사례 발표

삼차원 모바일 라이브러리의 설계 및 개발

이 성 섭* 박 곤 호**

목 차

1. 서 론

이동 기기와 이를 위한 기반 기술이 급격히 발달함에 따라 사용자의 요구가 다양해지면서 다양한 그래픽 컨텐츠를 선호하게 되었다. 모바일 플랫폼에서 그래픽 기술이 많이 적용되는 대표적 어플리케이션으로 모바일 게임이 있지만, 현재 하드웨어의 한계로 이차원 영상 기반의 그래픽으로만 제작되고 있다. 그러나 시장의 추세와 기술 성장으로 모바일 폰에서도 삼차원 그래픽이 대중화될 전망이다. 모바일 플랫폼의 삼차원 기술은 게임뿐만 아니라 애니메이션이나 아바타 등 여러 어플리케이션에서 유용하게 쓰일 것이다.

본 논문에서는 모바일 플랫폼상에서 삼차원 어플리케이션을 개발하는데 사용되는 삼차원 그래픽 라이브러리를 개발하였다. 구현한 라이브러리는 기존의 모바일 플랫폼 삼차원 그래픽 라이브러리에서는 제공하지 않던 광원(Light Source), 그로쉐이딩(Gouraud Shading), 텍스처 매핑(Texture Mapping) 등의 고급 랜더링 기능을 제공한다. 또한 기존의 Java 3D 프로그래머가 쉽게 이용할 수 있도록 Java 3D의 클래스 구조와 유사하게 설계 및 구현하였다. 이를 이용하면 모바일 환경에서 사용될 수 있는 삼차원 어플리케이션을 만들 수 있다. 삼차원 게임은 물론 시뮬레이션, 애니메이션까지 적용될 수 있으며, 기존의 모바일 기반의 라이브러리와는 달리 객체 지향 디자인 장식을 고려하여 설계하였기 때문에 어플리케이션 개발이 아주 용이하다. 이번에 개발한 라이브러리의 기능은 삼차원 모델의 표현, 뷰잉 파이프라인(Viewing Pipeline), 조명모델, 쉐이딩(Shading), 은면제거(Hidden Surface Removal), 텍스처 매핑(Texture Mapping)의 여섯 부분으로 나눌 수 있다.

본 논문의 구성은 2절에서는 삼차원 라이브러리의 구성 요소 및 구현을 설명하고, 3절에서는 성능평가를 보여줄 것이다. 4절에서는 결론과 향후 연구 방향을 제시할 것이다.

2. 삼차원 라이브러리 구성 요소 및 구현

2.1 삼차원 물체 표현 방식

컴퓨터 그래픽스에서 삼차원 물체를 표현하는 방식에는 크게 다각형 메쉬(polygonal mesh), 매개

* (주)일렉트릭아일랜드 부설연구소 선임연구원
** (주)일렉트릭아일랜드 사장

변수를 이용한 곡면, 그리고 복셀(voxel)을 이용한 것 등이 있다. 이들 표현 방식 중에서 본 라이브러리에서는 다각형 메쉬방법을 이용하였다. 본 논문의 삼차원 라이브러리에서는 삼차원 물체 표현을 위해 다각형 메쉬 중에서도 삼각 메쉬(triangular mesh)를 이용하여 구현했다. 다각형 메쉬는 다각형을 연속적으로 연결하여 삼차원 물체를 표현하는 방식으로써, 그 표현 방법이 간단하고 어떤 곡면이든 근사할 수 있기 때문에 널리 사용되고 있다.

2.1.1 구현한 클래스

2.1.1.1 Shape3D 클래스

(1)물체에 대한 텍스처 정보를 저장하는 Appearance 객체와 물체의 기하 정보를 저장하는 Geometry 객체를 멤버로 가진다.

(2)setGeometry, setAppearance 메소드 - Geometry 멤버, Appearance멤버를 지정할 수 있다.

(3)getGeometry, getAppearance 메소드 - Geometry 멤버, Appearance멤버를 얻을 수 있다.

