임베디드 시스템용 OpenVG 참조 구현

이상윤[o1], 이경희[1], 김성환[2], 정지훈[2], 최병욱[3]

[1]한국전자통신연구원 임베디드SW연구단

{syllee, kyunghee}@etri.re.kr

[2](주)파인원

{babynewton, jihoonc}@pineone.com

[3]한양대학교 정보통신대학교 정보통신학부

buchoi@hanyang.ac.kr

Reference Implementation of OpenVG for Embedded System

Sang-Yun Lee[o1], Kyung-Hee Lee[o1,] Sung-Hwan Kim[2], Ji-Hoon Chung[2], and Byung-Uk Choi[3]

[O1]Electronics and Telecommunications Research Institute, Embedded S/W Research Division

[2]Pineone Communications Co., Ltd., R&D Division

{babynewton, jihoonc}@pineone.com

[3]Division of Information and Communications, Hanyang University

buchoi@hanyang.ac.kr

요 약

본 논문에서는 크로노스 그룹에서 제정한 스케일러블 벡터 그래픽 하드웨어 가속을 위한 표준인 OpenVG를 소프트웨어 렌더링 방식으로 구현한 참조 구현을 제안한다. EGL과 OpenVG 엔진이 다양한 임베디드 환경에 쉽게 이식이 가능하도록 설계한 방식을 제시한다. 또한 성능 개선을 위해, 채택한 수학 함수와 알고리즘의 선택 배경을 기술하고 최적의 렌더링 방법을 제안한다. 소프트웨어 렌더링 방법으로 구현한 OpenVG를 통해 벡터 이미지를 화면에 출력하는 모습을 보인다. 또한, 호환성 테스트 툴인 CTS의 테스트 결과를 제시하며, 기존 참조 구현인 Hybrid 사의 참조 구현과 성능 비교 실험 결과를 보인다.

1. 서 론

최근에, 벡터 그래픽을 이용한 응용에 대한 요구가 늘어나고 있다[1]. 특히, SVG 뷰어, 휴대용 지도 서비스, E-Book 리더, 게임, 스케일러블 사용자 인터페이스 등에서 벡터 그래픽 기술이 널리 활용되고 있다[2].

OpenVG는 Flash, SVG와 같은 벡터 그래픽 라이브러리를 위한 저수준 하드웨어 가속 인터페이스를 제공하는 로열티 무료, 크로스 플랫폼 API이다. OpenVG는 작은 스크린 디바이스 상에서 경쟁력 있는 사용자 인터페이스와 텍스트를 제공하기 위해 고품질 벡터 그래픽이 이식 가능한 가속을 요구하는 핸드헬드 디바이스를 주로 타깃으로 하고 있다. 동시에, OpenVG는 극도의 저전력 수준에서 유동적으로 상호 작용 성능을 제공하기 위해 하드웨어 가속을 가능케 한다. OpenVG는 크로노스 그룹에 의해 제정된 표준이며, 2005년 7월에 버전 1.0 이 최초로 공개되었다[3].

OpenVG는 하드웨어 가속을 통해 성능이 극대화 될 수 있도록 제안된 규격이다. 하지만, 규격을 검증하거나, 에뮬레이터를 통해 OpenVG 응용 프로그램을 미리 동작해 보거나, OpenVG가 지원되는 하드웨어가 없을 때는, 소프트웨어 렌더링 방식으로 동작하는 참조 구현이 필요하다.

또한, OpenVG를 지원하는 전용 하드웨어가 생산되기까지는 시간이 많이 걸리고, 비용측면에서도 소프트웨어 렌더링 방식이 훨씬 절감된다. 뿐만 아니라, 향후 CPU와 임베디드 기기의 성능 향상에 따라, 소프트웨어 렌더링으로 대체될 가능성도 높다. OpenVG 전용 하드웨어를 이용하려면, 시스템마다 이를 설치해야 되기 때문에, 추가적인 비용이 발생해 단말기의 단가가 높아지는 문제가 있고, 이에 따라 확산이 느릴 수 밖에 없다.

