Journal of Korea Multimedia Society Vol. 14, No. 3, March 2011(pp. 440-448)

OpenGL 상에서 OpenGL SC 에뮬레이션 라이브러리 구현

백 낙 훈†

요 약

스마트 폰, 개인용 기기 등을 비롯한 다양한 시스템들에서 그래픽 기반의 사용자 인터페이스를 개발하기 위한 OpenGL 계열 렌더링 표준에 대한 수요가 증가하고 있다. 또한, 항공용, 군사용, 의료용, 차량용 분야의 수요를 중심으로 형성된 세이프티-크리티컬(safety-critical) 시장에서는 OpenGL의 세이프티-크리티컬 프로파일로 개발된 OpenGL SC 표준이 중요한 역할을 담당한다. 본 논문에서는 OpenGL SC 표준을 비용 대비 효과적으로 제공하기 위해서, 기존의 임베디드 시스템들에서 비교적 널리 사용되고 있는 OpenGL 1.x 파이프라인 상에서 OpenGL SC 에뮬레이션을 제공하는 방법을 제안한다. 우리가 제안하는 방법은 임베디드 시스템에서 낮은 개발비로 OpenGL SC 표준을 제공할 수 있으며, 임베디드 시스템용 PC 개발 환경에서의 에뮬레이션을 위한 필수 요소이기도 하다. 최종 결과는 리눅스 기반 시스템과 VxWORKS 기반 시스템에서 각각 표준에 맞게 작동하고, 적합한 실행 속도를 보였다.

Implementation of OpenGL SC Emulation Library over OpenGL

Nakhoon Baek†

ABSTRACT

The needs for the OpenGL-family of the rendering library standards are highly increasing, especially for the graphical human-machine Interface on the various systems including smart phones and personal information devices. Additionally, in the case of safety-critical market for avionics, military, medical and automotive applications, OpenGL SC, the safety critical profile of the OpenGL library plays the major role for the graphical interfaces. In this paper, we represent our OpenGL SC emulation library on the OpenGL 1.x rendering pipeline, which is widely available on the existing embedded systems, to provide the features of OpenGL SC standard cost-effectively. Our method can provide the OpenGL SC features at the low development cost on the embedded systems, and its implementation is also one of the fundamental elements for the emulation of embedded systems in the PC environment. Our final result now works on both of Linux-based and VxWORKS systems, showing correct execution results at the reasonable speed.

Key words: OpenGL SC(OpenGL SC), OpenGL(OpenGL), Safety-Critical(세이프티 크리티컬), Cost-Effective Implementation(비용 대비 효과적 구현)

※ 교신저자(Corresponding Author) : 백낙훈, 주소 : 대구시 북구 산격동 1370 경북대학교 IT대학 컴퓨터학부(702-701), 전화 : 053)950-6379, FAX : 053)950-6369, E-mail : oceancru@gmail.com
접수일 : 2010년 8월 17일, 수정일 : 2010년 11월 4일
완료일 : 2011년 1월 4일
† 정회원, 경북대학교 컴퓨터학부 부교수
※ 이 논문은 2010년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(과제번호: 2010-0028106).

1. 서 론

휴대폰을 비롯한 다양한 임베디드 시스템과 휴대용 기기들에서는 그래픽 기반의 사용자 인터페이스를 구현하기 위해서 OpenGL 및 이에 연관된 3차원 렌더링 표준들이 널리 사용되고 있다. 현재는 실질적 표준 관리 기구 중의 하나인 크로노스 그룹(Khronos Group) [1]에서 OpenGL [2], OpenGL ES (for embedded systems) [3], OpenGL SC (safety critical profile) [4] 등을 비롯한 OpenGL 계열의 모든 그래픽스 표준에 대한 문서를 관리하고 있다. OpenGL이나 OpenGL ES의 기능을 제공하는 하드웨어 장치나 소프트웨어들은 소비자를 대상으로 하는 상업적 시장에서 쉽게 구할 수 있다.

항공용, 군용, 의료용 및 자동차 전장용 어플리케이션을 대상으로 하는 세이프티-크리티컬(safety-critical) 시장에서는 OpenGL의 safety-critical profile로 개발된 OpenGL SC가 사용자 인터페이스 개발의 중요한 역할을 담당한다. 세이프티-크리티컬 시장의 성장에 따라 3차원 그래픽스 API (application program interface)에 대한 요구도 급격히 증가하고 있다. 의료용이나 자동차 전장용 어플리케이션 등에서는 소비자 대상의 시장에서도 OpenGL SC 표준이 필요해 지고 있다.

