DOI: 10.3745/KIPSTA.2009.16-A.3.159

OpenGL을 이용한 OpenGL ES 1.1 구현

이 환 용[†] · 백 낙 훈[††]

요 약

본 논문에서는 데스크탑과 같이 OpenGL 기능이 전용 하드웨어로 제공되는 환경을 대상으로, OpenGL ES 1.1 표준을 효율적으로 구현하는 방법을 제시한다. OpenGL ES는 기존의 OpenGL 표준을 바탕으로 하였지만, 고정소수점 연산, 이를 수용하는 버퍼 기능, 완전히 새로운 텍스처 처리 등의 도입으로, 임베디드 시스템에 특화된 3차원 그래픽스 라이브러리로 새롭게 제정되어, 구글 안드로이드, 애플 아이폰, 플레이스테이션3 등에서 공식 3차원 그래픽스 API로 채택되었다. 본 논문에서는 OpenGL ES의 특징적 자료형인 고정소수점 표현에 대한 산술 연산들을 개선하였고, 특히 고정소수점 자료형들을 부동소수점 형태로 변환하여 하위의 OpenGL API로 넘기는 과정에서 표준을 준수하면서도 효율적인 처리가 가능하도록 하였다. 새로 도입된 고정소수점 자료형을 허용하는 버퍼 기능에 대해서는 변환된 자료들을 별도의 메모리 공간에서 관리하는 방식으로 속도 향상에 중점을 두었으며, 요구 사항이 완전히 달라진 텍스처 처리 부분은 전체 기능을 별도의 소프트웨어로 완전히 새롭게 구현하였다. 최종 구현 결과인 OpenGL ES 라이브러리는 OpenGL ES 1.1 표준에 규정된 총 200여 함수를 제공하며, 표준인증 테스트를 완전히 통과하여 1.1 표준을 완벽히 만족시켰음을 보였다. 수행 속도 면에서는 OpenGL ES에 특화된 응용 프로그램들에 대한 처리 속도 측정에서 기존의 구현 사례들에 비해 최대 33.147배의 속도 향상을 가져왔으며, 동일한 범주의 구현 사례들 중에서 가장 빠른 구현 결과이다.

키워드 : OpenGL ES, OpenGL, 데스크탑구현, 표준인증 테스트

OpenGL ES 1.1 Implementation Using OpenGL

Hwanyong Lee[†] · Nakhoon Baek[††]

ABSTRACT

In this paper, we present an efficient way of implementing OpenGL ES 1.1 standard for the environments with hardware-supported OpenGL API, such as desktop PCs. Although OpenGL ES was started from the existing OpenGL features, it becomes a new three-dimensional graphics library customized for embedded systems through introducing fixed-point arithmetic operations, buffer management with fixed-point data type supports, completely new texture mapping functionalities and others. Currently, it is the official three dimensional graphics library for Google Android, Apple iPhone, PlayStation3, etc. In this paper, we achieved improvements on the arithmetic operations for the fixed-point number representation, which is the most characteristic data type for OpenGL ES. For the conversion of fixed-point data types to the floating-point number representations for the underlying OpenGL, we show the way of efficient conversion processes even with satisfying OpenGL ES standard requirements. We also introduced a simple memory management scheme to mange the converted data for the buffer containing fixed-point numbers. In the case of texture processing, the requirements in both standards are quite different and thus we used completely new software-implementations. Our final implementation result of OpenGL ES library provides all of over than 200 functions in OpenGL ES 1.1 standard and completely passed its conformance test, to show its compliance with the standard. From the efficiency viewpoint, we measured its execution times for several OpenGL ES-specific application programs and achieved at most 33.147 times improvements, to become the fastest one among the OpenGL ES implementations in the same category.

Keywords : OpenGL ES, OpenGL, Desktop Implementation, Conformance Test

1. 서 론

OpenGL ES는 PC와 워크스테이션 환경에서 널리 쓰이고 있는 OpenGL을 임베디드 시스템 용으로 특화시킨 3차원 그래픽스 라이브러리이다. OpenGL ES는 2003년 크로노스 그룹(Khronos Group)에서 표준 명세(standard specification)[1]를 발표한 이후, 사용이 증가하여 최근에는 임베디드 시스템에서의 표준 3차원 그래픽스 라이브러리가 되어가고 있다[2,3]. OpenGL ES는 심비안(Symbian) OS와 구글 안드로이드(Android) 플랫폼의 공식 3차원 그래픽스 API이며[4,5],

※ 본 연구는 지식경제부 지방기술혁신사업(RTI04-03-02) 지원으로 수행되었음.
† 정 회 원 : 경북대학교 전자전기컴퓨터학부 박사과정, ㈜휴원 기술이사
†† 종신회원 : 경북대학교 전자전기컴퓨터학부 부교수(교신저자)
논문접수 : 2008년 12월 29일
수 정 일 : 1차 2009년 4월 29일
심사완료 : 2009년 4월 29일

애플 아이폰(iPhone) SDK, 판도라 콘솔(Pandora console) 등에서도 채택하고 있다[6,7]. 또한, PlayStation 3에서는 Cg와 함께 공식 그래픽스 API로 채택되어, 3차원 게임 시장에서도 산업계 표준으로 부각되고 있다[8].

OpenGL ES는 OpenGL의 기능을 임베디드 시스템에 적용하자는 취지에서 제정된 그래픽스 표준이지만, OpenGL의 단순한 다운그레이드(downgrade) 판이나, 부분 집합으로 볼 수는 없다. 임베디드 시스템의 하드웨어 특성을 반영하는 과정에서, 원래의 OpenGL 표준에서 일부 기능이 제거된 대신, 임베디드 시스템을 위한 새로운 기능이 추가되었으므로, OpenGL ES는 임베디드 시스템을 위한 개선판으로 보아야 할 것이다[9,10].

OpenGL ES의 기능을 제공하는 다양한 형태의 제품들이 나와 있다. 그 중 일부는 하드웨어 형태를 가지며, 비교적 고가의 휴대폰 및 임베디드 시스템들을 중심으로 채택되고 있다. 반면, 중저가 단말기 시장에서는 소프트웨어로 구현된 OpenGL ES이 제공되고 있으며 AlexGL[11], Vincent 3D[12] 등의 제품들이 출시되어 있다.

이러한 임베디드 시스템 상에서의 하드웨어 및 소프트웨어 구현과는 달리, 이미 존재하는 그래픽스 라이브러리 위에 별도의 레이어 형태로 OpenGL ES의 기능을 제공하는 경우도 있다. 이들 거의 모두가 기존의 OpenGL을 하위 레이어로 사용하기 때문에, 통상 ES-on-GL 이라는 명칭으로 불리우기도 한다.