본 논문에서는 소프트웨어 렌더링 방법을 이용해서 다양한 임베디드 기기에 쉽게 이식될 수 있는 OpenVG 참조 구현을 제안한다. 또한, 실험을 통해 제안된 참조 구현이 기존의 참조 구현보다 성능 측면에서 우수함을 보인다.

2. OpenVG와 EGL 엔진 설계

2.1 시스템 아키텍처

본 논문에서 제안하는 OpenVG 참조 구현에 대한 시

스템 구조가 그림 1에 나타나 있다.

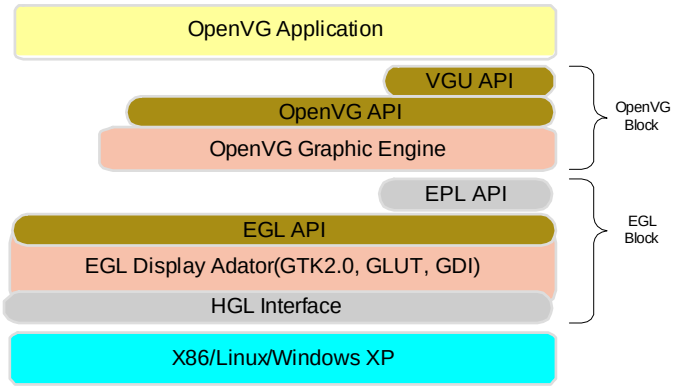


그림 1. 시스템 구조

OpenVG 참조 구현은 EGL(Embedded Graphics Library) 블록과 OpenVG 블록으로 구성되어 있다.

EGL은 OpenGL|ES 혹은 OpenVG 와 같은 렌더링 API(통칭하여 클라이언트 API라고 부름)와 기반이 되는 네이티브 플랫폼 윈도우 시스템 사이의 인터페이스이다. EGL은 클라이언트 API들이 그리는 렌더링 표면을 생성하기 위한 메커니즘을 제공한다[4].

본 논문에서는 EGL Display Adapter를 통해, 클라이언트 API가 네이티브 플랫폼 윈도우 시스템에 접근할 수 있도록 설계하였다. EPL(Embedded Platform Library) API는 EGL 규격에는 포함되어 있지 않지만, 클라이언트 API를 구현하기 위해서 필요하고 시스템마다 이식되어야 할 API 들이다. 예를 들어, Surface로부터 프레임 버퍼를 반환하는 기능, 메모리 할당과 해제 기능 등이 이에 속한다.

HGL(Hardware Graphic Library)인터페이스는 EGL을 네이티브 그래픽 시스템에 연결하는 기능을 수행한다. EGL은 그 자체가 하드웨어 시스템을 추상화하기 위해 만들어진 규격이지만, 최소한의 작업을 통해 여러 네이티브 플랫폼 윈도우 시스템에 EGL을 이식하기 위해서는 HGL과 같은 별도의 포팅 레이어가 필요하다.

OpenVG API는 2D 벡터 그래픽 라이브러리이고 VGU API는 고수준의 2D 벡터 그래픽 유틸리티 API이다. OpenVG 그래픽 엔진은 OpenVG API와 VGU API가 필요로 하는 핵심 기능을 제공한다.

2.2 EGL 엔진의 구조

EGL 엔진을 구성하는 블록 구조가 그림 2에 나타나 있다. Display Manager는 그래픽 출력을 담당하는 객체인 Display를 생성하고 관리한다. State Manager는 함수 수행시 발생하는 오류 값을 저장한다. Thread Manager는 여러 개의 프로세스 또는 쓰레드가 동시에 EGL을 사용할 경우 발생할 수 있는 경쟁 상황을 회피하기 위한 기능을 제공한다. 이 세 객체는 단 하나만 생성될 수 있는 객체이다. Manager Factory는 이 세 객체들이 유일성을 갖도록 한다.

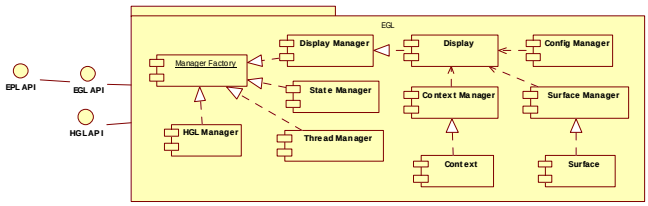


그림 2. EGL 엔진의 구조

2.3 OpenVG 엔진의 구조

OpenVG 엔진을 구성하는 블록 구조가 그림 3에 나타나 있다.

