Table)이라 한다.

Active Edge를 생성하기 위해서는 우선 시스템 메모리에 저장되어 있는 Edge를 로드하여, 몇 번째 스캔라인에 해당하는 Edge인지 판별한다. 이 후 해당 스캔라인에 해당하는 엣지 테이블 정보를 읽어와 현재의 Edge 정보를 갱신하여 Linked List로 저장한다. 그리고 엣지 테이블도 새로운 엣지로 갱신을 하여 저장한다. 이 경우에 하나의 Edge를 처리하는데 있어 2번의 Stream.load와 Stream.store가 발생하게 되며, Edge의 수가 많은 복잡한 그림의 경우, 메모리 엑세스에 의한 속도 저하가 더욱 심각해 진다. 이 문제를 해결하기 위해, 코어의 내부 공용 레지스터인 LMB를 이용하여 메모리액세스를 줄였다. LMB를 활용한 Active Edge 생성 과정은 그림 3에 나타내었다.

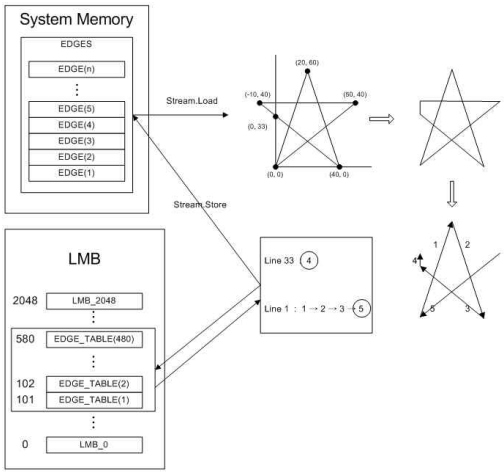

Fig. 3. Processing of Active Edge using LMB
그림 3. LMB를 활용한 Active Edge 생성 과정

Stream 명령어를 줄이기 위해서 Edge Table의 저장 공간을 시스템 메모리가 아닌 프로세서 내부의 LMB로 활용하였다. LMB는 하나의 원소마다 X, Y, Z, W 네 개의 컴포넌트로 구성되어 있으며, 각 코어마다 2048 vectors로 크기가 할당되어 있다. 코어 내의 스레드들은 하나의 LMB를 공유하여 사용한다. 이 중에서 480 vectors를 Edge Table로 사용하였으며, 이는 타겟 디스플레이 해상도를 최대 640*480으로 지정했기 때문이다. Edge Table에 저장되는 데이터는 실제 시스템 메모리의 주소로써, 첫 번째 엣지의 Base Address 정보에 Offset 값을 더하여 Edge Table에 저장된다. 이로써 2개의 Stream 명령어를 줄일 수 있다. 또한, 스캔라인 렌더링 시에 엣지 테이블에 계속 접근하여 해당 라인에 존재하는 액티브 엣지들을 불러오게 되는데 이 때에도, 시스템 메모리 대신 LMB를 사용함으로써 메모리 접근을 상당히 줄일 수 있다. 또한, 액티브 엣지 생성시 기울기가 0인 엣지는 생성하지 않는다. 해당 엣지를 생성하지 않아도 렌더링 시 같은 결과를 나타낼 수 있기 때문이다. 따라서, 기울기가 0인 엣지들에 대한 연산을 제거함으로써 속도를 더욱 향상 시킬 수 있다.

나. Rasterization

Rasterization은 Path에 의해 형성된 shape의 내부를 채우게 될 색상의 반영 비율을 계산하는 과정이다. 그림 4와 같이 각 픽셀에 대하여 색상이 채워지는 비율을 계산하여 저장하게 되는데, 이 값을 Coverage Value 라고 한다. 이 값은 컬러 정보는 포함하지 않은 비율 값만을 나타내며, Per Pixel Operation 단계에서 실제 컬러 값과 연산을 통해 출력될 색상을 결정한다.

본 논문에서는 OpenVG Specification에서 정의하고 있는 Better, Faster 그리고 Nonantialiased의 3단계의 렌더링 퀄리티를 지원하기 위하여 Super Sampling을 이용한 Anti-aliasing[5][6]을 적용하였다.