이러한 형태의 구현이 요구되고, 또 활발히 사용되는 이유는 크게 다음의 4가지로 볼 수 있다. 우선, OpenGL이 이미 제공되는 시스템이라면, OpenGL ES를 완전히 새로 구현하는 것보다는 별도의 레이어 형태로 구현하는 쪽이 훨씬 경제적이고, 특히 OpenGL이 하드웨어로 지원되는 경우에는 월등히 우수한 성능을 보인다. 예를 들어, Mozilla 웹 브라우저에서는 OpenGL ES 모듈의 구현 시에, 데스크탑 환경에서의 사용을 염두에 두고, 하위 레이어로 OpenGL을 사용하도록 하고 있다[13].

다음으로, OpenGL ES가 시급하게 필요한 경우에는 OpenGL이 이미 제공되는 시스템을 이용하여, 단기간에 OpenGL ES의 기능을 제공할 수 있다. 최초의 OpenGL ES 참조 구현(reference implementation)이 바로 이런 방식으로 구현되어, 1.0 표준의 발표와 거의 동시에 실제로 동작하는 OpenGL ES 라이브러리가 리눅스 환경에서 제공되었다[14].

현재, OpenGL ES 응용 프로그램을 개발하기 위한 효율적인 개발 환경이 필요하나 일반적으로 임베디드 시스템에서의 개발은 데스크탑 환경에서의 개발에 비해 매우 불편하므로, 데스크탑에서 개발된 OpenGL ES 응용 프로그램을 임베디드 시스템으로 포팅하여 테스트하는 방식으로 주로 개발하고 있다. 따라서 하드웨어 가속이 되는OpenGL ES 데스크 탑 개발 환경은 응용 프로그램 개발에 필수적인 요소가 되고 있다[2, 3].

OpenGL ES 전체를 소프트웨어 형태로 구현하고자 하는 경우, 전체 구현의 초기 단계에서 OpenGL을 하위의 래스터라이저(rasterizer)로 사용하여, 초기 작업을 수행할 수 있다. 이는 저자들이 별도로 구현에 성공한 AlexGL[11]에서 채택한 방식이기도 하다.

본 논문에서는 데스크탑 환경에서와 같이 OpenGL이 전용 하드웨어로 제공되는 상황에서 OpenGL ES 표준을 가장 효과적으로 구현하는 것을 목표로 하여, 전체 구현의 구조를 제안하고, 이를 구현하는 과정에서 발생하는 기술적 문제들과 그 해결 방법들을 제시한다. 이제까지 OpenGL 상에서의 OpenGL ES 구현은 몇몇 사례가 있지만, 저자들이 아는 한도 내에서는 이에 관한 연구결과가 발표된 적이 없었다. 따라서, 본 논문은 OpenGL 상에서의 OpenGL ES 구현에 대한 최초의 연구 발표라는 의의도 가진다. 또한, 최종 결과물은 크로노스 그룹의 표준인증 테스트(conformance test)를 통과하여 표준 명세를 100% 만족시켰으며, 기존 구현 사례들과의 성능 평가에서도 뚜렷한 성능 향상을 보였다.

2절에서 OpenGL ES 표준과 이에 관련된 연구 결과들을 소개한다. 3절에서는 우리가 택한 설계와 자세한 구현 방법들을 설명하겠다. 4절에서 구현 결과에 대한 실험결과들을 제시하고, 기존의 구현 결과들과의 비교 평가를 보이겠다. 마지막으로 5절에서 결론과 향후 과제들을 제시한다.

2. 관련 연구

이 절에서는 OpenGL ES 표준과 크로노스 그룹에서 제공하는 표준인증 테스트를 소개한 후, OpenGL 상에서의 OpenGL ES 구현 사례들을 살펴보겠다.

2.1 OpenGL ES 표준

OpenGL ES는 휴대폰, 게임용 콘솔, 차량용 장치 등을 포함하는 다양한 임베디드 시스템에서 2차원 및 3차원 그래픽스 기능들을 완벽하게 제공하기 위한 목적으로 제정된 API 표준이다. OpenGL ES는 근본적으로 임베디드 시스템에서의 그래픽스 성능을 가속하기 위한 저수준(low-level) 인터페이스를 담당한다. 이를 제정한 크로노스 그룹은 삼성전자, LG전자, SKT, IBM, 썬, 노키아, 소니, HP, 모토로라, 애플, AMD, ARM, nVIDIA, TI 등등의, 사실상 OpenGL ES의 공급자와 고객인 단말기 업체, 그리고 3D 응용 및 컨텐츠 개발에 관련된 전세계 주요 업체들이 망라되어 있고, OpenGL ES는 공동 이익을 위해 로열티가 필요 없는 표준으로 발표되었다[15].

OpenGL ES는 이미 산업계 표준으로 쓰이고 있는 OpenGL의 영향을 강하게 받아, OpenGL의 일부 함수들만 모은 부분집합 형태를 그 근거로 하였으나, 임베디드 시스템의 하드웨어 및 소프트웨어 특성을 반영하기 위해 OpenGL과는 다른 특성들도 상당수 포함하고 있다. 예를 들어, 기존의 OpenGL이 고성능의 부동소수점 연산(floating-point operation)을 기반으로 개발되었음에 비해, OpenGL ES는 정수 연산만 가능한 하드웨어를 채택하는 중저가 임베디드 시스템을 위해 고정소수점 연산(fixed-point operation)만을 사용하여 3차원

16bit integer part, 16bit fractional part

$$value = (integer.fraction)_2 \times 2^{-16}$$

(a) 고정소수점 표현 방식: OpenGL ES용의 16.16 형식

a sign bit, 8bit exponent, 23bit fraction

$$value = (-1)^{sign} \times (1.fraction)_2 \times 2^{(exponent-bias)_2}$$

(b) 부동소수점 표현 방식: IEEE-754

(그림 1) 부동소수점 및 고정소수점 표현 방식

그래픽스 기능을 제공할 수 있다. OpenGL ES에서는 이에 따라, 부동소수점과 고정소수점 연산 모두를 사용할 수 있는 Common Profile과 고정소수점 연산만을 사용하는 Common-Lite Profile의 2가지 세부 표준을 제공한다[16].

OpenGL ES에서 사용하는 고정소수점 표현은 32비트 정수 자료형을 변형한 방식으로, (그림 1)에서 보는 바와 같이, 전체 32비트 중에서, 상위 16비트를 정수부분으로 할당하고, 하위 16비트를 실수부분으로 사용하는 방식이다[17]. 이 방식은 IEEE 754[18]와 같은 부동소수점 표현에 비해서 표현할 수 있는 숫자의 범위가 좁다는 단점을 가지지만, 소수점의 위치가 고정됨으로써, 정수 연산만 가능한 하드웨어에서도 빠른 처리가 가능하다는 장점을 가진다.

OpenGL ES 1.0 표준[1]은 OpenGL 1.3 표준을 기반으로, 2003년 7월에 발표되었다. 이 표준은 임베디드 시스템에 적용된 최초의 3차원 그래픽스 표준이라는 점에서 큰 의의를 가지지만, 보완이 필요하여, 2004년 8월에 OpenGL 1.5 표준을 기반으로 하는 OpenGL ES 1.1 표준[16]이 발표되었다. 또한, 기존의 그래픽스 하드웨어들에 프로그래머블 파이프라인(programmable pipeline)을 도입한 OpenGL 2.0의 발표에 맞추어, 2005년 8월에는 OpenGL ES 2.0이 제정되었다[19].

