3D 그래픽스 가속 하드웨어 기술

Acceleration Hardware Technology of 3D Graphics

IT 융합 · 부품 기술 특집

조승현 (S.H. Cho) MMP개발팀 연구원
박성모 (S.M. Park) MMP개발팀 팀장
엄낙웅 (N.W. Eum) IT SoC연구본부 본부장

목 차

3D 그래픽스 관련 산업의 눈부신 성장은 GPU 기술의 발전을 기반으로 이루어졌다. GPU는 기존의 고정된 기능의 파이프라인을 벗어나 프로그램 가능한 형태로 발전하였으며 GPU의 프로그램 능력과 성능의 꾸준한 향상이 이루어지고 있다. 최근에는 GPU 내부의 연산 집중도의 불균형을 해결하기 위한 연구와 GPU의 연산능력을 다른 응용분야에 이용하기 위한 연구가 진행중에 있다. GPU를 이용한 3D 그래픽스 응용프로그램 개발을 위해서 산업 표준의 API들이 존재하는데 데스크톱용 API에서 필수 기능만을 골라 간략화한 모바일 기기용 프로파일 또한 정의되고 있다. 모바일 기기에 사용되는 GPU도 프로그램 가능한 구조로 진화하고 있으며 대중화되기 위해서는 전력소모를 낮추기 위한 노력이 필요하다.

I. 서론

지난 몇 년간 컴퓨터 3D 그래픽스 관련산업은 규모 면에서 가파른 성장을 보였을 뿐만 아니라 그 응용분야도 컴퓨터 게임으로부터 애니메이션 영화, 가상 캐릭터 등으로 폭넓게 확대되었다. (그림 1)은 확대된 그래픽스 응용분야의 예를 보이고 있다. 이러한 그래픽스 산업의 발전은 3D 그래픽스 처리를 가속하기 위한 하드웨어, 즉 GPU 기술의 눈부신 진보가 있었기에 가능했다. 최근에는 고사양의 GPU 성능을 요구하는 3D 그래픽스 응용기술이 많이 개발되면서 GPU 성능 향상의 구심점이 되고 있다. PC와 게임콘솔 등에 사용되는 GPU에는 가장 최신의 기술이 동원되어 그래픽 성능을 극대화시키고 있으며 여기에 사용되었던 기술이 휴대용 모바일 기기에도 점차 적용되고 있다.

(a) ETRI '디지털 액터'

(b) 픽사 애니메이션 '카'

(그림 1) 그래픽스 응용분야의 확대

<표 1>은 GPU가 장착되는 응용제품들에 대한 시장전망을 연평균성장률(CAGR)로 나타내고 있다[1]. 현재 GPU 시장의 대부분을 차지하고 있는 PC 산업의 경우 데스크톱 PC 시장의 비중이 줄어들고 노트북 PC 시장이 연평균 10.5% 성장하여 그 비중이 더욱 커지며, 통합 칩셋 시장은 연평균 9.2%로 꾸준하게 성장할 전망이다. 한편 게임 콘솔과 디지털 TV 시장의 경우 연평균 각각 35.5%, 22.2%라는 높은 성장률을 기록할 것으로 예견된다. 휴대용 단말기 시장의 경우 이미 연 8억 대 정도의 규모이고 2008년에는 10억 대를 넘어서며, 거의 대부분 멀티미디어 기능을 포함할 것으로 보인다. 이상에서 살펴볼 때, 과거 데스크톱 PC용 그래픽 카드를 중심으로 성장해온 GPU 시장이 디지털 TV, 게임 콘솔 등에까지 대폭 확대되고 노트북 PC, 통합 칩셋, 휴대용 기기에서의 GPU 사용도 꾸준히 증가될 것임을 알 수 있다.

〈표 1〉 GPU 응용제품 시장전망

(단위: Mega units)

	2005	2006	2007	2008	CAGR(%)
Desktop PC Discrete	82.1	82.3	81.6	80.4	-0.7
Notebook PC Discrete	21.2	21.3	25.6	28.6	10.5
Integrated Chipsets	162.2	183.5	195.2	211.2	9.2
Handheld	817.9	922.9	994.4	1071.9	9.4
Digital TV	73.7	98.3	114.9	134.5	22.2
Game Consoles	17	26.5	39.7	42.3	35.5

