모바일 기기를 위한 스캔라인 엣지 플래그 방식의 2D 벡터 그래픽 래스터라이저 설계

박정훈* · 이광엽* · 정태의**

*서경대학교 컴퓨터공학과 · **서경대학교 컴퓨터과학과

A Design of 2D Vector Graphics Rasterizer with a Modified Scan-line Edge flag Algorithms for Mobile Device

Jeonghun Park* · Kwangyeob Lee* · Taeui Jeong**

*Seokyeong University Department of Computer Engineering

**Seokyeong University Department of Computer Science

E-mail : seiclub@skuniv.ac.kr

요 약

벡터 그래픽스는 수학적 정보를 이용하여 이미지를 표현하기 때문에 이미지 손상 없이 쉽게 확대 축소가 가능하며, 비트맵 방식으로 표현되는 이미지보다 더 작은 파일 크기를 가진다. 본 논문에서 제안하는 벡터 그래픽 래스터라이저는 개선된 스캔라인 엣지 플래그 방식을 사용하여 설계되었으며 클리핑과 슈퍼샘플링 과정을 같이 수행한다. OpenVG 2D 벡터 이미지를 사용하여 검증되었다. 본 논문에서 제안하는 가속기는 Tiger image의 렌더링에 초당 5 프레임의 성능을 가진다

ABSTRACT

Vector Graphics describes an image with mathematical statements instead of pixel information, Which enables easy scalability without loss in image quality and usually results in a much smaller file size compared with bitmap images. In this paper, we propose Vector Graphics Rasterizer for mobile device with scan-line edge flag algorithm. Proposed Vector Graphics Accelerator was verified with OpenVG 2D Vector image. The estimated performance of proposed Accelerator is 5 frame per second with Tiger image.

키워드

Vector Graphics, OpenVG, Scanline

Ⅰ. 서 론

벡터 그래픽은 기하학적 정보를 기술해 이미지를 나타내는 그래픽 방식이며 비트맵 그래픽은 각 픽셀에 대한 색상 정보를 가지고 이미지를 나타내는 방식으로 벡터 그래픽의 경우 이미지를 확대하거나 축소했을 때 계단현상 등의 이미지 손상이 발생하지 않고 원본 이미지의 품질을 유지하기 때문에 한 번의 이미지 제작으로 해상도에 상관없이 동일한 품질로 사용될 수 있으며, 애니메이션 표현이 매우 용이하고, 프레임 수가 많아지더라도 데이터 크기가 비교적 작은 장점이 있어 각종 모바일 기기용 애플리케이션이나 모바일 서비스에 매우 적합한 기술이지만 사진 같은 이미지를 표현하기에는 부적합하고 디코딩 속도가 느리다는 단점이 있다.

최근 모바일 및 소비자 기기에 사용되는 그래픽 유저 인터페이스가 날로 복잡해지고, 많은 기기에 게임 기능 및 멀티미디어 환경 도입되면서 고 화질의 유저 인터페이스 및 텍스트, 애니메이

션과 같은 고 수준의 벡터 그래픽 환경의 필요성이 증대되고 있다.[1]

모바일 시스템에서 벡터 그래픽스에 대한 필요성이 증가함에 따라 Khronos Group에서 2005년 7월 벡터 그래픽스 렌더링 부분을 표준화하고 가속화하기 위한 표준안으로 OpenVG 1.0을 발표하였으며 2007년 1월 이를 확장시킨 OpenVG 1.0.1을 발표하였다. OpenVG는 Flash와 SVG같은 벡터 그래픽 라이브러리들을 위한 하드웨어 가속 인터페이스를 제공하는 공개된 플랫폼 독립적인 API이다.[2]

본 논문에서는 모바일 기기에서 OpenVG API를 지원하며 Vector Graphics 연산을 가속화 하기 위한 하드웨어 구조를 제안한다.

II. OpenVG 파이프라인 구조

OpenVG Specification에서 제안하는 파이프라인은 그림 1과 같다.

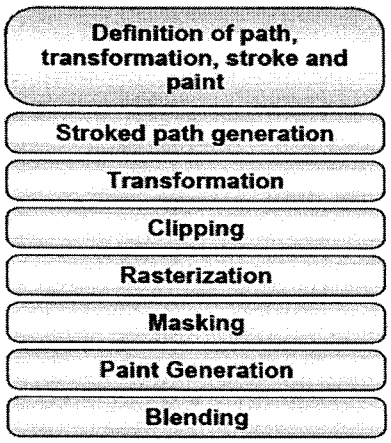

그림 1. OpenVG 파이프라인

OpenVG Specification에서 제안하는 파이프라인은 OpenVG가 수행하는 기능들을 개념적으로 설명하기 위한 파이프라인이다.