2.1.1.2 GeometryArray 클래스

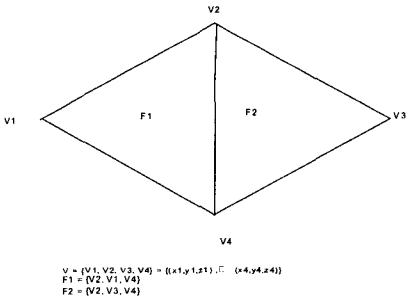
(1)꼭지점 위치, 색, 법선 벡터, 텍스처 좌표 정보와 면(face) 정보를 저장

(2)꼭지점의 위치는Coordinate, 색은 Col, 법선 벡터는 Normal, 텍스처 좌표는 Texture 멤버에 이들에 각각 대한 포인터 역할을 하는 정수 형의 멤버들이 있다.

(3)GeometArray 클래스의 하위 클래스로 PointArray, LineArray, TriangleArray, Quad Array 클래스가 있는 Java 3D와는 달리 본 삼차원 엔진에서는 삼각 메쉬만 지원하므로 GeometryArray 클래스의 하위 클래스는 삼각형에 대한 정보를 저장하는 TriangleArray 클래스만 구현하였다.

(4)setCoordinate, setColor, setNormal, setTexture Cooidinate 메소드 - 각 꼭지점의 위치, 색, 법선 벡터, 텍스처 좌표를 지정할 수 있다.

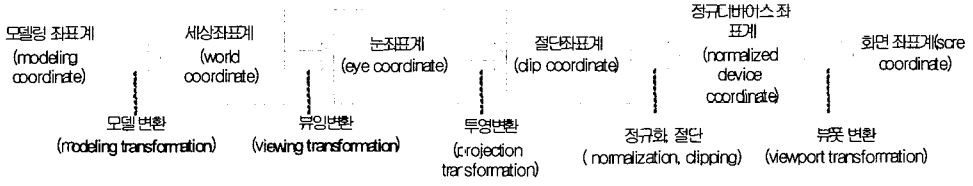
(5)getCoordinate, getColor, getNormal, get TextureCooidinate 메소드 - 각 꼭지점의 위치, 색, 법선 벡터, 텍스처 좌표를 얻을 수 있다.



(그림 1) 다각형 모델 표현 방법

2.2 뷰잉 파이프라인

삼차원 물체는 뷰잉 파이프라인(viewing pipeline) 처리를 거쳐서 화면에 그릴 수 있도록 이차원 상의 위치를 변환해야 한다. 즉, 삼차원 물체가 놓여 있는 모델링 좌표계는 뷰잉 파이프라인을 통해 화면 좌표계로 전환한다. 뷰잉 파이프라인 작업을 도식화하면 (그림 2)와 같다. 뷰잉 파이프라인 과정을 거치면 물체 좌표계가 화면 좌표계로 변



(그림 2) 뷰잉 파프라인 처리 과정

환된다. 최초의 물체 좌표계에서의 좌표로부터 화면 좌표계의 좌표를 얻기 위한 과정은 상당히 복잡한 과정을 거쳐야 얻어진다. 각 과정은 사진 촬영의 단계와 상당히 유사하다. 다음 (그림 3)은 뷰잉 파이프라인의 각 과정과 사진촬영 간의 관계를 도식화한 것이다.

2.2.1 구현한 클래스

2.2.1.1 ViewingPipeline 클래스

(1)삼차원 물체를 화면에 보여주는 작업을 위한 정적 메소드들로 이루어진 클래스다.

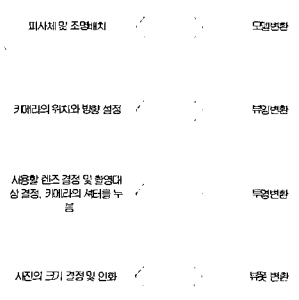
(2)render 메소드 뷰잉 파이프라인 작업의 시작점. 삼차원 물체를 표현하는 Shape3D객체를 인자로 받아 그것으로부터 Geometry객체를 추출하고, 투영 변환 행렬, 뷰잉 변환 행렬, 모델 변환 행렬을 곱하여 각 꼭지점에 곱해져야 할 변환 행렬을 구한다. 렌더링 모드를 나타내는 멤버 변수의 값에 따라 화면에 drawWireFrame, drawFlat, drawGouraud 메소드 중 하나를호출하며 앞에서 추출된 Geometry객체와 행렬 곱셈을 통하여 구하여진 변환행렬을 인자로 넘긴다.

