Journal of Korea Multimedia Society Vol. 12, No. 3, March 2009(pp. 335-344)

임베디드 시스템을 위한 OpenVG 구현

이환용[†], 백낙훈[††]

요 약

기존의 2차원 그래픽스 환경에서는 비트맵이나 래스터 위주의 연산들이 주가 되었지만, 최근에는 범위성(範圍性, scalability)을 지원하기 위해서, 임베디드 시스템과 웹 브라우저를 중심으로 2차원 스케일러블 벡터 그래픽스 기능(scalable vector graphics feature)을 제공하고 있다. 현재는 Flash, SVG 등이 활발히 사용되고 있으며, 이를 지원하기 위한 하위 라이브러리 표준으로는 크로노스 그룹(Khronos Group)의 OpenVG가 실질적 API 표준(de facto API standard)의 역할을 담당하고 있다. 이 논문에서는 OpenVG 표준의 구현 결과인 AlexVG의 설계 및 구현 과정, 최종 결과를 제시한다. AlexVG의 구현은 설계 당시부터 또다른 실질적 표준인 SVG-Tiny와의 연계를 염두에 두었고, 현재 OpenVG의 응용 프로그램들은 물론이고, SVG-Tiny 표준에 따른 미디어 파일들을 재생할 수 있는 능력을 제공한다. 제공하는 기능 면에서 본다면, AlexVG는 OpenVG 적합성 검사(conformance test)를 100% 통과하였으며, SVG-Tiny 적합성 검사의 그래픽스 관련 부분도 100% 통과하였다. 성능 면에서는 자원의 제한이 심한 휴대용 기기들과 임베디드 기기들에서의 효율성에 초점을 맞추었다. 그 결과로, 기존의 참조 구현(reference implementation)에 비하여 획기적인 속도 향상을 가져 왔으며, 특히 ARM 등의 저성능 CPU에서도 다른 라이브러리나 하드웨어 지원 없이 우수한 실행 속도를 보이고 있다.

Implementation of OpenVG on Embedded Systems

Hwanyong Lee[†], Nakhoon Baek[††]

ABSTRACT

Embedded systems and web browsers have started to provide two-dimensional vector graphics features, to finally support scalability of graphics outputs, while traditional graphics systems have focused on the raster and bitmap operations. Nowadays, SVG and Flash are actively used while OpenVG from Khronos group plays the role of a de facto low-level API standard to support them. In this paper, we represent the design and implementation process and the final results of an OpenVG implementation, AlexVG. From its design stage, our implementation aims at the cooperation with SVG-Tiny, another de facto standard for embedded systems. Currently, our overall system provides not only the OpenVG core features but also variety of OpenVG application programs and SVG-Tiny media file playing capabilities. For the conformance with the standard specifications, our system completely passed the whole OpenVG conformance test suites and the graphics output portions of the SVG-Tiny conformance test suites. From the performance point of view, we focused on the efficiency and effectiveness especially on the mobile phones and embedded devices with limited resources. As the result, it showed impressive benchmarks on the small-scale CPU's such as ARM's, even without neither any other libraries nor acceleration hardware.

Key words: OpenVG, SVG-Tiny, vector graphics(벡터 그래픽스)

※ 교신저자(Corresponding Author) : 백낙훈, 주소 : 대구시 북구 산격동(702-701), 전화 : 053)950-6379, FAX : 053)950-6369, E-mail : occancru@gmail.com
접수일 : 2008년 10월 7일, 완료일 : 2009년 1월 2일
[†] 정회원, (주)휴원 기술이사, 경북대학교 전자전기컴퓨터학부 박사과정
(E-mail : hylee@hul.com)
[††] 정회원, 경북대학교 전자전기컴퓨터학부 부교수
※ 본 연구는 지식경제부 지방기술혁신사업(RTI04-03-02) 지원으로 수행되었음.

1. 서 론

최근의 2차원 그래픽스 시스템들은 기존의 래스터 그래픽스 기능이나 이미지 처리 기능들에 추가하여, 스케일러블 벡터 그래픽스(scalable vector graphics) 기능에 초점을 맞추고 있다. 벡터 그래픽스는 확대나 축소 시에 계단 현상(jaggy effect) 등의 이미지 손상이 발생하지 않으며, 원본의 품질을 그대로 유지하기 때문에 디스플레이 장치의 크기나 해상도에 무관하게 동일한 품질을 제공할 수 있다는 장점을 가진다[1,2]. 이는 특히 그래픽스 출력을 다운사이징해서 사용하는 경우가 많은 소형 임베디드 시스템에 적합하다.

데스크탑 환경의 벡터 그래픽스 표현을 위해서는 Macromedia사의 Flash[3]와 Adobe사가 제안한 SVG(Scalable Vector Graphics)[4]가 주로 사용되고 있다. 특히 웹 환경에서는 SVG가 중시되어, 다양한 SVG 응용 프로그램들이 사용 가능하다. SVG는 원래 XML[5]의 2차원 그래픽스 출력이나 그래픽스 응용 프로그램들을 위한 언어에서 유래되었으므로, 웹 브라우저에서는 SVG의 내용을 직접 출력하는 형태가 적합하다. FireFox, Safari, Opera 등 대부분의 웹 브라우저들은 SVG 뷰어를 내장하고 있다[6,7]. 반면에 Internet Explorer는 SVG 형태의 내용을 보이기 위한 별도의 플러그-인(plug-in)을 필요로 한다. 현재는 Adobe사의 SVG 플러그-인[8]이 널리 알려져 있다.

