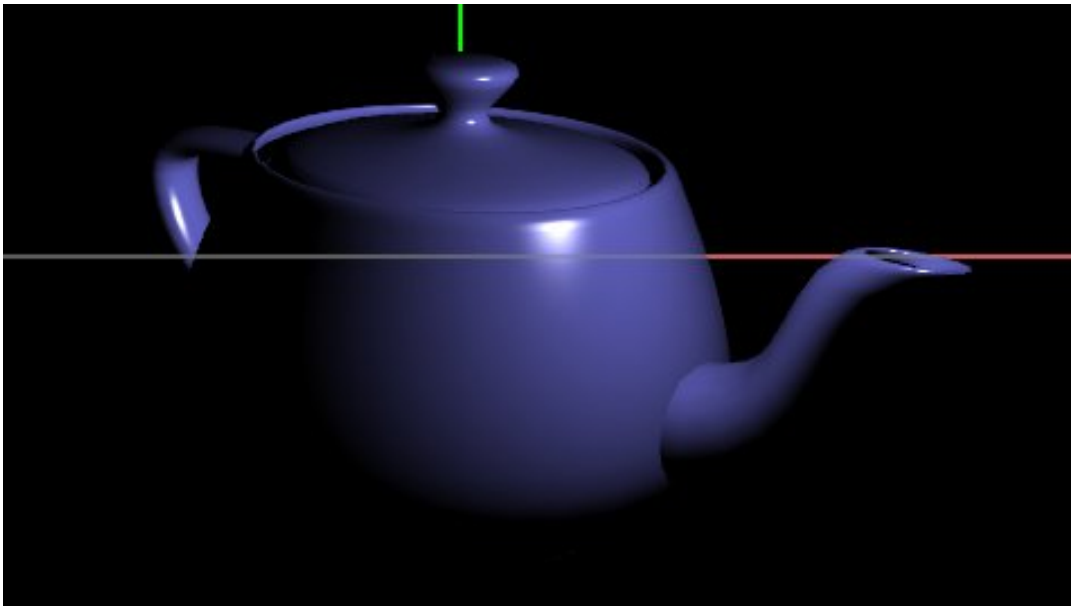


OpenGL ES2.0 기초 강좌

나의별

소개글

OpenGL ES 2.0의 Shader Programming을 이용하여, MFC와 Android용 기초 강좌

1. MFC EGL 구성

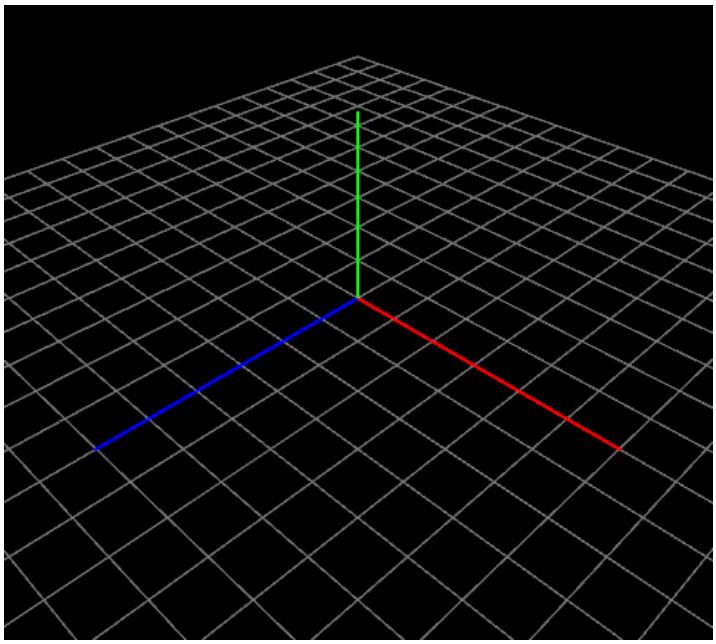
2. Android EGL 구성

목차

[GLES20] 01. Android 3D Programming을 위한 MFC 프로젝트 설정

2011.11.02 20:20

OGLES20Template



네이버 카페 3D Programming의 매니저 이신 바른생활님이 Android용으로 GLSL 강좌를 정리 해 보는 것도 좋을 것 같다는 추천이 있어서 한번 도전해 봅니다 ^^;

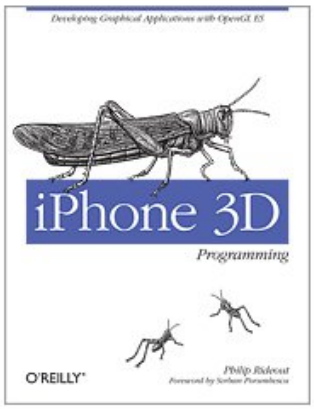
구성은 바른 생활님이 http://cafe.naver.com/gld3d/401 페이지 에서 만들고 있는 내용 위주로 진행할 까 생각 중입니다.