현재 OpenGL ES의 사용은 크게 OpenGL ES 1.1과 OpenGL 2.0이 공존하고 있지만, 임베디드 시스템에서는 향후 수년간은 OpenGL ES 1.1의 사용이 월등히 많을 것으로 예상하고 있다[3]. 이는 2.0 표준에서 프로그래머블 파이프라인을 채택하면서, 고정소수점을 위한 Common-Lite Profile을 포기해야 했고, 이에 따라, 2.0 표준은 부동소수점 계산을 위한 별도의 하드웨어를 가진, 비교적 고가의 임베디드 시스템에서만 사용 가능하다는 제약을 가지기 때문이다. 또한 OpenGL ES 1.1과 하향 호환성이 존재하지 않아 기존에 개발된 많은 수의 OpenGL ES 1.1 응용 프로그램들을 수행할 수 없다는 단점이 있다. 본 논문에서는 현재 시점에서 가장 적절하다고 판단되는 OpenGL ES 1.1 표준을 대상으로 한다.

2.2 OpenGL ES 표준인증 테스트

OpenGL ES의 표준들을 제정하고, 발표하는 크로노스 그룹에서는 OpenGL ES의 표준마다 대응되는 표준인증 테스트(conformance test)를 제공한다. 이 테스트는 70여종의 OpenGL ES 응용 프로그램들로 구성되어 있고, 테스트를 수행할 OpenGL ES 라이브러리를 사용하도록 컴파일하여 그 실행결과를 검사한다. 각 프로그램의 출력은 OpenGL ES 표준에 준하여 구현된 경우의 출력과 대조하여, 사용된 OpenGL ES 라이브러리가 표준을 위배한 경우가 있는지를 검사한다. 표준 인증은 OpenGL ES의 가장 기본적인 기능 검증 역할을 하며, 검증 인증이 되지 않은 제품에는 OpenGL ES라는 명칭이나 로고의 사용이 법적으로 금지되어 있다[20].

2.3 OpenGL 상에서의 OpenGL ES 구현

OpenGL ES 1.0 표준의 발표 이후에, 크로노스 그룹에서는 OpenGL ES가 구현 가능하다는 것을 증명하고, 다른 구현 사례들이 참고하기 위한 목적으로, 참조 구현(reference implementation, RI)을 제공하였다. 이 참조 구현은 기능이 작동한다는 것을 보여주는 것이 주목적이었기 때문에, 리눅스 운영체제에서 기존의 OpenGL 1.3 구현을 이용하여 OpenGL 상에서의 OpenGL ES 구현 방식으로 구현되었다[14].

이후, OpenGL을 사용하지 않는 완전한 소프트웨어 구현으로 AlexGL[11], Vincent 3D[12] 등의 구현 사례가 있었고, 최근에는 nVIDIA의 GoForce[21], ARM의 Mali[22]와 같은, 하드웨어로 구현된 OpenGL ES 칩들이 나와 있다. 이러한 하드웨어 구현들은 현재로서는 고가 시장에서만 가능한 선택이고, 아직까지는 소프트웨어 구현들이 더 많이 사용되고 있다.

본 논문에서 다루고자 하는, OpenGL을 하위 래스터라이저(rasterizer)로 사용하여 OpenGL ES를 구현하는 방식도 몇 가지 사례가 시도되었다. 이미 언급한 OpenGL ES 1.0 참조 구현은 1.0 표준에 대한 것이고, 성능은 별로 고려하지 않았기 때문에, 현재의 1.1 표준 기준으로는 효용성이 많이 떨어진다.

이후, Hybrid사에서 OpenGL ES의 상업적 구현을 제공하면서, 데스크 탑 PC들에서 사용할 수 있도록, 윈도우 운영체제용 OpenGL에 기초한 OpenGL ES 1.1 구현을 제공하였다. 이 구현은 Gerbera 라는 이름으로, PC에서의 OpenGL ES 1.1 응용 프로그램 개발을 위해 사용되었다[23]. 다만, 이 구현은 표준인증 테스트를 통과하지 못했고, Hybrid사가 nVIDIA에 통합되면서, 현재는 계속적인 지원 여부가 불투명하다.

Imagination Technologies사의 PowerVR SDK에는 임베디드 시스템에 대한 윈도우 운영체제용 에뮬레이터가 들어있고, 이 에뮬레이터의 일부로 윈도우용 OpenGL 상에서 구현된 OpenGL ES 1.1 라이브러리가 들어있다[24]. 제작사에서는 이 구현의 표준인증 테스트 통과 여부에 대해서 언급하지 않고 있지만, 비정상적인 출력 사례가 알려져 있다.

이외에도 dgles[25]와 같은 공개 OpenGL ES 구현들이 있었지만, 현재는 모두 지원이 중단되어 라이브러리 자체를 구하는 것도 여의치 않은 상황이 되었다. 따라서, 본 논문에서

는 새로 개발한 OpenGL 상에서의 OpenGL ES 구현에 대한 비교 대상으로, Hybrid사의 Gerbera와 Imagination Technologies 사의 PowerVR 구현을 사용하였다. 이들은 모두 윈도우 운영체제에서 테스트가 가능하고, 자주 쓰이면서 쉽게 구할 수 있다는 면에서 비교의 대상이 되었다.

좀더 특이한 구현 사례들로는 Java2 ME 상에서 OpenGL ES 1.1과 유사한 형태의 라이브러리를 제공한 경우[26]와 OpenGL ES 2.0 파이프라인 상에서 OpenGL 1.1 기능을 제공한 경우[27]도 있었다. 이들은 OpenGL ES 1.1에 대한 요구가 증가하면서, 다양한 환경에서 이를 제공하려는 노력들로 볼 수 있을 것이다.

3. 설계 및 구현

이 절에서는 OpenGL에 기초한 OpenGL ES 구현에서 고려해야 하는 사항들을 살펴보고, 실제 우리가 구현 과정에서 해결하여야 했던 문제점들을 제시하겠다. 이는 사실상 모든 OpenGL에 기초한 OpenGL ES 구현들이 직면하게 될 문제점들이지만, 기존 구현들에서는 완전한 해결에 실패하여 표준인증 테스트를 통과하지 못한 경우도 있었다.

3.1 전체 구조와 라이브러리 레벨의 문제 해결

일반적인 OpenGL ES의 구현은 (그림 2)(a)에서 보는 바와 같이, 하위의 그래픽스 하드웨어와 상위의 응용 프로그램 사이에서 3차원 그래픽스 기능을 제공하는 역할을 담당한다. OpenGL 상에서의 OpenGL ES 구현에서는 (그림 2)(b)에서 보는 바와 같이, OpenGL ES의 내부에서 하위 래스터라이저(rasterizer)로 OpenGL을 사용하는 방식을 택하였다. 그래픽스 하드웨어 위에 OpenGL 라이브러리를 추가하여, 3차원 그래픽스 기능을 제공하고, OpenGL ES에서는 이를 가상 하드웨어 장치로 사용하는 방식이다.