컴퓨터 그래픽스의 상업적 응용들에 있어서는 임베디드 시스템에 대한 고려가 강조되고 있다. 최근 모바일 시스템의 성능이 향상되면서 GUI, MMS (multimedia message service), 내비게이션, SVG 플레이어와 같은 다양하고 고수준의 그래픽스 환경을 필요로 하는 응용 프로그램들이 많이 탑재되고 있다. 이러한 응용 프로그램들을 텍스트나 비트맵 형태로 처리하기에는 한계가 있고, 다양한 그래픽스 기능을 지원하기 위해서 벡터 그래픽스가 대두되고 있다. SVG의 경우에는 휴대폰이나 PDA등의 임베디드 시스템을 위한 SVG-Tiny[1]가 별도로 제공된다. SVG-Tiny는 제한된 자원을 가지는 장치들에 특화되어, 현재는 3세대 이후 휴대폰의 필수 요소가 되고 있다.

SVG-Tiny가 기존의 표준을 임베디드 시스템으로 다운사이징하려는 전략임에 비해, OpenVG는 처음부터 임베디드 시스템을 위한 표준으로 탄생하였다. 크로노스 그룹(Khronos Group)[9]은 로열티가 필요 없는 공개 API 표준들을 제정하려는 목표로 설립된 산업체 컨소시엄이다. 크로노스 그룹은 현재 임베디드 시스템을 위한 다양한 그래픽스 표준들을 제공하고 있다. 그 중에서 OpenVG[10,11]는 임베디드 시스템에서의 2차원 벡터 그래픽스 출력을 지원하기 위해 제정되었고, Flash나 SVG와 같은 벡터 그래픽스 라이브러리의 하위 레벨 인터페이스를 제공하는 크로스-플랫폼 API이다.

이 논문에서는 현재 상업적으로 서비스되고 있는 OpenVG 엔진인 AlexVG를 제시한다. AlexVG는 2005년 9월에 세계 최초의 상업적 OpenVG 구현으로 공표되었으며, 2006년 12월부터는 해외 판매되는 휴대폰 모델들에 사용되면서 시장에 진입했다[12]. AlexVG는 특히 OpenVG와 SVG-Tiny를 동시에 지원하도록 설계하여 비용 대비 효과적인(cost-effective) 구현에 성공했다.

2005년 8월에 OpenVG 1.0 공식 규격(official specification)이 발표된 이후, 몇몇 구현 사례들[13-16]이 나왔지만, 거의 모두가 OpenGL[17] 또는 OpenGL ES[18], Qt[19] 등의 다른 그래픽스 라이브러리를 기초로 구현되어 독립성이 떨어졌다. 반면, 우리의 구현은 하드웨어 레벨의 인터페이스 함수들까지 구현한 완전한 소프트웨어 구현이기 때문에, 현재의 어떠한 상업적 하드웨어에도 그대로 탑재가 가능하고, 다른 부대 비용이 드는 것도 아니라는 장점이 있다.

2장에서 OpenVG와 이에 연관된 표준들을 설명하겠다. AlexVG의 설계와 구현에 대한 설명은 3장에 제시한다. 4장에서는 구현 결과와 성능 평가가 나올 것이고, 마지막으로 5장에 결론을 제시하고 끝을 맺겠다.

2. 기존 연구 및 배경 지식

이 장에서는 OpenVG와 이에 연관된 API들인 SVG, SVG-Tiny, JSR 등을 설명한다. 또한, OpenVG의 구현 사례들을 보이고, 우리의 구현 결과와의 관계를 설명하겠다.

2.1 OpenVG와 그 구현 사례들

OpenVG는 2차원 벡터 그래픽스 시스템의 가속을

위한, 플랫폼 독립적인 공개 표준이다. OpenVG의 공식 규격에 명시된 대로, 이 표준은 PostScript, PDF, Flash, Java2D, SVG 등의 2차원 API와 유사한 역할을 담당한다. 크로노스 그룹에서는 OpenVG 개발자들에게 도움을 주고자, OpenVG 적합성 검사 도구(conformance test suites)[20]를 제공한다. 이 적합성 검사는 어떤 OpenVG 구현이 OpenVG의 공식 규격을 얼마나 만족시켰느냐를 측정하는 척도로 볼 수 있다. 제대로 된 구현이라면, 적합성 검사를 100% 통과해야 한다.

OpenVG 그룹은 2004년 6월부터 작업에 착수하여, 2005년 8월에 1.0 규격이 공식적으로 발표되었다. 이 때, Hybrid사에서 제공하는 참조 구현(reference implementation)[13]도 제공되었지만, 이의 목적은 규격을 만족시키는 구현이 가능하다는 것을 증명하는 데에 있었고, 실제 성능은 매우 떨어졌다. 상업적인 구현은 이 논문에서 제시하는 AlexVG가 2005년 9월에 공표된 것이 최초이다.

이후로는 2006년에 AmanithVG[14], 2007년에는 ShivaVG[15], QtOpenVG[16] 등이 각각 발표되었다. 이들은 모두 OpenGL, OpenGL ES, Qt 등의 다른 라이브러리를 기반으로 구현되었기 때문에 상업적인 사용에서는 문제가 될 수 있다. 또한, 이들 구현이 공식 규격을 얼마나 만족시켰는지와 성능이 어느 정도인지에 대한 자료가 제대로 발표되지 않은 상황이다. 2007년에 발표된 또다른 OpenVG 구현[21]에서는 OpenVG 적합성 검사에서 약 73%의 통과율을 보고하고 있다. 우리의 구현 결과가 100% 통과율을 보이는 것과 비교해 보면, 이러한 비상업적 구현들과의 단순 비교는 무리가 따른다. 2008년에 발표된 OpenVG를 가속하기 위한 하드웨어 구현[22]의 경우에도 적합성 검사에 대한 보고가 누락되어 있다. 이러한 다른 구현 사례들과의 비교는 4장에서 구현 결과를 보이면서 좀더 자세히 다루겠다.

