韓國 컴퓨터情報學會 2008 제38차 하계학술발표논문집
第16卷 第1號, 2008. 06.

모바일 벡터 그래픽 가속기 설계를 위한 OpenVG API 구현

김 영 옥*, 노 영 섭*, 오 삼 권**

Implementation of OpenVG API for Mobile Vector Graphics Accelerator

Young-Ouk Kim*, Young-Sup Ro*, Sam-Kwan Oh**

요 약

최근 모바일 시스템의 성능이 향상되면서 다양한 형태의 동적인 메뉴 구성과, 메일 및 이차원 지도 등의 표현에 벡터 그래픽을 도입하고 있다. 모바일 기기에서 사용되는 벡터 그래픽 처리 기술은 Flash Lite, SVG(Scalable Vector Graphics)등이 널리 사용되고 있는데 두 가지 모두 소프트웨어 방식으로 사용되고 있다. 매크로미디어사의 Flash Lite는 연산에 많은 메모리를 필요로 하고, SVG는 웹 표준에 맞춘 스크립트 해석 기반으로 구동 속도가 느리다. 모바일 컴퓨팅 환경에서 벡터 그래픽스에 대한 필요성과 사용빈도가 증가함에 따라 메모리를 적게 사용하고 하드웨어 가속기를 지원 할 수 있도록 저 수준의 API(Application Programming Interface)인 OpenVG 1.0을 크로노스 그룹(Khronos Group)에서 제정하였다. 본 논문은 모바일 사용 환경에 맞추어 사용될 수 있도록 OpenVG 1.0에 기반한 API를 구현하고 실험하였다. 구현된 API는 느린 소프트웨어의 한계를 벗어나기 위해 하드웨어 가속기 설계에 적합하도록 각각의 API 블록 및 형태를 하드웨어 파이프라인 형태의 관점에서 설계하였고, 구현된 API를 윈도우즈 환경에서 기능을 검증하였다.

▸ Keyword : OpenVG 1.0, Flash Lite, SVG, Khronos Group, Accelerator

• 제1저자 : 김영옥
* 서울벤처정보대학원대학교 임베디드 시스템학과, ** 호서대학교 컴퓨터공학과

1. 서론

벡터 그래픽은 주어진 이차원이나 삼차원 공간에 선이나 형상을 배치하는데 있어 일련의 명령들이나 수학적 표현을 통해 디지털 이미지를 만드는 것이다. 물리학에서의 벡터는, 크기와 방향을 둘 다 동시에 갖는 것을 말한다. 벡터 그래픽에서는 사용자들의 창작 활동 결과물인 그래픽 파일이, 일련의 벡터 서술문의 형태로 창작되고 저장된다. 예를 들면, 벡터 그래픽 파일에는 선을 그리기 위해 각 비트들이 저장되어 있는 대신에, 연결될 일련의 점의 위치가 들어 있다. 그로 인해 파일 크기가 작아지는 결과를 가져오게 된다. 또한, 벡터 그래픽 파일은 래스터 이미지 파일보다 수정하기 쉬운 장점이 있다.