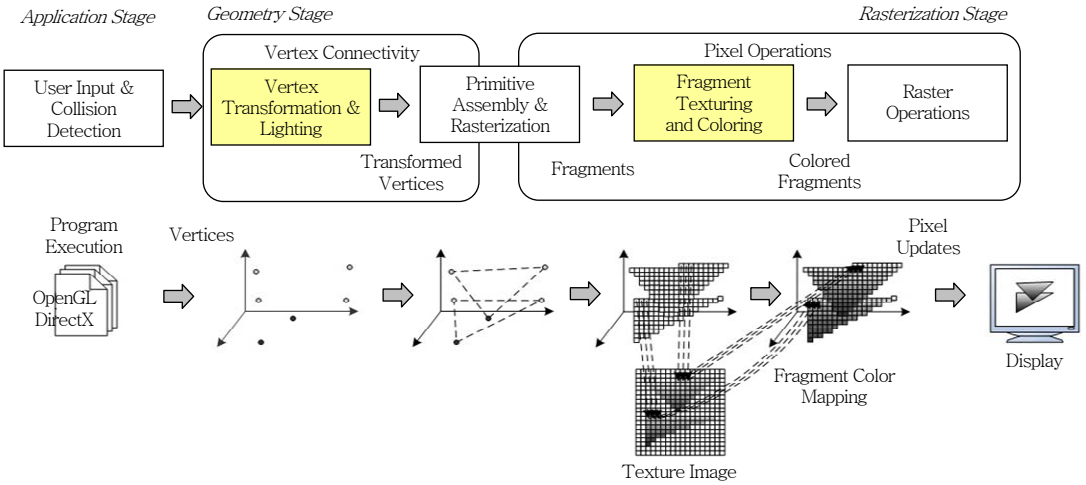
<자료>: ATI, 2006.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 3D 그래픽스 처리 과정에 대하여 간략히 설명한 후 GPU가 발전해온 과정을 살펴 봄으로써 현재의 GPU가 갖는 구조의 특성을 이해한다. Ⅲ장에서는 최신의 GPU에 도입되고 있는 통합 셰이더와 GPU를 다른 분야에 활용하기 위한 연구에 대하여 알아본다. Ⅳ장에서는 GPU의 발전과 응용프로그램의 고도화에 기여할 뿐만 아니라 이들 사이의 교량역할을 해주는 표준 API에 대해 살펴본다. Ⅴ장에서는 휴대용 모바일 기기에 사용되는 그래픽스 가속 하드웨어 기술에 대해 소개한다. 마지막으로 본 고에서 다루어졌던 내용들에 대해서 정리함으로써 결론을 맺는다.

Ⅱ. GPU의 발전과정

1. 3D 그래픽스 처리과정

일반적으로 3D 그래픽스의 처리과정은 크게 응



(그림 2) 3D 그래픽스 처리과정

용프로그램 단계(application stage), 지오메트리 단계(geometry stage), 그리고 래스터라이제이션 단계(rasterization stage)의 세 단계로 구분된다[2]. (그림 2)는 3D 그래픽스 처리과정을 간략하게 보여주고 있다. 응용프로그램 단계는 CPU에서 수행되며 사용자 입력 처리와 3D 오브젝트간 충돌 등과 같은 물리적 연산을 담당하고 지오메트리 단계에 버텍스(vertex) 데이터를 제공하는 역할을 한다. 버텍스는 3D 오브젝트를 이루고 있는 프리미티브, 즉 점 그 자체, 선의 양 끝점, 삼각형 또는 다각형의 꼭지점이다. 버텍스 데이터는 버텍스 좌표, 색깔, 텍스처 좌표, 법선 벡터 등을 포함한다.

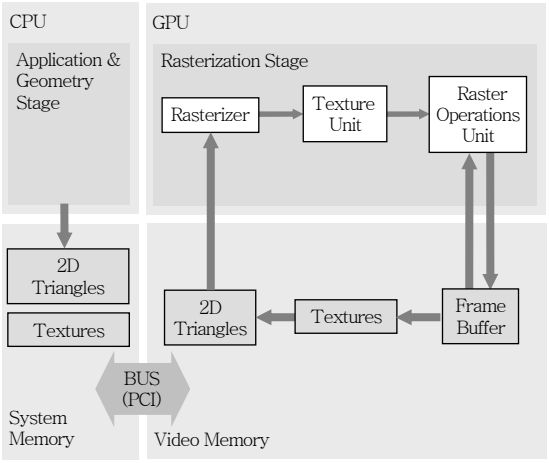
지오메트리 단계에서는 응용프로그램 단계로부터 입력된 각각의 버텍스에 대해 지오메트리 변환(transformation) 연산과 광원에 의한 색깔 값이 계산된다. 변환 연산은 변환 행렬과 버텍스 벡터의 곱셈에 해당하며 빛에 대한 계산은 광원의 특성에 따라 연산이 달라진다. 그 다음, 버텍스는 프리미티브 단위로 묶인 뒤 내부를 채우고 있는 각 픽셀의 색을 결정하기 위해 래스터라이제이션 단계로 보내지게 된다.

마지막 단계인 래스터라이제이션 단계에서는 미리 로딩되어 있는 텍스처 이미지를 외부 메모리로부터 읽어와 텍스처 매핑(texture mapping)을 하고 안개 효과(fog effect), 투명도(alpha) 반영 등의 나머지 픽셀별 연산을 통해 각 픽셀의 색깔 값을 결정하여 최종 결과를 프레임 버퍼(frame buffer)에 저장한다.

