KIPS Tr. Comp. and Comm. Sys.
Vol.3, No.4 pp.125~128 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2014.3.4.125

OpenGL ES 2.0 Emulation on Desktop PCs

Nakhoon Baek†

ABSTRACT

OpenGL ES(OpenGL for Embedded System) 2.0 is one of the most widely used 3D graphics API(application progrma interface) standard for smart phones and tablet PCs at this time. During programming with this API, they prefer desktop environment rather than the target mobile environment, which has relatively low computing power. Thus, we need to emulate the OpenGL ES 2.0 API on the desktop PCs, where only OpenGL API libraries are available. In this paper, we present technical difficulties and their solutions to emulate OpenGL ES 2.0 on desktop PCs. Our final implementation of OpenGL ES 2.0 emulation library works on desktop PCs and passed over more than 96% of the official CTS(conformance test suites) to prove the correctness of our implementation. Additionally, for the commercially available benchmark programs, our implementation shows equivalent execution speeds to the previous commercial OpenGL ES 2.0 implementations.

Keywords : OpenGL ES, Emulation, Mobile Graphics, Embedded System

데스크탑 상에서의 OpenGL ES 2.0 에뮬레이션

백 낙 훈†

요 약

OpenGL ES(OpenGL for Embedded System) 2.0은 현재 스마트 폰과 태블릿 PC에서 가장 널리 사용되고 있는 3차원 그래픽스 API표준이다. 이를 이용하는 개발과정에서는 상대적으로 성능이 떨어지는 모바일 환경보다는 데스크 탑 환경이 선호된다. 따라서, OpenGL 라이브러리만 제공되는 데스크 탑 환경에서, 모바일 그래픽스 환경에서의 OpenGL ES 2.0 API를 그대로 에뮬레이션 할 필요가 있다. 본 논문은 PC 상에서 OpenGL ES 2.0 을 에뮬레이션하기 위해, 기술적 문제점들을 극복하는 방법들과 이에 따른 구현 결과를 제시한다. 구현된 OpenGL ES 2.0 에뮬레이션 라이브러리는 데스크 탑 PC 상에서 동작하고, 공식적인 검증 테스트(conformance test suite)의 96%이상을 통과하여, 구현의 정확성을 보였다. 또한, 상업적으로 제공되는 벤치마크 프로그램들에 대한 테스트에서 기존의 상업적 구현 사례와 동등한 수행 속도를 보였다.

키워드 : OpenGL ES, 에뮬레이션, 모바일 그래픽스, 임베디드 시스템

1. 서 론

애플 아이폰, 구글 안드로이드 등의 스마트 폰과 태블릿 PC에서는 3차원 GUI가 매우 중요한 역할을 담당한다[1]. 이에 따라, 최근의 임베디드 시스템들에서는 3차원 그래픽스 출력을 중요시하여, 대부분의 임베디드 시스템에서 이를 지원하고 있다. 비영리 표준 제정 단체인 크로노스 그룹(Khronos Group)에서 제정한 OpenGL ES(OpenGL for Embedded Systems)는 임베디드 시스템에서의 효율적인 3차원 그래픽스 출력을 위한 실질적 표준(de facto standard)이다. 현재 OpenGL ES는 중저가 시스템을 위한 고정 그래픽스 파이프라인(fixed graphics pipeline)을 제공하는 ES 1.1[2]과 고가 시스템을 위한 프로그래머블 파이프라인(programmable pipeline)을 제공하는 2.0[3] 및 2.0을 개선한 3.0[4]이 나와 있다.

이러한 분류는 데스크탑 시스템에서의 발전 추이를 그대로 따라가는 것으로, OpenGL ES 1.1은 OpenGL 1.3에서 파생되었고, OpenGL ES 2.0은 OpenGL 2.1[5]부터 도입된 프로그래머블 파이프라인과 GLSL(OpenGL shading language)[6]에 기반하고 있다. OpenGL ES 3.0은 2012년 8월에 표준이 제정되었지만, 기능상으로는 ES 2.0 버전의 확장판으로 설정되었고, 아직까지는 사용된 사례가 미미하다. 따라서, OpenGL ES 3.0은 아직까지 충분히 활성화되지 않은 상황으로 보이고, 본 논문에서는 현시점에서 가장 널리 사용되는 OpenGL ES 2.0을 대상으로 한다.

