모바일용 2D Vector Graphics에 효율적인 Rasterizer 설계

박재규* · 이영호* · 정준모** · 이광엽*
*서경대학교 컴퓨터공학과 · **서경대학교 전자공학과

Effective design of 2d vector graphics rasterizer for mobile device

Jaekyu Park* · Yeongho Lee* · Junmo Jeong** · Kwangyeob Lee*
*Seokyeong University Department of Computer Engineering
**Seokyeong University Department of Electronic Engineering
E-mail : jkpark@skuniv.ac.kr

요 약

본 논문에서는 OpenVG Specification에서 제안한 파이프라인을 기능별, 혹은 연산별로 그룹화 하여 하드웨어 구현에 적합한 새로운 파이프라인을 제안하였다. 래스터라이저에서는 스캔라인 알고리즘과 엣지 플래그 알고리즘의 장점들을 포함하는 스캔라인 엣지 플래그 알고리즘을 구현하여 적용하였으며, Non-Zero 룰을 만족하기 위해 엣지의 방향에 따라 Winding 횟수를 기록하기 위한 추가 버퍼를 이용하였다. 또한, 래스터라이저는 안티 앨리어싱을 위해 슈퍼 샘플링 과정을 수행한다. 액티브 엣지 생성 시 클리핑을 동시에 수행하여 이후 과정에서의 불필요한 연산을 줄였고, 액티브 엣지들의 정렬을 수행하지 않는 방법을 사용하여 처리 속도를 향상 시켰다. 본 연구에서 설계된 OpenVG Rasterizer는 크로노스 그룹에서 제공하는 샘플 이미지들을 사용하여 검증하였다.

키워드

Vector Graphics, OpenVG, Scanline, fill rule

Ⅰ. 서 론

최근 모바일 기기의 성능이 향상되면서 작은 화면에 고화질의 유저 인터페이스 및 텍스트, 애니메이션과 같은 고수준의 그래픽 환경의 필요성이 증대되고 있다.

벡터 그래픽스는 픽셀 정보로 이미지를 표현하는 비트맵(래스터) 그래픽스와 다르게 수학 수식과 좌표를 이용하여 이미지를 표현한다. 기존의 비트맵 그래픽스 방식에서 애니메이션을 표현할 경우 각 프레임 별로 별도의 이미지를 제작해야 하는 반면, 벡터 그래픽스의 경우 하나의 이미지에서 좌표 값을 변화시킴으로써, 애니메이션을 표현할 수 있으며, 이미지 품질의 손실 없이 확대/축소가 가능하여 해상도에 관계없이 동일한 이미지를 표현할 수 있다.

벡터 그래픽스는 정점의 위치정보와 색상 정보 등으로 이미지를 표현하기 때문에, 각 픽셀의 값을 모두 가지고 있는 비트맵 방식에 비해 이미지 크기도 작아 모바일 기기에서 고 수준의 그래픽 환경을 구현하는데 적합한 방식이라 할 수 있다.

그림 1은 동일한 해상도의 벡터와 비트맵 방식의 이미지를 각각 8배 확대하여 비교한 것이다. 오른쪽 결과에 보이는 것과 같이 비트맵 방식의 경우에는, 픽셀을 확장하여 확대를 하기 때문에 이미지의 확대 시 손실이 발생한 것을 알 수 있다. 반면에 벡터 방식의 경우에는, 정점의 위치 정보를 가지고 수식적으로 확대하여 이미지를 표현하기 때문에, 손실 없이 확대가 가능하게 된다.

모바일 시스템에서 벡터 그래픽스에 대한 필요성이 증가함에 따라 그래픽스 표준 API들을 제정하는 Khronos Group에서 벡터 그래픽스 렌더링 부분을 표준화하고 가속화하기 위한 표준안으로 OpenVG를 발표하였다. OpenVG는 Flash와 SVG같은 벡터 그래픽 라이브러리들을 위한 하드웨어 가속 인터페이스를 제공하는 공개된 플랫폼 독립적인 API이다.[1]

본 논문에서는 OpenVG Specification에서 제안하는 파이프라인보다 빠른 성능을 나타낼 수 있도록 새로운 파이프라인을 제안한다. 또한, 래스터라이제이션 단계에서 OpenVG가 지원해야 하는 안티 앨리어싱과 2가지 채움 기능들을 지원하며, 불필요한 연산들을 제거함으로써, 모바일 기기에서 OpenVG API를 지원하기 위한 효율적인 래스터라이저 구조를 제안한다.