과거에는 앞서 살펴본 모든 단계를 CPU에서 처리하였으나 특화된 하드웨어를 통해 지오메트리 단계 또는 래스터라이제이션 단계를 가속시킴으로써 CPU의 부담을 덜어주고 3D 그래픽스 연산이 갖는 병렬처리가 가능한 특성을 살려 렌더링 성능을 높이기 위해 GPU가 등장하였다. GPU의 발전과정을 살펴보면 GPU를 구성하는 내부 하드웨어 구조의 변화, CPU와 GPU간의 데이터 전달을 위한 버스 성능의 개선에 따라 다음과 같이 크게 3세대로 구분할 수 있다[3].

2. 1세대 – 래스터라이제이션 단계를 가속하던 GPU

1990년대 중반에 등장한 초기의 GPU는 3D 그래픽스 래스터라이제이션 단계만을 가속할 수 있는 간단한 구조를 가지고 있었다. 비교적 적은 연산으로 3D 오브젝트에 사실감을 더하기 위해 그림이나 사진을 덧씌우는 텍스처 매핑이 가능했으며 GPU가 발전하면서 멀티-텍스처링(multi-texturing)이 가능해져 한 오브젝트에 매핑 가능한 텍스처의 수가 점점 늘어났다. 이 시기의 GPU와 CPU와의 통신에



(그림 3) 래스터라이제이션 단계 가속의 GPU

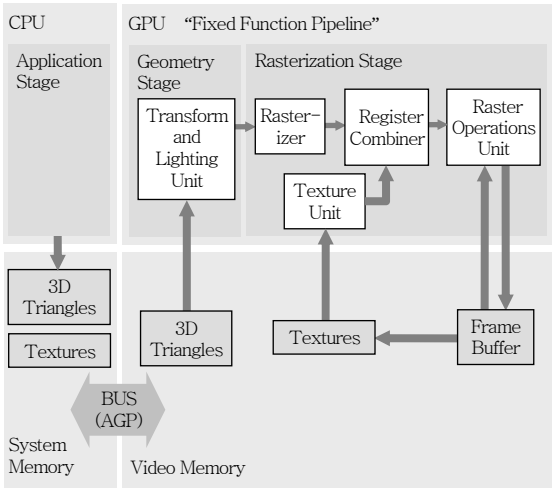
는 전통적인 방식의 PCI가 사용되었다. (그림 3)은 래스터라이제이션 단계만을 가속했던 초기형태의 GPU의 구조를 보여주고 있다.

3. 2세대 - 지오메트리 연산이 가능한 고정된 기능의 GPU

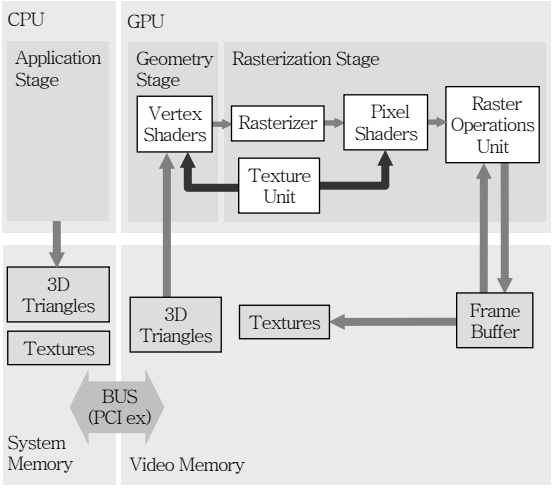
1990년대 후반에 등장하여 2000년대 초반까지 많이 사용되었던 GPU는 지오메트리 단계를 포함한 전체 3D 그래픽스 연산이 하드웨어로 가속되는 형태를 갖추었다. 지오메트리 단계는 3D 모델의 버텍스 변환과 빛에 대한 연산을 포함했다. 3D 모델은 고정된 기능의 하드웨어를 통해 짧은 시간 안에 렌더링되었고 멀티텍스처링 수는 보다 늘어났다. 따라서, CPU-GPU간의 데이터 전송을 위해 필요한 대역폭(bandwidth)은 급격히 늘어났다. AGP는 이러한 문제를 해결하기 위해 고안되어 사용되었다. (그림 4)는 지오메트리 단계와 래스터라이제이션 단계를 가속해주는 고정된 기능의 GPU의 구조를 보여준다.