이에 따라, OpenGL SC의 비용 대비 효과적인 구현이 자연스럽게 요구되고, 특히, 가능하다면 상용 기성품(COTS, commercial off-the-shelf) 장비들로 구현되기를 요구하고 있다[5-7]. 반면에, 현재로서는 널리 사용되는 OpenGL SC 구현 결과물을 찾기는 어렵다. 본 논문에서는 임베디드 시스템을 대상으로 하는 OpenGL SC의 비용 대비 효과적 구현을 목표로 하였고, 이를 위해서 현용의 임베디드 시스템들에서 폭넓게 사용 가능한 OpenGL 1.x 제품들에 주목하였다. 최종적으로 우리는 OpenGL 1.x의 고정 렌더링 파이프라인(fixed rendering pipeline) 위에서 OpenGL SC 에뮬레이션 라이브러리를 제공함으로써, OpenGL SC 표준의 모든 기능을 저가격에 제공할 수 있음을 보인다.

본 논문에 제안하는 방식의 에뮬레이션 라이브러리가 필요한 이유는 다음과 같다.

- **비용 대비 효과의 향상**: OpenGL SC의 기능들을 완전히 새로 개발할 수도 있지만, 이미 시장에는 OpenGL 또는 OpenGL ES의 인증된 하드웨어와 디바이스 드라이버들이 충분히 합리적인 가격에 제공되고 있다. 본 논문에서는 이러한 하드웨어들을 충분히 이용하여 상대적으로 낮은 가격에 OpenGL SC의 기능을 제공할 수 있다는 것을 보이고자 한다.
- **빠르고 안정적인 구현 방법**: 완전히 독립적인 OpenGL SC 하드웨어와 소프트웨어 구현들을 제공하기에 앞서, 이의 중간 단계로, OpenGL 라이브러리를 내부 렌더링 엔진의 구현에 사용하는 방식으로 매우 안정적인 제품을 에뮬레이션 형태로 단기간에 개발하여 제공할 수 있다.
- **효율적인 개발 환경의 제공**: 임베디드 시스템들은 보통 시스템의 성능 한계 때문에 일반적인 개발 도구로 직접 사용하기에는 무리가 따른다. 이에 따라, 대부분의 임베디드 시스템용 어플리케이션들은 PC 환경에서 개발된 후에 대상 임베디드 시스템으로 다운로드하는 방식을 택한다. 이러한 크로스 컴파일 환경에서는 임베디드 시스템과 PC 모두에서 작동하는 에뮬레이션 라이브러리가 반드시 필요하다. 우리는 OpenGL SC를 위한 이러한 방식의 에뮬레이션 라이브러리를 제공하는 것이기도 하다.

이제까지의 OpenGL SC 구현은 하드웨어 칩 위에 직접 드라이버를 제공하는 방식을 택하여, device driver 개발 비용이 많이 들어갈 수도 있다. 또는 전체 구현을 software로 시도한 경우도 있는데, 이 역시 모든 기능을 소프트웨어로 구현하기 때문에, 역시 많은 비용이 들어갈 수 있다.

우리는 최소 비용으로 기능을 제공하기 위해, 일종의 layer 형태로 OpenGL SC emulator를 만들었다.

2장에서는 OpenGL SC의 이전까지의 개발 사례들을 소개하고, 유사한 라이브러리들에 대한 에뮬레이터 개발 사례들을 살펴 본다. 3장에서는 전체적인 설계와 이에 따른 자세한 구현 방법, 텍스처 처리 파이프라인 등을 차례로 설명한다. 4장에서 구현 결과를 보이고, 5장에서 결론 및 향후 과제를 제시한다.

2. 기존 연구 결과

개방형 표준인 OpenGL SC (safety critical profile)은 항공용, 산업용, 군사용, 차량용 어플리케이션을

위한 세이프티-크리티컬 시장에서의 요구을 만족시키기 위해 제정되었다. 이 표준은 세이프티-크리티컬한 소프트웨어의 검증을 쉽게 하고, 재현성을 높이며, 실시간 요구에 부응하고, 이미 사용 중인 세이프티-크리티컬 어플리케이션의 포팅을 돕는다[4].

현재 사용 중인 OpenGL SC 1.0.1 표준에서 지원하는 어플리케이션 영역은 다음과 같다[1].

● 항공용 : 미국 연방항공국(Federal Aviation Administration)에서 규정한 항공기 조정실용 소프트웨어를 위한 DO-178B 검증 과정에서는 계기판, 항법장치, 제어장치 등에서 100% 신뢰할 수 있는 그래픽스 드라이버를 요구한다.

● 차량용 : 차량용 통합 계기판 어플리케이션에서는 OpenGL SC가 제공하는 세이프티-크리티컬 분야에의 신뢰성을 요구한다.