OpenGL ES 2.0에서 도입된 프로그래머블 파이프라인은 이전까지의 고정 파이프라인에서 하드웨어로 고정되어 있던 기능들을 필요에 따라 최적화 할 수 있도록, Fgi. 1에서와

※ This investigation was financially supported by Semiconductor Industry Collaborative Project between Kyungpook National University and Samsung Electronics Co. Ltd.
† 종신회원 : 경북대학교 컴퓨터학부 교수(Smart Life 실현을 위한 SW인력양성 사업단)
논문접수 : 2014년 1월 16일
심사완료 : 2014년 3월 13일

같이, 버텍스 쉐이더(vertex shader)와 프래그먼트 쉐이더(fragment shader)를 제공하고, 이들을 프로그래밍 할 수 있는 쉐이더 랭귀지(shader language)로 ESSL(OpenGL ES shader language)를 지원한다[7]. 이러한 방식은 그래픽스 하드웨어를 매우 효과적으로 사용할 수 있도록 하여, 낮은 전력 소모로도 더 빠른 시간 내에 더 우수한 출력을 가져올 수 있기 때문에, 점차 사용이 확대될 것으로 기대된다.

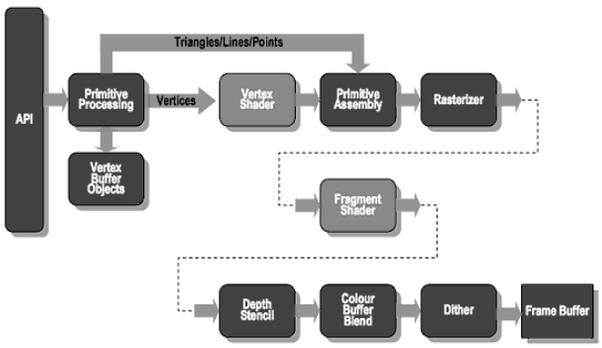


Fig. 1. OpenGL 2.0 programmable graphics pipeline [3]

최근의 스마트 폰을 비롯한 많은 임베디드 시스템들에서 OpenGL ES 2.0 표준에 부합되는 3차원 그래픽스 기능을 필요로 하지만, 실제 프로그램의 개발에 있어서는 임베디드 환경에서의 직접 개발은 효율성이 떨어진다. 따라서, 많은 임베디드 프로그램들이 PC 환경에서 개발된 후에 다운로드 과정을 거쳐 대상이 되는 임베디드 시스템에서 실행되는 방식을 쓴다. 이때, PC에서는 임베디드 환경을 에뮬레이션 할 필요가 있고, 본 논문에서는 PC 환경의 OpenGL 2.1 표준을 이용하여 스마트 폰이나 태블릿 PC 환경의 OpenGL ES 2.0을 에뮬레이션한 결과를 보인다.

저자가 아는 한, OpenGL ES 2.0의 에뮬레이션에 대한 학술적 발표는 이제까지 없었다. 반면, OpenGL ES 2.0의 상업적 구현 사례는 이미 존재하고 있다. 대표적인 것으로는 Imagination Technology 사의 PowerVR[8], ARM 사의 Mali[9], NVIDIA사의 Tegra[10] 등이 있다. 이들은 모두 PC 상에서의 OpenGL ES 2.0 에뮬레이터를 제공하고 있지만, 그 성능이나, 구현 방식에 대해서는 언급하지 않고 있다.

본 논문의 2장에서는 구현을 위한 설계와 실제 구현 과정에서의 세부 사항들을 설명한다. 3장에서는 구현 결과가 제시되고, 이로부터 4장에서 결론과 향후 과제를 보이겠다.