4. 3세대 - 프로그램이 가능한 GPU

2000년대 초반에는 래스터라이제이션 단계에 프로그램 가능한 그래픽용 프로세서인 픽셀 셰이더(pixel shader)가 포함되었으며 얼마 지나지 않아 지오메트리 단계도 버텍스 셰이더(vertex shader)를 포함하게 되었다. 이로써, 이전의 고정된 형태의 GPU가 갖던 표현의 한계를 극복하여 게임 제작자 또는 3D 콘텐츠 개발자의 의도대로 보다 실제에 가까운 표현이 가능하게 되었다. 이러한 형태의 GPU는 현재 가장 보편적으로 사용되고 있으며 CPU와의 보다 빠른 통신을 위해 AGP 방식에서 벗어나 PCI Express를 사용한다. (그림 5)는 버텍스 셰이더와 픽셀 셰이더를 포함하는 프로그램 가능한 GPU의 구조를 보이고 있다. 초기의 셰이더는 어셈블리어 형태의 저수준 언어로 프로그램 되었지만 최근에



(그림 4) 고정된 기능의 GPU



(그림 5) 프로그램이 가능한 GPU

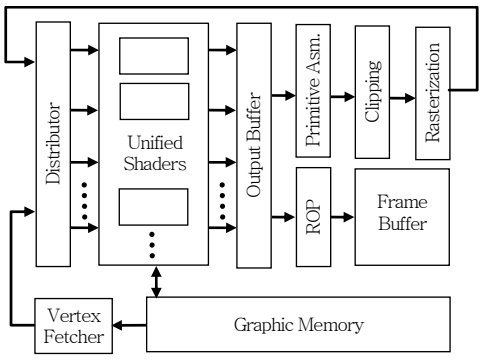
는 Cg[4], HLSL[5], GLSL[6] 등과 같이 C언어와 유사한 상위 수준의 셰이딩 언어(shading language)가 개발되어 셰이더 프로그램의 작성이 용이해졌다.

PC, 워크스테이션, 게임콘솔 등에 사용되는 GPU 기술은 성능을 극대화시키기 위해 각 단계의 연산이 동시에 처리가 가능한 그래픽의 특성을 이용하여 끊임없이 하드웨어 면적을 넓혀가고 있다. 또한, 버텍스 및 픽셀 셰이더의 프로그램능력(programmability)을 극대화하여 표현의 한계를 허물고 있다. 이들 셰이더는 복잡한 소수점 연산에 대해 범용 CPU의 성능을 능가하는데다 프로그램능력의 향상으로 과학적 시뮬레이션 등 다른 분야에의 활용 가치도 높다. 가장 최근에는 응용프로그램의 특성에 따라 연산량이 버텍스 셰이더 또는 픽셀 셰이더 어느 한 쪽에 집중되는 단점을 극복하고 하드웨어 사용의 효율성을 극대화시키기 위해 기능이 통합된 통합 셰이더(unified shader)[7]를 GPU에 내장하는 움직임도 일고 있다.

Ⅲ. GPU 관련 최신기술

1. 통합 셰이더

GPU에 사용되는 셰이더는 특화된 일종의 프로세서로, 3D 그래픽스 응용프로그램으로부터 전달받은 데이터 또는 GPU 내부의 다른 하드웨어 블록으로부터 전달 받은 데이터를 사용자가 기술한 프로그램의 입력으로 받아들여 처리할 수 있다. 3D 그래픽스의 처리과정에서 각각의 버텍스 및 픽셀 데이터는 서로 독립적이고 컴포넌트별로 수행되는 연산이 동일한 경우가 대부분이기 때문에 고성능을 목표로 하는 GPU는 다수의 버텍스 셰이더와 픽셀 셰이더를 병렬로 갖고 있으며 셰이더 내부에도 컴포넌트별 병렬처리 능력을 향상시키기 위한 구조를 택하고 있다[8],[9]. 하지만 경우에 따라 지오메트리 단계 또는 래스터라이제이션 단계 중 어느 한 쪽에만 연산이 집중되는 경우가 많기 때문에 하드웨어 자원, 즉



(그림 6) 통합 셰이더를 포함하는 GPU의 구조 예

셰이더의 효율적인 활용이 어려운 단점이 있다.

최근에는 이러한 단점을 보완하여 GPU의 막강한 성능을 충분히 활용하기 위해 통합 셰이더가 등장하였다[10],[11]. 통합셰이더는 버텍스 셰이더, 픽셀 셰이더의 구분이 없이 동일한 구조의 셰이더가 하나의 거대한 집단을 이루고 있으며 이들 각각의 셰이더는 경우에 따라 버텍스 셰이더 또는 픽셀 셰이더로 동작할 수 있기 때문에 경우에 따라 버텍스 셰이더와 픽셀 셰이더간의 연산 집중도의 불균형 문제를 해결할 수 있다. 뿐만 아니라 통합 셰이더는 하나의 공통된 명령어 세트(instruction set)를 사용하기 때문에 셰이더 프로그래밍이 보다 용이해졌으며 하드웨어 추가 없이 지오메트리 셰이더 등 다른 기능의 셰이더를 구현할 수 있다는 장점을 가진다. 하드웨어 (그림 6)은 통합 셰이더가 사용된 한 GPU의 내부 구조의 예를 나타내고 있다.