OpenVG 엔진은 응용이 사용하기 위한 OpenVG API와 EGL이 사용하기 위한 Context Sync API를 제공한다. Context Sync API는 EGL이 생성한 Context와 OpenVG 내부에서 생성되는 VGContext 간의 동기화를 제공한다. 동기화가 필요한 요소에는 컨텍스트의 생성, 소멸, 현재의 설정 상태 등이 있다.

VG State Manager는 OpenVG의 수행시 발생하는 오류값을 저장 및 관리한다. VG Context Manager, Paint Manager, Image Manager, Path Manager는 각각 VG Context, Paint 객체, Image 객체, Path 객체를 생성하고 관리한다. VG Manager Factory는 이들 객체들이 singleton 동작을 할 수 있도록 보장하는 역할을 수행한다. Rasterizer는 궤적, Image, Paint 정보를 조합하여 EGL의 surface가 제공하는 프레임 버퍼에 그리는 기능을 수행한다.

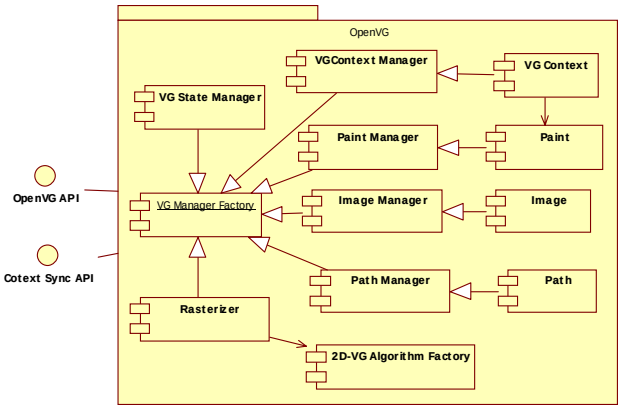


그림 3. OpenVG 엔진의 구조

2.4 OpenVG/EGL 엔진 설계 요구 사항

OpenVG는 데스크탑 PC 뿐만 아니라, 서버에서도 동작할 수 있으나, 주로 임베디드 기기에서 활용될 목적으로 개발되었다. 임베디드 환경은 하드웨어뿐만 아니라 플랫폼, 소프트웨어 등이 다양하기 때문에 항상 포팅 문제가 생긴다. 따라서, 한번 개발한 OpenVG를 다양한 임베디드 기기에 쉽게 탑재되도록 하려면, 아키텍처 설계에서부터 포팅을 고려하여야 한다. 즉, 환경 변화에 따라 수정할 부분이 최소화 될 수 있도록 포팅 레이어가 존재

하여야 한다. EGL에 기반하여 동작하는 OpenVG는 EGL의 수정 또는 확장에 따라 OpenVG를 수정하는 일이 최소화 될 수 있도록 두 모듈이 약하게 결합되어 있어야 한다.

임베디드 환경은 일반적으로 CPU의 성능이 떨어지고 메모리가 제한적이다. 따라서, 알고리즘의 선택에 있어서 최적의 성능을 낼 수 있는 알고리즘과 메모리 및 전력을 적게 소비하는 알고리즘을 채택하여야 한다.

3. OpenVG 참조 구현을 위한 새로운 특징

3.1 수학 함수

OpenVG의 각 그래픽 객체를 그리기 위한 연산 과정에서 수학 함수의 사용이 빈번하다. 수학 함수를 계산하는 방식은 두 가지로 나뉜다. 첫 번째는 미리 계산된 값을 가진 테이블 값을 참조하는 방법이고, 두 번째는 유한 차수의 테일러 급수를 계산하는 방법이다[5]. 전자의 경우는 메모리의 소모가 큰 반면, 후자의 경우는 시간을 많이 소모한다[6].