몇몇 작업들은 동시에 수행 가능하며, 렌더링 환경 설정에 따라 수행하지 않는 파이프라인도 존재한다. 따라서 실제 구현에 있어서 파이프라인을 연산을 수행하는 단위별로 분류하여 그룹화 시킬 수 있다.

본 논문에서는 OpenVG 모델에 대하여 그림 2와 같은 파이프라인을 제안한다.

Coverage 값과 관계되는 클리핑과 시저링 유닛을 래스터라이저에 포함하였다. 래스터라이제이션 과정에 클리핑을 포함함으로써 화면 영역 밖으로 나가는 엣지를 생성하지 않아 연산량을 줄였다.

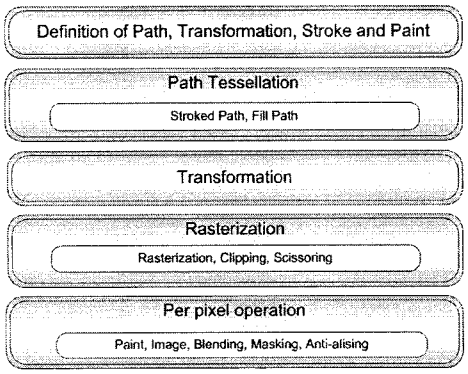

그림 2. 제안하는 파이프라인

또한 각 픽셀 당 연산을 수행해야 하는 페인트 생성 과정과 블렌딩, 마스킹, 안티 앨리어싱 과정을 픽셀 오퍼레이션 단계에서 수행 한다.

III. 부동소수점 표현 방식

벡터 이미지를 생성하는 과정은 다양하고 복잡한 실수연산을 필요로 하며, 이는 상대적으로 처리 성능이 낮은 임베디드 시스템에서 중요한 문제이다. 일반적으로 실수 연산을 위해 Fixed-point와 Floating-point의 2가지 방식이 존재하며 이 2가지 방식은 각각 장단점을 가지고 있다. Fixed-point 연산의 경우 연산을 위한 별도의 연산기 없이 정수 연산기를 이용하여 실수 연산을 수행할 수 있다는 장점이 있다. 그러나 같은 데이터 패스를 가지는 부동 소수점 표현방식에 비해 수의 표현 범위가 작고, 소프트웨어에서는 부동소수점 데이터 형식을 사용하므로 이를 변환하기 위한 별도의 처리 과정을 필요로 한다.[3] 반면 부동 소수점 연산의 경우 이를 위한 별도의 연산기를 필요로 한다는 단점이 있으나, IEEE-754 표준으로 정의되어 있어 하드웨어와 API의 연동 시 추가적인 데이터 타입의 변형 없이 직접적으로 넘겨 줄 수 있어 다양한 플랫폼에 쉽게 이식될 수 있다는 장점이 있다.[4]

24비트 부동 소수점 형식의 실수 연산의 경우, 수의 표현범위가 IEEE-754 단정도 형식에 비해 감소하지만, OpenVG Specification에 명시되어 있는 규정을 만족하며 모바일 기기 대상으로 VGA급 해상도를 표현하기엔 충분한 수의 표현 범위를 가진다.

IV. 래스터라이제이션

가. 클리핑

클리핑은 실제 렌더링이 수행되어질 영역 밖으

로 나가는 엣지들을 제거하여, 이후 파이프라인에서는 렌더링 영역 외부에 해당하는 영역에 대한 연산을 수행하지 않음으로서 연산량을 줄이는 방법이다. 렌더링 영역의 위쪽과 아래쪽 그리고 오른쪽으로 완전히 벗어나는 엣지는 이후 렌더링에 영향을 미치지 않는다. 따라서 해당되는 영역의 엣지들은 무시한다. 렌더링 영역에 일부만 걸쳐지는 엣지들은 렌더링 영역에 맞추어 새로운 좌표를 할당한다. 또한 수평한 엣지도 렌더링에 영향을 미치지 않으므로 제거한다.