2. GPGPU

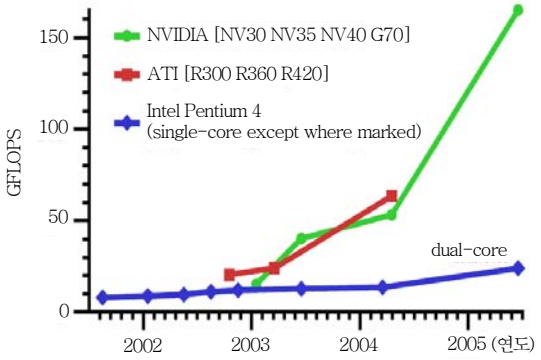
GPU의 성능이 발전되고 PC 및 워크스테이션에 널리 퍼져 사용됨에 따라 GPU의 연산능력(computing power)을 3D 그래픽스 이외의 다른 분야에도 활용하기 위한 연구가 활발히 진행되고 있다[12]. 이러한 시도는 GPU가 막대한 메모리 대역폭과 CPU를 능가하는 성능을 가진 값비싼 하드웨어임을 생각하면 당연하게 여겨지며, 뿐만 아니라 앞서 살펴본 바와 같이 고정된 구조를 벗어나 프로그램이 가능하며 유연한 구조로 발전하고 있기 때문에

필수적이라 할 수 있겠다. (그림 7)은 최근 GPU와 CPU의 성능의 발전을 부동소수점 연산을 기준으로 비교한 것이다. 최근 몇 년간 더욱 가파르게 증가하고 있음을 알 수 있다.

그러나, 이러한 GPU 성능의 비약적 발전은 고도로 병렬화된 컴퓨터 그래픽스 처리과정으로부터 최고의 성능을 도출하기 위해 만들어진 특화된 구조에서 기인한 것으로 일반적인 응용분야에 그대로 적용시켜 단순히 계산능력만큼의 성능향상을 기대하기는 어렵다. 또한, 하드웨어 구조가 급격히 변하는 경우가 많고 대부분 외부에 공개되지 않으며 프로그래밍 모델이 기존과 다르고 프로그래밍 환경에 제약이 있는 등의 제한사항도 존재한다.

대부분의 경우 한 화면을 구성하기 위한 프래그먼트의 수가 버텍스 수보다 많으므로 GPU 내부에서 가장 높은 연산능력을 필요로 하는 부분은 픽셀 셰이더이다. 버텍스 셰이더와 픽셀 셰이더의 수가 고정된 GPU가 버텍스 셰이더보다 많은 수의 픽셀 셰이더를 갖는 이유도 바로 여기에 있다. 따라서 GPGPU 프로그램은 주로 GPU의 픽셀 셰이더를 활용하도록 작성되고 있다.

사용자는 먼저 GPU를 통해 수행하고자 하는 응용프로그램에서 데이터 병렬성을 찾아내서 서로 독립적이며 병렬로 수행될 수 있는 여러 부분으로 나누어 각각을 픽셀 셰이더 프로그램으로 작성한다. 프로그램의 입·출력 데이터는 배열 형태로 텍스처 메모리에 저장할 수 있다. 픽셀 셰이더는 각각의 프래그먼트에 대해 수행이 되고 프래그먼트의 수는 폴리곤(polygon)의 크기에 의해 결정되므로 출력 데이터 범위의 조절은 GPU로 전달하는 버텍스의 좌표를 조절하는 것으로 가능하다. 보통의 경우 화면과 평행한 사각형을 이용한다.



(그림 7) CPU와 GPU의 성능 발전 비교

Ⅳ. 3D 그래픽스 가속을 위한 API

1. OpenGL

OpenGL은 Open Graphics Library의 약자로 1990년 SGI에 의해 고안된 3D 그래픽스 하드웨어를 위한 소프트웨어 인터페이스로써, 3D 그래픽스 응용프로그램을 만들기 위한 API이다[13]. OpenGL은 로열티가 없는 공개 표준으로 하드웨어 제작자와 이를 이용하는 응용프로그램 개발자가 서로 독립적으로 일하는 것을 가능하게 한다. 개발 초기부터 Sun, DEC, SGI 등의 다양한 플랫폼에서 Windows 95, X Windows, Windows NT, OS/2 등 다양한 운영 시스템에 이식되어 성능이 검증된 OpenGL은 최근 크로노스 그룹[14]의 일부가 된 OpenGL ARB 워킹 그룹에 의해 제정과 관리가 이루어지고 있다. OpenGL은 주로 데스크톱 PC나 워크스테이션에서 사용된다.