Hybrid 참조 구현은 후자의 경우를 채택해 수학 함수 연산에 의한 성능 저하를 유발하지만[7], 본 논문에서는 테이블 참조 방식을 채택하여 성능 향상을 꾀했다.

3.2 정렬 알고리즘

OpenVG에서 정렬은 테셀레이션을 기반으로 하는 렌더링 알고리즘에서 사용된다. 즉, 스캔 선을 지나는 정점을 x좌표 순서로 정렬할 때와 여러 개의 자르기 사각형을 정렬할 경우가 있다. 하이브리드 참조 구현에서는 거품 정렬을 채택하였지만, 본 논문에서는 병합 정렬을 채택하였다. 거품 정렬이 $O(N2)$의 복잡도를 갖는 반면, 병합 정렬은$O(NlogN)$의 복잡도를 가지므로 성능을 향상시킬 수 있다.

3.3 개선된 래스터 렌더링 알고리즘

렌더링이란 벡터 그래픽 객체를 디스플레이에 그리는 동작을 말한다[8]. 렌더링 방식에는 벡터 그래픽 객체를 매번 그리는 벡터 렌더링과 이미지의 각 픽셀들이 가지는 컬러 값을 계산하여 그리는 래스터 렌더링 방식이 있다. 래스터 렌더링은 벡터 렌더링에 비해 여러 가지 면에서 장점을 갖는다. 첫째, 궤적의 개수 증가 및 궤적의 면적 증가에 따른 복잡도가 벡터 렌더링보다 낮다. 두번째, 채움 규칙을 적용하기 위한 계산은 정점 그리기 단계에서 모두 이루어지므로 벡터 렌더링에 비해 연산량이 적다. 세번째, 정점을 직접 그리지 않기 때문에 정점 그리기를 위한 수학 연산이 필요 없다.

그러나, 이 방식은 실제로 화면에 출력되지 않는 부분까지 모두 계산하기 때문에 불필요한 연산으로 시간을 소비한다. 본 논문에서는 이러한 문제점을 해결하기 위해서 개선된 렌더링 알고리즘을 제안한다. 그림 4는 본 논문에서 제안하는 개선된 래스터 렌더링 알고리즘을 나타낸다.

제안된 렌더링 알고리즘의 절차는 다음과 같다: (1) 각각의 자르기 사각형에 대해 궤적 경계와 겹치는 영역이 발견되면 해당 부분에 대해 픽셀 색을 계산한다. (2) 겹치는 영역이 발견되지 않으면 다음 자르기 사각형에 대해 같은 연산을 수행한다. (3) 해당 스캔 라인과 교차하는 정점이 발견되지 않으면 다음 라인으로 스캔 라인을 옮긴다.