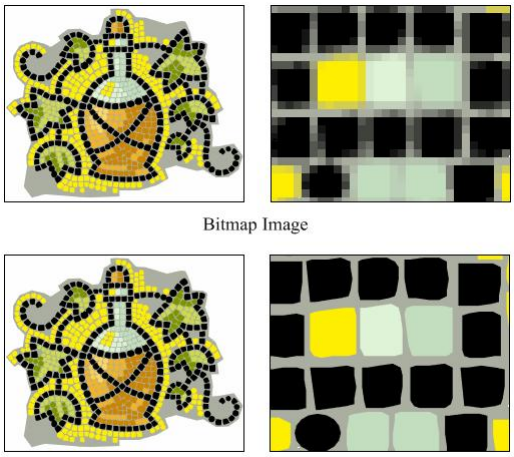

그림 1. 벡터/비트맵 이미지

Ⅱ. 파이프라인

OpenVG Specification에서 제안하는 파이프라인은 그림 2와 같다.[2]

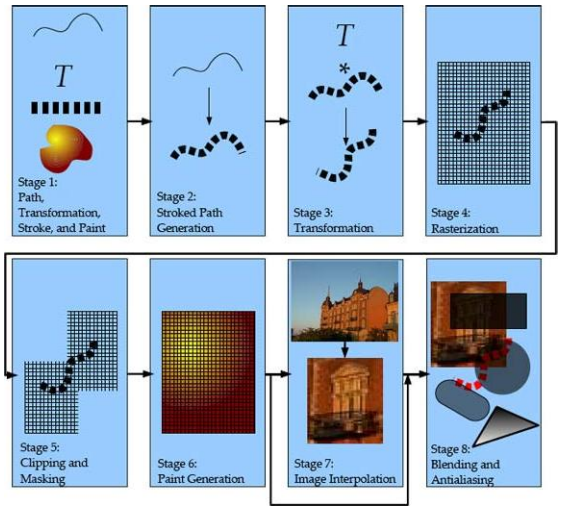

그림 2. OpenVG 파이프라인

이것은 OpenVG가 수행하는 기능들을 개념적으로 설명하기 위한 파이프라인으로써, 8단계로 이루어져 있다. 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 Specification에서 제안하는 OpenVG 파이프라인의 기능을 포함하면서 효율적으로 연산이 가능하도록 그림 3과 같은 파이프라인을 제안한다. 본 논문에서 제안하는 파이프라인은 총 6단계로 구성되어 있으며, 각각의 연산 단계에 부가적으로 OpenVG Specification에서 제안하는 파이프라인의 기능들이 포함되어 있다.

액티브 엣지 생성 단계에서 클리핑을 동시에 수행함으로써, 이후 과정에서 랜더링 영역을 벗어나는 불필요한 엣지들에 대한 연산을 수행하지 않는다. 또한, 래스터라이제이션 단계에서 안티앨리어싱을 위해 슈퍼샘플링을 수행하여 coverage value 값을 생성한다. 이것은 이후 픽셀 오퍼레이션 단계에서 사용된다.

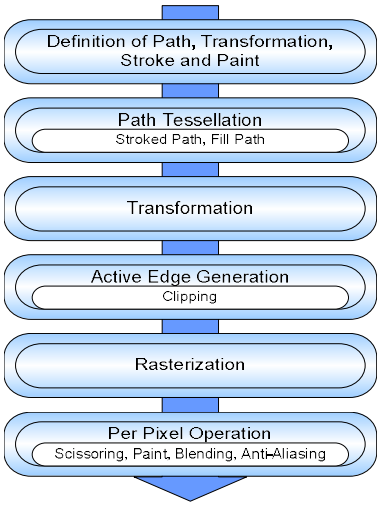

그림 3. 제안하는 파이프라인

Ⅲ. 래스터라이제이션

1. 슈퍼 샘플링

OpenVG에서 요구하는 3가지 타입의 Anti-Aliasing 모드를 지원하기 위해 Center Sampling, 4-Queens, 8-Queens 패턴을 사용하였으며, Super-Sampling 단계에서 샘플링을 수월하게 진행할 수 있도록 Transformation 단계에서 이미지를 Y축으로 8배 Scaling 하는 작업을 수행한다.