나. 엣지 플래그 알고리즘

래스터라이저는 스캔라인 단위로 연산 수행을 하기 위해 슈퍼샘플링을 포함하는 스캔라인 엣지 플래그 알고리즘[5] 을 적용하였다. 스캔라인과 엣지가 교차하는 위치에 플래그 값을 기록해 두고 플래그 사이를 색상으로 채운다. 스캔라인 엣지 플래그 알고리즘에 대한 기본적인 방법은 그림 3에 도시하였다.

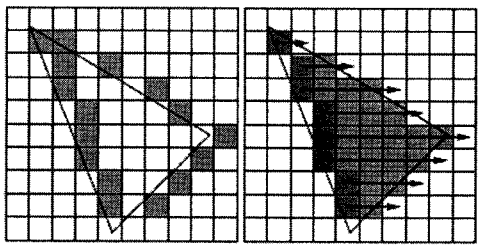

그림 3. 스캔라인 엣지 플래그 알고리즘

기본적인 엣지 플래그 알고리즘의 경우 OpenVG에서 지원해야 하는 2가지의 Fill-Rule 중 Even-Odd fillrule밖에 수행하지 못한다. Non-Zero fillrule을 수행하기 위해서는 엣지의 방향에 따라 Winding 회수를 세기 위한 추가 버퍼를 필요로 한다. 렌더링 과정은 Even-Odd fillrule 을 수행할 때와 동일하다. 차이점은 Even-Odd fillrule을 수행하기 위해선 플래그가 기록된 부분에서 비트를 반전시켜 채우기를 실행하는 반면, Non-Zero fillrule에선 엣지의 방향에 따라(+1 혹은 -1) Winding 회수를 더해주며 이때 Winding 회수가 0이 아닐때 채우기를 수행한다.

다. 스캔라인 엣지 플래그 알고리즘

스캔라인 단위의 렌더링을 수행하지 않을 경우, 렌더링을 수행 할 해상도 크기의 커다란 버퍼를 가져야 한다. 만약 4샘플링 안티 앨리어싱을 수행한다면 버퍼의 크기는 렌더링을 수행 할 화면의 크기보다 4배 큰 버퍼를 가지게 된다. 하지만 스캔라인 단위로 렌더링을 수행할 경우, 렌더링을 수행 할 해상도의 Width 크기의 버퍼만을 가지게 되므로 효율적이라 할 수 있다[3].

기본적인 스캔라인 알고리즘의 경우 스캔라인 단위의 연산 수행을 위해 AET(Active Edge Table)을 생성하고 이를 X좌표 순으로 정렬하는 과정을 필요로 하였다. 스캔라인 알고리즘을 수행하기 위해서 AET 를 정렬하는 과정은 복잡하고 추가적인 메모리 오퍼레이션으로 인한 오버헤드가 발생한다.

라. 제안하는 래스터라이저

본 논문에서 제안하는 래스터라이저는 AET를 정렬하는 과정 없이 렌더링을 수행하도록 설계되었으며 이를 위해 내부에 스캔라인 크기의 마스크 버퍼를 가진다. 또한 Non-Zero fillrule을 수행할 수 있도록 Winding 회수를 기록할 수 있는 Winding Buffer를 가진다. 엣지는 엣지가 시작되는 Y좌표와 종료되는 Y좌표, 그리고 해당 엣지의 시작 Y좌표에서 교차되는 X좌표와 기울기 값을 담고 있다. 만약 시작점 Y가 종료점 Y보다 작다면 엣지의 방향은 1이 되며, 시작점 Y가 종료점 Y보다 크다면 엣지의 방향은 -1이 된다. 이를 이용하여 Winding 회수를 산출할 수 있다.

마. 슈퍼 샘플링

안티 앨리어싱을 위한 슈퍼 샘플링 좌표를 위해 n-look 패턴을 이용하였으며, 각 모드에 따라 8개와 16개의 샘플링을 수행한다. 안티 앨리어싱이 적용되지 않는 경우, 각 픽셀의 중심에서 한번 샘플링을 수행한다. 슈퍼샘플링은 하나의 픽셀에 여러 개의 샘플 포인트를 두어 여러 개의 샘플을 취해 coverage value를 계산하는 방법이다. 각 샘플 포인트는 샘플 가중치(sample weight)를 가지고 있으며, 샘플링 된 샘플 가중치를 이용하여 coverage value를 결정한다. 각 픽셀에서 모든 샘플 포인트는 shape의 외부인지 내부인지 판별한다. 만약 shape의 내부라면 coverage value에 샘플 가중치가 더해지게 된다.