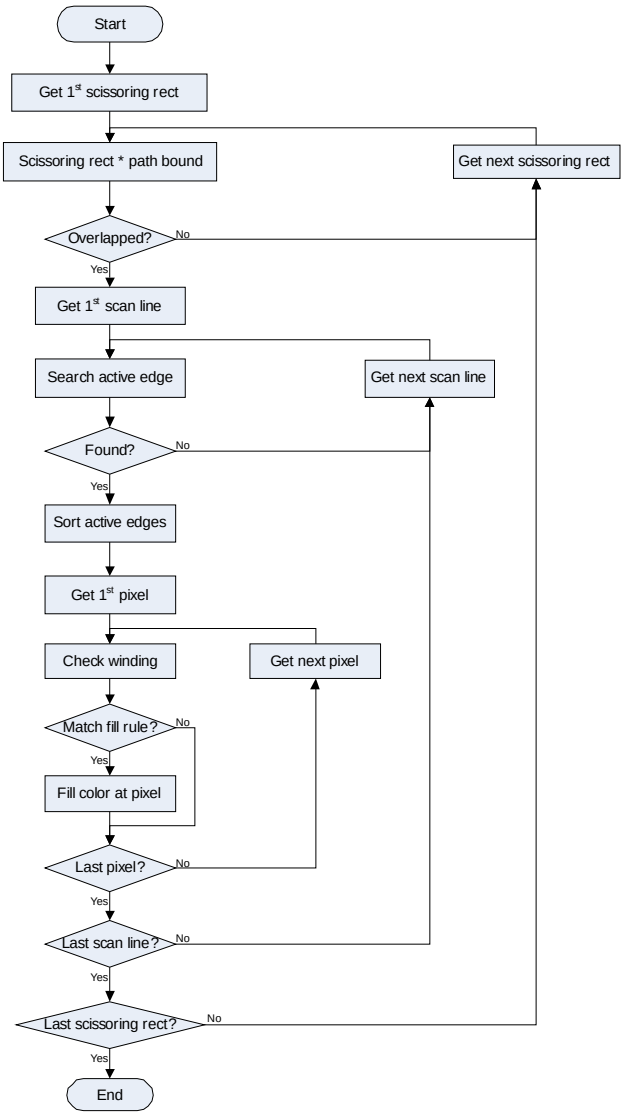


그림 4. 제안된 렌더링 알고리즘

즉, 제안된 방식은 픽셀 값을 계산하기 위해 방문하는 대상을 현저히 줄임으로써 기존 방식보다 빠르게 벡터 그래픽을 표시할 수 있다.

4. 임베디드 환경을 위한 설계 포인트

4.1 EGL과 OpenVG의 결합성

Hybrid 참조 구현은 EGL과 OpenVG가 각각 생성하는 객체에 대한 자료 구조를 서로 공유한다. 따라서, EGL 혹은 OpenVG 블록 중 어느 한 블럭을 수정하면 다른 블럭에도 영향을 미친다. 예를 들면, OpenGL|ES를 지

원하도록 EGL의 자료 구조를 확장하여 사용할 때, OpenGL|ES 응용 프로그램만을 개발하더라도 OpenVG를 위한 객체도 EGL내에 생성되기 때문에 자원 낭비가 발생한다.

본 논문에서는 이런 문제점들을 해결하기 위해, EGL과 OpenVG의 자료 구조를 분리하였고, 서로의 자료 구조와 상태 정보를 은닉하였다. 또한, OpenVG가 EGL API 혹은 EPL API 호출을 통해 EGL 엔진 기능을 사용할 수 있도록 설계하였다.

4.2 언어 의존성

Hybrid 참조 구현은 C++ 언어로 구현되었다. C++ 언어는 데스트탑 환경에서는 매우 강력하나, 임베디드 환경에서 사용되기에는 많은 문제점이 있다. 첫번째 상속이나 가상 함수를 실행시키는 속도가 느리다. 두번째, 어떤 컴파일러는 C++ 언어를 완전히 지원하지는 않는다. 따라서, 본 논문에서는 임베디드 기기에 쉽게 이식이 될 수 있고 실행 속도가 빠른 C 언어를 채택하였다. 또한, C++ 언어에서 쉽게 지원되는 객체 지향 개념을 C 언어에서도 제공될 수 있도록 구조체에 정보를 가공하고 처리하는 함수를 정의하였다.

4.3. 객체의 유일성 확보를 위한 싱글톤 패턴

객체 중에는 여러 개의 복사본 생성이 허용되지 않는 객체가 있다. EGL 블럭에서는 각 Factory가, OpenVG 블록에서는 Context Manager, State Manager, Image Manager, Algorithm Factory가 그런 객체들이다. 우리는 객체의 유일성 확보를 위해 싱글톤 패턴을 도입하였다. 싱글톤 패턴이 없이 설계하면, 각 모듈들은 이러한 객체들을 인지하고 생성을 회피해야 한다. 또는, 전역 변수 저장소에 이 객체들을 등록하고 사용해야 한다. 하지만, 플랫폼마다 전역 변수 저장소의 구조가 다르므로 다른 플랫폼에 이식할 경우 기존 코드를 수정해야 하는 문제가 발생한다.

5. 구현 및 실험

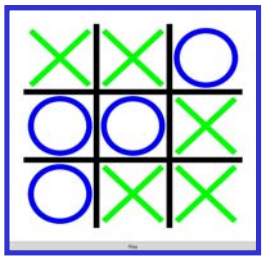
5.1 구현

본 논문에서는 EGL과 OpenVG를 구현하고 다양한 환경에 포팅을 해 봄으로써 제안된 설계 방식이 임베디드 환경에 쉽게 적응할 수 있는지 검증하였다.