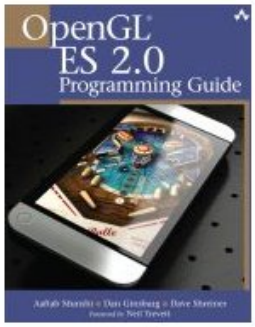
중간 중간 NeHe Tutorial과 PowerVR SDK의 내용도 추가해 볼 생각 입니다.

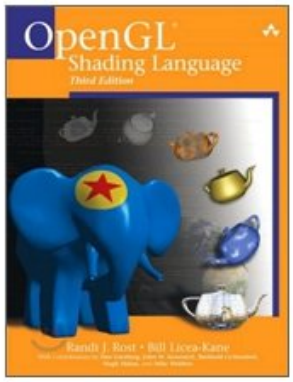
이곳에 정리하는 내용은 3D의 기본적인 사항인
PROJECTION MATRIX, MODEL MATRIX, VIEW MATRIX, EULER/QUATERNION TRASFORMATION,
FRAME BUFFER OBJECT, VERTEX BUFFER OBJECT, COLOR BUFFER ARRAY, DEPTH BUFFER ARRAY, STENCIL BUFFER ARRAY, VERTEX, TEXTURE, LIGHT, BLEND 등에 대해서 일일히 서술하지는 않도록 하겠습니다.

위의 내용은 바른생활님 3D Programming 카페나 기타 다른 OpenGL ES 용 도서를 참고하셔서 병행 공부해 나가면 좋을 것 같습니다.

저는 OpenGL Super bible 과 iPhone 3D Programming Developing Graphical Applications With OpenGL ES(Oreilly), OpenGLES 2.0 Programming Guide(Addison Wesley), OpenGL Shading Language 3/E(Addison Wesley) 등의
책을 추천합니다.







참고로 제가 구현한 Android Native Renderer 엔진은 위의 iPhone 3D Programming 책에서 소개한 Native Renderer를 기초로 만들었으며, 차후 혹여나 상용화를 하실 경우 라이센스 상에 문제가 발생할 수 있습니다.
(책에 보면, 저자가 소스에 대해서 O'REILLY 출판사에 권한이 있음으로,
소스 사용시 문의는 해달라는 요청 글등이 있었던것 같네요. ^^;)

들어가기에 앞서서 밝혀 두지만, 저 또한 3D를 마스터 한 상태는 아닌 1인 이기 때문에, 내용 자체가 저의 주관적으로 이해한 것을 바탕으로 정리될 것입니다.

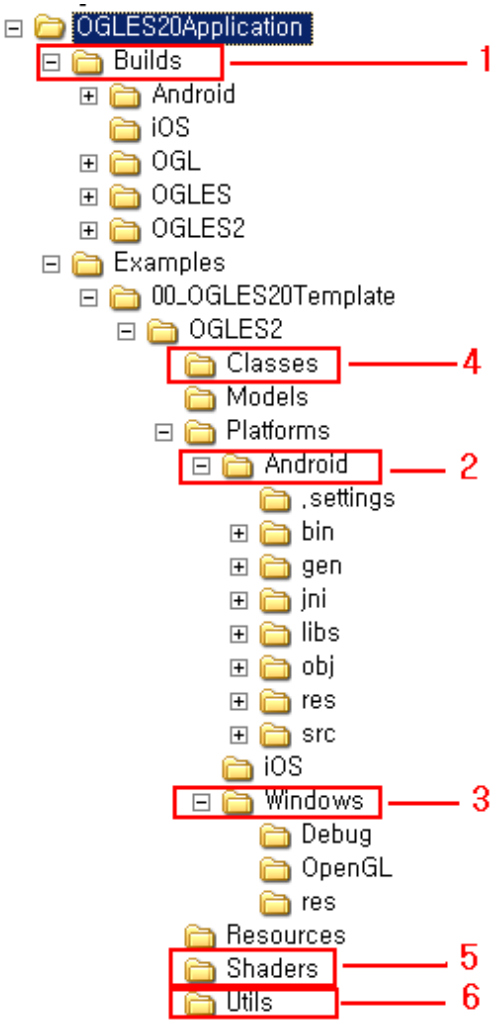
혹 틀린 부분이 있다면 바로 바로 지적해 주십시요 ^^;