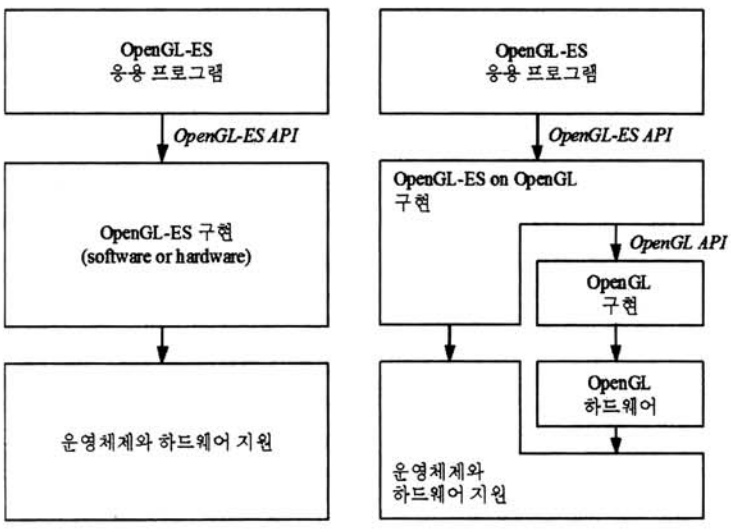
OpenGL을 하위 래스터라이저로 사용하는 경우에는 OpenGL ES가 필요로 하는 많은 기능들이 하위에서 이미 제공되므로, 구현에서 많은 이점을 얻을 수 있다. 반면에, 이러한 방식을 채택한다고 해서, 모든 구현이 그대로 쉽게 처리되는 것은 아니고, OpenGL ES와 OpenGL 표준의 차이에서 오는 문제점들을 해결해야 한다.

우선, 하위에 사용되는 OpenGL에 대해서 적합한 버전이 제공되도록 하여야 한다. OpenGL ES 1.0 표준과 1.1 표준은 각각 OpenGL 1.3과 1.5 표준을 기반으로 하므로, 대응되는 OpenGL 버전이 확보되어야 한다. 리눅스 계열 운영체제에서는 X 윈도우 서버에 이미 최신 OpenGL이 탑재된 경우가 많으므로, 별다른 문제가 되지 않지만, 윈도우 운영체제에서는 그래픽 카드 제작사에서 제공하는 디바이스 드라이버를 설치해야 하고, 이를 구동하기 위한 별도의 라이브러리를 사용해야 한다. 우리는 최신의 디바이스 드라이버를 설치하고, GLee 라이브러리[28]를 사용해서 이 문제들을 해결하였다.

3.2 고정소수점 연산의 추가

OpenGL ES에서는 고정소수점 자료형과 그에 따른 연산들을 제공하여야 한다. 이에 따라서, 우리는 최우선으로 OpenGL ES 1.1 표준을 준수하는 고정소수점 자료형을 제공하였고, 내부적으로 필요한 연산들을 작성하여 내장하였다.

OpenGL ES 1.1 표준에서는 고정소수점 자료형들 사이의 연산에 대해서는 별도로 언급하고 있지 않지만, 실제 구현에 있어서는 내부 처리를 위해 더하기, 빼기, 곱하기, 나누기 등의 기본적인 산술 연산들은 물론이고, 삼각함수, 로그, 지수 함수 등의 수학 함수들이 제공되어야 한다. 우리는 매우



(a) 일반적인 경우 (b) OpenGL 상에서의 구현

(그림 2) OpenGL ES의 구현 (a) 일반적인 경우 (b) OpenGL 상에서의 구현

효율적으로 구현된 고정소수점 자료형을 위한 소프트웨어 라이브러리를 내장하는 방식으로 전체적인 수행 속도를 높이도록 하였으며, 이는 최종적인 실험 결과에서 기존의 구현 사례들에 비해 상당한 속도 향상을 가져오는데 큰 공헌을 하였다.

3.3 자료형에 따른 매개변수 처리

OpenGL 상에서의 OpenGL ES의 구현은 내부적으로 OpenGL ES의 각 함수에 주어진 매개변수를 하위의 OpenGL API에 어떻게 전달할 것인가의 문제를 가져온다. 특히, 고정소수점 자료형이 추가됨으로써, 많은 함수들이 적절한 변환 과정을 거쳐야만 처리가 가능하다. 이러한 처리 방법은 함수의 유형에 따라 서로 다른 전략을 택하게 되는데, 크게 보아서는 다음의 3가지 유형으로 나눌 수 있다.

[유형 1] 그대로 통과시켜도 되는 경우

일부 함수들은 OpenGL ES 1.1 표준이 OpenGL 1.5 표준을 정확히 그대로 따라간 경우가 있다. 어긋나는 부분이 전혀 없이, 정확히 일치하는 이러 유형의 함수들에 대해서는 별도의 처리 없이, 모든 매개변수를 그대로 통과시켰다. 대표적인 예로는 다음에 실제 구현에 사용된 의사 코드(pseudo code)가 제시된 glMultMatrixf(…), glLightf(…) 등이 있으나, 전체적인 비율은 그렇게 높지 않다.

```
void glMultMatrixf(const GLfloat* m) {
  /* call to underlying OpenGL */
  func_ptr fptr = function pointer to OpenGL's glMultMatrixf(...);
  call (*fptr)(m);
}

void glLightf(GLenum light, GLenum pname, GLfloat param) {
  /* call to underlying OpenGL */
  func_ptr fptr = function pointer to OpenGL's glLightf(...);
  call (*fptr)(light, pname, param);
}
```

[유형 2] 간단한 자료형 변환으로 처리가능한 경우

OpenGL ES 표준에서 부동소수점 자료형을 사용하는 거의 모든 함수들은 대응되는 고정소수점 자료형 방식의 함수들을 Common-Lite Profile에서 제공하고 있다. 이들은 주어진 고정소수점 자료형을 부동소수점 자료형으로 변환한 후, OpenGL에서 제공하는 함수를 호출하는 방식을 택하였다. 이러한 예로는 다음에 제시된 glMutlMatrixx(…), glLightx(…) 등이 있으며, 전체적으로는 상당히 많은 함수들이 이 유형에 속한다. 내부 구현에서 사용된 fixed_to_float(…) 함수는 부동소수점으로 표현된 값들을 고정소수점 자료형으로 변환하는 기능을 담당하고, 수행 속도를 높이기 위해 매크로 함수로 구현되었다.

```
void glMultMatrixx(const GLfixed* m) {
  /* type conversion */
  GLfloat fm[16]
  fm[0..15] = fixed_to_float(m[0..15])
  /* call to underlying OpenGL */
  func_ptr fptr = function pointer to OpenGL's glMultMatrixf(...);
  call (*fptr)(fm);
}

void glLightx(GLenum light, GLenum pname, GLfloat param) {
  /* type conversion */
  GLfloat fparam = fixed_to_float(param);
  /* call to underlying OpenGL */
  func_ptr fptr = function pointer to OpenGL's glLightf(...);
  call (*fptr)(light, pname, fparam);
}
```