(3)initialize 메소드 - render메소드가 실행되기 전에 호출되어 각 변환행렬 및 깊이 버퍼를 초기화한다.

(4)transform 메소드 물체를 구성하는 각 꼭지점마다 render메소드에서 구한 변환행렬을 곱하여 각 꼭지점의 정규화된 디바이스 좌표계의 좌표를 구한다.

(5)setLookAt 메소드 카메라를 설정한다.

(6)setFrustum 메소드 뷰 볼륨의 모양을 설정한다



(그림 3) 뷰잉 파이프라인의 각과정과 사진촬영과의 비교

2.3 조명

삼차원 물체를 사실적으로 표현하기 위해서는 그 물체가 발산하거나 반사하는 빛을 고려해야 한다. 물체의 재질 및 빛의 위치, 세기 등에 따라 우리의 눈이 인지하는 정보는 달라진다. 컴퓨터 그래픽스에서는 이러한 것을 고려하여 조명 모델을 제시하였다. 이 절에서는 본 논문에서 구현한 조명 모델에 대해서 살펴보도록 하겠다.

<표 1> 조명 모델

조명 모델	설 명
주변광 (ambient light)	주변광은 일정한 방향을 가지지 않고 비추는 빛이다. 물론 빛의 근원은 있지만, 광선은 일정한 공간이나 화면 등에 반사되어 비춰져서 방향성을 상실한다.
난반사광 (diffuse light)	난반사광은 특정 방향으로부터 비춰지지만 표면에 고르게 반사된다. 또한 주변광과는 달리 광원이 있어 광원의 위치와 세기에 따라 강도(intensity)가 달라진다. 본 프로젝트에서 구현한 조명 모델이다.
전반사광 (specular light)	전반사광은 난반사광과 마찬가지로 방향성은 있지만 특정 방향으로 정확히 반사된다. 주로 반들반들한 물체에서 관찰된다.

2.3.1 구현 메소드

2.3.1.1 illuminate 메소드

(1)Shading 클래스의 멤버 함수로서, 난반사광 모델을 이용하여 빛의 강도를 구한다.

(2)빛의 강도는 광원의 위치를 가리키는 좌표에서 꼭지점의 위치를 가리키는 좌표를 빼서 구한 벡터와 꼭지점의 법선 벡터를 내적하여 빛의 강도를 구한다. 단, 두 벡터를 내적하기 전에 두 벡터는 정규화 과정을 거쳐서 그 크기가 1인 단위 벡터가 된다.

2.3.1.2 setPLight, getPLight 메소드

(1) ViewingPipeline의 멤버 함수로서, 광원의 위치를 지정하거나 얻는다.

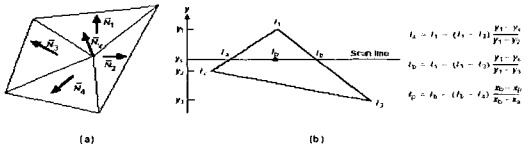
(2) 본 프로젝트는 광원이 하나 존재하도록 구현하였으므로 단 하나의 광원의 위치만 지정할 수 있다.

2.4 쉐이딩

사실적인 영상을 만들기 위해서는 물체 고유의 색과 조명의 위치 및 방향을 고려하여 각 다각형 안을 적절한 색으로 채워야 한다. 이렇게 다각형의 색상을 결정하는 것을 쉐이딩이라고 한다. 본 논문에서는 삼차원 라이브러리에서는 널리 사용하고 있는 플랫 쉐이딩과 그로 쉐이딩을 구현하였다.