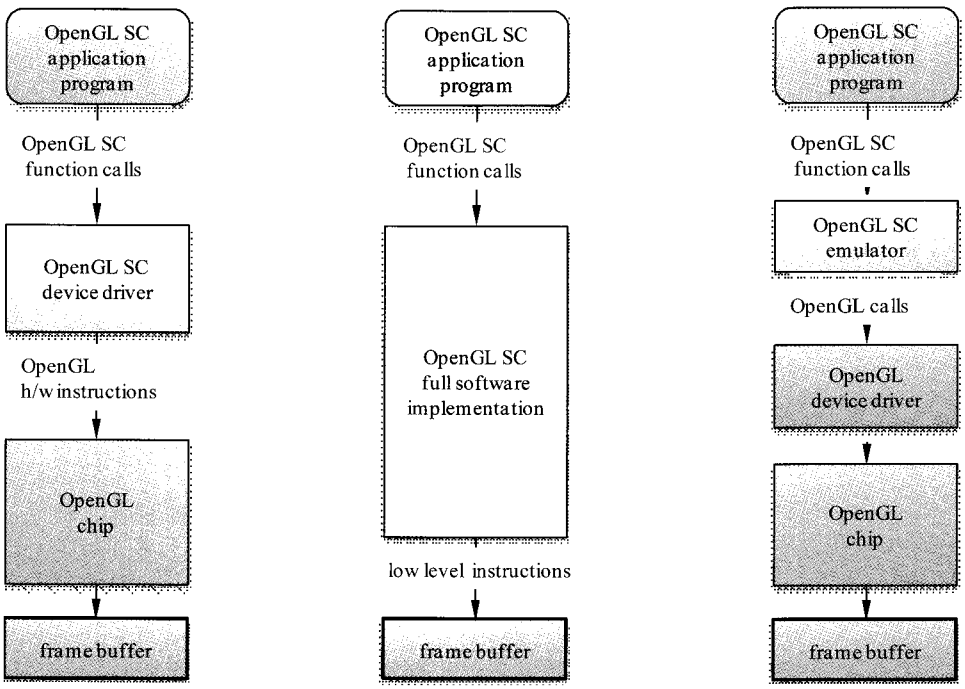
● 산업용 : 발전설비, 운송장비, 네트웍 관리 등의 분야에서는 세이프티-크리티컬 분야의 요구 사항들을 충족시키는 상용 장비들이 사용되어야 한다.

● 의료용 : 의료용 데이터의 실시간 디스플레이는 특히 수술 목적으로는 100% 신뢰성을 요구한다.

● 군사용 : 많은 부분이 군용 항공기들에 치중하고 있지만, 휴대용 기기들에서의 임베디드 트레이닝이나 시각화 부분도 증가하고 있다.

현재 사용 가능한 OpenGL SC 구현들은 그렇게 많지 않다. 이들은 크게 2가지 종류로 나눌 수 있는데, 반도체 칩을 사용해서 하드웨어 가속이 가능한 것들과, 완전히 소프트웨어로 구현된 것이다. 전자의 경우는 그림 1(a)에서와 같이, 전용의 OpenGL SC 반도체 칩을 사용하거나, 다른 OpenGL 계열의 반도체 칩 위에 전용 디바이스 드라이버를 사용하는 방식이다. 이들은 반도체 칩과 디바이스 드라이버의 개발 비용을 필요로 한다. 후자의 경우는 그림 1(b)에서와 같이, 전체를 소프트웨어로 구현하게 되는데, 경우에 따라서는 처리 속도가 만족스럽지 못할 수 있다. 이들에 대해서 차례로 살펴 보겠다.

OpenGL SC를 필요로 하는 시장이 아직까지 충분히 성장하지 않았기 때문에, OpenGL SC 전용 chip을 새로 개발하기 보다는 이미 시장에 출시된



(a) OpenGL 계열의 반도체 칩을 사용하는 경우 (b) 전체를 소프트웨어로만 구현하는 경우 (c) OpenGL 디바이스 드라이버 위에 에뮬레이션 라이브러리로 구현하는 경우

그림 1. OpenGL SC의 구현 방법들

OpenGL 또는 OpenGL ES 칩을 사용해서, 별도의 OpenGL SC 드라이버를 제공하는 방식이 더 경제적일 수 있다. ALT Software Inc는 ATI 사의 OpenGL 또는 OpenGL ES 칩들을 기반으로 자체 개발한 별도의 디바이스 드라이버를 제공하는 방식을 택하고 있다[8]. 지금은 Presagis 사에 합병된, Seaweed 사에서는 nVIDIA 사의 OpenGL 칩들을 기반으로 역시 전용의 디바이스 드라이버를 개발하였다[9]. 이 방식은 하드웨어 칩의 지원을 받으므로, 고속의 처리가 가능하지만, 완전히 새로운 디바이스 드라이버를 개발해야 하므로, 실제로는 상당한 개발 비용을 필요로 한다.

Quantum3D 사에서 개발한 IGL 178과 같은 시스템은 full software 방식으로 OpenGL SC를 지원하고 있다[10]. 이 방식은 순수 소프트웨어 방식으로 개발되어, 발견된 버그를 수정하거나, 새로운 하드웨어 시스템으로의 이전이 용이하다는 장점을 가진다. 반면에, 하드웨어의 지원을 받기가 상당히 곤란하므로, 속도의 저하를 피할 수 없다. 다만, 이 방식에서는 비교적 낮은 가격으로, 버그에 재빠르게 대처하는, 상당히 안정적인 시스템을 제공하는 것이 가능하기 때문에, Vincent 3D사의 Vincent SC [11], ALT 사의 OpenGL SC software render [8] 등도 full software로 구현되어 있다.