일부 함수들의 경우, OpenGL ES에서는 부동소수점 자료형과 고정소수점 자료형을 모두 지원하지만, OpenGL에서는 배정도 부동소수점 자료형만을 제공한다. 이 경우에는 부동소수점 타입의 함수와 고정소수점 타입의 함수들 양쪽 모두에서 자료 변환이 필요하다. 대표적인 예로는 OpenGL의 glOrtho(…) 함수에 대응되는, 다음의 glOrthof(…), glOrthox(…) 함수들을 들 수 있다. 내부 구현에 사용된 float_to_double(…)와 fixed_to_double(…) 매크로 함수들은 각각 부동소수점과 고정소수점 자료형에서 배정도 부동소수점 자료형으로의 변환을 담당한다.

```
void glOrthof(GLfloat left, GLfloat right, GLfloat bottom,
              GLfloat top, GLfloat near, GLfloat far) {
  /* type conversion */
  GLdouble dleft, dright, dbottom, dtop, dnear, dfar
       = float_to_double(left, right, bottom, top, near, far);
  /* call to underlying OpenGL */
  func_ptr fptr = function pointer to OpenGL's glOrtho(...);
  call (*fptr)(dleft, dright, dbottom, dtop, dnear, dfar);
}

void glOrthox(GLfixed left, GLfixed right, GLfixed bottom,
              GLfixed top, GLfixed near, GLfixed far) {
  /* type conversion */
  GLdouble dleft, dright, dbottom, dtop, dnear, dfar
       = fixed_to_double(left, right, bottom, top, near, far);
  /* call to underlying OpenGL */
  func_ptr fptr = function pointer to OpenGL's glOrtho(...);
  call (*fptr)(dleft, dright, dbottom, dtop, dnear, dfar);
}
```

[유형 3] 각 매개변수들을 세밀하게 검사해야 하는 경우.

OpenGL ES가 OpenGL의 단순한 다운그레이드 버전이나 부분 집합이 아니라는 사실은 많은 함수들의 매개변수에서 두 표준이 서로 다른 처리를 요구하고 있다는 것에서 잘 드러난다. 예를 들어, glMaterial(…) 함수는 두 표준 모두에서

void glMaterialf(GLenum *face*, **GLenum** *pname*, **GLfloat** *param*);

형태로 정의되어 있지만 매개변수의 처리는 서로 판이하다. 즉, OpenGL ES 표준에서는 매개변수 face의 값으로 GL_FRONT_AND_BACK만을 허용하지만, OpenGL에서는 GL_FRONT나 GL_BACK의 값에 대해서도 대응되는 처리를 수행한다. 따라서, OpenGL ES 함수에서 face의 값을 제대로 검사하지 않으면, OpenGL ES 표준에서는 error로 정의된 상황에서 아무런 error가 발생하지 않고, 엉뚱한 부작용(side-effect)가 발생할 수 있다. 우리의 구현에서는 이런 경우를 모두 찾아내어, 다음과 같이 처리하도록 하였다.

```
void glMaterialf(GLenum face, GLenum pname, GLfloat param) {
  switch (face) {
  case GL_FRONT_AND_BACK:
       /* call to underlying OpenGL */
       func_ptr fptr
            = function pointer to OpenGL's glMaterialf(...);
       call (*fptr)(face, pname, param);
       break
  case GL_FRONT, GL_BACK:
  default:
       /* error valuesin OpenGL ES */
       invoke error handling routine;
       break
  }
}
```

glMaterialxv(…) 함수에서는 위의 glMaterialf(…) 함수의 처리에 추가해서, pname에 주어진 값들에 따라, 뒤의 param이 가리키는 배열에서 가져와야 할 매개변수 값들의 개수가 달라진다. 또, 이 값들이 모두 고정소수점 자료형에서 부동소수점 자료형으로 하나씩 변환되어야 하므로, 이를 처리하는 과정은 각 경우에 따라 처리되어야 한다.

우리의 구현에서는 OpenGL ES 1.1 에서 제공해야 하는 모든 함수들을 표준 명세서에 따라 엄격하게 해석하여 가능한 모든 경우에 대하여 적절한 자료형 변환이나, 에러 처리를 수행하도록 하였다.

3.4 완전히 새로운 기능의 추가

OpenGL ES 1.1 표준에서 제시한 많은 함수가 위의 3가지 유형들 중의 하나에 포함되어, 비교적 정형화된 처리가 가능했지만, 일부 함수는 OpenGL 1.5 표준과는 함수 이름을 제외하고는 완전히 다른 기능을 요구하기도 한다.

우선, OpenGL 1.5에서 제공하는 버퍼(buffer) 기능을 OpenGL ES 1.1에서도 제공하여야 하는데, 고정소수점 자료형의 도입에 따라, OpenGL을 하위 래스터라이저로 사용하더라도 직접적인 구현은 불가능한 상황이 발생한다. 버퍼 기능은 glDrawArrays(…) 함수나 glDrawElements(…) 함수에서 실제 화면 출력을 하기 전에, 이들에 필요한 정점(vertex), 색상(color), 법선 벡터(normal vector), 텍스처 좌표(texture coordinate)들을 저장한 배열을 별도의 메모리 영역에 저장해 두는 역할을 수행한다. 근본적인 문제는 OpenGL은 부동소수점이나 정수형으로 저장된 좌표나 색상값들을 해석할 수 있지만, OpenGL ES에서 추가로 제공하는 고정소수점 자료형에 대해서는 처리가 불가능하다는 것이다.

비교적 쉬운 해결책은 glDrawArrays(…) 나 glDrawElements(…) 와 같은 출력 함수들을 수행하는 시점에, 별도의 메모리에 저장되어 있던 좌표나 색상값들을 OpenGL에 직접 넘겨주는 대신, 고정소수점 자료들을 하나하나 자료형 변환 과정을 거치도록 하는 것이다. 실제로 최초의 OpenGL ES 1.0 참조 구현에서는 이 방식을 택하였으나, 처리 속도의 심각한 저하를 피할 수 없다. 버퍼 기능이 도입된 이유가 좀 더 빠른 처리를 위한 것이었음을 감안하면, 좀더 효율적인 처리 방법이 필요하다. 우리의 구현에서는 이러한 자료형 변환 작업을 glDrawArrays(…)나 glDrawElements(…) 함수가 호출될 때마다 매번 수행하는 대신, 한번 변환된 자료들은 별도의 메모리에 저장하여 관리하는 방식을 택하여 속도 향상을 이루었다.