2.2 SVG와 SVG-Tiny

1999년 WWW 컨소시엄은 SVG 또는 스케일러블 벡터 그래픽스라고 불리는 공개 포맷을 개발하기 시작하였다. WWW 컨소시엄이 2001년 9월에 발표한 SVG 1.0 규격[23]은 Macromedia사의 Flash에 비해 동등 이상의 그래픽 및 애니메이션 기능을 제공하고, 추가적인 장점들을 가지고 있다. SVG는 전체 기능을 지원하는 SVG[23]와 모바일 기기용으로 최적화된 SVG-Tiny[1]로 구분되어 표준화 작업이 진행되고 있으며, 현재 1.2 버전이 배포되어 있다. 모바일 표준 기구인 3GPP는 미국과 유럽에서 사용할 차세대 이동전화를 위한 플랫폼을 선정하면서 SVG-Tiny를 멀티미디어 메시징 서비스(MMS, Multimedia Messaging Service)에 필수적인 포맷으로 채택하였다[24].

OpenVG는 원래 SVG-Tiny 1.2 규격의 렌더러[1]에서 요구하는 모든 그래픽스 기능들을 제공하고자 하였다. SVG의 관점에서 본다면, OpenVG는 SVG-Tiny의 렌더링 함수들을 거의 대부분 제공하므로, OpenVG 구현 위에 SVG-Tiny를 구현하는 것은 상당히 용이해 보인다. 하지만, 이들을 면밀하게 비교해 보면, OpenVG는 SVG-Tiny의 기능 중에서 장면 그래프(scene graph), 파싱, 애니메이션, 링킹, 스크립팅 등을 제공하지 않으므로, 이들은 별도로 구현하여야 한다. 이를 감안하더라도, OpenVG는 SVG-Tiny의 대다수 기능을 제공하고 있으므로, 우리는 휴대폰과 같은 임베디드 시스템에 OpenVG 엔진을 구현함은 물론, 이에 기초하여 SVG-Tiny의 기능도 구현하고자 한다.

2.3 Java Specification Requests

SVG 외에도 몇몇 JSR(Java Specification Request)들이 OpenVG와 밀접한 관계를 가진다. 우선, JSR226[25]은 SVG-Tiny 1.1에 대한 Java 바인딩이다. 나중에는 SVG와 OpenVG에 관련된 JSR 프로젝트들도 시작되어, JSR271[26]에서는 MIDP (Mobile Information Device Profile)[27]을 지원하기 위해, SVG-Tiny 1.2에 기초한 2차원 벡터 그래픽스와 리치 미디어 기능을 정의하고 있다. JSR287[28]에서는 OpenVG와 호환되는 렌더링 API 함수들을 정의하였다. JSR271에서는 3세대 Mobile Information Device Profile (MIDP3)를 정의할 예정인데, 여기에 스케일러블 벡터 그래픽스 함수들이 포함될 것이다. 크로노스 그룹의 OpenVG 워크그룹은 Flash 7 기능과 폰트 렌더링을 제공하는 새로운 OpenVG 1.1 규격을 제정하여 2008년 12월에 발표하였다. 이에 따라, OpenVG에서는 JSR271, JSR287의 모든 렌더링 기능을 지원하고 Flash 7 기능 및 폰트 렌더링을 제공하게 되었다.

3. AlexVG의 구현

이 장에서는 AlexVG의 핵심 기능들의 구현에 대해 설명한 후, 이 핵심 기능 위에 별도의 레이어 형태로 구현된 SVG-Tiny 지원 기능들을 보이겠다.

3.1 구현 전략

2개 이상의 그래픽스 표준들이 유사한 기능을 제공할 때는 이들을 한꺼번에 구현하는 것이 효과적이다[29]. 이 때, 가능하면 설계 단계부터 이러한 가능성을 염두에 두고, 이를 반영하는 것이 좋고, 실제 구현에 있어서는 각 표준을 지키면서 서로가 원활하게 돌아가는 방법을 찾아야 하는 것이 관건이 된다. OpenVG와 SVG-Tiny의 경우에는 비교적 쉽게 서로의 관계를 정립할 수 있다. 즉, OpenVG 구현을 이미 가지고 있는 경우에는 이를 바탕으로 SVG-Tiny를 위한 응용 프로그램들을 구현하는 것이 자연스럽다. 이론적으로는 SVG-Tiny에서 요구하는 렌더링 기능의 거의 대부분이 OpenVG 렌더링 파이프라인으로 구현될 수 있다. 이 경우, OpenVG 파이프라인을 사용하는 방식이 더 작은 프로그램 크기로 더 뛰어난 성능을 보일 것이다.

3.2 AlexVG 엔진 구현

OpenVG의 구현은 전체 구조에 대한 분석에서 출발한다. 그림 1에서와 같이, OpenVG에는 내부적으로 하드웨어 장치의 바로 위에 EGL이라고 불리는 장치 제어 레이어(device controller layer)가 존재한다[30]. OpenVG 자체는 다시 코어, API(또는 언어 결합 language binding), 유틸리티 등으로 구성된다. OpenVG 시스템의 위에는 BREW, Java 등으로 작성된 응용 프로그램들이 위치할 수 있다. OpenVG에서는 장치 의존적인 부분들을 장치 제어 레이어인 EGL로 분리시켰으므로, OpenVG 엔진의 핵심 부분들은 다양한 플랫폼에서 그대로 사용될 수 있다.