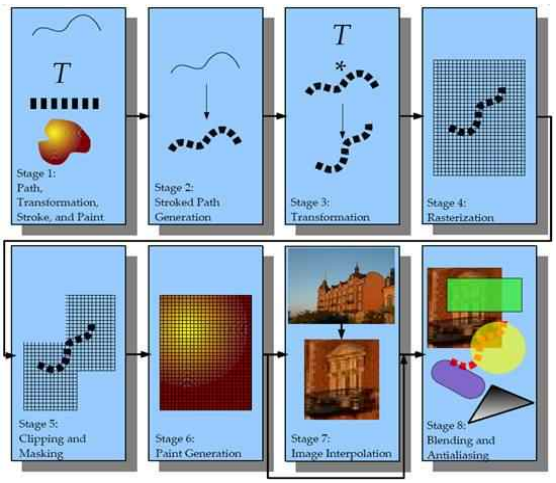
래스터 그래픽은 비트맵 그래픽에 상응하는 개념으로서, 색상표현이 정교해서 사진 작업등에 많이 이용되며, 대량의 픽셀들 조합으로 표현되는 그래픽이므로 픽셀수가 많을수록 색감이 정교하게 표현되는 반면 용량이 커지게 된다. 또한, 기존 비트맵 등으로 구성된 이미지는 확장되거나 변경되는 경우 이미지의 외곽 형태에 소위 계단현상이 발생하여 사실적이거나 회화적인 표현이 제한되나, 벡터로 구현된 이미지는 확대되거나 또는 축소되는 경우에도 수학 연산자인 매트릭스를 이용함으로써 상기와 같은 문제점이 극복되어 자연스럽게 표현될 수 있는 강점이 있다. 이런 이유로 최근 모바일 임베디드 기기에서는 사용자 인터페이스를 기존의 비트맵 방식에서 벗어나 벡터 위주 혹은 벡터와 삼차원 그래픽의 혼용으로 표현하기도 한다. 벡터 그래픽에 대한 이와 같은 현상은 휴대폰, PDA(Personal Digital Assistant), PMP(Portable Multimedia Player) 등의 모바일 기기 위주의 디지털 컨버전스 제품에서 많이 적용되고 있으며, 사용자 인터페이스를 기반으로 멀티미디어 메일과 위치 기반 시스템, 교육용 콘텐츠, 오락, 광고 등의 분야에 사용되고 있다. 현재 모바일 환경에서 사용되는 벡터 그래픽 처리 기술로는 매크로미디어(Macromedia)사의 플래시와 W3C의 SVG[1]가 이용되고 있다.

본 논문에서는 임베디드 기기에 그 활용성이 높아지고 있는 벡터 방식의 그래픽 엔진인 OpenVG(Open Vector Graphic)[2]를 분석하여 임베디드 기기의 가속기 설계에 적합하도록 소프트웨어를 설계하였다.

논문의 제 2절에서는 OpenVG 벡터 그래픽에 대한 관련 연구를 기술하여 사전 지식을 습득하고, 3절에서는 하드웨어 가속기 설계에 적합하도록 OpenVG API의 기하 연산 처리와 렌더링 처리 설계를 위해 각 기능 부분을 설명하고 적용된 알고리즘을 살펴본다. 제 4절에서는 설계된 OpenVG API를 이용하여 구현한 소프트웨어를 검증 및 고찰하고, 5절에서 결론을 맺는다.

2. OpenVG 벡터 그래픽 관련 기술

OpenVG API는 크로노스 그룹에서 제정한 이차원 벡터 그래픽스 이며, 현재 웹 기반에서 많이 사용되는 플래시와 SVG와는 다른 저 수준의 API 지원과, 적은 메모리를 사용하고, 하드웨어 가속화를 지원하기 위한 플랫폼 독립적인 오픈 규격이다. OpenVG의 기본적인 시스템 구성은 [그림 1]과 같으며, 각 파이프라인은 전, 후처리로 나누어 그 기능별 블록이 구별되어 있다.



<그림 1> OpenVG 파이프라인

2.1 기하 연산 처리

[그림 1]의 파이프라인을 살펴보면 단계(stage) 1 – 4에 해당되는 처리 구간이 기하처리 연산임을 알 수 있으며, 기하 연산 단계에서는 이차원의 패스(path) 데이터를 기반으로 수학적 연산을 위한 매트릭스 값을 갖고 패스의 변환(model transform)을 수행한다. 변환이 수행되는 대상은 이차원 패스이고 각 패스를 이루는 정점을 이용하여 기본적인 스트로크(stroke)와 대상 객체를 채울 페인트(paint) 형태를 만들어 일차적인 선분을 만들어 낸다. 만들어진 선분은 3x3 행렬의 곱셈 연산이 주를 이루게 되는데 행렬의 곱셈 연산은 일반적인 곱셈과 덧샘 연산으로 행하여진다. 또한 사용자가 원하는 좌표, 도형 등의 정보를 입력 받아, 그리고자 하는 파라미터를 결정하기도 한다. 예를 들면, 스트로크 하는 선의 두께나, 채