OpenGL은 GPU의 발전과 밀접하게 관련이 있어 초기에는 고정된 기능의 파이프라인 형태를 유지하다가 프로그램 가능한 파이프라인의 형태를 갖추게 되었다. OpenGL 1.0, 1.1, 1.2, 1.3, 1.4, 1.5는 고정된 기능의 파이프라인에 해당하며 상위 버전으로 갈수록 기능이 추가되었고 1.5 버전 이후로는 고정된 파이프라인을 갖는 버전은 만들어지고 있지 않다. 반면 OpenGL 2.0과 최근에 발표된 2.1 버전은 버텍스 프로세서와 프래그먼트 프로세서를 갖는 프

● 용 어 해 설 ●

버텍스 프로세서: 앞서 살펴본 버텍스 셰이더와 동일하게 사용된다.

프래그먼트 프로세서: 앞서 살펴본 픽셀 셰이더와 동일하게 사용된다.

로그램 가능한 파이프라인을 갖는다. 이들 프로세서를 프로그래밍 하기 위한 GLSL도 함께 발표되고 있다. OpenGL 2.x는 프로그램 가능하도록 파이프라인에 큰 변화를 주었지만 앞선 버전들에 대한 호환성을 유지하고 있다. 다시 말해 1.x에서 실행되던 응용프로그램이 2.x에서 수정 없이 실행된다. 한편, OpenGL ARB는 가장 최근에 OpenGL 3.0에 대해 공식적으로 발표하였으며 사양서는 최종 승인단계를 거쳐 배포될 예정이다.

OpenGL ES는 3D 그래픽을 지원하는 내장 시스템(embedded system)을 위한 API로 크로노스 그룹에 의해 제정과 관리가 이루어지고 있다. OpenGL ES는 OpenGL과 마찬가지로 로열티가 없는 공개 표준이며, OpenGL의 부분집합으로 구성된 프로파일이다. OpenGL ES는 소프트웨어와 GPU간의 유연하면서도 강력한 저수준의 인터페이스를 제공하는 한편 메모리가 작은 환경과 저전력이 요구되는 환경에 적합하여 주요 모바일 및 내장 시스템 플랫폼 환경에서 3D 게임과 다양한 고급 3D 그래픽 기능을 제공하는 데 기여하고 있다. OpenGL ES는 OpenGL을 기반으로 하고 있으므로 특별히 새로운 기술을 배울 필요 없이 응용프로그램 개발의 시너지 효과를 기대할 수 있다.

OpenGL ES 1.1은 고정된 파이프라인을 갖고 있으며 1.0과 완전한 호환이 가능하다. 반면에 2.0은 프로그램 가능, 즉 버텍스 및 프래그먼트 프로세서를 포함하며 1.x와의 완벽한 호환을 지원하지 않는다. OpenGL이 고정된 기능의 파이프라인을 갖는 버전을 더 이상 제정하고 있지 않는 것과 달리 OpenGL ES는 1.x와 2.x가 각각 성능 개선을 위한 개정이 진행중에 있다.

2. DirectX Direct3D

우리가 잘 아는 마이크로소프트사의 DirectX는 Windows 운영체제 기반 컴퓨터를 멀티미디어 응용프로그램을 위한 이상적인 플랫폼으로 만들기 위해 기술된 기술 모음으로써, DirectX를 사용하면 2D 그래픽, 비디오, 3D 그래픽, 오디오 등 다양한 멀티미디어 구성요소를 이용하여 응용프로그램을 개발할 수 있다. 이 가운데 Direct3D는 DirectX 중에서 가장 큰 부분을 차지하며 활발한 개정이 이루어지고 있다. Direct3D는 가장 보편적으로 많은 게임 개발업체들이 지원하는 API이다[15].

Direct3D 역시 GPU의 발전 과정과 밀접하게 연관되어 있는데, 최근에는 Direct3D의 다음 버전이 발표되면 하드웨어 제작사들은 이를 지원하는 GPU를 경쟁적으로 내놓고 있는 추세이다. GPU가 나오기 시작하던 초기에는 OpenGL의 사용만이 3D 그래픽스 응용프로그램을 개발하기 위한 유일한 효율적인 방법이었지만 DirectX의 초기 버전이 발표되고 빠른 속도로 새로운 기법들을 추가해감으로써 인기를 얻기 시작하였다. 대부분의 사용자들이 Windows 운영체제를 사용하고, 따라서 PC용 3D 게임 역시 대부분 Windows를 위해 개발되기 때문에 DirectX의 입지는 더욱 굳건해지고 있다.

Direct3D와 OpenGL은 완전히 일치하지는 않지만 적어도 비슷한 기능을 지원하는 API들이 많으며 한 쪽에서 가능한 일은 대부분 다른 쪽에서도 가능하다. Direct3D는 새 버전을 발표하면서 꾸준히 새로운 버전의 '셰이더 모델'을 발표하고 있다. 셰이더 모델은 버텍스 셰이더, 픽셀 셰이더 등을 프로그래머 입장에서 바라본 모델로 입출력 레지스터, 사용가능한 명령어 등을 정의한다. 현재 최신의 Direct X 10에서는 기존의 버텍스 셰이더와 픽셀 셰이더 이외에 새롭게 지오메트리 셰이더가 추가된 '셰이더 모델 4.0'을 지원하도록 하고 있다.