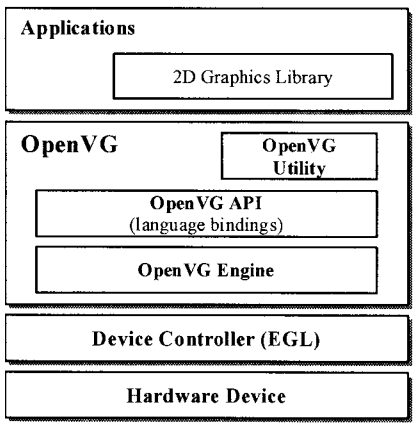


그림 1. OpenVG의 계층 구조

OpenVG의 실제 구현에서는 어떠한 구조를 사용해도 문제가 되지는 않지만, 공식 규격에서 규정한 결과가 그대로 나와야 한다. 이 때문에, OpenVG 구현은 OpenVG 공식 규격[11]에서 제시한 8단계 파이프라인을 채택하는 것이 일반적이다. 우리는 전체 설계 단계에서부터 안정적이면서도 효율적인 구현을 목표로 설정하여, AlexVG 구현은 공식 규격의 8단계 파이프라인 구조를 그대로 채택하되, 성능 향상을 위해 구조적인 변경을 하였다. 예를 들면, Stage 5의 클리핑을 Stage 4의 래스터 연산보다 먼저 수행함으로써 불필요한 영역에서의 래스터 연산 과정을 생략하도록 했으며, Stage 7의 이미지 보간은 Stage 6과 동시에 수행하여 Image Multiply, Stencil 기능의 성능을 향상시켰다.

OpenVG 파이프라인의 전체 구조는 그림 2와 같다. 이 그림에서는 공식 규격에서와 같이, 그래디언트(gradient) 효과를 사용해서 굵은 점선을 그리는 예를 보여주고 있다. 좀더 자세한 사항은 공식 규격[11]에 제시된 설명을 참고하기 바란다. 응용 프로그래머의 입장에서는 그림의 stage 1에서와 같이,

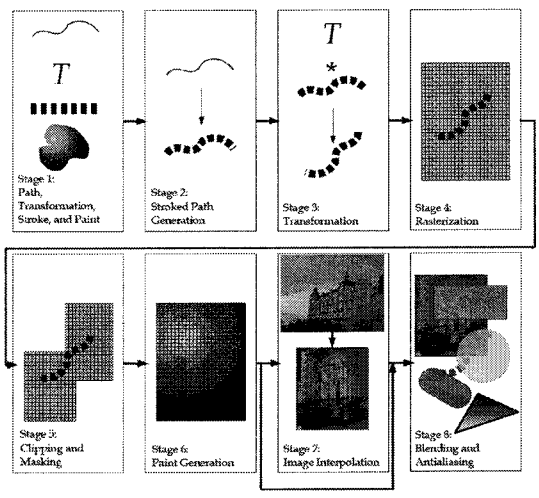


그림 2. OpenVG 파이프라인

OpenVG의 API 함수들을 사용해서 출력 경로와 드로잉 매개변수들을 설정하게 된다. 나중에 vgDrawPath(…), vgDrawImage(…) 등의 출력 함수를 호출하면 stage 2에서 7까지의 과정을 거쳐, 설정된 값들에 대응되는 비트맵 형태의 이미지가 만들어진다. 마지막 단계인 stage 8에서는 이 새로 만들어진 이미지가 기존의 이미지와 합성되어, 새로운 화면 출력이 만들어진다. 각 오브젝트 별로 이런 단계를 반복함으로써, 최종 이미지가 만들어진다.

AlexVG는 OpenVG 파이프라인 구조를 바탕으로, C 언어를 사용하여 2차원 벡터 그래픽스 엔진의 형태로 개발되었다. 이 벡터 그래픽스 엔진은 stage 1에서 3에서 처리해야 하는 경로(path)와 오브젝트 정보들에 대해서는 오브젝트 단위의 처리를 수행하고, 그 이후의 stage 4에서 8까지에서는 프래그먼트(fragment) 단위의 처리를 수행한다.

그래픽스 엔진에서 필수적인 실수 연산 과정에서는 효율적인 처리를 위해서 Q16.16 형태의 고정 소수점 표현(fixed-point representation)을 사용하였다[31,32]. 아직까지는 임베디드 시스템들에서 사용하는 대부분의 CPU 코어들이 정수 연산을 사용하여 실수 연산을 소프트웨어로 시뮬레이트하고 있는 상황이다. 부동 소수점 연산 대신 고정 소수점 연산을 도입하면 특히 이러한 소형 CPU들에서 상당한 성능 향상을 가져올 수 있다. 고정 소수점 연산에 대해서는 참고 문헌 [31]과 [32]을 참고하기 바란다.

AlexVG 엔진의 개발 과정에서 우리는 다음과 같은 특징적인 구현을 하였다.