우고자 하는 색상의 결정, 방법의 결정 등이 이에 해당된다. OpenVG의 각 그래픽 객체를 그리기 위한 연산 과정에서 연산을 위한 함수의 사용이 빈번하다. 여기서 함수를 계산하는 방식은 두 가지로 나뉘는데, 첫 번째는 미리 계산된 테이블 값을 참조하는 형태이고, 두 번째는 유한 차수의 테일러 급수를 계산하는 방식이다[3]. 전자의 경우는 메모리의 소모가 큰 반면, 후자의 경우는 시간을 많이 소모한다[4]. 참고로 본 논문에서는 테이블 참조 방식을 택하여 속도 향상과 이후 하드웨어 가속기 설계를 하기 위해 API를 설계하였다. 기본적인 스트로크 속성이 결정된 후 패스 생성은 대상 패스의 아웃라인을 먼저 정의한다. 이는 패스 정보가 스트로크 되어야 한다면, 입력한 패스 정보를 일정한 두께의 폭으로 위치 값(offset)을 수행하고, 기타 대시(dash) 형태로 그릴 경우는 이를 반영한 새로운 경로를 생성해야 한다. 기하 처리의 다음 단계는 사용자 좌표(user coordinate)로부터 화면 좌표(screen coordinate)로 변환하는 과정이 수행된다. 여기서 좌표 변환이 일어나 최종 출력하고자 하는 프레임 버퍼의 원점 좌표 간 행렬과 벡터 간의 곱셈 연산이 수행된다. 따라서 위와 같은 형태의 선분과 스트로크 패스 생성을 이용하여 일차적인 기하 연산 처리를 거쳐 현재 패스가 영향을 주는 픽셀의 평균값을 구한다. 여기서 구한 평균값은 저장되어 후처리의 안티알리어싱에서 사용하게 된다. 기본적인 벡터 데이터가 비트맵 형식의 데이터로 변환되는 단계로 전처리를 수행한다.

2.2 렌더링 벡터 처리

기하 연산 처리의 수행이 끝난 후 생성된 패스의 픽셀 평균값에 각종 효과(이미지 섞기, 잘라내기, 칼라 섞기 등)등의 과정을 렌더링 처리에서 수행한다. 렌더링이란 벡터 그래픽 객체를 디스플레이에 그리는 동작을 말한다[5]. 기하 연산 처리 이후 최종 생성될 패스에 대해 화면의 원하는 영역에만 출력하기 위해 잘라내기(clipping)와 마스킹(masking)이 이루어진다. 잘라내기란 주어진 이차원 선분의 보기 영역(view port)을 벗어나는 선분을 제거하는 것을 말한다. 따라서 렌더링 처리에서 불필요한 연산을 사전에 제거하여 생성될 패스에 대해서만 여러 효과를 적용한다. 잘라내기와 마스크 연산 이후 실제 어떤 색이 칠해질지를 결정하는 단계로 단색으로 칠하는 경우와, 단색 스트로크 그래데이션을 이용한 채우기(fill)와 스트로크, 패턴 채우기 등을 수행하게 된다. 페인트 생성 단계에서는 크게 세 가지 형태의 페인트로 구별되어야 한다. 첫째 칼라 페인트와 둘째 그래디언트 페인트, 마지막으로 패턴 페인트 형태가 있다. 칼라 페인트는 모든 픽셀에 대해 주어진 칼라 값으로 채우는 형태이고, 그래디언트는 리니어(linear)와 래디얼(radial) 형태로 주어진 선분 영역에 칠하기 효과를 준다. 페인트 생성 과정 이후 OpenVG API에서 주어지는 이미지를 이용하여 각 픽셀에 이미지 칼라를 더하는 보간 과정을 처리한다. 이 과정은 이미지에 적용된 매트릭스의 인버스(inverse) 매트릭스를 사용해 보간된 이미지 값으로 각 픽셀의 값을 계산한다. 이 과정은 OpenVG API의 그리기 형태 값에 따라 수행 여부를 판별하여 진행한다. 이는 벡터 그래픽의 그리기 값 중 이미지를 사용하지 않는 솔리드(solid)와 이미지 형태로 구별하여 수행하기 때문이다. 일반적으로 이미지를 사용하지 않는 솔리드 형태로만 벡터를 표현할 수 있기 때문이다. 페인트 생성과 이미지 보간이 완료되면 렌더링 수행의 마지막 단계로 블렌딩(blending) 단계를 거쳐 최종 완성된 형태의 픽셀을 만들어 낸다. 위와 같은 단계의 파이프라인을 거쳐 출력하길 원하는 패스 또는 이미지 등의 객체에 대해 그리기 파라미터를 설정하고 [그림 1]의 단계 1 이후 OpenVG API의 그리기 함수를 호출하면 그래픽스 파이프라인에서 사용자가 정의한 그리기 파라미터를 분석하여 그릴 객체를 비트맵 정보로 변환한 후(단계 2 - 단계 7), 이미 출력된 비트맵과의 합성 여부를 판별하여 합성이 이루어진다(단계 8). 이와 같은 객체에 대해 반복 수행하여 한 장의 완성된 이미지가 생성된다.