2. 설계 및 구현 전략

OpenGL ES 2.0이 데스크탑 용의 OpenGL 2.1에서 파생되었기 때문에, 에뮬레이터 구현이 용이할 것으로 보일 수도 있지만, 내부적으로는 OpenGL 2.1에서 파생된 부분 집합 성격이 아니라 임베디드 시스템의 특성을 반영하기 위한 다양한 기능 보강이 단행되었다. 예를 들어 IEEE 754 형식의 부동소수점 표현 방식만 지원하는 OpenGL에 비해 OpenGL ES 2.0은 많은 API 함수들에서 signed 15.16 형태의 고정소수점 표현을 필수적으로 지원하여야 한다. 또한, OpenGL에서는 익스텐션(extension) 기능으로 분류되어 있는 프레임버퍼 오브젝트(framebuffer object: FBO) 기능[11]이 코어(core) 기능으로 재분류 되면서 이를 지원하기 위한 API 들이 필수적으로 추가되어야 한다.

쉐이더 랭귀지(shader language) 측면에서는 GLSL 보다 ESSL 표준이 나중에 제정되면서 자료형 검사가 좀더 엄격해 지고, 상당수의 빌트인(built-in) 변수들이 이름이 바뀌거나 상수 값이 바뀌는 변화를 겪었다. 이에 따라, ESSL로 작성된 프로그램은 GLSL과 완전한 호환이 되지 않으며, 많은 경우에 GLSL에서는 문제가 없던 쉐이더 프로그램들이 ESSL에서는 더 엄격한 문법으로 인해 컴파일 오류를 발생시켜야 한다. 이는 결과적으로, ES 2.0 에뮬레이터들은 궁극적으로 ESSL 프로그램을 파싱해서 독자적으로 신택스(syntax) 및 시맨틱스(semantics) 오류를 충분히 찾아낸 후, 아무런 문제가 없는 ESSL 프로그램에 대해서만 GLSL로의 변환을 단행하여 해당 코드를 출력시켜 주어야 하는 방식이 필요하다. 본 논문에서는 바로 이런 방식으로 ESSL-to-GLSL 번역기를 구현하여 내장하였으며, 이 작업이 상당한 부분을 차지하였다.

에뮬레이터 전체에 대한 설계는 크게 코어 파이프라인, 쉐이더 랭귀지, 프레임버퍼, 익스텐션의 4부분으로 구성되어 있다. 다음 절들에서는 이들 각각을 차례로 설명하겠다.

2.1 코어 파이프라인 구현

본 논문에서 제공하는 모든 OpenGL ES 2.0 라이브러리 API 함수들은 크로노스 그룹에서 제공하는 표준 헤더 파일(header file)들인 gl2.h, gl2ext.h, gl2platform.h 등을 기준으로, 표준에서 정의한 형태를 그대로 사용하여 100% 호환성을 확보하였다.

실제 구현에서는 vertex attribute, vertex buffer object (VBO)에 대한 고정소수점 표현을 지원하는 부분이 상당한 부분을 차지했다. OpenGL 하드웨어는 부동소수점 표현만을 지원하므로, 모든 고정소수점 표현은 일단 부동소수점 표현으로 변환되어야 OpenGL 함수를 이용할 수 있다.

vertex attribute에 고정소수점 자료형을 사용하는 경우에는 가능한 한 자료 변환을 뒤로 미루어서, 불필요한 자료 변환을 피하도록 하였고, glDrawElements(…)나 glDrawArrays(…) 함수를 통하여 그래픽스 출력이 일어나야만 하는 시점에 자료 변환을 수행하도록 하였다.

다음으로, vertex buffer object(VBO)에서도 고정소수점 자료형을 사용할 수 있는데, VBO에서는 한번 저장된 자료를 반복해서 사용한다는 특성을 가지므로, 자료 변환을 한번 수행한 후에는 변환된 자료를 별도의 메모리에 저장하고, 그 이후는 매번 저장된 자료를 그대로 사용하게 하였다. 이러한 "변환 후 저장" 방식은 최종 구현 결과에서 주목할 만한 속도 향상을 가져왔다.

2.2 쉐이더 랭귀지 지원

OpenGL ES 2.0은 OpenGL ES Shader Language (ESSL)를 지원하기 위해, 내부적으로 "쉐이더 랭귀지 컴파일러(shader language compiler)"를 제공해야 한다. 본 논문에서는 완전히 새로운 풀-스케일 ESSL 컴파일러를 제공하는 대신에, ESSL로 작성된 쉐이더 프로그램을 GLSL로 변환

하는 "ESSL-to-GLSL 번역기"를 채택하였다. 풀-스케일 ESSL 컴파일러를 제공하는 경우에는 특정 GPU에 의존적인 어셈블리 코드들이 나와야 하지만, 번역기 방식은 PC 시장의 다양한 하드웨어 구성을 지원하기에 더 적합하다.