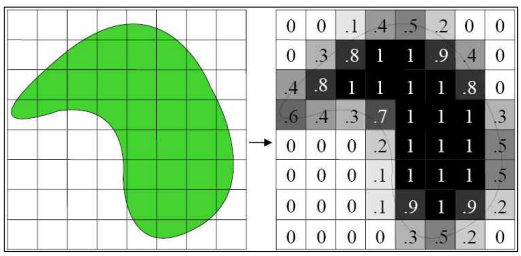

Fig. 4. Rasterization processing
그림 4. Rasterization 과정

(1) Super Sampling

Super Sampling은 픽셀당 한 개 이상의 샘플을 추출하여 Anti-aliasing을 구현하는 알고리즘이다. 이는 하나의 픽셀을 더 작은 단위로 분할하여, 분할된 서브 픽셀에 여러 개의 샘플 포인트를 둔다. 샘플 포인트는 각각 샘플 가중치(sample weight)를 갖고 있으며, 각 샘플 포인트를 판별하여 그 가중치의 합으로 Coverage Value를 계산한다. 샘플 포인트가 모두 shape의 내부로 판별될 경우 가중치의 합은 1이 된다.

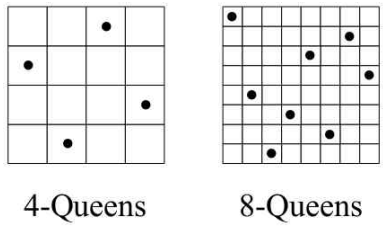

Fig. 5. N-Queens Sampling-position
그림 5. N-Queens 패턴 샘플링

본 논문에서는 n-Queens 패턴을 사용하여 샘플링을 구현하였으며, 이를 위해 Transformation 단계에서 이미지를 Y축으로 n배 Scaling 하는 작업을 수행한다.

그림 6은 슈퍼 샘플링 수행 후 그 결과를 Mask Buffer에 저장하는 과정을 나타낸다. 각 샘플링 포인트에서는 이미지의 내/외부를 판별하여, 외부일 경우는 0 내부일 경우는 1을 Buffer에 할당한다. Mask Buffer의 각 비트들은 서로 다른 가중치 값을 갖고 있으며, 이 가중치와 Bit값을 통하여 Coverage Value 값을 산출한다.

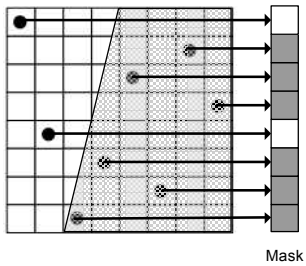

Fig. 6. Mask value creation
그림 6. Mask 값의 생성

(2) Per Pixel Operation

픽셀 오퍼레이션 단계는 색상의 채움 여부와 래스터라이제이션 단계에서 결정된 Coverage Value를 통하여 실제 출력될 픽셀의 색상 값을 결정하고, 출력하는 단계이다. 이 단계는 픽셀 당 연산을 수행해야 하기 때문에 Shape의 크기가 클수록 많은 시간이 소요된다.

렌더링은 그림 7과 같이 엣지 플래그 알고리즘을 통해 수행된다.

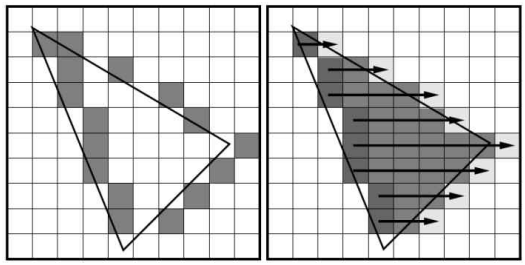

Fig. 7. Edge flag algorithm
그림 7. 엣지 플래그 알고리즘

각 스레드들이 하나의 라인을 처리하게 되며, 좌측에서 우측으로 진행해가면서 해당 픽셀의 Mask Buffer 값을 읽어와 연산을 수행한다. 하나의 픽셀마다 Mask Buffer 값을 읽어와 연산을 하여 그 결과인 출력 컬러를 프레임 버퍼에 저장해야 하기 때문에, 하나의 라인에 대해 width * 2번의 메모리 읽기/쓰기 동작이 수행되어야 한다. 이러한 많은 메모리 연산은 속도 저하를 불러오기 때문에 한 번에 여러 개의 Mask Buffer를 불러오도록 하고, 이에 대한 연산을 수행하는 동안 프리패치를 통하여 다음 연산에 필요한 Mask Buffer를 불러오도록 구현하였다.