1. 폴더 구성

기본적으로 프로젝트를 Android Java 3D가 아닌, Native Renderer로 3D를 구성할 예정 입니다.
Java로 짤 경우, Debugging이 쉬운 장점이 있지만, Native Renderer 보다 속도가 좀 느리고, 아이폰이나 윈모 등 다른 플랫폼으로 이식하기에는 어렵다는 단점도 있습니다.

이에 Native Renderer를 C++로 짜서 Android, iOS, Windows Mobile 등에 이식하게 편하게 만들어 볼 예정 입니다.
Native Renderer의 경우, Debugging이 어려운 점이 있지만, Renderer 부분은 MFC로 호환 되게금 해서 Visual Studio로 연계할 예정 입니다.

프로젝트의 기본 구성은 아래와 같습니다.



폴더 설명

1. Builds : OpenGL, EGL Library 와 android ndk-r5b에서 사용할 stlport등이 있다.
향후 추가적인 prebuilt library 등은 이 폴더에 추가할 예정 이다.

2. Android : Android Project 폴더 이다.
안드로이드로 App 개발 시 필요한 Activity 등이 있으며, Native Renderer용 JNI도 이곳에 있다.

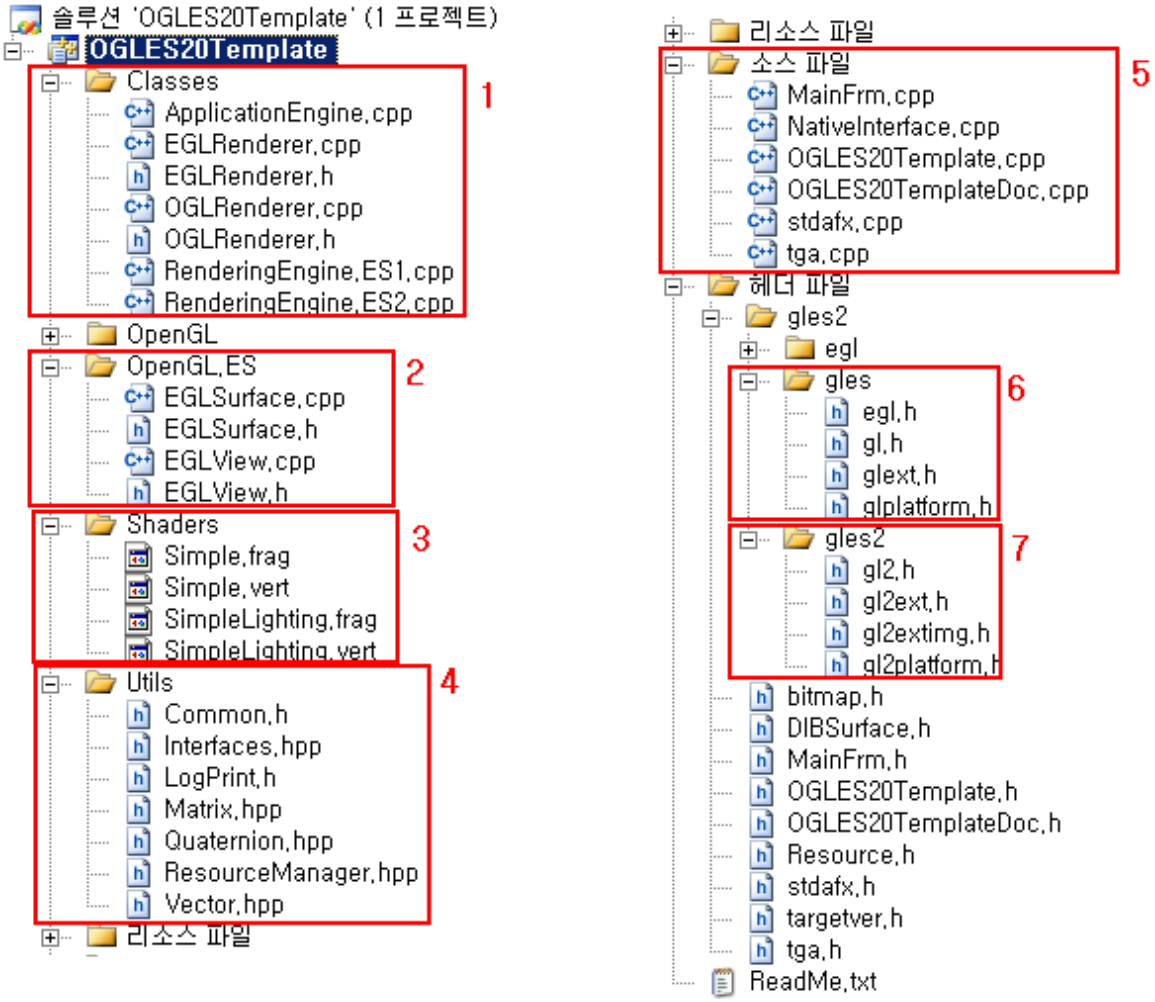
3. Windows : MFC Project 폴더 이다.
MFC을 구성하는 Frame, View, Document 파일등과 GLView, EGLView, Native Renderer Interface 등이 있다.