구현한 번역기는 크게 "전처리기(preprocessor)"와 "코어 처리기(core processor)"로 구성된다. 전처리기는 #define, #version, #pragma와 같은 ESSL 전처리기 명령들을 처리하고, 전처리가 끝난 쉐이더 랭귀지 소스 코드는 OpenGL ES 2.0을 기준으로 엄격한 신택스(syntax), 시맨틱스(semantics) 체크를 거친다. 코어 처리기 부분은 ESSL 전용으로 설계된 DFA(deterministic finite automata)와 PDA(push-down automata)로 구성된다. 각각은 C-like 언어인 ESSL 언어를 파싱(parsing)해야 하고, 특히, ESSL에 추가된 Ada 언어의 일부 기능을 지원하는 시맨틱스 처리 부분을 추가했다.

이러한 번역기 방식의 처리는 최초의 1번만 수행된다. 최초에 번역이 완료된 후에는 컴파일된 오브젝트 코드를 저장해서, 이것을 반복해서 사용하도록 했다. 따라서, 본 논문의 시스템은 최초의 ESSL-to-GLSL 번역 시에 약간의 지연(delay)이 발생할 수 있지만, 이후로는 GLSL로 작성한 경우와 비교해도 전혀 속도 저하가 일어나지 않는다는 장점을 가진다.

2.3 프레임버퍼 익스텐션 지원

데스크탑 용의 OpenGL에서는 별도의 framebuffer extension[11]을 제공해서, 그래픽스 출력을 표준 프레임 버퍼(frame buffer)뿐만 아니라, 렌더 버퍼(render buffer)나 2D 텍스처(texture)로도 보낼 수 있도록 했다. OpenGL ES 2.0에서는 이것이 익스텐션이 아니라 코어(core) 기능이 되었으므로, 반드시 제공되어야 한다. 다행히 현재 사용되는 OpenGL 2.0 그래픽 카드들이 framebuffer extension을 대부분 제공하고 있지만, OpenGL ES 2.0에서는 세부적으로 좀더 향상된 기능들을 제공할 수 있다고 정의되어 있다. 본 논문에서는 별도의 소프트웨어 에뮬레이션 코드들을 추가해서 OpenGL ES 2.0에서 설정한 모든 framebuffer 기능들을 제공하도록 했다. 즉, 기반이 되는 데스크탑 용의 OpenGL 2.0에서 제공하는 framebuffer extension 보다 더 향상된 기능을 제공하고 있다.

2.4 익스텐션 구현

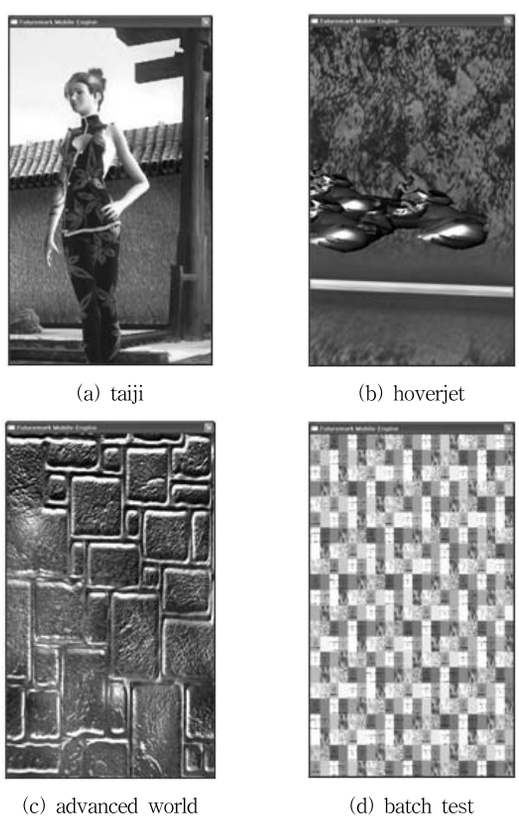
OpenGL ES 2.0에서는 핵심 명세서(core specification)를 제정한 이후에 발생한 기술적 진보나 사용자들의 요구 사항들을 지원하기 위해서, 다양한 익스텐션(extension)들을 제공한다[12]. 시스템을 구현하는 입장에서는 핵심 명세서를 그대로 구현하는 것도 중요하지만, 다양한 익스텐션들을 추가 제공할 필요도 있다. 시스템 구현이 완료된 후에 고려하지 않았던 익스텐션을 추가하기 위해서는 경우에 따라서는 이미 구현된 파이프라인 아키텍처에 비교적 큰 수정을 가해야 할 수도 있다. 본 논문에서는 설계 단계부터 OpenGL ES 2.0 표준의 핵심 기능은 물론이고, 미리 Table 1의 11개 익스텐션들을 설계에 반영하여 최종 구현에 성공하였다.