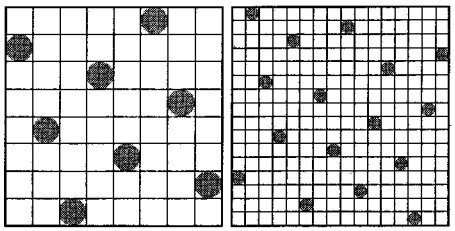

그림 4. 8, 16, n-rooks 패턴

V. 검증 및 결과

Tiger Sample Image의 연산 수행 분석 결과

Floating point의 Addition과 Multiplication 및 Square root, Division 이 집중적으로 사용됨을 알 수 있다. 이는 Tessellation 과 Paint 단계에서 수학적 연산이 많이 사용되기 때문이며, 이러한 연산들로 인해 임베디드 보드 상에서 OpenVG를 Floating Point로 구현하는데 H/W 구현을 통한 속도의 향상을 필요로 한다. 표 1은 Tiger Image를 그리기 위해 수행되는 연산의 회수를 분석하여 도시하였다.

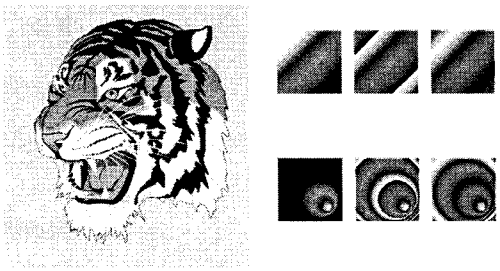

그림 5. Tiger & Gradient Image

Operation	Count
addition	3,313,847,888
multiplication	181,431,340
square root	2,269,697
division	346,706
sign	409

표 1. Tiger Image 에 사용된 연산 횟수

그림 5에서 보여준 Gradient 이미지를 생성하기 위해 수행한 연산을 분석하여 표 2에 나타내었다.

Operation	Linear Gradient	Radial Gradient
Addition	812,787	1,110,439
Multiplication	570,507	1,210,219
Square root	7	19,207
Division	27,791	66,191

표 2. Gradient Image 에 사용된 연산 횟수

그림 5에 보여진 이미지를 렌더링 하는데 수행되는 시간을 Khronos Group에서 제공하는 레퍼런스와 비교 측정하여 표 5, 6과 7에 도시하였다. 본 논문에서 제안하는 Khronos Group에서 제공하는 레퍼런스는 OpenVG를 개발 할 개발자들이 개발 시 필요한 기능 및 연산의 구현에 초점을 두고 있으며, 속도는 고려하지 않고 구현되었다. 본 논문에서 제안하는 OpenVG는 스펙에서 제안하는 기능 구현 및 속도에 중점을 두고 설계하였다.

Image	Rendering Time
Tiger	208ms
Radial Gradient	41ms
Linear Gradient	37ms

표 3. 렌더링 연산 수행 시간

VI. 결 론

본 논문에서는 OpenVG 하드웨어 가속을 위한 파이프라인 분석 및 2D 벡터 그래픽스 파이프라인을 구성하는 알고리즘을 연구하여, OpenVG 래스터라이저 아키텍처를 구성하였다.

모바일 환경에 적합하도록, 적은 자원을 사용하여 소프트웨어와 하드웨어 구현에 있어 추가비용을 줄일 수 있는 24비트 부동 소수점 데이터 형식을 사용하였으며 OpenVG Specification에서 제안한 파이프라인을 기능별, 혹은 연산별로 그룹화 하여 하드웨어 구현에 적합한 새로운 파이프라인을 제안하였다.

본 연구에서 설계된 OpenVG는 크로노스 그룹에서 제공하는 Tiger Sample Image와의 결과 이미지 비교를 수행하여 동작 및 기능이 정확하게 동작함을 검증하였고, 검증 프로그램 수행을 통하여 OpenVG에서 제공해야 하는 여러 가지 기능들의 구현 및 검증을 수행하였다.

참고문헌

[1] Ashley Stevens, "*ARM MaliTM Graphics solution for Mobile device*", http:// www. arm.com 2007

[2] Khronos Group Inc. "*OpenVG Specification Version 1.0.1*" http://www.khronos.org/openv g/, January 2007

[3] Gene Frantz, Ray Simar "*Comparing Fixed and Floating Point DSPs*, Texas Instruments 2004

[4] ARM "*Fixed Point Arithmetic on the ARM*" , Application Note 33, ARM, September 1996

[5] Kiia Killio "*Scanline edge-flag algorithm for antialiasing*" EG UK Theory and Practice of Computer Graphics 2007