우리의 경우는 그림 1(c)에서와 같이, 이미 상업적으로 완성된 OpenGL 디바이스 드라이버 위에 OpenGL SC emulator를 구현하고자 한다. 이미 존재하는 그래픽스 파이프라인 위에 다른 그래픽스 라이브러리를 구현하면 비용 대비 효과나 다른 시스템에의 이식성 등에서 장점이 있다. 상업적으로 판매되는 그래픽스 라이브러리 위에서 다른 그래픽스 라이브러리를 에뮬레이션하는 사례들이 알려져 있다. OpenGL ES 표준의 경우에는 OpenGL ES 2.0 위에 OpenGL ES 1.1을 구현한 적이 있고[12], 이 과정에서 OpenGL ES 1.1만의 독특한 기능들을 지원하기 위해서 OpenGL ES 2.0 파이프라인의 일부를 보강하기도 했다. 또다른 사례로는 데스크 탑 용의 OpenGL 1.5 표준 위에서 OpenGL ES 1.1을 성공적으로 제공한 것이 있다[13].

현재까지 공개된 연구결과들 중에서는 OpenGL SC를 에뮬레이터 방식으로 구현한 적은 없다. 우리의 에뮬레이터 방식 구현은 2가지 면에서 의의를 가지는데, 우선, OpenGL 1.1 파이프라인과 ARB_multitexutre extension을 지원하는, 최소 사양의 임베디드 시스템들에서도 OpenGL SC를 최소 비용으로 에뮬레이션할 수 있음을 보였다. 또한, OpenGL SC를 위한 PC 기반의 개발 환경을 제공하기 위해서는 데스크 탑 OpenGL 위에서도 동작하는 에뮬레이터가 반드시 필요하다.

에뮬레이터 방식의 구현은 하위의 라이브러리(underlying library)를 잘 선택하면, 최소의 비용으로 원하는 기능을 구현할 수 있지만, 둘 사이의 차이점을 메우는 것이 단순하지만은 않은 것이 보통이다. 우리의 경우도 상당한 수의 기술적 문제들을 만났고, 다음 절에서 보는 바와 같이, 이들을 하나씩 해결해 나가야 했다. 또한, 에뮬레이터 방식은 통상 하위의 라이브러리보다는 느려지게 되므로, 이러한 속도 저하를 최소화할 필요가 있다[14].

3. 설계 및 구현

3.1 구현 전략

OpenGL SC는 데스크 탑 용의 OpenGL 1.3 표준[15]을 기반으로 하여, 총 101개의 API 함수들을 제공한다. 그림 2는 OpenGL SC 렌더링 파이프라인의 전체 블록 다이아 그램이다. 이들 API 함수들은 코어(core) API 함수들과 OES_single_precision, EXT_paletted_texture, EXT_shared_texture_palette 등의 extension 들로 구성된다[4].

텍스처에 관계된 마지막 2개의 extension들은 기존의 많은 항공용 어플리케이션들에서 2D 텍스처 매핑 용으로 요구하고 있는 기능이다. 이들은 텍스처 데이터로부터 별도의 색상표(color table)을 추출해서, 색상표를 변화시키거나, 여러 개의 텍스처들이 하나의 색상표를 공유(share)하는 등의 작업이 가능하게 한다[16]. 본 논문에서는 이들 모든 extension들을 지원하고자 한다.

대부분의 OpenGL SC 코어 API 함수들은 OpenGL 1.1 코어에서 지원 가능하다. 다만, 다중 텍스처 유닛(multiple texture units)을 지원하기 위한 일부 텍스처 함수들은 OpenGL 1.3 이후의 코어에서만 지원된다. 이 기능은 OpenGL 1.1 또는 1.2 코어에 포함되어 있지는 않지만, 이들 하위 버전의 코어에서도 별도로 ARB_multitexture extension을 지원한다