〈표 2〉 조명 모델

쉐이딩 모델	설명 및 장단점
플랫쉐이딩	- 플랫 쉐이딩은 각 다각형마다 대해 조명을 계산하여 색 강도(intensity)를 결정하여 이 색상으로 다각형을 채우는 방식이다. 즉 평면을 이루는 다각형의 법선 벡터는 계산하고, 이 법선 벡터를 이용하여 조명을 계산한다. 하나의 다각형마다 조명을 계산하기 때문에 계산량이 적다는 장점이 있지만, 다각형의 경계 부분에서 색이 급격히 변하는 단점을 가지고 있다.
그로쉐이딩	- 그로 쉐이딩은, 평면의 법선 벡터를 이용하는 방식과는 달리, 각 꼭지점의 법선 벡터를 이용하여 색상을 계산하는 방식이다. 꼭지점에 인접하는 평면 법선 벡터를 평균하여 꼭지 점의 법선 벡터를 구한 후, 다각형을 이루는 각 꼭지점의 색 강도를 결정한다. 그런 다음 이를 보간하여 다각형 내부의 각 점에서의 색 강도를 결정한다. 이 방법은 조명 계산을 각 꼭지점마다 하므로 조명 계산량은 적지만, 다각형 내부의 각 점에 대해 보간을 하기 때문에 많은 계산을 필요로 한다. - 그로 쉐이딩의 장점은 다각형 내의 각 점마다 다른 색으로 채우기 때문에 플랫 쉐이딩에 비해 사실감 있는 표현을 할 수 있다는 점이다.



(그림 4) 그로 보간법에 의한 색강도의 결

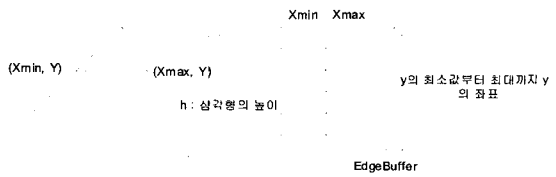
2.4.1 구현 클래스와 메소드

2.4.1.1 Shading 클래스

(1) 플랫 쉐이딩과 그로 쉐이딩을 위한 클래스다.

(2) fillTriangleFlat 메소드

Point4f 배열 객체, Color 객체, Graphics 객체를 인자로 받아 삼각형을 하나의 색으로 채운다. 3개의 Point4f를 원소로 가지는 배열은 세 개의 꼭지점에 대응된다. 삼각형을 화면에 그리기 때문에 Point4f에서 x, y 좌표만 쓰인다. 우선 삼각형의 높이를 구해야 한다. 가장 높은 점과 가장 낮은 점의 y 좌표를 먼저 구하고 저장한다. 삼각형의 높이는 최대 y 좌표에서 최소 y 좌표를 빼서 계산한다. 그 다음 삼각형을 채우기 위해서 경계점을 찾는 작업이 필요하다. 경계점은 삼각형의 세 변의 자취에 의해 만들어진 점과 일치한다. 경계 점은 삼각형의 세 변의 자취에 의해 만들어진 점과 일치한다. 여기서 우리는 브레젠햄(Bresenham)의 선 긋기 알고리즘을 세 번 이용하였다. 그러나 직접 화면에 점을 찍는 대신, 에지 버퍼(edge buffer)에 점을 찍었다(그림 5 참조). 에지 버퍼는 각각의 y 좌표마다 최소와 최대 x 좌표를 저장한다(삼각형은 볼록 다각형(convex polygon)이기 때문에 각각의 y 값에 대응되는 2개의 x좌표가 존재한다). 에지 버퍼가 한번 채워지면 y의 최소값부터 최대값까지 y 값을 하나씩 증가시키며 최소 x 값부터 최대 x 값까지 수평 스캔라인을 주어진 색으로 그린다.