4. Classes : Renderer 소스가 있는 폴더 이다.
이 폴더는 MFC와 Android에서 공용으로 사용한다.

5. Shaders : Shader Language인 vertex shader와 fragment shader가 있는 폴더 이다.
이 폴더는 MFC와 Android에서 공용으로 사용한다.

6. Utils : Renderer를 구성하는데 필요한 수학 함수, 리소스 매니저 등과 같은 도우미 모듈등이 있는 폴더 이다.
이 폴더는 MFC와 Android에서 공용으로 사용한다.

2. MFC 프로젝트 구성



프로젝트 패키지 설명

1. Classes : 렌더링 클래스가 모여 있다.
템플릿 예제에서는 Grid 형의 바닥과 X-Y-Z 축 을 그리고 있다.
실제 사용되는 파일은 RenderingEngine.ES1.cpp와 RenderingEngine.ES2.cpp이며,
EGLRenderer와 OGLRenderer는 더 이상 사용 되지는 않는다.
(렌더러 모듈은 iPhone 3D Programming 책 내용과 유사한 구성을 따르고 있으며, 책의 1장 내용과 비교해서 보시면 됩니다.)
이 폴더는 MFC와 Android에서 공용으로 사용한다.

2. OpenGL.ES : MFC에서 사용할 EGLView로서 libEGL.lib 구현화 한 부분이다.
EGView.cpp는 MFC의 CView를 상속 받은 User View이다.
EGLSurface.cpp는 EGL을 생성한 Surface 모듈이다.