3. 벡터 처리를 위한 OpenVG API 설계

2절에서 언급한 OpenVG 파이프라인을 기본 토대로 본 논문에서 설계한 이차원 벡터 그래픽 API의 전체 흐름도는 [그림 2]와 같다. [그림 2]에서 EGL(Embedded Graphics Library)은 OpenVG API가 실제 수행되어 각종 정보를 입력할 컨텍스트(context)와 최종 픽셀이 저장되는 프레임 버퍼(surface)등의 초기화 과정으로 이루어진다. 다음 오브젝트 형태(type) 판별은 이차원 패스 객체 혹은 이미지를 그리기 위한 객체로 구별된다. 이는 OpenVG API가 두 객체 그리기를 따로 지원하며, 각각의 객체는 자신의 형태별 데이터를 건네받아 OpenVG API를 구동하기 위한 각종 파라미터와 이미지 데이터 혹은 패스 데이터를 메모리에 저장하는 단계이다. 이후 기하 연산 처리 엔진은 이차원 오브젝트의 기하 변화를 처리하기 위해 매트릭스를 이용한다. 변환은 이차원 오브젝트의 기하 변화를 나타내는 3x3 행렬 매트릭스를 만들고 이차원 오브젝트를 변환시킨다. [그림 2]의 칠하기 패스 생성 부분(fill path generation)에 세그먼트 라인 변환(segment to line convertor)을 포함한다. 세그먼트 라인 변환은 곡선이나 호와 같은 세그먼트를 직선으로 보간 한다. 렌더링 처리

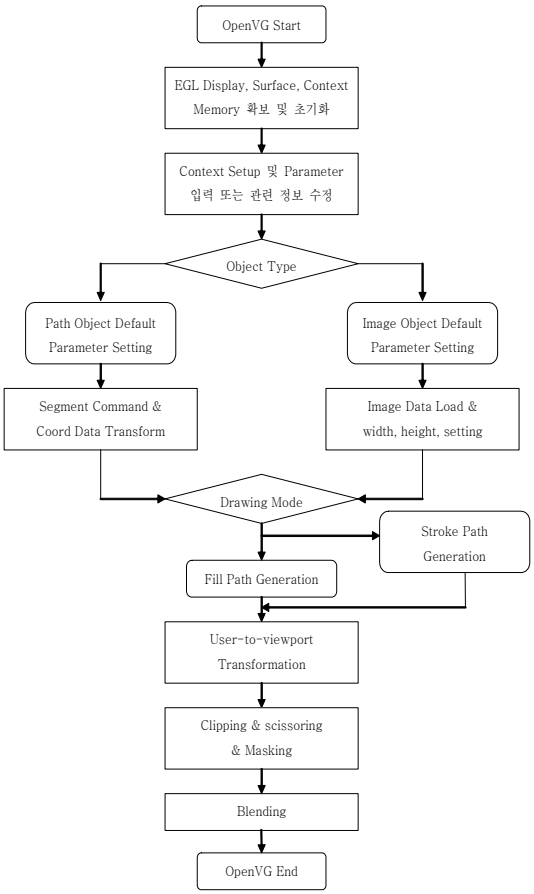
부분은 앞서 기하 단계에서 생성된 액티브 에지(active edge)를 이용하여 실제 컬러 정보를 구하게 된다. 잘라내기 검사는 주어진 사각형 안의 영역만 그리게 하고 페인트 생성은 각 페인트 형태에 따라 오브젝트 안에 채워질 픽셀 정보를 구한다. 알파 마스킹은 앞서 구해진 픽셀 정보의 알파에 대한 마스크 값을 곱한다. 블렌드 연산에서는 현재 구해진 소스 칼라와 프레임 버퍼에 있는 목적 칼라 간에 섞기 연산을 한다.