(그림 5) 삼각형을 그리기 위해 필요한 Edge Buffer의 구

(3)fillTiangleGouraud 메소드

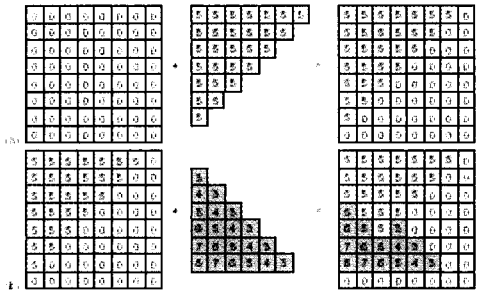
fillTriangleGouraud 메소드는 보간을 수행하여 결정된 색으로 삼각형을 채운다. 삼각형 내부의 각 픽셀에 대해 보간을 수행해야 하므로 fillTriangleGouraud 메소드가 fillTriangleFlat 메소드보다 복잡도가 높다. fillTriangleGourad에서 주어진 꼭지점들은 y 값들에 의해 정렬된다. 여기서 가장 높은 꼭지점을 v0, 가장 낮은 점을 v2, 다른 하나를 v1이라 한다. 삼각형의 높이는 v0의 y 값에서 v2의 높이 값을 빼서 쉽게 구할 수 있다. 그러고 나서 깊이 값, 색 강도, 텍스처 좌표에 대해 보간이 수행된다. 각 변의 깊이, 색, 텍스처 좌표에 대한 정보는 모드 에지 버퍼에 저장된다. 에지 버퍼가 채워지면, 플랫 쉐이딩과 마찬가지로 y 값을 증가시키며 수평 스캔 라인을 긋는다. 그러나 스캔 라인이 쏘아질 때, 깊이, 색, 텍스처 좌표가 다시 보간되어야 한다. 각 픽셀은 각각 보간되어진 깊이, 색, 텍스처 정보에 의해 그려진다.

2.5 은면 제거

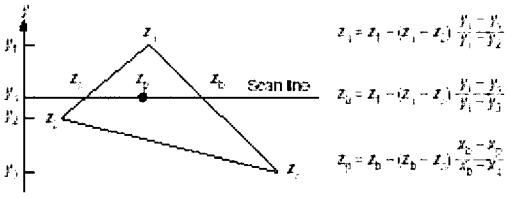
〈표 3〉 은면 제거 알고리즘

은면 제거 알고리즘	설명 및 장단점
Z 버퍼 알고리즘	뷰잉 파이프라인 과정에서 각 꼭지점의 깊이, 즉 투영된 이후의 z 좌표는 그 꼭지점이 화면에 나타나게 될 픽셀에 해당되는 Z 버퍼에 저장된다.(그림 6 참조). 또한 세 꼭지점이 이루는 삼각형 내부의 깊이 값은 각 꼭지점을 보간하여 저장한다(그림 7 참조). 픽셀이 그려질 때마다 그 픽셀의 깊이 값은 그것에 해당하는 Z 버퍼에 저장되어 있는 깊이 값과 비교된다. 만약에 그려지는 픽셀의 깊이가 저장된 Z-버퍼에 저장된 깊이보다 크면 그려진다(모든 깊이 값은 음수이고 더 큰 깊이 값은 그 픽셀이 눈으로부터 더 가까이 있다는 것을 의미한다). 만약 픽셀이 그려지도록 결정되면, 그 픽셀의 깊이 값은 Z 버퍼에 저장되고 화면(혹은 화면버퍼)에 그려진다.
후면 선별 알고리즘	(그림 8)에서 평면의 법선 벡터의 z 값이 양수를 가리키는 평면 G, H, E, C는 전면, 평면의 법선 벡터의 z 값이 음수를 가리키면 후면이 된다. 후면은 카메라의 위치에서 보이지 않는 면이므로 렌더링할 필요가 없을 경우가 많다. 따라서 Z 버퍼 알고리즘을 적용하기 전에 후면들을 제거한다면 계산량을 훨씬 줄일 수 있다. 이러한 방법을 후면 선별이라 한다.

화면에서 보이지 않은 다각형들은 제거하는 것을 은면 제거라고 한다. 은면 제거의 방법에는 Z-버퍼 알고리즘과 후면 선별이 있고, 본 논문은 이 두 가지 알고리즘을 모두 이용하여 은면 제거를 구현하였다.