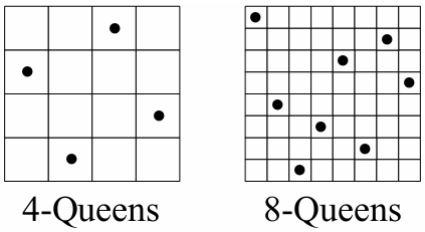

그림 4. N-Queen Sampling-Position

그림 5는 슈퍼 샘플링 수행 후 Mask Buffer에 저장될 Mask 값이 결정되는 과정을 보여준다. 8개의 샘플링 지점이 현재 그려질 이미지의 내부인지 외부인지를 판단하고, 포함되면 1 아닐 경우 0 값을 할당한다. 마스크 값은 Mask Buffer에 저장되며, 추후 픽셀의 Coverage Value로 변환되어

픽셀 연산 단계에서 사용된다.

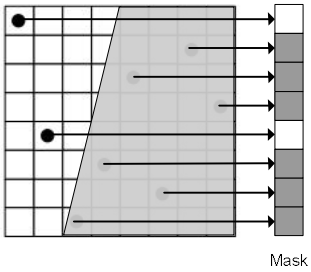

그림 5. Mask 값 생성 과정

2-1. 스캔라인 알고리즘

스캔라인 단위의 렌더링을 수행하지 않을 경우, 렌더링을 수행 할 해상도 크기의 커다란 버퍼를 가져야 한다. 8샘플링 안티 앨리어싱을 수행한다면 버퍼의 크기는 렌더링을 수행 할 화면의 크기보다 8배 큰 버퍼를 가지게 된다. 하지만 스캔라인 단위로 렌더링을 수행할 경우, 렌더링을 수행 할 해상도의 Width 크기의 버퍼만을 가지게 되므로 효율적이라 할 수 있다.

기본적인 스캔라인 알고리즘의 경우 스캔라인 단위의 연산 수행을 위해 AET(Active Edge Table)을 생성하고 이를 X좌표 순으로 정렬하는 과정을 필요로 한다.

2-2. 스캔라인 엣지 플래그 알고리즘

스캔라인 단위로 연산을 수행하기 위해 슈퍼샘플링을 포함하는 스캔라인 엣지 플래그 알고리즘을 적용하였다.[3][4][5] 기본적인 엣지 플래그 알고리즘에서는 스캔라인과 엣지가 교차하는 위치에 플래그 값을 기록해 두고 플래그 사이를 색상으로 채운다. 엣지 플래그 알고리즘에 대한 기본적인 방법은 그림 6에 도시하였다.

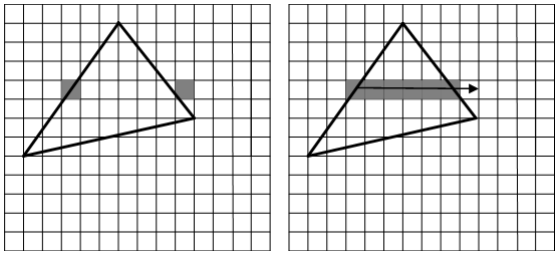

그림 6. 엣지 플래그 알고리즘

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 fillrule을 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 아닐 때 채우기를 수행한다.

2-3. 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.

Ⅳ. 액티브 엣지 생성

스캔라인 단위로 랜더링을 수행하기 위해서는 그려질 전체 Edge 중 어느 Edge들이 현재 스캔라인에 영향을 미치는 가를 파악해야 한다. 랜더링을 수행할 스캔라인에 영향을 미치는 Edge를 Active Edge라고 하며, Active Edge들을 각 스캔라인 별로 분류하여 저장해 놓은 공간을 Active Edge Table(AET) 이라 한다.

전통적인 스캔라인 알고리즘에서는 두 정점을 연결하는 엣지들을 생성하고 이를 Y 좌표에 따라 정렬(Sorting)하는 과정을 수행해야 한다. 벡터 그래픽을 표현하기 위한 Path의 개수가 많을 경우 엣지의 개수는 기하급수 적으로 늘어나며, 이를 Y 좌표에 따라 정렬하는 과정에서 많은 시간이 소요된다. 본 논문에서 설계한 2차원 벡터 그래픽스 Software에서는 AET을 구성하는데 있어서 Y축 좌표를 기준으로 정렬하는 기능을 수행하지 않는다.

그림 7은 전통적인 방식의 ET / AET 이다.