• 효율적인 고정 소수점 연산: 고정소수점 연산에 있어서 정밀도를 높이기 위해서는 좀더 정밀한 계산을 해야 하므로 연산속도가 느려진다. 이를 해결하기 위해 필요한 부분에 필요한 정밀도만을 제공하는 방식으로 계산을 효율성을 높였다. 예를 들어 일반적인 좌표 계산에서는 곱하기 연산에서 오버플로우가 발생할 수 있기 때문에, 64비트로 확장된 곱셈을 수행해야 하지만, 컬러 연산에서는 사용되는 값의 제한으로 오버플로우가 발생하지 않으므로 32비트 곱셈만으로 충분한 정밀도를 유지할 수 있다.

• 라인 단위의 렌더링 엔진: 프래그먼트 연산을 픽셀 단위가 아니라 라인 단위로 수행하는 방식을 통해 불필요한 메모리 참조를 제거하고 데이터의 캐쉬 히트 비율을 높일 수 있었다. 임베디드 시스템의 그래픽 엔진 성능은 데이터를 가져오는 연산의 성능에 따라 상당히 크게 좌우된다. 라인 단위의 알고리즘은 데이터의 로컬리티를 높여 주어 캐쉬 히트 비율을 상당히 높였다.

• 임베디드 시스템을 위한 최적화: PC상에서의 개발과 임베디드 시스템상에서의 개발은 많은 부분에서 차이가 있다. 예를 들어 임베디드 시스템에서는 함수 호출에 대한 부하가 크지 않은 반면, 나눗셈과 곱셈의 효율이 급격히 떨어진다. 이러한 특성을 고려하여 임베디드 시스템에 최적화된 구현을 하였다.

• 응용 프로그램과 동시 구현: 그래픽 API 와 동시에 SVG 응용 프로그램을 개발하는 방식으로 구현함으로써 발생할 수 있는 문제점과 성능 특성을 미리 검토할 수 있도록 하였다.

위와 같은 구현의 결과로 최종 결과물에서는 다음과 같은 장점들을 가지게 되었다.

• 경제적인 구현: AlexVG 엔진과 이에 결합되어 사용되는 VGU(OpenVG Utility library)는 OpenVG의 원래 기능은 물론이고, SVG-Tiny에서 제공하는 모든 그래픽스 기능들을 제공한다. 이렇게 OpenVG와 SVG-Tiny를 동시에 구현하는 방식은 OpenVG, SVG는 물론이고, 가까운 시일 내에 이에 연관된 JSR들까지 지원할 수 있는 경제적인 구현 방법이 될 것이다.

• 이식성(portability)과 구성성(configurability)이 좋은 시스템: 일관된 모듈 구조를 채택하여, 시스템 의존적인 코드들은 특정 모듈에만 국한시켰다. 특정 하드웨어에 기반한 형태가 아니기 때문에, AlexVG 엔진은 휴대폰, PDA. PMP, 셋탑 박스 등의 다양한 임베디드 시스템에 쉽게 이식할 수 있고, BREW나 WIPI 등의 기존 소프트웨어 플랫폼과 결합시키기도 용이하다. 하드웨어 성능에 좌우되는 부분들이 몇 개 모듈에 집중된 구조이기 때문에, 임베디드 시스템에서의 처리에 있어, 품질과 처리 속도 사이에서 적당한 성능을 보이도록 튜닝하는 작업도 비교적 용이하게 수행할 수 있다.

• 가속성이 우수한 시스템: OpenVG에서 요구하는 기하 처리 알고리즘들은 상당히 복잡하면서도 기존의 그래픽스 구현 방식과는 다른 임베디드 시스템을 위한 구현 및 최적화를 수행하였으며 이를 통해 전반적인 성능 향상을 가져왔다.

(a) anti-aliased map and navigation (b) tiger: path stroke and fill (c) outline fonts (d) stroke parameters

(e) image effects (f) sample animation (g) drawings

그림 3. OpenVG 프로그램의 실행 화면들

전체 구현 과정 내내, 엔진 자체는 물론이고 다양한 데모 프로그램들이 ARM9, XScale, XScale MMX 등의 다양한 하드웨어 플랫폼과 Linux, Windows CE, Nucleus 등의 운영 체제 조합들에 대해서 실행되어졌다. Windows 2000, Windows XP 기반의 PC 환경을 위해서는 고정 소수점 연산 대신 부동 소수점 연산을 지원하는 변형된 버전도 사용가능하다. 그림 3은 현재 사용 가능한 프로그램들 중의 일부에 대한 실행 화면들이다.

3.3 SVG-Tiny 플레이어의 개발

채택한 구현 전략이 제대로 구현되었는지를 검사하기 위해서, AlexVG의 구현 결과를 이용하는 SVG-Tiny 플레이어로 AlexVG 플레이어 B110을 개발하였다. 그림 4에서 보는 바와 같이, 이 응용 프로그램은 SVG의 파싱에서 출발하여, 해석된 SVG 구성요소 트리 구조를 순회하면서 대응되는 속성을 사용하여 화면 출력을 수행한다. 즉, 출력될 도형의 외형이나 이미지에 관련된 구성요소들은 그래픽스 출력을 위해, OpenVG의 vgSet(…) 함수들을 사용해서 필요한 속성들을 설정하고, OpenVG의 vgDrawPath(…)