(그림 6) Z버퍼 알고리즘을 이용한 Z버퍼 업데이트의 예



(그림 7) 깊이 값의 보간

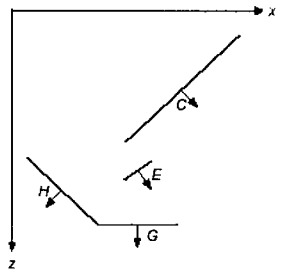
2.5.1 구현 메소드

2.5.1.1 ViewingPipeline 클래스의 backFaceCull 메소드

(1)후면선별 알고리즘을 구현한 클래스다.

(2)후면 제거 연산은 두 벡터를 필요로 한다. 우선, 법선 벡터가 사용된다. 여기서 우리는 어떤 면에 대한 법선 벡터를 N이라 한다. 또한 카메라로부터 면을 향하는 벡터를 R이라 한다. 여기서, 우리는 R을 구하기 위해 무작위로 면을 구성하는 한 꼭지점을 고르고 카메라 위치로부터 그 꼭지점 좌표를 뺀다. N과 R은 반드시 정규

화(normalize)되어야 한다. 마지막으로 우리는 두 벡터간에 각을 계산할 수 있다. 만약 그 각이 90도를 넘으면, 그 면은 뷰잉 파이프라인에서 제거된다.



(그림 8) 전면과 후면

2.6 텍스처 매핑(Texture Mapping)

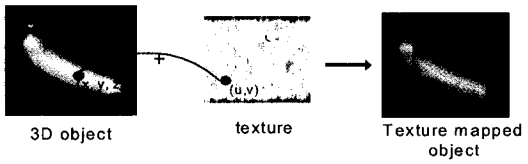
컴퓨터 그래픽스 렌더링의 목표 중 하나는 실제의 상황을 카메라로 촬영한 듯 사실적인 영상을 생성하는 것이다. 물체의 사실성을 높이기 위해서 다각형 메쉬 모델의 다각형 수를 늘리는 방법이 있다. 하지만 이는 많은 계산량과 메모리 공간을 필요로 한다. 뿐만 아니라 정교한 모델을 만들기도 어렵다. 다른 방법으로는 텍스처 매핑이 있다. 이는 다각형 메쉬 모델을 렌더링할 때 미리 생성해 놓은 이미지 데이터를 적용하여 화면을 생성하는 방법이다. 텍스처 매핑 기법 중 널리 쓰이는 것이 이차원 텍스처 매핑이다. 이차원 텍스처 매핑은 미리 생성된 이미지를 다각형 메쉬 모델이 덮어 싸서 마치 모델 표면에 그러한 이미지가 존재하는 것처럼 보이게 하는 것이다.

2.6.1 이차원 텍스처 매핑 과정

이차원 텍스처 매핑을 하는데 있어 가장 먼저 해야 할 작업은 텍스처 이미지와 기하 물체간의 대응관계를 설정하는 것이다. 다시 말해서 텍스처를 물체에 입힐 때, 물체 표면 상에 존재하는 물체 공간의 점 (x, y, z)을 대응되는 텍스처 이미지의 점 (u, v)에 효과적으로 대응시키는 것이다. 물체가 화면에 투영된 이후 각 픽셀에 대한 RGB값에 대해 적절한 색으로 스크린에 그려주는 작업을 래스터화(rasterization)이라고 한다. 텍스처 매핑 기법을 적용할 때는 바로 래스터화를 하는 시점으로써 각 픽셀에 해당하는 텍스처 색깔을 구해, 물체의 기반 색깔과 혼합을 하여 텍스처를 입혀준다.

2.6.2 대응 함수

대응 함수는 (그림 9)와 같이 삼차원 물체 상의 임의의 좌표에 대응되는 텍스처의 좌표를 결정하는 함수다. 다각형 메쉬 모델의 각 꼭지점은 텍스처 좌표를 가진다. 본 프로젝트에서는 각 꼭지점의 텍스처 좌표는 이미 주어진다고 가정한다. 실제로 마야 같은 모델링 툴을 이용하여 모델 데이터를 만들면 텍스처 좌표도 같이 저장된다. 각 꼭지점은 해당 텍스처 좌표가 가리키는 텍스처 이미지 픽셀로 대응되고 삼각형 내부의 픽셀은 삼각형의 꼭지점들을 보간하여 만들어진 텍스처 좌표가 가리키는 텍스처 이미지 픽셀로 대응된다. 보간하는 방법은 앞에서 설명한 그로 쉐이딩에서 사용한 방법과 유사하다.