처음에는 윈도우즈 XP를 기반으로 EGL과 OpenVG를 구현하였으나, 포팅 레이어를 수정하여 쉽게 리눅스와 WIPI 플랫폼[9]에 이식할 수 있었다.

윈도우즈 XP 용 EGL은 처음에 네이티브 윈도우 시스템을 접근하기 위해 GDI(Graphic Device Interface)를 이용하여 구현하였으나, Hybrid 참조 구현과의 성능 비교를 위해 GLUT를 이용하여 포팅하였다. GLUT를 이용하면 리눅스에 이식할 때 코드 수정을 줄일 수 있는 장점이 있다.

그림 5는 우리가 구현한 참조 구현을 통해서 벡터 그래픽을 화면에 출력한 모습이다. 그림 5(a)는 Tic-Tac-Toe 게임에서 보여지는 이미지를 그린 것이고, 그림 5(b)는 벡터 그래픽의 대표 이미지라고 할 수 있는 호랑이 이미지를 그린 것이다. 호랑이 이미지는 305개의 궤적으로 구성되어 있다. 그림 5(c) 는 핸드폰에 탑재하여 호랑이 이미지를 출력한 모습이다.



(a) Tic-Tac-Toe

(b) PC상에서의 타이거

(c) WIPI에서의 타이거 이미지

그림 5. 구현 예제

그림6은 호랑이 이미지가 벡터 그래픽으로 출력되는 과정을 보여준다. 전체 궤적 305개중 각각 25%, 50%, 75%, 100% 의 궤적이 그려졌을 때의 이미지를 나타내고 있다.

(a) 25%

(b) 50%

(c) 75%

(d) 100%

그림 6. 호랑이 렌더링 과정

5.2 CTS 테스트 결과

본 논문에서 제안한 참조 구현이 표준 규격을 얼마나 준수하는지 검증하기 위해 크로노스 그룹에서 배포하는 OpenVG 호환성 테스트 툴인 CTS(Conformance Test Suites)1.0.0[10] 를 통해 테스트를 수행하였다. 그 결과 약 73% 의 성공율을 보였다. 표 1과 표 2는 각기 제안된 참조 구현과 Hybrid 사의 참조 구현의 항목별 CTS 통과율을 보여준다.

표 1. 제안된 참조 구현의 CTS 실험 결과

시험 항목	테스트 개수	성공	실패	성공율(%)
Parameter	10	8	2	80.0
Matrix	11	11	0	100.0
Clearing	3	3	0	100.0
Scissoring	5	4	1	80.0
Masking	2	2	0	100.0
궤적	48	37	11	77.1
Image	10	6	4	60.0
Paint	10	5	5	50.0
Image Filter	3	2	1	66.7
VGU	12	5	7	41.7
합계	114	83	31	72.81

표 2. Hybrid 참조 구현의 CTS 실험 결과

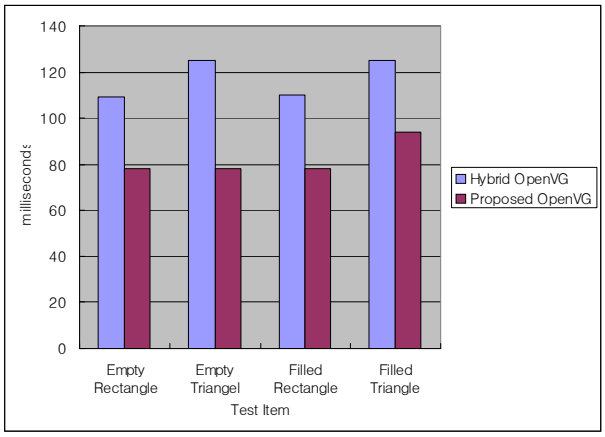
시험 항목	테스트 개수	성공	실패	성공율(%)
Parameter	10	8	2	80.0
Matrix	11	11	0	100.0
Clearing	3	3	0	100.0
Scissoring	5	4	1	80.0
Masking	2	1	1	50.0
궤적	48	47	1	97.9
Image	10	9	1	90.0
Paint	10	6	4	60.0
Image Filter	3	2	1	66.7
VGU	12	9	3	0.75
합계	114	100	14	87.7