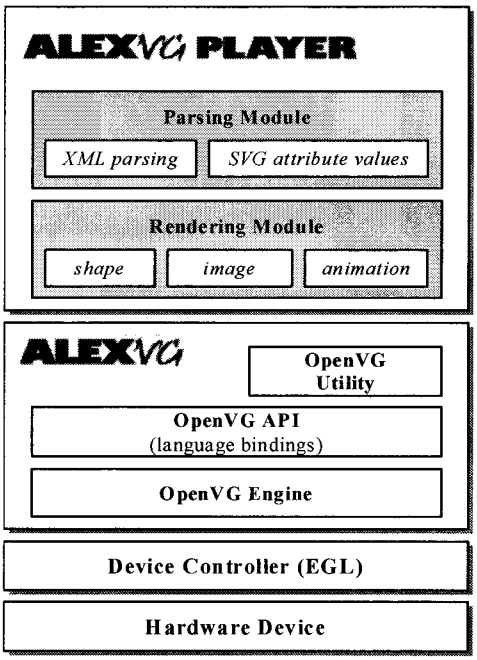
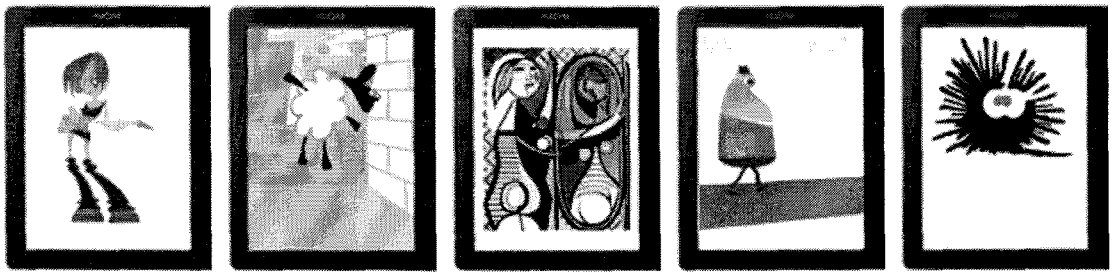


그림 4. AlexVG 플레이어의 계층 구조

나 vgDrawImage(…) 함수를 사용해서 화면 출력이 이루어진다.



(a) guitar.svg (b) lamb.svg (c) picasso.svg (d) retro.svg (e) spud.svg

그림 5. SVG-Tiny 예제 문서들

AlexVG 플레이어 B110은 현재 구현이 완료되어 다양한 플랫폼 상에서 상업적으로 서비스되고 있다. 그림 5는 AlexVG 플레이어 B110의 기능 검사에 사용되었던 다양한 SVG 문서들 중의 일부이다. 이 응용 프로그램은 OpenVG 구현 상에서 SVG-Tiny 응용 프로그램을 구현하는 좋은 예제가 될 것이다.

4. 구현 결과

구현 결과를 분석하기 위해서는 크게 정확도(correctness)와 성능(performance) 측면을 볼 수 있을 것이다. 구현이 주어진 규격을 얼마나 준수했는지를 보아야 할 것이고, 또, 실행 속도가 얼마나 빠른지도 측정해야 할 것이다. 정확도 측면에서는 OpenVG 적합성 검사(conformance test)가 하나의 척도가 될 것이다. 이것은 어떤 OpenVG 구현이 OpenVG 공식 규격의 요구 사항을 얼마나 정확하게 구현했는지를 검사하는 프로그램들의 집합으로 구성되어 있다. AlexVG는 현재 적합성 검사를 100% 통과했으며, 이를 통하여 공식 규격의 준수를 인증받았다. 반면에, 다른 OpenVG 구현들이 적합성 검사를 얼마나 통과하는 지에 대해서는 이렇다할 자료가 없는 상황이고, 유일하게 한 구현 사례[21]에서 약 73%를 통과했다고 주장하고 있다. OpenVG를 사용해야 하는 개발자의 입장에서는 적합성 검사 통과 여부가 어느 구현을 사용해야 할지에 대한 신뢰할 만한 척도가 될 수 있을 것이다.

다음으로는 구현된 결과가 어느 정도의 성능을 보이는 지를 측정해야 할 것이다. 문제가 되는 것은 다른 OpenVG 구현들이 OpenGL, OpenGL ES, Qt 등의 별도의 그래픽스 라이브러리를 필요로 하기 때문에, 특정 하드웨어나 특정 운영 체제에 대해서만 작동하는 경우가 많아, 제대로 된 비교가 힘들다는 것이다. 크로노스 그룹에서 공식적으로 제공하는 참조 구현(reference implementation)은 소스 코드(source code)가 공개되어 있고, 윈도우 환경에서 동작되므로, 비교적 수월하게 그 성능을 측정할 수 있다. 그래서, 우리는 AlexVG와 공식 참조 구현을 모두 WindowsXP에서 컴파일하여 실행 시간을 비교하였다. Intel Core2 6600 2.4GHz CPU와 2MB 램을 가진 PC에서 2개의 응용 프로그램을 대상으로 한 결과가 표 1에 제시되어 있다. tiger 예제는 PostScript용의 상당히 복잡한 예제를 OpenVG 응용 프로그램으로 전환한 것이고, calder의 경우는 회화 작품을 벡터 그래픽스 형태로 바꾸어 화면에 출력한 것이다. 실험 결과에서 보듯이, AlexVG의 성능은 공식 참조 구현에 비해서 월등한 우위를 보였다. 공식 참조 구현이 단순히 규격을 만족시키는 구현이 가능하다는

표 1. 예제 프로그램의 성능 평가

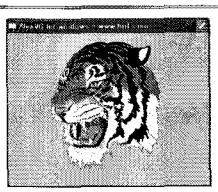
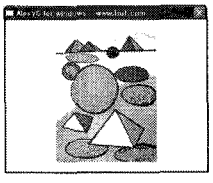
	렌더링 소요 시간	
	AlexVG	참조 구현
 tiger	15 msec	1,843 msec
 calder	< 1 msec	172 msec

표 2. SVG-Tiny 파일의 실행 결과

미디어 파일	크기	frames per second
guitar.svg	18.6 KB	30.8
lamb.svg	38.1 KB	16.0
picasso.svg	109.0 KB	5.6
retro.svg	14.7 KB	44.4
spud.svg	5.4 KB	100.0

것을 목표로 하였기 때문에, 성능 면에서는 상당히 떨어질 수 밖에 없음을 보여주는 예가 될 것이다.