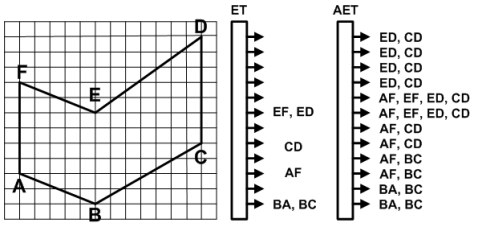

그림 7. 전통적인 방식의 ET / AET

이 방식에서는 Y축 좌표에 따라 정렬된 엣지들을 이용하여 ET(Edge Table)을 구성한다. ET은 Y축 해상도 만큼의 슬롯을 가지고 있으며, 각 엣지는 엣지가 시작되는 Y좌표에 해당하는 ET의 슬롯에 저장된다. 그 후 스캔라인 연산을 위해 AET에서는 Y좌표를 증가(감소) 시켜가며 현재 진행중인 스캔

라인 좌표에 해당하는 ET에 저장된 엣지들을 가져온 후, 이를 X좌표에 따라 다시 정렬을 수행하고 렌더링을 수행 한다.

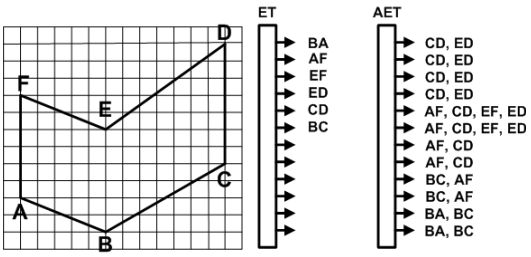

그림 8. 제안하는 방식의 ET / AET

본 논문에서 구현한 ET 와 AET의 구성 방식은 그림 8과 같다. 전통적인 방식과의 차이점은 ET를 구성하기 위해 엣지 정렬을 수행하지 않는 점이다. 따라서 모든 엣지들은 생성되는 순서대로 ET에 저장이 된다. 또한 AET를 구성함에 있어서도 X좌표 순으로 정렬을 수행하지 않는다. Y축 정렬과 X축 정렬을 하지 않지만 AET를 구성하는 엣지들은 같다. 따라서, 엣지 정렬을 수행하지 않는 만큼 연산 회수가 줄어 연산 속도에서 이득을 얻을 수 있다.

또한, 기울기가 0인 엣지는 생성하지 않는다. 그림 7과 8을 보면 기존 방식에선 5개의 엣지가 생성되나, 본 논문에서 구현한 방식에선 4개의 엣지가 생성된다. 엣지의 생성 개수가 적을수록 Active Edge를 생성하기 위한 연산을 수행하지 않으므로 성능이 향상된다.

추가적으로, Active Edge를 생성하는 과정에서 Clipping을 수행하여, 실제 렌더링 영역을 벗어나는 Edge에 대한 불필요한 연산을 수행하지 않는다.

Ⅴ. 검증 및 결과

그림 9는 Khronos Group에서 제공하는 샘플 이미지들이다.

그림 9. 테스트 이미지

이 그림들을 이용하여 본 논문에서 제안하는 래스터라이저의 성능을 측정하였고, 그 결과를 기존의 래스터라이저와 비교하여 표 1에 나타내었다.

Image	AmanithVG	Proposed
Tiger	28ms	24ms
Pelican	8ms	4ms
Picture	49ms	28ms
Dude	15ms	9ms

표 1. 이미지 렌더링 수행 시간

Ⅵ. 결 론

본 논문에서는 OpenVG 파이프라인 분석 및 2D 벡터 그래픽스 파이프라인을 구성하는 알고리즘을 연구하여 OpenVG API를 지원하는 Rasterizer 아키텍쳐를 구성하였다.

OpenVG Specification에서 제안한 파이프라인을 기능별, 혹은 연산별로 그룹화 하여 하드웨어 구현에 적합한 새로운 아키텍처을 제안하였으며, 액티브 엣지를 구성함에 있어, 정렬을 수행하지 않아 연산 속도를 향상 시켰다.

본 연구에서 설계된 OpenVG rasterizer는 크로노스 그룹에서 제공하는 샘플 이미지를 통해 결과 비교를 수행하여 동작 및 기능 검증을 수행하였다.

※ 본 논문은 서울특별시 산학연 협력사업과 지식경제부 시스템반도체 2010사업 지원으로 제작되었으며 IDEC 지원장비를 활용하였습니다.

참고문헌

[1][2] Khronos Group Inc. *"OpenVG Specification Version 1.0.1"* http://www.khronos.org/openvg/, January 2007

[3] B.Ackland and N.Weste, "The edge flag algorithm – a fill method for raster scan displays", IEEE Trans. Computers, January 1981

[4] K. Kallio, "Scanline edge-flag algorithm for antialiasing", Theory and Practice of Computer Graphics Conference, 2007

[5] M.E.Goss and K.Wu, "Study of supersampling methods for computer graphics hardware antialiasing", HP Labs Technical Reports, Dec. 2000