12개의 스레드가 공유하는 LMB의 크기는 2048 vectors이며, 이중에서 환경 변수나 상수 값 등으로 사용되는 곳을 제외하고, 남은 저장 공간에 Mask Buffer를 저장할 수 있도록 하였다. Scratch Counter를 사용하면 각 스레드에 번호의 할당이 가능하며, 이렇게 할당된 카운터 값으로 LMB의 사용 공간을 나눌 수 있다. 따라서, 그림 8과 같이 LMB를 12개로 나누어 각 스레드가 별도의 저장 공간으로 사용이 가능하다.

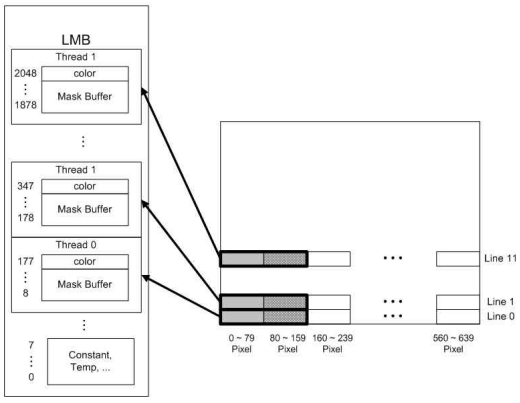

Fig. 8. Pixel procesing using LMB
그림 8. LMB를 사용한 픽셀 연산

Mask Buffer로 사용되는 LMB공간은 1920 vectors 이며, 12개의 스레드가 나누어 사용하므로, 하나의 스레드는 160 vectors의 저장 공간을 갖는다. 픽셀 오퍼레이션을 수행하기 위해서는 현재 처리해야 할 픽셀들에 대한 Mask Buffer 값과 프리패치를 통해 읽어올 Mask Buffer 값을 저장해야 하기 때문에, 이를 나누어 한번에 80개의 픽셀에 대한 Mask Buffer를 저장하도록 하였다.

또한, 최종적으로 출력될 컬러 값은 한 스레드당 16bpp(bit per pixel)*80pixel으로써 이것은 10vectors의 저장 공간을 필요로 한다. 따라서 하나의 코어에서 컬러를 저장하기 위한 공간은 120 vectors이고, Mask Buffer와 컬러를 저장하기 위해서 총 2040 vectors 크기의 LMB를 사용한다.

결과적으로 한 라인에 대해 각각 8번(640/80pixels)의 메모리 읽기/쓰기 동작을 수행하고, 추가적으로 load 명령어 수행 시 프리패치를 통해 메모리 지연시간을 줄임으로써 속도 향상을 가져올 수 있었다.

III 구현 및 검증

멀티 코어 GP-GPU의 구조적 특징을 이용하여 설계한 OpenVG 가속기를 검증하기 위해 테스트 드라이브를 사용하여 검증 하였다. 테스트 드라이브는 System Verilog의 DPI를 이용하여 C와 HDL을 연동할 수 있도록 구현한 통합 검증 시스템이다.

Fig. 9. OpenVG smaple verification
그림 9. OpenVG 샘플 검증

검증 이미지는 크로노스 그룹에서 제공하는 Tiger Sample Image를 사용하였다. Tiger Sample Image는 304개의 Path로 구성된 복잡한 이미지로 OpenVG의 다양한 기능들을 복합적으로 사용하여 동작, 기능 검증에 가장 많이 사용된다.