SVG-Tiny 플레이어인 AlexVG 플레이어 B110에서의 성능 측정을 위해서는 그림 5에서와 같은 SVG file들이 사용되었고, 이들을 ARM9 216MHz CPU에서 240x320 해상도의 컬러 LCD 화면에 최종 출력하기까지의 시간을 측정하였다. 표 2에서 보는 바와 같이, 실제 응용에서 흔히 사용되는 SVG 파일들에 대해서 꽤 높은 프레임 수(frame rate)를 보여주고 있으며, 이는 AlexVG 플레이어 B110이 휴대폰 환경에서도 SVG-Tiny 플레이어로서의 역할을 충분히 수행할 수 있음을 보여 준다.

5. 결 론

이 논문에서는 OpenVG의 상업적 구현인 AlexVG를 제시하였다. AlexVG는 2차원 그래픽스 환경을 제공하며, 실질적 표준인 OpenVG와 SVG-Tiny의 전체 기능을 제공한다. 이전까지의 OpenVG 구현 사례들이 OpenGL나 OpenGL ES와 같은 3차원 그래픽스 라이브러리를 기반으로 개발되었던 것에 비해, AlexVG는 제한된 자원을 가지는 휴대용 임베디드 시스템에서의 효율적이고 비용 대비 효과가 큰 구현에 초점을 맞추어서, 완전히 독립적인 그래픽스 엔진으로 개발되었다. AlexVG 엔진은 2005년부터 상업적 제품으로 제공되고 있다[12].

AlexVG 시스템은 OpenVG의 전체 기능에 대한 완전 독립적인 구현이라는 장점을 가진다. 기존의 OpenVG 구현 사례인 AmanithVG 나 Hybrid OpenVG 등의 거의 모든 OpenVG 구현들이 OpenGL, GLUT, Qt 등의 그래픽스 라이브러리나 하드웨어를 기반으로 했던 데에 비해, AlexVG는 그래픽스 버퍼를 다루는 하드웨어 제어 함수부터 구현한, 완전히 독립적인 구현이다.

또한, 임베디드 시스템에 최적화된 구현을 제공하여, AlexVG 엔진은 전체 코드 사이즈가 120KB 미만이며 요구되는 Static RAM도 QVGA 기준으로 30KB에 불과하다. 이는 제한된 자원만을 가지는 소형 임베디드 시스템에서 상당히 큰 장점으로 작용하여, 중저가 임베디드 단말기에서도 다양한 벡터 그래픽스 기능을 제공할 수 있다.

본 논문의 구현 결과는 AlexVG는 OpenVG 적합성 검사를 100% 통과했으며, 2006년부터 상업적으로 판매되는 휴대폰들에 일부 탑재되어 시장에서 판매되고 있다. 이는 AlexVG가 이미 안정화 단계에 접어 들었고, 상업적으로 검증된 시스템임을 보여준다. 또한 비용 대비 효과적인 구현을 목표로 하여 AlexVG 위에 SVG-Tiny 지원 모듈을 추가함으로써 SVG-Tiny 기능들을 효과적으로 지원한다. 이들 기능은 SVG-Tiny 적합성 검사를 통과하였으며, 이를 통하여 SVG-Tiny 플레이어와 같은 상업적인 응용 소프트웨어들을 이미 제공하고 있다.

최근 모바일폰의 그래픽 사용자 인터페이스의 중요성의 증대되고, Full Browsing 기능 및 지도 서비스를 포함한 위치기반 서비스에 대한 수요 증대는 OpenVG와 같은 표준 가속 플랫폼의 중요성을 더욱 증대시켜 주고 있다.

현재 다양한 표준과 제품군들이 OpenVG와 SVG-Tiny를 그래픽 기본 엔진으로 사용하려는 추세이다. 예를 들면 Adobe사의 Flash Lite Player, Microsoft사의 Silverlight 등과 같은 리치 미디어 플레이어는 OpenVG 상에서 수행될 수 있도록 하고 있으며, Qualcomm사의 uiOne, Acrodea 사의 vividUI와 같은 GUI 플랫폼은 SVGTiny를 기반으로 하고 있다. 또한 Linux 환경의 대표적인 벡터 미들웨어인 Cairo도 OpenVG를 지원하며, MPEG4-Part20 LASeR 표준 역시 SVG 기반으로 되어 있다. 이와 같이 OpenVG와 SVG는 임베디드 시스템의 그래픽 응용 및 서비스의 중심역할을 할 것이다.

단순한 하드웨어 혹은 소프트웨어만을 통한 구현 뿐만 아니라, 다양한 가속 하드웨어들인 멀티미디어 칩, 3차원 그래픽스 가속 칩, 영상처리 가속 칩, DSP 하드웨어 등의 고성능 기능을 이용하여 OpenVG 파이프라인의 일부를 가속할 수 있다. 이를 통해 좀 더 다양한 플랫폼에서 높은 성능의 OpenVG를 사용하

는 것이 가능해 질 것이다. 다음 단계에서는 OpenGL ES 하드웨어, 멀티미디어 하드웨어에서 OpenVG를 가속하는 제품들을 개발하고 있으며, OpenVG 1.1 제품이 완성 단계에 있다.