3.1 기하 연산 부분 구현

OpenVG API 기하 연산은 사용자에 의해 주어진 매트릭스 값을 갖고 일차적인 수학 연산을 한다. 매트릭스는 매트릭스 변환에서 읽어 들인 패스 좌표를 행렬로 변환하고, 이후 대시 생성을 한다. 대시 생성은 스트로크 페인트 시 대시 값을 체크하여 새로운 라인 좌표들을 생성한다[6]. 대시는 서브 스트로크를 수반하여 생성되는데, 이는 라인 두께, 라인 끝의 모양, 선이 이어지는 부분의 모양에 따라 새로운 라인들을 생성하고 이후 API 내부에서 액티브 에지를 생성한다. 액티브 에지 생성은 앞에서 생성된 좌표에 따라 렌더링 엔진에서 사용될 시작점 x축(startx), 시작점 y축(starty), 끝점 x축(endx), 그리고 끝점 y축(endy) 값을 가지는 라인 좌표를 만든다. [그림 2]의 칠하기 패스 생성 부분 까지가 대부분 기하 연산 부분을 API로 구현하는 것이고, [그림 1]의 단계 1-3 까지가 오브젝트 단위로 기하 처리를 하는 부분으로 설계 하였다.

3.2 렌더링 연산 부분 구현

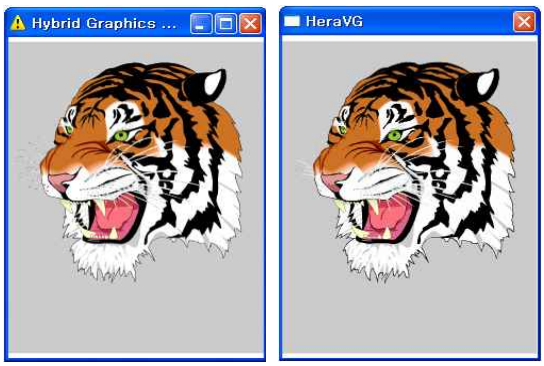
기하 처리기를 거쳐 생성된 에지 값은 래스터라이저 셋업단에서 채우기 규칙을 검사하고 스팬(span)될 최저 x축(xmin), 최고 x축(xmax), 스텝(step), 그리고 알파 값을 구한다. 이후 OpenVG API 파이프라인 단계에서 자르기 검사를 거쳐 각 페인트 타입에 따른 픽셀 정보를 구한다. 렌더링 연산은 벡터 처리를 위해 기울기(slope)와 거리 값을 구해낸다. OpenVG에서 정렬은 테셀레이션을 기반으로 하는 렌더링 알고리즘을 사용한다. 즉, 스캔 선을 지나는 x좌표 순서로 정렬할 때와 여러 개의 자르기 사각형을 정렬할 경우가 있다. 본 논문에서는 패스 생성 블록에서 이차원 x, y의 크기를 병합 정렬 알고리즘을 이용하여 오름차순으로 정렬된 데이터는 픽셀 생성을 위해 선과 선(line by line) 형태로 변형되어 벡터의 여러 요소를 표현한다. [그림 2]의 스트로크 패스 생성(stroke path generation)은 래스터라이저 셋업 블록을 포함하고, 이 블록은 사용자가 그리고자 하는 오브젝트를 위해 기하 단계에서 생성된 액티브 에지를 이용하여 채우기 규칙을 적용하고 y좌표에 따른 최저 x축 ,최고 x축, 그리고 알파값을 생성한다. 이후 페인트 생성에서 구해진 픽셀 정보의 알파 값은 알파 마스킹 연산에서 알파 마스크 값과 곱해지고 API의 최종 부분인 블렌딩(blending)에서 API에 주어진 모드에 따라 솔리드 페인트, 이미지 모드로 프레임 버퍼의 픽셀 정보와 합쳐져 최종 스크린에 보여 질 컬러 정보를 프레임 버퍼에 다시 쓰게 되고, OpenVG API를 종료하게 된다.