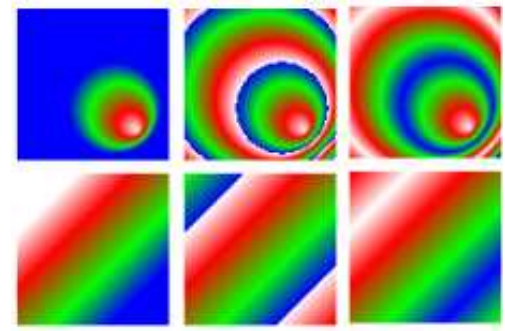

Fig. 10. Gradient paint verification
그림 10. 그래디언트 페인트 검증

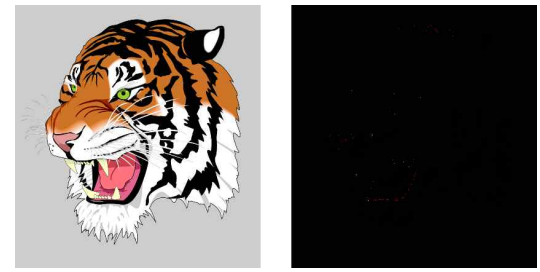

Fig. 11. Output image
그림 11. 출력 결과 이미지

그림 11은 크로노스 그룹에서 공개한 샘플 이미지[7]와 본 논문에서 구현한 OpenVG 가속기의 출력을 비교한 결과이다. 오른쪽 그림에서 비교 픽셀의 색상이 어두울수록 원본 이미지와의 색상 차가 적고, 색상이 밝아질수록 원본 이미지와의 색상 차가 많이 발생한 것이다. 결과화면에 출력 된 픽셀은 총 720,376개 이며 이중 61개의 픽셀에서 색상차이를 보였다. 구현된 가속기에서 사용되는 32bit Fixed Point 연산과 GP-GPU 내부에서 사용되는 24bit 연산기에 의해 약간의 차이가 발생하였지만, 99%의 정확도를 보임을 알 수 있다.

실제 동작 여부를 검증하기 위해 그림 12와 같이 Xilinx 사의 Virtex-6보드인 ML605보드를 사용하였다. 7개의 코어로 GP-GPU를 구성하였으며, 윈도우 환경에서 PCI Express 버스를 통해 컴파일된 프로그램과 데이터 입력이 이루어지며 DVI포트에 연결된 모니터로 출력 결과를 확인한다.

검증은 Tiger 이미지를 이용하였으며, 기존의 알고리즘과 수행 속도를 비교하였다. 그림 13은 Tiger 이미지의 렌더링 시간을 각각의 기능 적용 여부 별로 나누어 그래프로 나타낸 것이다.

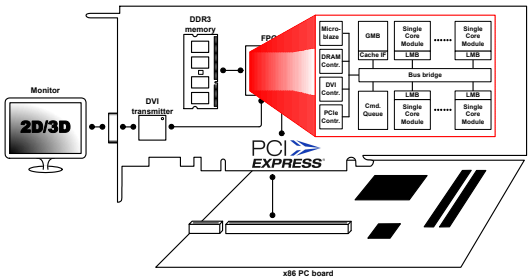

Fig. 12. Verification Environment of FPGA
그림 12. FPGA 검증 환경

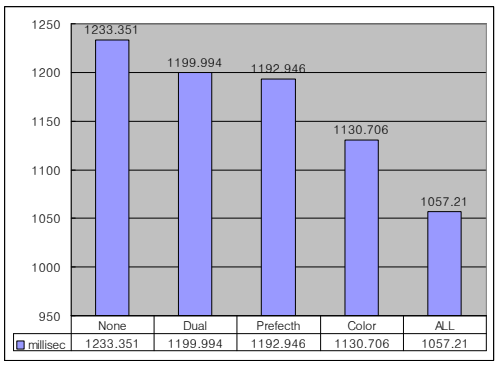

Fig. 13. Comparing the performance of implementations
그림 13. 구현 기능의 성능 비교

None의 경우는 멀티 스레드를 이용한 48개의 스캔라인 처리 이외에 다른 기능은 사용하지 않은 결과이다. Dual 항목은 None 상태에서 듀얼 페이즈 명령어를 적용하여 수행한 결과이고, Color는 None 상태에서 80개의 픽셀데이터들을 스트림으로 처리하는 경우의 결과이다. 마찬가지로 Prefetch는 None 상태에서 Prefetch만을 적용한 결과이며, 마지막 항목인 All은 Dual과 Prefetch, Color를 모두 적용하였을 때의 결과를 나타낸다.