텍스처 처리에 있어서는 OpenGL과 OpenGL ES 사이에 많은 차이가 존재한다. 예를 들어, glCompressedTexImage2D(…)와 glCompressedTexSubImage2D(…)는 함수 이름과 매개변수의 형식만 일치할 뿐, 완전히 새로운 구현이 필요하다. 따라서, 둘의 내부는 완전히 다르며, OpenGL쪽의 처리를 그대로 사용할 수는 없어, 텍스처 처리에 있어서는 많은 부분에서 완전히 새로운 방식의 구현이 필요하였다. 우리의 경우에는 임베디드 시스템에서의 사용을 위해 OpenGL ES 전체를 소프트웨어로 구현한 AlexGL[11]의 코드들을 이식하여 이 문제를 해결하였다.

4. 구현 결과

본 논문에서는 3절에서 설명한 기법들을 사용하여, 최종적으로 OpenGL ES 표준인증 테스트를 완전히 통과하는 OpenGL ES 1.1을 OpenGL 상에서 구현하였다. 이 절에서는 최종 구현 결과를 평가하고, 기존의 구현 사례들과 비교하겠다. 표준을 구현한 결과물에 대한 평가에서는 표준에서 정의한 대로 구현되었는지에 대한 정확성(correctness)과, 구현의 효율성(efficiency)를 따져 볼 수 있다.

OpenGL ES를 제정한 크로노스 그룹에서는 정확성을 평가하는 기준으로 표준인증 테스트를 제공한다. OpenGL ES 표준인증 테스트는 OpenGL ES를 구현한 라이브러리나 하드웨어가 표준 문서를 준수했는지를 평가하는 프로그램으로,

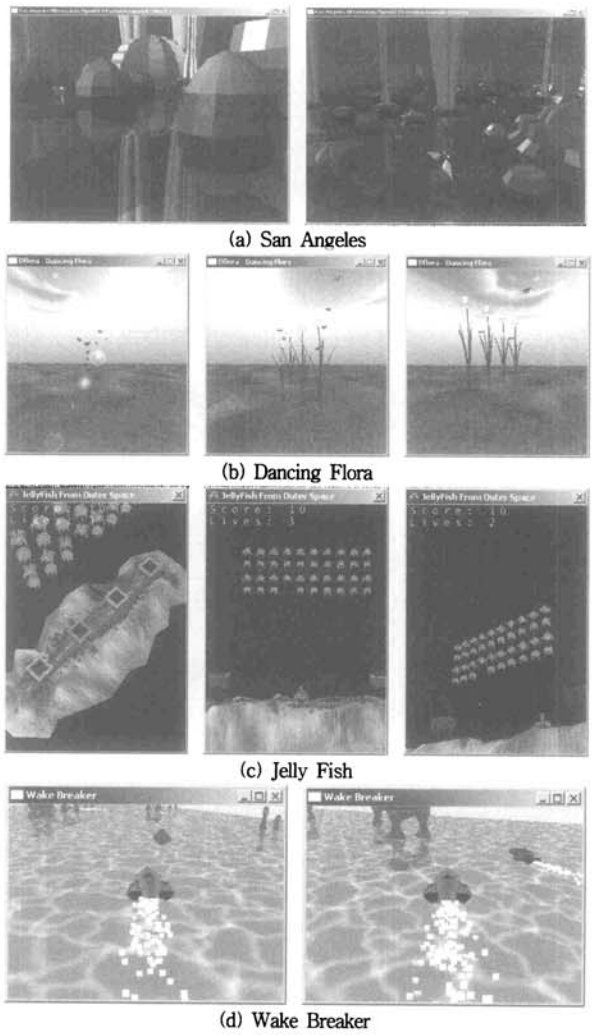
표준 문서를 정확히 지킨 프로그램은 표준인증 테스트에서 실시하는 모든 검사 과정을 완벽히 통과해야 한다.

OpenGL 상에서의 OpenGL ES 구현 결과를 표준인증 테스트로 검사하기 위해서는 한가지 기술적인 문제가 걸리는데, 하위 레이어로 사용된 OpenGL 라이브러리가 OpenGL의 표준을 준수했느냐는 것이다. 이 문제는 이미 OpenGL ES 1.0의 참조 구현을 발표하면서 제기된 문제점이기도 하다[14]. 본 논문에서는 nVIDIA 사의 FX3400 그래픽스 카드에 최신 OpenGL 드라이버와 GLee 라이브러리[28]를 사용하였고, 최소한 이 경우에는 하위의 OpenGL 구현이 문제를 일으키지는 않았으며, 최종적으로 우리의 구현은 OpenGL ES 1.1 표준인증 테스트를 통과하였다.

3절에서 설명한 바와 같이, 우리의 OpenGL ES 구현은 수행 속도의 향상을 위해서도 상당한 노력을 기울였다. 수행 속도의 효율성을 평가하기 위해서는 기존의 OpenGL ES 1.1 라이브러리 중에서 비교적 우수하다는 평가를 받으면서 널리 사용되고 있는 Hybrid사의 Gerbera와 Imagination Technologies사의 PowerVR MBX를 비교 대상으로 사용하였다. 비교의 방법으로는 OpenGL ES의 특성을 잘 활용했다고 알려진 OpenGL ES 프로그램을 대상으로, 각각의 라이브러리를 사용하여 최종 수행 시간을 측정하였다. 기존의 OpenGL 프로그램들을 단순히 OpenGL ES로 포팅한 예제들은 부동소수점 연산만을 사용하거나 하위의 OpenGL이 제공하는 기능만을 사용하므로, 평가용으로는 적합하지 않다. 이런 이유로, 우리는 2005년에 개최된 OpenGL ES 코딩 챌린지(coding challenge)에 입상했던 프로그램들에 주목했다. 이들은 OpenGL ES의 특성을 사용하여 최대한의 효율을 달성하도록 제작되었으므로, OpenGL ES 라이브러리의 비교 평가에 적합하다고 판단하였다. 이들 프로그램 중에서 특정 컴파일러 환경을 필요로 하는 1개 프로그램을 제외하고, 나머지 4개 프로그램, San Angeles, Dancing Flora, Jelly Fish, Wake Breaker를 비교의 대상으로 삼았다.

San Angeles와 Dancing Flora는 각각 미래 도시와 춤추는 꽃들을 주제로 하는 컴퓨터 애니메이션을 1분 50초와 2분 47초에 걸쳐 실시간에 재생하는 프로그램이다. 두 프로그램 모두 고정소수점 자료형, 버퍼 기능, 텍스쳐 등을 충분히 사용하는 좋은 예제였고, 각각의 실행 화면은 (그림 3)의 (a), (b)와 같다. Jelly Fish는 1970년대 후반에 선풍적인 인기를 끌었던 아케이드 게임인 Space Invaders의 3차원 버전이고, Wake Breaker는 간단한 모터보트 레이싱 게임이다. 둘 모두 사용자의 키 입력에 반응하는 대화형 게임으로, 실행 화면은 (그림 3)의 (c), (d)와 같다. 이들 게임 프로그램은 첫 스테이지를 완료하기까지의 시간을 측정하여, 각각 평균 1분 16초와 1분 15초간 수행되었다.