현재 SVG, JSR226, JSR271, JSR287 등의 OpenVG에 직접 관련된 모든 표준들을 하나로 통합하려는 추세이므로, AlexVG 역시 이런 추세에 맞추어 나가려 한다. 장기적으로 본다면 AlexVG는 다양한 2차원 그래픽스 표준들과 그 응용 프로그램들을 지원하는 통합된 프로그래밍 환경이 될 것이다[33].

참 고 문 헌

[1] W3C, *Scalable Vector Graphics(SVG) Tiny 1.2 Specification*, W3C Recommendation, 2006.

[2] T. Evans, *Introducing Macromedia Flash Lite 1.1*, Macromedia Developer Center, 2004.

[3] Macromedia Inc., *Macromedia flash developer center*, http://www.adobe.com/devnet/flash.

[4] W3C, *Scalable vector graphics*, http://www.w3.org/Graphics/SVG/.

[5] W3C, *Extensible Markup Language*, http://www.w3.org/XML/.

[6] Mozilla Developer Center, *SVG in FireFox*, http://developer.mozilla.org/en/docs/SVG_in_Firefox.

[7] Opera Software, *SVG in Opera*, http://www.opera.com/products/desktop/svg/.

[8] Adove Systems Inc., *SVG zone*, http://www.adobe.com/svg/.

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

[10] Khronos Group, *OpenVG home page*, http://www.khronos.org/openvg/.

[11] Daniel Rice, *OpenVG Specification*, version 1.0.1, Khronos Group, 2005.

[12] Huone Inc., *Huone home page*, http://www.hu1.com/.

[13] Hybrid Graphics Ltd., *OpenVG reference implementation*, http://www.hybrid.fi/main/extra/openvg_ri.php.

[14] Mazatech, *AmanithVG home page*, http://www.amanithvg.com/.

[15] Pixel Infinity, *ShivaVG: open-source ANSI C OpenVG*, http://ivanleben.blogspot.com/ 2007/07/shivavg-open-source-ansi-c-openvg.html.

[16] Z. Rusin, *QtOpenVG*, http://zrusin.blogspot.com/2007/01/openvg_116899762231694078.html.

[17] M. Segal and K. Akeley, *The OpenGL Graphics System: A Specification*, version 2.1, Silicon Graphics, 2006.

[18] D. Blythe, A Munshi, and J. Leech, *OpenGL ES Common/Common-Lite Profile Specification*, version 1.1, Khronos Group, 2007.

[19] Trolltech, *Qt cross-platform application framework*, http://trolltech.com/products/qt/.

[20] Khronos Group, *OpenVG Conformance Test Process*, OpenVG Workgroup, 2005.

[21] S.-Y. Lee, S. Kim, J. Chung, and B.-U. Choi, "Salable vector graphics (openVG) for creating animation image in embedded systems," *Knowledge-Based Intelligent Inform. and Eng. Sys., 11th Int'l Conf.(KES 2007)*, 2007.

[22] D. Kim, K. Cha, and S. Chae, "A high-performance OpenVG accelerator with dual-scanline filling rendering", *IEEE Trans. on Consumer Electronics*, Vol.54, No.3, pp. 1301-1311, 2008.

[23] W3C, *Scalable Vector Graphics(SVG) Full 1.2 Specification*, W3C Recommendation, 2006.

[24] The 3rd Generation Partnership Project, *3GPP home page*, http://www.3gpp.org/.

[25] Java Community Process, *JSR226: Scaleable 2D Vector Graphics API for J2ME*, Java Community Process, 2005.

[26] Java Community Process, *JSR271: Mobile Information Device Profile 3*, Java Community Process, 2006.

[27] Sun Microsystems Inc., *Mobile information device profile(MIDP)*, http://java.sun.com/

products/midp/.

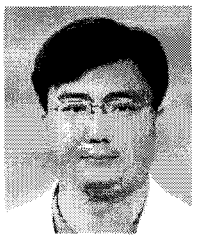
[28] Java Community Process, *JSR287: Scaleable 2D Vector Graphics API 2.0 for Java ME*, Java Community Process, 2006.

[29] N. Baek and S. R. Maeng, "A unified reference model for CGM/CGI and its implementation," *Computer Standards & Interfaces*, Vol.18, No.5, pp. 451-467, 1997.

[30] Khronos Group, *Native Platform Graphics Interface 1.2*, Khronos Group, 2005.

[31] ARM, *Fixed Point Arithmetic on the ARM*, Application Note 33, ARM, 1996.

[32] D. Hough, "Applications of the proposed IEEE-754 standard for floating point arithmetic," *IEEE Computer*, Vol.14, No.3, pp. 70-74, 1981.



이 환 용

1990년 KAIST 전산학과 학사
1992년 POSTECH 대학원 컴퓨터공학과 석사
1995년 POSTECH 대학원 컴퓨터공학과 박사과정 수료
POSTECH 정보통신연구소 연구원
현재 (주)휴원 기술이사, 경북대학교 전자전기컴퓨터학부 박사과정
관심분야 : 임베디드소프트웨어, 그래픽스, 가상현실, 그래픽사용자인터페이스

백 낙 훈

1990년 KAIST 전산학과 학사
1992년 KAIST 전산학과 석사
1997년 KAIST 전산학과 박사
현재 경북대학교 전자전기컴퓨터학부 부교수
관심분야: 컴퓨터 그래픽스, 임베디드 소프트웨어