(그림 9) 이차원 텍스처 매핑 함수

2.6.3 구현 클래스와 메소드

2.6.3.1 Texture2D 클래스

(1)이차원 텍스처 관련 연산을 위한 클래스다.

(2)setImage 메소드 png 형식의 텍스처 이미지를 로딩하여 각 픽셀의 RGB값을 정수 형 배열에 저장한다.

(3)getPixel 메소드 텍스처 좌표를 인자로 받아서 텍스처 이미지에서 텍스처 좌표에 해당하는 RGB값을 반환한다.

2.6.3.2 구현방법

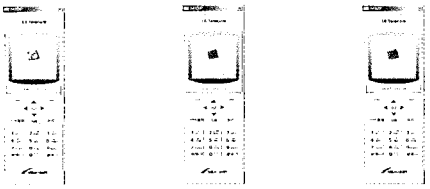
Texture2D클래스를 이용하여 이차원 텍스처 이미지가 로딩하고, 그로 쉐이딩 루틴에서 텍스처 좌표 보간을 하여, 텍스처 좌표가 가리키는 이미지의 RGB값을 받아서 칠한다.

3. 예제 및 성능 평가

3.1 예제 및 실행

3.1.1 랜더링 모드 전환

예제 프로그램에서는 세가지 렌더링 모드를 지원한다. 랜더링 모드는 와이어 프레임, 플랫 쉐이딩, 그리고 텍스처 매핑 등이 있다(그림 10 참고).

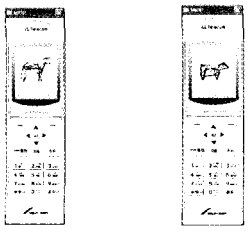


(그림 10) 렌더링 모드 변환(왼쪽부터 와이어 프레임, 플랫 쉐이딩, 텍스처 매핑)

3.1.2 카메라 위치 변환과 모델 파일 바꾸기

카메라 위치 변환은 (그림 11)과 같이 핸드폰의 방향키로 카메라를 상하좌우로 이동할 수 있고, 1번 버튼을 누르면 카메라가 앞으로, 3번 버튼을 누르면 카메라가 뒤로 이동한다. 물체가 절단 영역벗어나면 절단이 일어난다. 모델 파일 바꾸기는 제공되는 model.obj파일을 OBJ 파일 형식의 다른 파일로 대체한다(그림 12 참고).

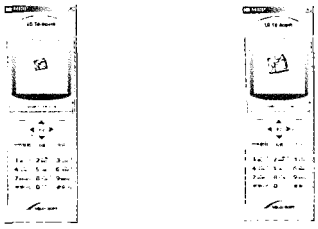
(그림 11) 절단이 일어나는 경우



(그림 12) 복잡한 모델을 로딩한 경우

3.2 결과 분석

본 논문의 삼차원 라이브러리를 실제 핸드폰으로 다운로드하여 60초 동안 여러 개의 화면을 생성해 낼 수 있는지 성능을 평가했다.



(a) (b)

(그림 13) 프레임률 검사 예제

모델이 화면에 디스플레이 되는 부분이 적은 경우(그림 13(a)) : 삼차원 라이브러리를 이용하여 정육면체로 테스트한 경우

	60초당 프레임 수	프레임/초
wireframe	15	0.25
flat shading	12	0.2
gouraud shading(with texture)	1.2	0.02

모델이 화면에 디스플레이 되는 부분이 큰 경우(그림 13(b)) : 위의 모델을 1.5배 확대한 후 테스트한 경우

	60초당 프레임 수	프레임/초
wireframe	15	0.25
flat shading	9	0.15
gouraud shading(with texture)	0.42	0.0069