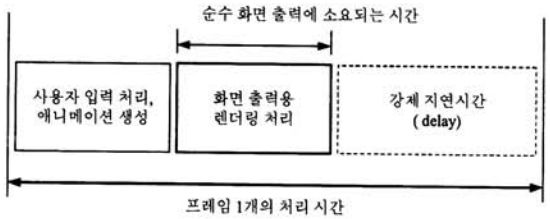
이들 프로그램들은 모두 애니메이션이나 게임의 한 화면씩을 생성하는 작업을 반복하는 루프 구조를 가지는데, 각 화면 사이에는 사용자 입력을 처리하거나, 애니메이션의 시간을 맞추기 위한 강제 지연 시간을 가진다. 수행 시간의 측정 시에는 (그림 4)에서와 같이, 강제 지연 부분을 제외하고, 순수 화면 출력에 소요되는 시간을 측정하였다.



(그림 3) 비교에 사용된 OpenGL ES 코딩 챌린지 입상 프로그램들

<표 1>은 각 프로그램에 대해서, 본 논문에서 구현한 라이브러리, PowerVR MBX라이브러리, Gerbera 라이브러리를 각각 사용하여 측정한 수행시간들이다. 각 프로그램은 Intel Xeon 3.40GHz CPU와 2GB RAM을 가지는 PC에 nVIDIA 사의 FX3400 그래픽스 카드를 장착하여, 라이브러리 별로 총 5회씩이 수행된 결과를 평균했다. 각 프로그램 별로는 평균 프레임 수와 프레임당 소요 시간을 측정하였고, 프레임당 소요 시간에 대해서는 평균, 표준편차, 최소값, 최대값을 제시하였다.

프로그램 별로 렌더링에 필요한 기능이 조금씩 다르고,



(그림 4) 반복되는 프레임 내에서의 시간 측정

〈표 1〉 측정된 수행 시간

(unit : msec)

implementations test programs			본 논문의 구현결과	PowerVR MBX	Hybrid Gerbera
SanAngeles	# frames		6,528	6,196	1,165
	elapsedtime per frame	average	7.472	17.107	92.704
		std. dev.	0.809	5.409	19.069
		minimum	0.001	0.001	0.001
		maximum	47.368	375.623	180.771
DancingFlora	# frames		9,808	7,272	9,013
	elapsedtime per frame	average	0.530	0.853	17.568
		std. dev.	0.372	3.892	7.402
		minimum	0.105	0.170	11.170
		maximum	15.412	95.437	175.201
JellyFish	# frames		3,616	3,705	7,598
	elapsedtime per frame	average	1.096	1.574	7.942
		std. dev.	0.735	3.394	2.163
		minimum	0.158	0.277	2.695
		maximum	35.803	155.640	42.573
WakeBreaker	# frames		1,397	1,888	1,656
	elapsedtime per frame	average	1.710	1.765	13.904
		std. dev.	0.502	4.478	4.671
		minimum	0.563	0.763	4.161
		maximum	15.563	164.802	36.796

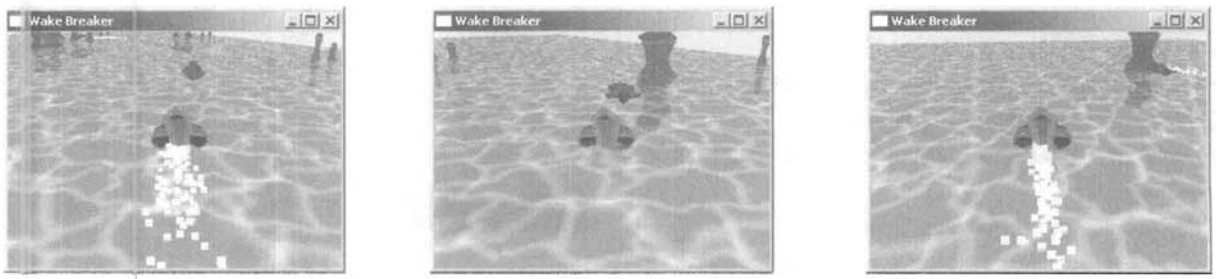
수행 도중에도 화면의 복잡도에 따라, 소요 시간에 차이가 났다. <표 2>에서와 같이, 프레임당 소요 시간의 평균값만을 따로 모아 보면, 비교에 필요한 4개 프로그램 모두에서 본 논문에서 구현한 결과가 가장 우수한 성능을 보였고, 다음으로 PowerVR MBX, 마지막으로 Gerbera의 순서가 되었다. 프로그램 별로 편차가 있지만, 본 논문의 구현 결과는 현재 상업적으로 판매되고 있고, OpenGL 상에서의 OpenGL ES 구현 사례 중에서 가장 우수하다고 알려진 PowerVR MBX에 비해서 최소 1.032배, 최대 2.289배의 속도 향상을 보였다. 또한, 또다른 구현 사례인 Gerbera에 비해서는 최소 7.246배, 최대 33.147배의 속도 향상을 가져왔다.

PowerVR MBX에 대한 가속 성능이 1.032배로 가장 속도 향상이 적었던 Wake Breaker 프로그램에서는 OpenGL에서는 직접 지원되지 않지만, OpenGL ES에서는 반드시 제공되어야 하는 PointSize extension을 사용하여 파도의 포말을 표현하였다. 이 기능은 하위의 OpenGL에서의 처리를 기대할 수 없고, 완전한 소프트웨어 처리가 필요하여, 처리 속도의 저하를 가져오는 주원인이 될 수 있다. 각 라이브러리별 수행 결과는 (그림 5)에서와 같이, 본 논문의 구현 결과나, Gerbera를 사용한 경우에는 파도의 포말이 제대로 표현되었으나, PowerVR MBX에서는 파도의 포말 부분이 생성되지 않는 경우가 빈번했다. 이는 PowerVR MBX에서의 구현 오류일 가능성이 크다고 보이며, 이 경우에 대해서는 다른 라이브러리들과의 단순한 속도 비교가 무의미해진다.

현재 OpenGL 상에서의 OpenGL ES를 구현한 라이브러리로는 본 논문에서 비교의 대상으로 선정한 PowerVR MBX와 Gerbera를 제외하고는 사실상 사용가능한 것들이 없다는 점을 감안하면, 본 논문의 최종 구현 결과는 최소한 이제까지의 실험결과들에서는 이제까지의 구현 사례들에 비해 상당한 속도 향상을 보여주었다.