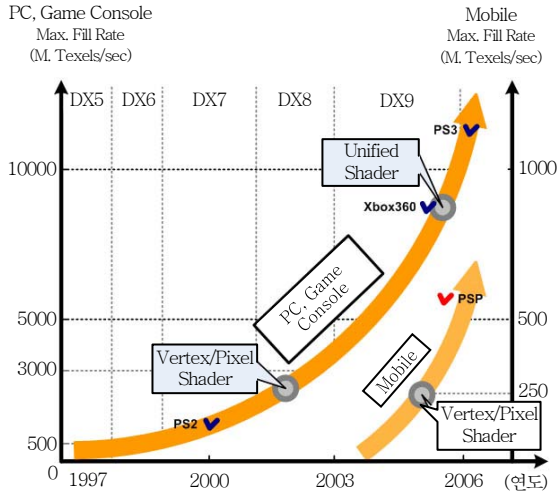
모바일 기기를 위한 마이크로소프트의 운영체제인 Windows Mobile 5.0에서 Direct3D Mobile을 사용하여 3D 그래픽 응용프로그램을 작성할 수 있다. Direct3D Mobile은 Direct3D 8에 가장 근접하며 Direct3D 9버전의 일부 요소도 포함하고 있다. Direct3D Mobile은 데스크톱용 버전들보다 작은 런타임을 갖도록 간단하게 설계되어 있다. OpenGL ES가 그러하듯, Direct3D Mobile 역시 데스크톱 버전에서 필수적인 부분만을 모아 놓은 프로파일이

라 할 수 있다. 그러나, Direct3D Mobile에서는 아직 버텍스 및 픽셀 셰이더를 지원하지는 않고 있다.

V. 모바일 기기를 위한 GPU

앞 절들에서 살펴본 바, 3D 그래픽 하드웨어 기술은 PC용 GPU의 발전을 통해 끊임없는 진화를 거듭하여 막강한 연산능력과 프로그램능력을 바탕으로 실사에 가까운 표현이 가능한 단계에 이르렀다. 이 과정에서 정립된 필수 기능들은 모바일용 기기를 위한 표준 API들에도 반영이 되고 있다는 것을 알 수 있었다. (그림 8)은 PC 또는 게임 콘솔에 사용되는 고사양 GPU의 성능이 발전하는 추세와 모바일용 GPU의 성능이 발전하는 추세를 보이고 있다. 각 화살표의 세로축이 각각 좌우에 따로 위치함에 유의하길 바란다. 성능 면에서 많은 차이가 있지만 모바일용 GPU 역시 프로그램 가능한 형태를 갖추어가고 있으며 휴대용 게임 콘솔과 게임폰을 중심으로 PC용 GPU에서 축적된 하드웨어 기술이 적용되고 있다.

일부 휴대용 게임콘솔은 상당히 비싼 하드웨어 가격을 감수하여 과거에 비하면 놀랄 만큼 뛰어난 그래픽 성능을 보이기도 하지만 이러한 발전이 휴대



(그림 8) High-end GPU 성능의 발전과 모바일 GPU 성능의 발전 추이

전화, PDA, PMP와 같은 나머지 모바일 기기에 빠른 속도로 번질 것인지는 아직 예측하기 어렵다. 모바일용 GPU가 시장에서 성공하기 위해서는 전력소모의 최소화가 가장 중요하다. PC용 GPU에서는 하드웨어를 병렬로 배치하여 가능한 많은 연산이 동시에 이루어지도록 하여 높은 성능을 이뤄낼 수 있었다. 하지만 모바일용 GPU는 하드웨어 면적이 제한되어 있고 배터리로부터 전원을 공급받기 때문에 하드웨어의 단순 확장보다는 단순하면서도 효율적인 구조를 도입하고 외부메모리 접근을 최소화하는 등의 노력이 필요하다. 모바일용 GPU에서 embedded DRAM을 접근 빈도가 높은 프레임 버퍼 등의 용도로 칩 내에 내장하는 것도 같은 이유에서 비롯되고 있다.

VI. 결론

본 논문에서는 3D 그래픽스 관련 기술 중에서 가속 하드웨어 기술, 즉 GPU 기술의 전반에 대해 살펴보았다. GPU의 발전은 컴퓨터 게임 및 엔터테인먼트 산업이 발전할 수 있었던 발판이 되었으며 GPU 응용제품은 이제 데스크톱 PC 뿐만 아니라 게임 콘솔, 모바일 기기, 디지털 TV로 확대되고 있다.