<그림 2> OpenVG API 소프트웨어 흐름도

4. 실험 및 검토

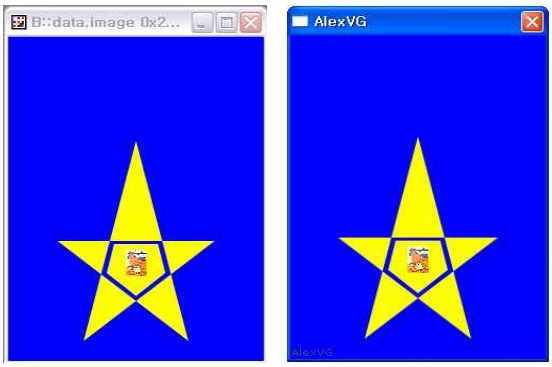
본 논문에서 구현한 API의 기능 검증을 위해 국내의 (주)휴원[7]에서 제작한 콘텐츠와 하이브리드(Hybrid)[4]사의 호랑이 이미지를 사용하여 각각을 윈도우즈 기반에서 검증 하였다.



(a) 하이브리드사의 호랑이 (b) 본 논문의 호랑이

<그림 3> 실험 영상 I

[그림 3]의 실험 영상을 보면, 하이브리드사가 SVG Tiny용으로 구현한 벡터 콘텐츠를 OpenVG API에서 사용하기 위해 변환한 것이며, 305개의 많은 궤적으로 구성되어 있다. 위의 호랑이 콘텐츠는 벡터 콘텐츠를 대표하는 많은 수의 궤적과 솔리드 형태 및 다양한 스트로크 형태를 포함하므로 OpenVG API의 많은 기능을 사용한다고 볼 수 있으며, 따라서 검증을 하기에 용이하다.



(a) 본 논문의 타일 채우기 (b) (주)휴원의 타일 채우기

<그림 4> 실험 영상 II

[그림 4]는 (주)휴원에서 제작한 타일 형태의 채우기 콘텐츠를 비교한 것이며, 이미지 모드를 먼저 타일 형태 중 채우기 속성으로 주고 솔리드 형태로 별 모양의 벡터를 그려 채우기를 한 형태이다.

위의 실험 영상 I, II에서 알 수 있듯이, 설계한 OpenVG API는 기능상의 여러 모드에서 정상 동작을 하였으며, 기존에 제작된 엔진들과 콘텐츠의 출력이 동일함을 알 수 있었다.

5. 결론

본 논문은 모바일 임베디드 기기에서 이차원 벡터 그래픽을 처리하기 위한 효율적인 OpenVG API 설계 및 관련 알고리즘을 제안하고 하드웨어 가속기 설계에 적합하도록 OpenVG API를 소프트웨어로 구현하였다. 구현한 OpenVG API는 기존의 소프트웨어 렌더링 엔진의 느린 속도를 보완하고, 하드웨어 가속기 설계에 이용되기 위해 OpenVG API의 스펙을 하드웨어 가속기 형태에 적합하도록 관련 파이프라인을 기하 연산과 렌더링 연산으로 나누어 구현하였으며 점차 증가하는 새로운 형태의 GUI(Graphic User Interface)를 필요로 하는 모바일 멀티미디어 기기의 반도체 설계 자산(intellectual property)으로 응용하기에 적당하다. 이후 본 논문에서 설계된 API를 이용해 범용의 하드웨어 가속 환경에서 사용할 수 있는 하드웨어 설계 연구가 필요하다.

참고문헌

[1] W3C, "Scalable Vector Graphics (SVG) Tiny 1.2 Specification Draft 13," W3C SVG Workgroup, April 2005.

[2] Khronos Group, "OpenVG Specification 1.0," Khronos Group, August 2005.

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

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

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

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

[7] 휴원, http://www.hu1.com.