〈표 2〉 수행시간의 상대적 비교

		본 논문의 구현결과	PowerVR MBX	Hybrid Gerbera
SanAngeles	average time(msec)	7.472	17.107	92.704
	ratio	1.000	2.289	12.407
DancingFlora	average time(msec)	0.530	0.853	17.568
	ratio	1.000	1.609	33.147
JellyFish	average time(msec)	1.096	1.574	7.942
	ratio	1.000	1.436	7.246
WakeBreaker	average time(msec)	1.710	1.765	13.904
	ratio	1.000	1.032	8.131
minimum ratio			1.032	7.246
maximum ratio			2.289	33.147



(a) 본 논문의 구현결과 (b) PowerVR MBX (c) Gerbera

(그림 5) Wake Breaker의 수행 결과 : 파도의 포말 출력에서 차이가 있음

5. 결 론

본 논문에서는 OpenGL ES 1.1 표준의 구현을 목표로 하여, 데스크탑에서와 같이, 전용 하드웨어에서 OpenGL 기능이 이미 제공되는 경우에 OpenGL ES 기능을 효과적으로 제공하는 방법을 제안하였다. 제안하는 방법에 따른 최종 구현결과는 크로노스 그룹에서 설정한 OpenGL ES 표준인증 테스트를 완전히 통과하여 OpenGL ES 1.1 표준을 완전히 만족시켰다. 이제까지 OpenGL ES 표준인증 테스트를 통과한 OpenGL상에서의 OpenGL ES 구현 사례가 없었음을 감안하면, 우리의 구현 결과는 최초로 표준을 준수한 성공 사례이다.

또한, 우리는 설계 단계에서부터 전체적인 효율성의 향상에 초점을 맞추어, 수행 속도를 높이기 위한 다양한 기법들을 도입하였다. 특히, 고정소수점 계산과 매개변수의 자료형 변환 과정에서 계산 속도의 향상을 이루었으며, OpenGL ES에 특화된 응용 프로그램들에 대한 수행 속도 측정에서 기존의 구현 사례들에 비해 최소 1.032배, 최대 33.147배의 성능 향상을 보였다.

종합적으로 보아, 본 논문의 구현 사례는 OpenGL 상에서의 OpenGL ES 구현 사례 중에서 유일하게 표준인증 테스트를 통과하였고, 기존의 구현 사례들에 비해서도 가장 빠른 수행 속도를 보이고 있다. 우리는 이 구현 사례를 바탕으로 표준인증 테스트 결과에 대한 크로노스 그룹의 공인 절차를 수행하고, 시장에서 널리 사용되는 OpenGL ES 벤치마크 테스트인 3DMark를 수행하기 위한 작업을 수행 중이다. 또한, 다양한 제품군으로의 확장도 추진 중이다. 다음 단계의 OpenGL ES 2.0 표준에 대해서는 OpenGL 2.0 표준을 바탕으로 새로운 구현이 요구되며, 우리는 이미 전체적인 구현 설계에 착수했다. OpenGL 이외의 그래픽스 라이브러리가 제공되는 시스템에서의 완전한 소프트웨어 및 하드웨어 구현 역시 필요하며, 궁극적으로는 이를 바탕으로 임베디드 시스템에서 대형 워크스테이션까지 일관된 인터페이스를 제공하고자 한다.

6. 감사의 글

San Angeles, Dancing Flora, Jelly Fish, Wake Breaker 프로그램의 사용에 각각 동의해준 원저자들, Jetro Lauha, Changzhi Li, Jacobo Rodriguez Villar, Peter Angstadt에게 감사드린다.

참 고 문 헌

[1] D. Blythe, OpenGL ES Common/Common-Lite Profile Specification, version 1.0.02(annotated), Khronos Group, 2004.
[2] A. Malizia, Mobile 3D Graphics, Springer, 162 pages, 2006.
[3] K. Pulli, T. Aarnio, V. Miettinen, K. Roimela and J. Vaarala, Mobile 3D Graphics: with OpenGL ES and M3G, 500 pages, Morgan Kaufman, Nov., 2007.
[4] S. Babin, Developing Software for Symbian OS, 2nd Ed, 432 pages, Symbian Press/Wiley, 2007.
[5] http://code.google.com/android/what-is-android.html
[6] http://developer.apple.com/iphone/
[7] http://www.openpandora.org/
[8] http://www.nvidia.com/object/IO_21111.html
[9] D. Astle and D. Durnil, OpenGL ES Game Development, Course Technology PTR, 320 pages, Sep., 2004.
[10] K. Pulli, T. Aarnio, K. Roimela, and J. Vaarala, "Designing graphics programming interfaces for mobile devices," IEEE CG&A, Vol.25, No.6, pp.66-75, 2005.
[11] http://www.hul.com/english/B-03_01.html
[12] http://www.vincent3d.com/
[13] http://wiki.mozilla.org/Canvas:3D/Historical
[14] http://www.khronos.org/developers/resources/
[15] http://www.khronos.org/
[16] A. Munshi and J. Leech, OpenGL ES Common/Common-Lite Profile Specification, version 1.1.12 (Full Specification), Khronos Group, Apr., 2008.
[17] ARM, Fixed Point Arithmetic on the ARM, Application Note 33, ARM, 1996.
[18] D. Hough, "Applications of the proposed IEEE-754 standard for floating point arithmetic," IEEE Computer, Vol.14, No.3, pp.70-74, 1981.
[19] A. Munshi, OpenGL ES Common Profile Specification, version 2.0.23(Full Specification), Khronos Group, Aug.,

2008.
[20] http://www.khronos.org/adopters/
[21] http://www.3d-test.com/interviews/nvidia_4.htm
[22] http://www.arm.com/news/21889.html
[23] http://www.embeddedstar.com/press/content/2004/2/embedded13008. html
[24] http://sourceforge.net/projects/dgles
[25] http://www.imgtec.com/powervr/insider/sdk/KhronosOpenGLES1xMBX.asp
[26] C.-H. Tu and B.-Y. Chen, "The architecture of a J2ME-based OpenGL ES 3D Library", 9th Conf. on Computer Aided Design and Computer Graphics, 5 pages, 2005.
[27] S. Hill, M. Robart and E. Tanguy, "Implementing OpenGL ES 1.1 over OpenGL ES 2.0," Int. Conf. on Consumer Electronics, pp.11-1.5, 2 pages, 2008.
[28] http://elf-stone.com/glee.php

이 환 용

e-mail : hylee@hul.com
1990년 KAIST 전산학과(학사)
1992년 POSTECH 컴퓨터공학과(석사)
1995년 POSTECH 컴퓨터공학과 박사과정 수료
1995년~2000년 POSTECH 정보통신연구소 연구원
현 재 경북대학교 전자전기컴퓨터학부 박사과정, (주)휴원 기술이사
관심분야 : 임베디드소프트웨어, 그래픽스, 가상현실, 그래픽사용자인터페이스

백 낙 훈

e-mail : oceancru@gmail.com
1990년 KAIST 전산학과(학사)
1992년 KAIST 전산학과(석사)
1997년 KAIST 전산학과(박사)
현 재 경북대학교 전자전기컴퓨터학부 부교수
관심분야 : 컴퓨터 그래픽스, 임베디드 소프트웨어