3. Shaders : GLSL용 Shader 파일들이 있다.
이 폴더는 MFC와 Android에서 공용으로 사용한다.

4. Utils : Renderer를 구성하는데 필요한 수학 함수, 리소스 매니저 등과 같은 도우미 모듈등이 있다.
이 폴더는 MFC와 Android에서 공용으로 사용한다.

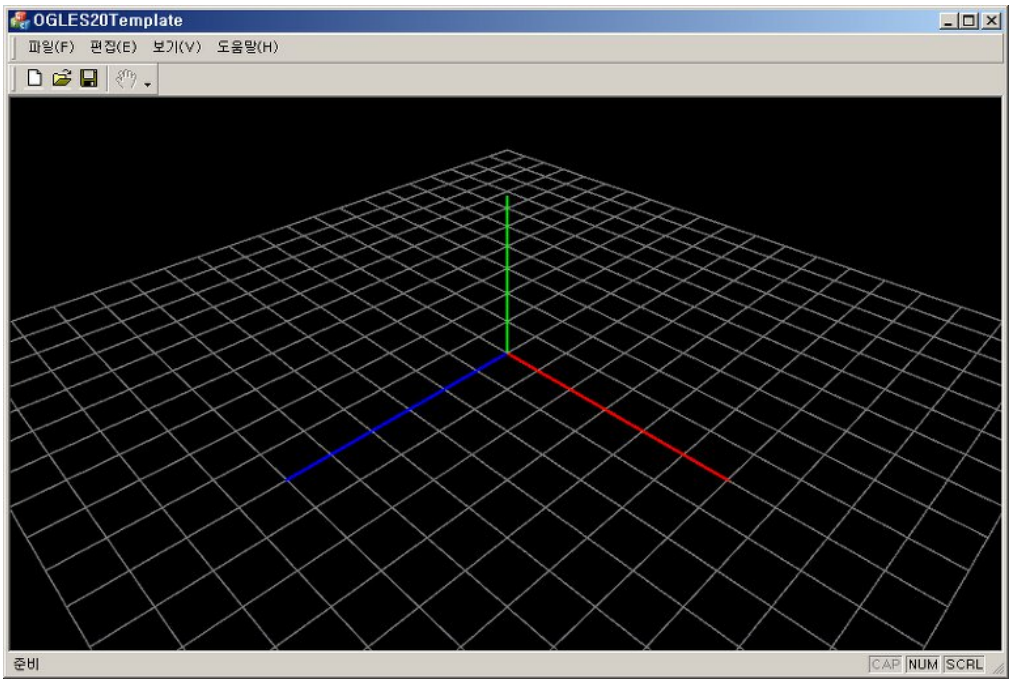
5. 소스 파일 : MFC의 Frame, View, Document의 소스 부분이다.
이중 NativeInterface.cpp는 ResourceManager에서 bmp와 tga 파일을 읽어 올때 거쳐 가는 Bridge 역활을 한다.

6. 헤더 파일.gles : OpenGL ES 1.0용 header 파일 이다.

7. 헤더 파일.gles2 : OpenGL ES 2.0용 header 파일 이다.

3. 실행 결과

MFC 에서 실행한 결과는 다음과 같습니다.



4. 예외사항 정리

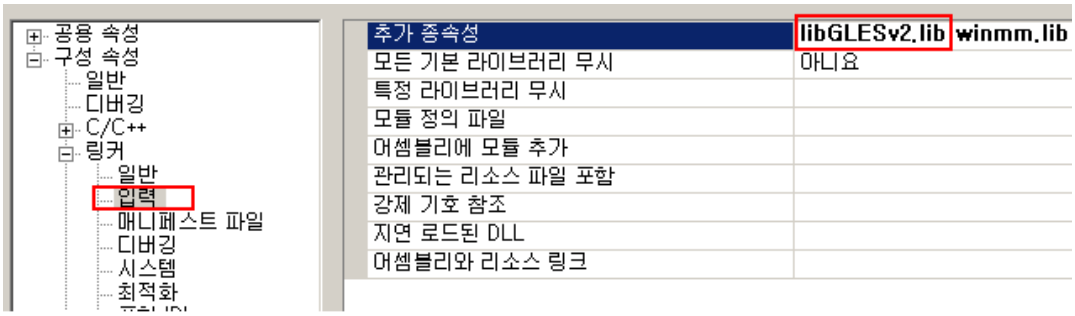
GLES 2.0은 GLES1.0의 하위 호환성을 배제하고 만들어 졌습니다.
이때문에, GLES2.0세서는 기존 GLES1.0에서 지원하는 glMatrixMode, glPushMatrix, glPopMatrix, glRotatef, glTranslatef, glScalcef 등등등 많은 API가 지원하지 않습니다.