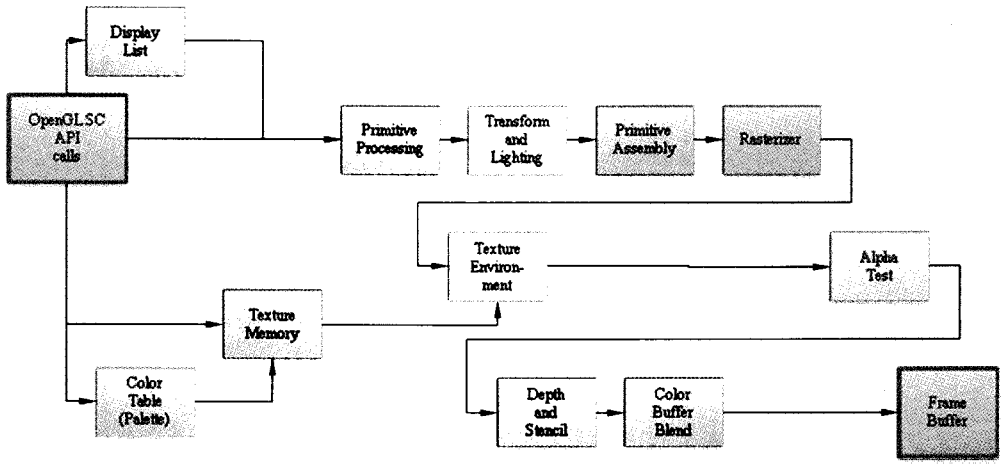


그림 2. OpenGL SC 렌더링 파이프라인

면 사용 가능하다[16]. 다만 이 경우에는 다중 텍스처 기능을 사용하기 위해서는 코어 API 함수 대신, ARB extension 에서 정의한, 형태가 다른 함수들을 사용해야 한다.

OpenGL SC 표준에 정의된 함수 이름들이 원래의 바탕이 된 OpenGL 표준에서도 똑 같은 이름으로 정의되어 있는 경우를 자주 보게 되지만, 이들의 이름이 같다고 해서, 기능까지 일치하지는 않는다. 세이프티-크리티컬한 장치들에 적합하도록 표준이 설정되면서, OpenGL SC의 함수들은 기존의 OpenGL 함수들과 같은 이름을 쓰더라도 받아들이는 매개변수 값들이 달라지고, 해당되는 기능에 변화가 생긴 경우가 많다. 따라서, OpenGL SC 표준의 요구 사항을 만족시키기 위해서는 좀더 강력한 오류 검사(error check)가 필요하고, 하위의 OpenGL 하드웨어가 받아들일 수 있도록 매개변수 값을 적절히 바꿔주는 수치 변환 (numerical conversion) 과정이 필요하다. 이들 추가 작업은 이전에 발표된 데스크 탑용 OpenGL 상에서의 OpenGL ES 1.1 구현 사례[13]에서와 같이, 케이스 별로 따로따로 수행되어야 했다.

우리는 ARB_multitexture extension이 장착된 OpenGL 1.1 코어 시스템을 대상으로, OpenGL SC의 에뮬레이션을 제공한다. 전체적인 구현 전략은 아래와 같다.

● OpenGL 1.1 코어에 해당하는 함수들: 이들은 모두 하드웨어로 구현되어 있고, 핵심적인 기능을 담당한다. 이들에 대해서는 하위의 OpenGL이 1.1 코어 함수들을 제공한다고 가정하고, 하드웨어 구현을 되도록 많이 사용하도록 한다. 다만, 이들이 OpenGL SC와 OpenGL에서 같은 이름을 가지는 함수라고 해서, 기능까지 같은 것은 아니다. 대부분의 함수에서 유효한 매개변수 값(valid parameter values)에 변화가 생겼고, 이에 따른 별도의 오류 처리가 필수적이다. 어떤 함수들은 적당한 수치 변환을 거쳐서 하위 OpenGL 함수를 호출해야 하는 경우도 있다. 이들은 이미 OpenGL ES 에뮬레이터의 구현 [13]에서 보인 바와 같이, 케이스 별로 따로 구현하여야 할 필요가 있다.

● ARB_multitexture extension : OpenGL SC에서는 OpenGL 1.3 코어를 기준으로 하는데, 위에서 언급한 OpenGL 1.1 코어에 속하는 함수들을 제외하면, 순수하게 OpenGL 1.3 코어에 속하는 함수들은 모두 multi-texture에 관계된 함수들이고, 하드웨어에서 텍스처 유닛을 2개 이상 지원해야 동작한다. 우리의 구현에서는 OpenGL 1.3 코어를 사용하는 대신, OpenGL의 ARB_multitexture extension을 사용하도록 했다. 이렇게 하면, ARB_multitexture extension을 지원하되, OpenGL 1.2 또는 OpenGL 1.3 코어는 지원하지 않는 장치들에서도 우리의 OpenGL SC 에뮬레이터가 작동할 수 있다.

● OES_single_precision extension : OpenGL SC에서 반드시 지원해야 하는, mandatory extension인데, 다행히 이들은 이미 존재하는 배정밀도(double precision) 자료형을 매개변수로 가지는 함수들에 대해서 단정밀도(single precision) 자료형 매개변수를

가지는 함수들을 제공하도록 하는 것이다. 따라서, 사용자가 호출하는 단정밀도 값들을 배정밀도 값으로 바꾼 후에, 적당한 오류 처리를 거쳐서 하위 OpenGL에서 제공하는 배정밀도 자료형 함수들을 호출해서 정확한 기능이 수행되게 하는 작업을 처리하면 된다.

OpenGL에서는 배정밀도와 단정밀도 실수 자료형이 혼재되어 있었다. OpenGL SC에서는 상대적으로 비중이 미미한 배정밀도 실수 자료형을 사용하는 함수들을 대응되는 단정밀도 실수 자료형을 사용하는 새로운 함수들로 대체하여, 일관성을 유지하려고 한다.