CTS가 실패로 통지한 항목 중에는 성공했을 때의 이미지와의 차이를 눈으로는 알 수 없는 항목들도 있었다. 이는 픽셀 값이 한 칸씩 밀려서 출력되는 경우였다. 또한, 실패한 경우를 통과시키기 위하여 코드를 수정해서 통과시키면, 이미 통과되었던 테스트 이미지가 실패하는 경우도 있었다. 이는 CTS의 테스트 이미지가 아직 안정화 되어 있지 않은 것으로 파악된다.

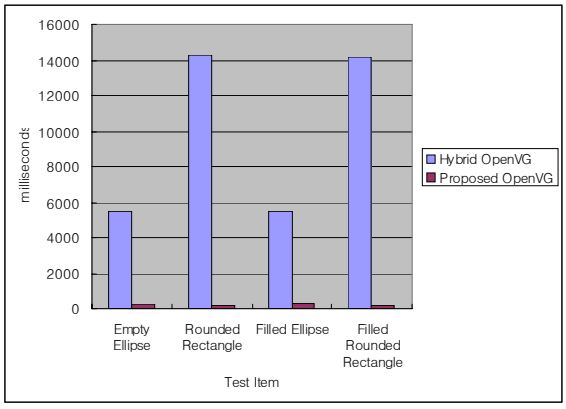
5.3 성능 측정

본 논문에서 제안하는 OpenVG의 성능 측정을 위해 Vgperf 라는 2D 벡터 그래픽 성능 측정 프로그램을 개발하였다. Hybrid 참조 구현과의 공정한 비교를 위해, 윈도우즈 XP에서 수행하였다. 하드웨어 환경은 인텔 코어 듀오 1.83GHz CPU, 2GB 램, 2MB(L2) 캐시 메모리, ATI Radeon X1400 그래픽 카드, 128MB 비디오 램이다.

그림 7은 궤적에 대한 성능 측정 결과를 보인다.



(a) 기본 궤적 성능 측정



(b) 복합 궤적 성능 측정

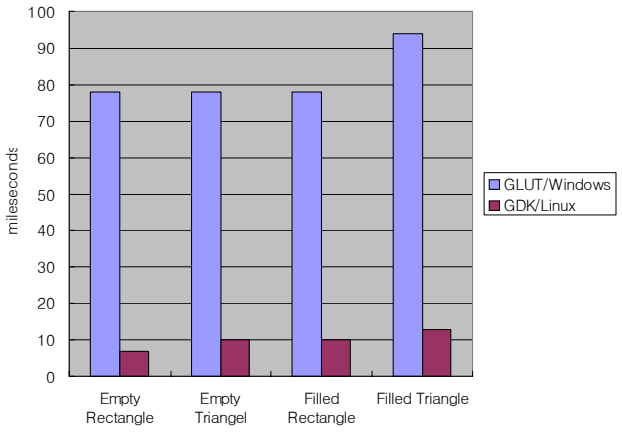
그림 7. 궤적 렌더링 성능 측정

그림 7(a) 는 기본적인 도형인 삼각형, 사각형을 그린 측정 결과를 나타내고 있고, 그림 7(b) 는 타원, 둥근 사각형과 같은 좀 더 복잡한 도형에 대한 측정 결과를 나타내고 있다. 그림에서 볼 수 있듯이, 본 논문에서 제안하는 OpenVG 가 Hybrid 참조 구현보다 기본 궤적인 경우 1.3-1.6 배 빠르며, 복잡한 궤적인 경우는 4.6-76 배나 빠른 렌더링 속도를 보이고 있다.