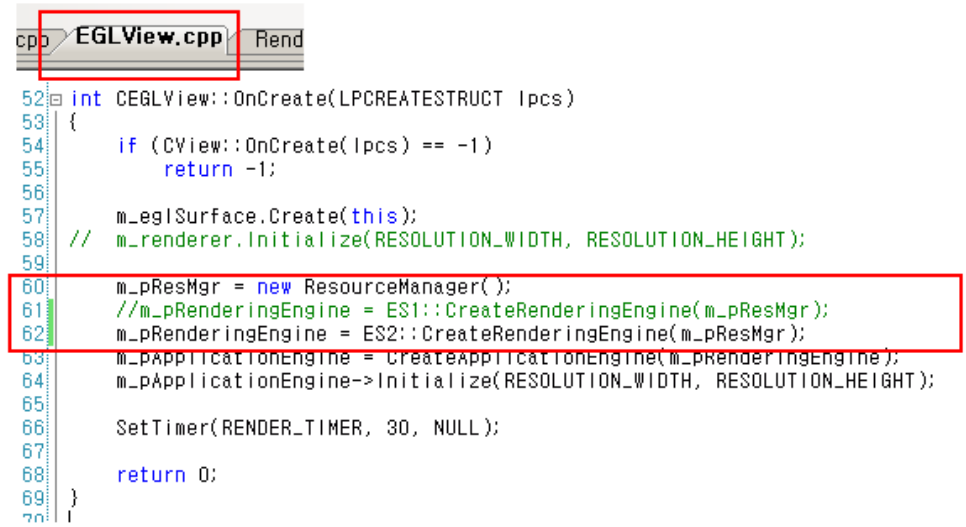
즉, 라이브러리에 담겨있는 header 파일 내의 내용도 서로 다릅니다.

하나의 OGLES20Template 프로젝트내에 GLES1.0과 GLES2.0을 담다 보니 서로 충돌이 나는 문제가 발생합니다.
이는 iOS 개발시에도 동일하며 보통 Macro 처리를 통해서 컴파일 시점에 GLES1.0으로 할지 GLES2.0으로 할지 결정 합니다.

MFC 프로젝트에서는 다음과 같이 수정해 줘야 두 버전을 사용할 수 있습니다.

4.1 GLES 2.0 으로 빌드하기

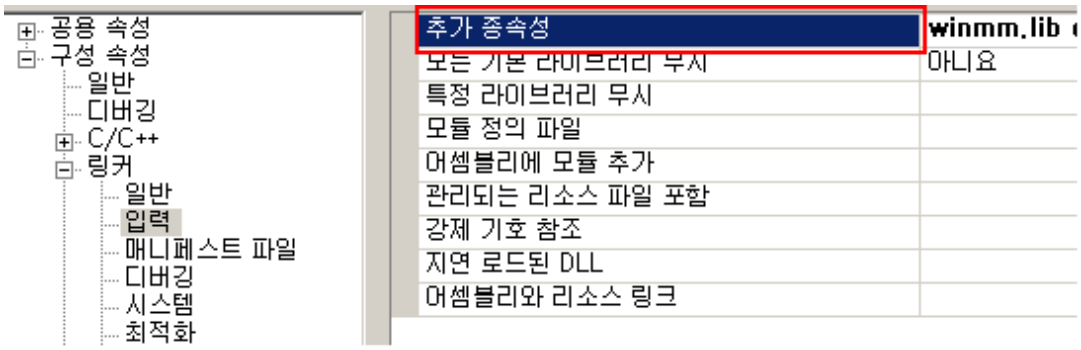


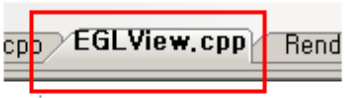


프로젝트 속성 창의 "링크 -> 입력 -> 추가 종속성" 탭에서 libGLESv2.lib를 추가해 줘야 합니다.

또한 EGLView.cpp의 CEGLView::OnCreate()에서 렌더러 생성을 RenderingEngine.ES2.cpp로 맞춰줘야 합니다.
이 부분은 iPhone 3D Programming 책에서 GLView.mm에서 처리하는 부분과 유사하게 처리 하였습니다.

4.2 GLES 1.0 으로 빌드하기






```
52  int CEGLView::OnCreate(LPCREATESTRUCT lpcs)
53  {
54      if (CView::OnCreate(lpcs) == -1)
55          return -1;
56
57      m_eglSurface.Create(this);
58  //  m_renderer.Initialize(RESOLUTION_WIDTH, RESOLUTION_HEIGHT);
59
60      m_pResMgr = new ResourceManager();
61      m_pRenderingEngine = ES1::CreateRenderingEngine(m_pResMgr);
62      //m_pRenderingEngine = ES2::CreateRenderingEngine(m_pResMgr);
63      m_pApplicationEngine = CreateApplicationEngine(m_pRenderingEngine);
64      m_pApplicationEngine->Initialize(RESOLUTION_WIDTH, RESOLUTION_HEIGHT);
65
66      SetTimer(RENDER_TIMER, 30, NULL);
67
68      return 0;
69  }
```

GLES1.0으로 빌드하기 위해서는 프로젝트 속성 창의 "링크 -> 입력 -> 추가 종속성" 탭에서 libGLESv2.lib를 삭제해 준 뒤에
또한 EGLView.cpp의 CEGLView::OnCreate()에서 렌더러 생성을 RenderingEngine.ES1.cpp로 맞춰줘야 합니다.
이 부분은 iPhone 3D Programming 책에서 GLView.mm에서 처리하는 부분과 유사하게 처리 하였습니다.