Table 1. Extensions supported by our implementation

OES_blend_func_separate
OES_blend_subtract
OES_fixed_point
OES_single_precision
OES_stencil_wrap
OES_texture_mirrored_repeat
OES_element_index_uint
OES_texture_3D
OES_texture_npot
OES_rgb8_rgba8
OES_packed_depth_stencil

3. 결과 및 분석

전체 시스템은 향후의 이식성을 높이기 위해서, 리눅스 기반의 데스크 탑 PC에서 ANSI C89 표준[13]의 C언어를 사용해서 개발되었다. 구현된 코드의 대부분은 하위 운영체제와는 무관하므로, 다른 운영체제나 다른 플랫폼으로의 이식이 용이하다. 개발이 완료된 후에는 MS Windows 기반의 PC에서 Visual Studio 2005를 사용하는 시스템으로 변환하여 성공적으로 동작함을 확인하였다.



(a) taiji (b) hoverjet

(c) advanced world (d) batch test

Fig. 2. Selected screen shots from the test programs

구현이 완료된 시스템은 표준 명세에의 적합성을 테스트하기 위해 크로노스 그룹에서 제공하는 검증 테스트(Conformance Test Suite: CTS)[14]를 사용하여 검증하였다. 현재까지의 구현 결과로는 총 1,196개의 CTS 테스트 중에서 37개를 실패하여, 총 96.9%의 CTS 성공률을 달성하였다. 일부 실패한 테스트는 하위의 OpenGL 하드웨어에서 발생한 호환성 위배가 원인이었다. 그 외에도, Fig. 2에서와 같이, 기존의 다양한 OpenGL ES 관련 프로그램들과 Future Mark 사의 벤치마크 프로그램들[15]에 대해서도 테스트가 수행되어, 별다른 문제가 없음을 확인하였다.

Table 2. Experimental results (unit: fps)

		Our Imp.	PowerVR	Ratio
intro	min	59.9	59.9	100.00%
	max	59.9	60.0	99.83%
taiji	min	19.8	19.9	99.50%
	max	26.0	27.7	93.86%
hoverjet	min	29.9	29.8	100.34%
	max	43.0	45.5	94.51%
advanced world	min	59.8	59.8	100.00%
	max	60.0	60.0	100.00%
batch test	min	59.0	59.8	98.66%
	max	59.8	60.0	99.67%
texture tiling	min	59.8	60.0	99.67%
	max	60.0	60.0	100.00%
unified shader	min	59.8	59.8	100.00%
	max	60.0	60.0	100.00%

실행 속도를 측정하기 위해서는 Intel Xeon 3.0GHz CPU와 8GB RAM, NVIDIA GeForce GTX260 그래픽스 카드를 장착한 시스템을 사용하였다. 실행 속도의 비교 대상으로는 현재 사용가능한 에뮬레이터들 중에서 가장 많이 사용되는, PowerVR 사에서 제공하는 에뮬레이터[8]를 사용하였다. 본 논문에서 구현한 시스템과 PowerVR 에뮬레이터를 대상으로 FutureMark 사의 3DMarkMobile ES 2.0 벤치마크 프로그램[15]을 수행한 결과는 Table 2와 같다. 표에서 보는 바와 같이, 두 시스템은 거의 동일한 수행 속도를 보였고, 일부 차이가 나는 테스트들에서도 측정 오차로 간주할 만한 1% 내외의 오차만을 보였다. 이로부터 본 논문의 시스템이 수행 속도 면에서도 기존의 상용 시스템들에 비해서 손색이 없음을 알 수 있다.