와이어 프레임의 경우는 모델 크기에 상관없이 일정한 프레임률이 나온다. 그러나 쉐이딩이 들어갈 경우에는 픽셀 연산을 해줘야 하므로 모델의 크기에 따라 결과가 많이 달라진다. 특히 텍스처 매핑과 그로 쉐이딩 렌더링을 동시에 수행할 경우 속도가 1/3

로 줄어들었다. 또한 플랫 쉐이딩 같은 경우도 모델이 더욱 커진다면 그 속도는 훨씬 줄어들 것이다. 왜냐하면 모든 뷰잉 파이프라인을 거쳐서 화면에 픽셀을 찍을 때 RGB값을 계산하는데 가장 많은 시간이 걸리기 때문이다. 텍스처 매핑을 한 그로 쉐이딩 모드에서는 픽셀 하나의 RGB 값을 결정하기 위해서 깊이, 법선, 텍스처 등을 보간해줘야 하고, 이를 하기 위해서는 많은 소수점 연산을 해야 한다. 하지만 KVM에서는 부동소수점 연산을 지원하지 않기 때문에 LG ez-java MIDP에서 제공하는 MathFP라는 클래스를 이용하였다. MathFP는 정수 형의 변수와 시프트(shift) 연산을 이용하면서 부동 소수점 연산을 해 준다. 실제로 정수 형의 변수를 부동 소수점 연산에 사용하기 위해서는 가상 머신이나 하드웨어에서 지원하는 것보다 불필요한 연산을 수행하기 때문에 많은 시간이 필요하게 된다.

4. 결론 및 향후 연구 방향

본 논문은 모바일 플랫폼 기반 삼차원 어플리케이션 개발에 이용 가능한 삼차원 라이브러리를 개발하였다. 결과적으로 삼차원 모형을 모바일 폰 화면에 나타낼 수 있게 되었고, 와이어 프레임은 물론 플랫 쉐이딩, 그로 쉐이딩, 텍스처 매핑 등 고급 렌더링 기능까지 구현하였다. 성능 평가를 통하여 본 논문의 삼차원 라이브러리를 이용하여 삼차원 렌더링 기능을 제대로 수행함을 보였다. 본 논문의 결과물로서 나온 삼차원 라이브러리는 모바일 플랫폼 기반의 삼차원 어플리케이션에서 이용이 가능할 것이다. 둠이나 퀘이크와 같은 삼차원 게임은 물론 삼차원 실시간 지리 정보 시스템, 아바타 무선 인터넷 채팅 등에 활용될 가능성이 충분하다. 3절의 성능평가를 보면 알 수 있듯이 렌더링 과정이 복잡할수록 실시간 어플리케이션에 적용하기는 어렵다. 이는 구현된 알고리즘의 문제라기보다는 휴대폰 하드웨어가 아직은 삼차원 그래픽에 적합한 기능을 제공하지 못하기 때문이다. 본 프로젝트에서 구현한 알고리즘은 대부분 그래픽스 시스템에서 자주 이용되며, 그 효율성 또한 이들 시스템을 통해 이미 입증되었다. 앞으로 삼차원 가속 기능 및 부동 소수점 연산을 지원하는 모바일 폰이 개발된다면 충분히 실시간 어플리케이션에 적용할 수 있을 것이다.

저자약력

이 성 섭

1998년 전남대학교 컴퓨터공학과(공학사)
2000년 전남대학교 대학원 컴퓨터공학과(석사)
2001년 (주)쉘컴정보통신 부설연구소
현재 (주)일렉트릭아일랜드 부설연구소 선임연구원
관심분야 : 실시간 스티리밍, 모바일 컨텐츠
이 메 일 : feelcool@electricisland.com

박 곤 호

1984년 조선대학교 전자공학과(공학사)
1990년 캐나다 토론토대학 데이터통신
1996년 정보통신부 서기관
2001년 한솔 PCS 상무
2002년 (주)위즈케뮤니케이션 사장
현재 (주)일렉트릭아일랜드 사장
관심분야 : 모바일 컨텐츠, 모바일 전자상거래
이 메 일 : khpark@electricisland.com