다음에는 Android Project 설정에 대해서 다뤄 보도록 하겠습니다.

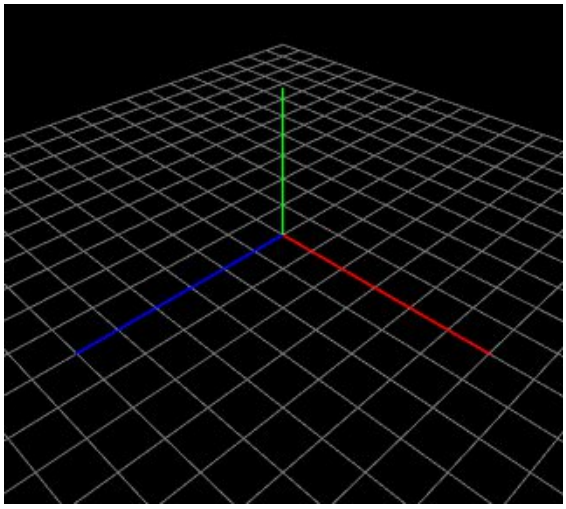
프로젝트는 16MB 정도로 커져 버려서, myastro google code 에 올렸습니다.

OGLES20 Template Application ported to mfc and android

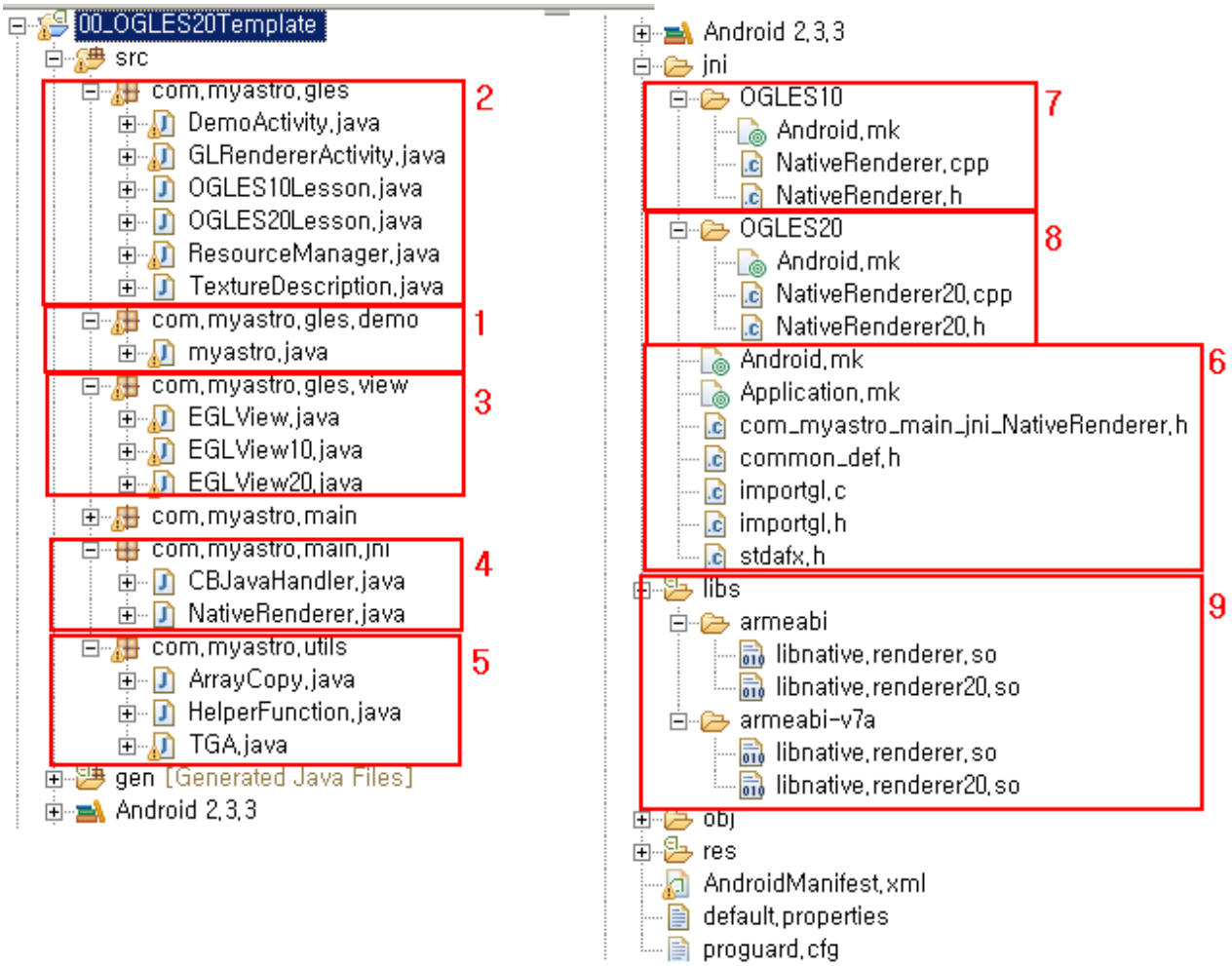
http://code.google.com/p/myastro/downloads/list