이것은 OpenGL ES 1.0과 1.1에서도 공통적으로 적용된 사항으로, 아래 4개 함수가 새로 도입되고, 이들은 내부적으로 주어진 단정밀도 실수 값들을 대응되는 배정밀도 실수 값으로 변환하여 적합한 OpenGL 함수를 호출하게 하는 처리가 필요하다.

- void glDepthRangef(GLclampf near, GLclampf far);
- void glFrustumf(GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat near, GLfloat far);
- void glOrthof(GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat near, GLfloat far);
- void glClearDepthf(GLclampf depth);

● EXT_paletted_texture extension : 이 기능도 OpenGL SC에서 반드시 지원해야 하는, mandatory extension이다. 문제는 EXT_paletted_texture extension은 OpenGL SC 표준에서는 이전의 avionics 응용 프로그램들을 지원하기 위해 필요로 하지만, 이 자체는 현재 사용되는 거의 대부분의 OpenGL 하드웨어나 라이브러리 들에서 지원되지 않는 것이다. 이 기능은 원래 indexed color를 지원하는 color palette를 사용하여 texture를 정의하는 것인데, 현재의 추세는 direct color를 사용하는 시스템이 증가하면서, OpenGL 카드들에서는 더 이상 지원되지 않는 경우가 많다. 우리는 이 문제를 해결하기 위해, 다음 절에서 설명하는, 완전히 새로운 텍스쳐 처리 파이프라인을 도입했다.

● EXT_shared_texture_palette: 이 기능은 OpenGL SC에서 선택적으로 지원할 수 있는, optional extension이다. 이 extension은 EXT_paletted_texture extension이 지원되는 경우에 한해서 추가되는 기능이므로, EXT_paletted_texture에서와 같이, 지원되는 OpenGL 카드들이 더 이상 존재하지 않는다는 같은 문제에 부딪치게 된다. 이 문제 역시 완전히 새로운 텍스쳐 처리 파이프라인의 도입으로 해결해서, 우리의 구현은 이 optional extension도 지원한다.

3.2 텍스쳐 파이프라인

EXT_paletted_texture와 EXT_shared_texture_palette extension들을 지원하기 위해서는 이를 지원하는 하드웨어를 찾는 것도 한 방법이지만, 현재 상업적으로 팔리는 그래픽스 카드 중에서는 paletted texture를 지원하는 그래픽스 카드를 찾을 수가 없었다. 이들 extension들은 OpenGL SC에서는 이미 개발된 항공용 소프트웨어들과의 호환성 문제로 계속 지원되어야 하지만, 일반적인 데스크탑 그래픽스 하드웨어들에서는 자연스럽게 도태된 기능들이다. 또한, 우리는 가능한 한 많은 그래픽스 카드에서의 구현을 목표로, OpenGL core 기능의 버전을 낮추려는 입장이므로, 이 기능을 지원하는 하드웨어를 찾는 것보다는 이 기능 자체를 소프트웨어로 제공하는 방법을 택하였다.

paletted texture를 지원하기 위해서는 glTexImage2D(…) 함수부터 시작해서 상당한 숫자의 텍스쳐 관련 함수들에 대해서 수정이 필요했고, 특히 shared texture palette를 지원하기 위해서는 그래픽스 컨텍스트(graphics context)에 별도의 색상표(color table)를 할당해 줄 필요도 있다. 각 색상표에는 인덱스 값 1개마다 (red, green, blue, alpha)의 4개 값이 저장되어야 하고, 이를 위해서 glColorTableEXT(…) 또는 glColorSubTableEXT(…) 함수들이 사용된다. 이에 따라, 전체 텍스쳐 처리 부분은 기존의 OpenGL 텍스쳐 처리 파이프라인을 확장하여, 그림 3에서와 같이 구현되었다.

텍스쳐를 정의하는 과정에서 internal format으로 RGBA를 선택하면, 기존의 OpenGL 에서 사용하는 RGBA 형태의 텍스쳐로 저장하게 되고, 이 형식에서는 텍스쳐 1픽셀당 red, green, blue, alpha 마다 각 1개 바이트씩, 총 4바이트가 할당된다. 이후의 처리는 이 4바이트에 저장된 텍스쳐 소스 컬러(texture source color)를 직접 사용한다.