4. 결론 및 향후 과제

본 논문에서는 현재 모바일 그래픽스용으로 가장 많이 사용되고 있는 OpenGL ES 2.0 표준의 응용 프로그램 개발에 반드시 필요한 에뮬레이터 환경을 데스크탑 PC들에서 제공하고자 하였고, 최종적으로 구현한 시스템과 그 결과를 제시하였다. 개발한 시스템은 크로노스 그룹의 검증 테스트에서 우수한 성공률을 보였고, 다양한 벤치마크 프로그램들과 예제 프로그램들에서 문제가 없음을 보였다. 실행 속도 면에서도, 기존의 상용 에뮬레이터와 동등한 성능을 보였다.

다음 단계로는 OpenGL ES GPU 에뮬레이터를 작성하여, 현재의 구현과 통합시킴으로써, 풀-소프트웨어로 구현된 에뮬레이터 시스템을 완성할 수 있다. 또한, 최근에 제정된 OpenGL ES 3.0 표준[4]으로 시스템을 개선할 예정이며, OpenGL SC [16], WebGL [17] 등의 다른 그래픽스 표준들에 대한 에뮬레이터들도 제작할 예정이다.

Reference

[1] T. Capin, K. Pulli and T. Akenine-Moller, "The State of the Art in Mobile Graphics Research," *IEEE Computer Graphics and Applications*, 28(4):74-84, 2008.

[2] D. Blythe, *OpenGL ES Common/Common-Lite Profile Specification*, Version 1.1.12 (Full Specification), 2008.

[3] A. Munshi and J. Leech, *OpenGL ES Common Profile Specification*, Version 2.0.25 (Full Specification), 2010.

[4] B. Lipchak, *OpenGL ES*, Version 3.0.2, 2013.

[5] M. Segal and K. Akeley, *The OpenGL Graphics System: A Specification*, Version 2.1, Dec., 2006.

[6] J. Kessenich, *The OpenGL Shading Language*, Language Version: 1.20, Sep., 2006.

[7] R. J. Simpson, *The OpenGL ES Shading Language*, Language Version: 1.00, May, 2008.

[8] Imagination, *PowerVR Insider*, http://www.imgtec.com/

[9] ARM, *Mali Graphics*, http://www.arm.com/products/multimedia/mali-graphics-hardware/index.php

[10] NVIDIA, *Tegra 4 Super Processors*, http://www.nvidia.com/object/tegra.html

[11] R. Barris, et. al, *OpenGL Extension #45*, GL_ARB_framebuffer_object, 2008.

[12] A. Munshi, *OpenGL ES 1.1 Extension Pack Specification*, Version 1.03, Jul., 2005.

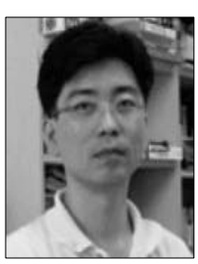
[13] ISO/IEC, *International Standard 9899: Programming Language C*, 1989.

[14] Khronos Group, *OpenGL ES 2.0 CTS (conformance test suites)*, http://www.khronos.org/opengles/adopters/

[15] FutureMark, *3D Mark @ Mobile ES 2.0*, http://www.futuremark.com/pressreleases/50623/

[16] B. Stockwell, *OpenGL SC: Safety-Critical Profile Specification*, version 1.0.1 Khronos Group, 2009.

[17] C. Marrin Ed., *WebGL Specification*, Draft 16, Khronos Group, 16 March, 2012.



백 낙 훈

e-mail : oceancru@gmail.com

1992년 한국과학기술원 전산학과(공학석사)

1997년 한국과학기술원 전산학과(공학박사)

2004년~현 재 경북대학교 컴퓨터학부 교수

관심분야 : Mobile Graphics, Real-time Graphics