듀얼 페이즈 명령어를 적용했을 경우, None의 경우에 비해 약 2.7%의 성능 향상을 나타내었으며, prefetch의 경우 기존 대비 3.2%의 성능 향상을 보였다. Color만 적용했을 시는 8.3%의 속도 향상이 있었으며, 이 세 가지 기능을 모두 적용하였을 때는 14.2%의 효율을 보였다.

OpenVG 가속기의 성능을 알아보기 위해 타 가속기와 렌더링 속도를 비교하여 표 1에 나타내었다.

Amanith VG는 OpenVG를 3D 하드웨어 가속을 이용하여 처리하는 제품으로 표에서 제시하는 수치는 320*240의 해상도를 기준으로 한 속도이다.

Table 1. Compare with rendering speed
표 1. 렌더링 속도 비교

	Multi-Core GP-GPU 100Mhz	Amanith VG	
		Freescale 533Mhz, FPU	TI OMAP 2420 330Mhz, FPU
Tiger	2.14	1.78	0.92

Frames/Sec

GP-GPU의 경우 640*480의 해상도를 기준으로 하였으며, 클럭 주파수의 차이도 존재하기 때문에 정확한 비교는 어렵지만, Amanith VG보다 낮은 클럭 주파수로 더 높은 해상도의 이미지를 처리함에 있어서도 더 나은 성능을 나타내는 것을 알 수 있다.

Ⅳ. 결 론

본 논문에서는 GP-GPU 환경에서 OpenVG 가속기를 구현하여 성능과 기능을 검증하였다. 검증에 사용된 GPU는 GP-GPU 구조로써 일반적인 GPU와 달리 그래픽 처리 이외에도 CPU의 연산을 나누어 처리가 가능한 장점을 갖고 있으며, 이를 멀티 코어로 구성함으로써, 성능의 향상과 저전력을 만족시킬 수 있도록 설계되었다. 이는 현재 모바일 기기에 장착 되어 있는 3D나 2D Vector, 동영상 가속을 위한 전용 가속기들을 하나의 GP-GPU로 대체하여 처리 할 수 있다는 실례를 보여주는 것으로써, 모바일 기기에서 GP-GPU의 가능성을 보여준다. 현재 구현된 OpenVG 가속기는 병렬 처리가 가능한 부분이 많지 않기 때문에 멀티코어에 의한 성능 향상이 크지 않지만, 좀 더 최적화된 알고리즘으로 개선할 경우 더욱 빠른 성능을 기대해 볼 수 있다.

참고문헌

[1] GPGPU. General Purpose Computation Using Graphics Hardware. http://www.gpgpu.org.

[2] 정형기, “능동형 스레드 관리기법을 적용한 멀티 스레드 멀티 코어 GP-GPU 설계”, 학위논문(박사), 2010년 6월

[3] M.E.Goss and K.Wu, “Study of supersampling methods for computer graphics hardware antialiasing”, HP Labs Technical Reports, Dec. 2000

[4] Ackland BD and Weste NH, “The edge flag algorithm: a fill method for raster scan displays” , IEEE Transactions on Computers archive, Volume 30 Issue 1, January 1981

[5] A. Schilling, "A new simple and efficient antialiasing with subpixel masks", ACM SIGGRAPH.

Computer Graphics, pp.133-141, July. 1991
[6] K. Doan, "Antialiased rendering of self-intersecting polygons using polygon decomposition", Proc. 12th Pacific Conf Computer Graphics and Applications, pp. 383-391, Oct. 2004
[7] OpenVG, Khronos OpenVG API Registry. http://www.khronos.org/registry/vg/

저 자 소 개

이 광 엽 (정회원)

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

박 종 일 (학생회원)

2011년 : 서경대학교 컴퓨터공학과 졸업 (공학사)
2011년 3월~ 현재 : 서경대학교 대학원 전자컴퓨터공학과 (석사과정)
<주관심분야> Embedded System, 3D Vision

이 찬 호 (정회원)

1987년 : 서울대학교 전자공학과 졸업 (공학사)
1989년 : 서울대학교 대학원 전자공학과 (공학석사)
1994년 : University of California Los Angeles, Dept. of Electrical Engr. (공학박사)
1995년 3월~현재 : 숭실대학교 정보통신전자공학부 교수
<주관심분야> SoC on-chip-network, 메모리 제어기, 영상인식, SoC 설계방법론, H.264 codec 구현