반면, internal format이 COLOR_INDEX8_EXT

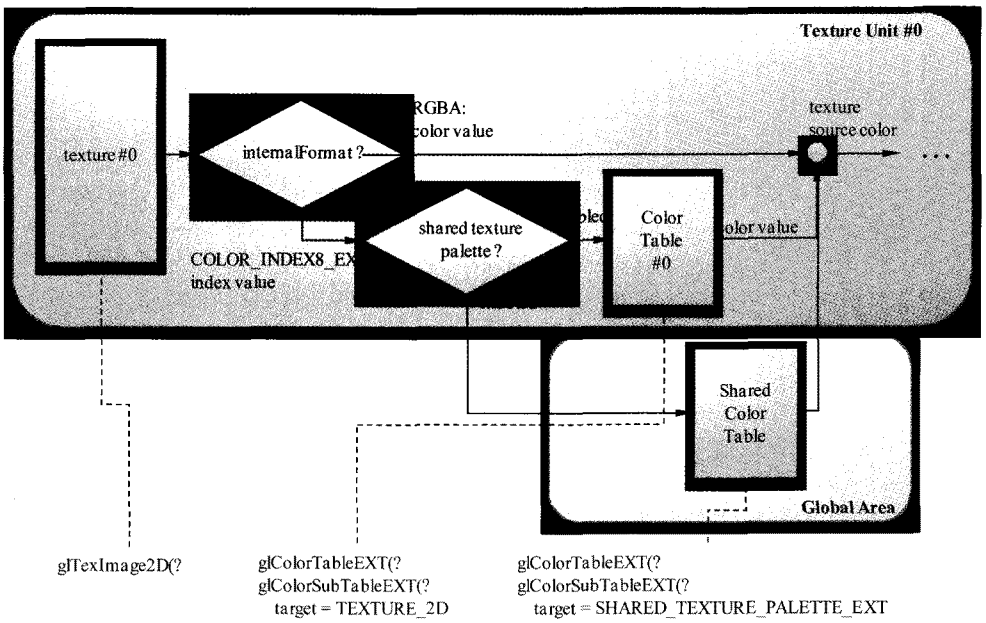


그림 3. 텍스처 파이프라인의 구현

로 설정된 경우는 텍스처 1픽셀당 1바이트의 컬러 인덱스(color index) 값이 저장되어 있고, 시스템에서는 미리 저장된 색상표에서 이 인덱스에 대응되는 RGBA 컬러 값을 가져와서, 텍스처가 실제로 필요로 하는 red, green, blue, alpha의 값, 총 4 바이트를 복원한다. 개념 상으로는 텍스처를 필요로 할 때마다 이런 복원 작업이 반복적으로 수행되어야 하지만, 우리는 처리 속도를 높이기 위해, 실제 필요로 하는 RGBA 형식의 텍스처를 복원한 후에는 별도의 메모리에 이를 저장해 두고 반복해서 사용하게 했다. 사용자가 텍스처 데이터를 새로 주거나, 색상표(color table) 정보를 새로 부여하는 경우에는 RGBA 형식의 텍스처에 변화가 있을 수 있으므로, 저장된 텍스처를 폐기하고, 복원 작업을 새로 수행하게 했다.

4. 구현 결과

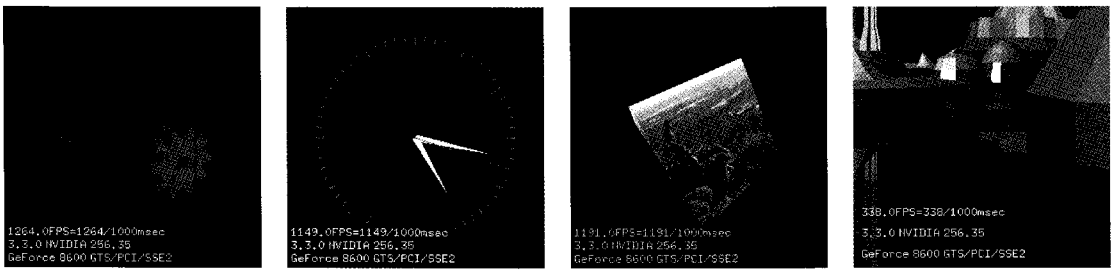
우리의 구현은 2단계로 나누어 진행되었는데, 처음의 구현 단계에서는 Linux 시스템에서 OpenGL 드라이버를 설치한 후, glut 라이브러리 [17]와 OpenGL 함수들을 사용하여 모든 함수들을 구현하였다. 이 시스템에서 최적화(optimization)와 디버그(debug) 작업이 대부분 수행되었고, 몇몇 OpenGL 칩들에 대해서 수행 결과를 분석하였다. 구현 결과가 OpenGL SC를 완전히 지원하는 지는 Khronos Group에서 제공하는 OpenGL SC CTS [18] 를 적용하여 분석하였고, 최종적으로 모든 테스트를 통과하였다.

다음 검증 단계에서는 비교적 저성능의 임베디드 시스템을 대상으로, OpenGL 1.2 코어 API와 multi-texture extension만 지원되는 하드웨어를 사용했다. 이 VxWORKS 기반의 시스템에서도 모든 OpenGL SC 프로그램들이 잘 작동함을 확인하였다.