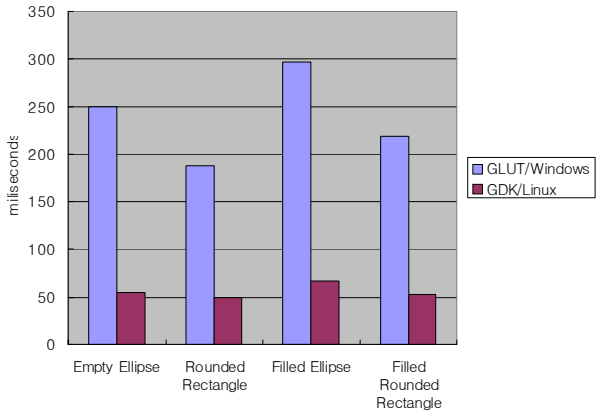
호랑이 이미지는 윈도우의 크기가 1024x768 인 경우 Hybrid 참조 구현이 약 7.3초 걸렸고, 본 논문에서 제

안한 OpenVG는 약 3.6초 걸렸다.

그림 8은 GLUT/Windows 와 GDK/Linux 상에서의 궤적 그리기의 성능 측정 결과를 보여준다.



(a) 기본 궤적 성능 측정



(b) 복합 궤적 성능 측정

그림 8. 윈도우즈와 리눅스 상에서의 성능 측정 결과

그림 8에서 볼 수 있듯이, 제안된 OpenVG 참조 구현은 리눅스 상에서의 구현이 윈도우wm 상에서의 구현보다 기본 궤적에서는 7.2배에서 11.1배, 복합 궤적에서는 3.7배에서 4.5배 빠르다. 실험을 통해 GLUT가 렌더링 속도에 악영향을 끼쳤음을 알 수 있다.

6. 결 론

벡터 그래픽 분야에서는 이미 매크로미디어의 플래쉬가 80% 이상의 시장을 점유하고 있다. OpenVG는 플래쉬보다 늦게 시작되었지만 업계 표준으로 정착이 되어가고 있고, 대부분의 그래픽 카드 업체들이 표준화에 참여하고 있기 때문에 조만간 시장 점유를 높여 가리라 예상된다.

본 논문에서는OpenVG를 소프트웨어 렌더링 방식으로 구현한 참조 구현을 제안하였다. 제안된 방식이 기존 Hybrid 참조 구현보다 우수한 성능을 보임으로써 우리는 소프트웨어 렌더링 방식의 성공 가능성을 보여주었다. 또한 임베디드 환경에 부합하도록 체계적으로 설계함으로써 다양한 플랫폼에 쉽게 이식할 수 있었다. OpenVG가 비교적 최근에 등장한 표준이다 보니 이에 대한 학계의 연구나 개발 사례가 많지 않다. 이 논문이 OpenVG 연구를 활성화 시키고 발전시키는 계기가 되길 기대한다.

향후에는 CTS 통과율을 높이고, 소프트웨어 렌더링 방식의 성능 개선에 관한 연구를 계속 진행할 예정이다. 또한, OpenVG 기반의 SVG 지원에 대한 연구를 진행할 예정이다.

참고문헌

[1] Kari Pulli, "New APIs for Mobile Graphics", Proceedings of SPIE - The International Soci ety for Optical Engineering, Vol. 6074, art. no. 607401, 2006.

[2] "Multi-stroke freehand text entry method using OpenVG and its application on mobile devices", LNCS , Vol. 3942, pp. 791-796, 2006.

[3] Khronos Group Std. OpenVG, Kronos Grouop Standard for Vector Graphics Accelerations, www.khronos.org, 2005.

[4] Khronos Group Std. EGL, Kronos Grouop Standard for Native Platform Graphics Interfaces, www.khronos.org, 2005.

[5] Alan Watt, 3D Computer Graphics 3rd Edition, Addison-Wesley, 2000.

[6] Hybrid Graphics Forum, OpenVG Reference Implementation, http://forum.hybrid.fi, 2005.

[7] R.C. Gonzalez & R.E. Woods, Digital Image Processing 2nd Edition, Addison-Wesley, 1992.

[8] Steven Harrington, Computer Graphics A Programming Approach 2nd Edition, McGraw Hill, 2006.

[9] KWISF, Wireless Internet Platform for Interoperability (www.wipi.org.kr), 2006.

[10] Huone, Confermance Test Suite for OpenVG, www.khronos.org, 2006.