3D 그래픽스 렌더링 과정은 응용프로그램으로부터 전달되는 버텍스를 처리하는 것으로 시작하여 여러 복잡한 연산을 거치는데, 과거의 GPU는 그 중 일부분 또는 전체를 고정된 기능의 파이프라인을 사용하여 가속하였다. 최근의 GPU는 3D 그래픽스에 특화된 프로세서를 내장하여 프로그램 가능한 구조를 갖는다. 통합 셰이더는 더욱 발전된 형태로 기존의 방식에서 경우에 따라 연산이 GPU의 어느 한 부분에만 집중되는 단점을 개선하고 새로운 개념의 셰이더를 추가할 수 있도록 해준다. GPU의 발전된 프로그램능력과 연산능력을 그래픽 이외의 다른 분야에도 이용하고자 하는 GPGPU에 대한 연구도 활발히 진행되고 있다.

OpenGL은 데스크톱 컴퓨터에서 GPU를 이용하여 3D 그래픽스 응용프로그램을 제작하기 위한 공

개 표준 API이다. OpenGL은 컴퓨팅 플랫폼과 운영체제로부터 독립적이며 로열티가 없다. OpenGL 2.x는 기존의 고정된 파이프라인에서 벗어나 프로그램 가능하도록 버텍스 프로세서와 프래그먼트 프로세서를 포함하며 이들이 수행할 프로그램의 작성에는 GLSL이 사용된다. OpenGL ES는 내장형 시스템을 위해 OpenGL에서 필수적인 부분만을 따로 모아 놓은 프로파일이다. 한편 마이크로소프트사의 DirectX Direct3D 역시 같은 목적을 갖는 API로 Windows 운영체제에서 사용된다.

휴대전화 등 모바일 기기에 사용되는 GPU에는 데스크톱 컴퓨터용 GPU에서 사용되었던 기술들이 빠르게 접목되고 있다. 그러나, 아직까지 크게 대중화되진 못하고 있는 실정이다. 모바일 기기는 배터리로부터 전원을 공급받기 때문에 GPU의 성능만을 고려하기 보다 전력소모의 최소화를 위해 절충해야 할 필요가 있다.

약 어 정 리

AGP	Accelerated Graphics Port
ARB	Architecture Review Board
CAGR	Compound Annual Growth Rate
GLSL	OpenGL Shading Language
GPGPU	General Purpose GPU
GPU	Graphics Processing Unit
HLSL	High Level Shading Language
PCI	Peripheral Component Interconnect

참 고 문 헌

[1] 이민영 외, "3D 그래픽칩 관련 기술 및 산업동향, 기업소개," *IT SoC Magazine*, 통권 12호, 2006. 5.

[2] Tomas Akenine-Möller and Eric Haines, "Real-Time Rendering," A K Perters Publishing Company, Second edition, 2002.

[3] Randy Fernando, Mark Harris, Matthias Wloka, and Cyril Zeller, "Programming Graphics Hardware," *Tutorial on EUROGRAPHICS 2004*, available URL: http://http.download.nvidia.com/developer/presentations/2004/Eurographics/EG_04_TutorialNotes.pdf

[4] W.R. Mark, R.S. Glanville, K. Akeley, and M.J. Kilgard, "Cg: A System for Programming Graphics Hardware in a C-like Language," *ACM Transactions on Graphics*, Vol.22, No.3, July 2003, pp.896-907.

[5] Microsoft HLSL, http://msdn2.microsoft.com/en-us/library/bb509638.aspx

[6] The OpenGL Shading Language, http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf

[7] Victor Moya, Carlos Gonzalez, Jordi Roca, Agustin Fernandez, and Roger Espasa, "Shader Performance Analysis on a Modern GPU Architecture," *Microarchitecture, 2005. MICRO-38. Proc. 38th Annual IEEE/ACM Int'l Symp. On*, 12-16 Nov. 2005.

[8] Chang-Hyo Yu, Ky Usik Chung, Donhyun Kim, and Lee-Sup Kim, "A 120Mvertices/s Multi-threaded VLIW Vertex Processor for Mobile Multimedia Applications," *Solid-State Circuits, 2006 IEEE Int'l Conf. Digest of Technical Papers*, Feb. 6-9, 2006, pp. 1606-1615.

[9] WIKIPEDIA, "GeForce 7 Series," available URL: http://en.wikipedia.org/wiki/GeForce_7_Series

[10] WIKIPEDIA, "GeForce 8 Series," available URL: http://en.wikipedia.org/wiki/GeForce_8_Series

[11] "Details of ATI's Xbox 360 GPU Unveiled," http://www.techreport.com/etc/2005q2/xbox360-gpu/index.x?pg=1

[12] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *In Eurographics 2005*, State of the Art Reports, Aug. 2005, pp.21-51.

[13] OpenGL specification is available at: http://www.opengl.org/

[14] OpenGL ES specification is available at: http://www.khronos.org/

[15] DirectX: http://msdn2.microsoft.com/en-us/library/bb219740.aspx