표 1은 다양한 테스트 단계 중의 하나로서, 그림 4에서와 같은 예제 프로그램들에 대해서, 같은 그래픽스 출력을 내는 프로그램을 OpenGL 및 OpenGL SC 라이브러리를 사용하도록 각각 구현한 후에, 각각의 수행 속도를 측정한 예이다. OpenGL 프로그램

표 1. 테스트 프로그램의 실행 속도 측정
(단위: FPS, frame per second)

	OpenGL 프로그램	OpenGL SC emulator를 이용한 경우	비율 (b/a)	지연율
gears	1325.5	1301.8	98.21%	1.79%
clock	1178.6	1159.0	98.34%	1.66%
spin	1261.3	1239.0	98.23%	1.77%
angeles	339.4	332.6	97.99%	2.01%
average			98.27%	1.73%



(a) grears (b) clock (c) spin (d) angeles

그림 4. 테스트에 사용된 프로그램들

은 OpenGL 1.5 코어를 기준으로 작성되어, 사각형, 다각형 등의 출력이 가능하지만, OpenGL SC에서는 삼각형만 출력할 수 있고, 텍스처 처리에서는 paletted texture를 지원하기 위해서 소프트웨어로 구현한 별도의 텍스처 처리 파이프라인이 돌아가야 했다. 그렇지만, 전체 수행 속도는 표에서 보는 바와 같이, 2% 미만의 수행 속도 저하만을 가져왔다.

5. 결 론

본 논문에서는 OpenGL SC의 코어 기능들은 물론이고, 모든 extension들이 multi-texture extension이 지원되는 OpenGL 1.1 하드웨어 상에서 에뮬레이터 형태로 구현될 수 있음을 보였다. 구현 결과에서는 이런 방식의 구현이 하드웨어에서 지원되지 않는 paletted texture 기능들을 완전히 소프트웨어로 구현하였음에도 2% 미만의 속도 저하만을 가져왔음을 보여준다. 이는 우리의 초기 설계와 구현 방식이 효과적이었음을 보이는 결과이다. 구현 결과는 OpenGL SC 의 공식 인증 테스트(conformance test)를 비롯한 다양한 예제 프로그램들을 적합하게 수행하였다.

다음 단계의 연구로는 임베디드 시스템에서 사용되는 다른 하드웨어 칩들인, 멀티미디어프로세서(multi-media processor)들이나, DSP 칩 등을 바탕으로 하는 OpenGL SC 구현이 가능하다. 궁극적으로는 완전 소프트웨어 구현까지 가능할 것이다.

참 고 문 헌

[1] Khronos Group, Khronos group home page, *http://www.khronos.org/*.

[2] M. Segal and K. Akeley, *The OpenGL Graphics System: A Specification, version 4.1 (core profile)*, Khronos Group, 2010.

[3] A. Munshi and J. Leech, *OpenGL ES Common Profile Specification, version 2.0.24 (full specification)*, Khronos Group, 2009.

[4] B. Stockwell, *OpenGL SC: Safety-Critical Profile Specification, version 1.0.1 (difference specification)*, Khronos Group, 2009.

[5] P. Cole, "OpenGL ES SC - Open Standard Embedded Graphics API for Safety Critical Applications," 24th Digital Avionics Systems Conference, 2005.

[6] M. Snyder, "Solving the Embedded OpenGL Puzzle - Making Standards, Tools, and APIs Work Together in Highly Embedded and Safety Critical Environments," 24th Digital Avionics Systems Conference, 2005.

[7] M. Beeby, "Aviation Quality COTS Software: Reality or Folly," 21st Digital Avionics Systems Conference, 2002.

[8] ALT Software, ALT software home page, *http://www.altsoftware.com/*.

[9] Presagis Inc., Presagis home page, *http://www.presagis.com/*.

[10] Quantum3D, Quantum3D home page, *http://www.quantum3d.com/*.

[11] Vincent Pervasive Media Technologies, Vincent 3D Rendering Library, *http://www.vincent3d.com/*.

[12] S. Hill, M. Robart and E. Tanguy, "Implementing OpenGL ES 1.1 over OpenGL ES

2.0," Digest of Technical Papers, IEEE International Conference on Consumer Electronics, pp.1-2, 2008.

[13] H. Lee and N. Baek, "Implementing OpenGL ES on OpenGL," *Proc. of the 13th IEEE International Symposium on Consumer Electronics*, pp.999-1003, 2009.

[14] 이환용, 백낙훈, "임베디드 시스템을 위한 OpenVG 구현," 멀티미디어학회논문지, 12권, 3호, pp. 335-344, 2009.

[15] J. Leech, *The OpenGL Graphics System: A Specification, version 1.3*, OpenGL ARB, 2001.

[16] J. Leech, "Appendix F. ARB Extensions," *The OpenGL Graphics System: A Specification, version 1.2.1*, OpenGL ARB, 1999.

[17] M. Kilgard, *OpenGL Programming for the X Window System*, Addison-Wesley, 1996.

[18] Khronos Group, OpenGL Safety Critical Profile, *http://www.khronos.org/opengles/sc/*.

백 낙 훈

1990년 한국과학기술원 전산학과 졸업 (학사)
1992년 한국과학기술원 전산학과 (공학석사)
1997년 한국과학기술원 전산학과 (공학박사)

2004년~현재 경북대학교 IT대학 컴퓨터학부 교수
관심분야 : 모바일 그래픽스, 리얼타임 그래픽스