http://code.google.com/p/myastro/downloads/detail?name=OGLES20Application.v10.zip&can=2&q=#makechanges

OGLES20Template



1. Android 프로젝트 구성



프로젝트 패키지 설명

1. com.myastro.gles.demo : Android App의 Main Activity 이다.

2. com.myastro.gles : List Activity이다.
GLES20.Lesson 과 GLES10.Lesson 두 가지 List Activity 가 있다.
대체로 GLES20에 대해서 다룰 예정 이지만, 시간 여유가 생기면, GLES10 용도 업데이트할 예정이다.

3. com.myastro.gles.view : EGL View용 모듈 이다.
EGLView.java : GLSurfaceView를 상속 받은 추상 클래스(abstract class) 이다.
또한 EGLView에는 사용자로 부터의 이벤트도 수신함으로 Touch 이벤트을 통한 Gesture 기능 구현은 이곳에 이루어져 있다.
EGLView20.java : EGLView를 상속 받아서 abstract method 들을 구현 한다.
GLES20 용 EGL Configuration으로 구성 되어 있다.

EGLView10.java : EGLView를 상속 받아서 abstract method 들을 구현 한다.
GLES10 용 EGL Configuration으로 구성 되어 있다.

4. com.myastro.main.jni : C++로 구현한 NativeRenderer 와 통신하는 Java native interface 이다.
NativeRenderer.java : GLES10용 NativeRenderer.cpp와 GLES20용 NativeRender20.cpp를 호출하기 위한 native method로 구성되어 있다.
CBJavaHandler.java : Java 단의 Main Activity Thread가 아닌 Native 단의 NativeRenderer 로 부터 Androd SDK내의 API를 이용할 때 사용하기 위한 모듈이다.
이 모듈에는 "utf-8 to ecu-kr" converting 함수, "bmp, png, jpg, tga"등과 같은 이미지 파일 로딩을 위한 함수가 있다..

5. com.myastro.utils : TGA 이미지 파일 로딩 클래스인 TGA.java(cocos2d java 버전에서 가져옴) 와 이미지 로딩후 Integer 형을 Byte 형으로 변환하기 위한 ArrayCopy.java 클래스가 있다.
TGA.java : From cocos2d(ZhouWeikuan-cocos2d-2eb179b)

6. jni : Native Renderer를 구성하는 Java Native Interface layer 이다.
com_myastro_main_jni_NativeRenderer.h : JNI의 헤더 파일 이다.(Java Native Interface 참고)
Anroid.mk : jni.GLES10에 있는 NativerRenderer.cpp 와 앞장에서 설명한 RenderingEngine.ES1.cpp 와 RenderingEngine.ES2.cpp를 컴파일 한다.
빌더는 ndk-r5b을 이용하였으며, STL 라이브러리의 경우, crystax 라이브러리가 아닌 stlport 라이브러리를 이용하였습니다.
stlport library 는 이전 페이지에서 언급했듯이 "OGLES20Application->Builds->Android->stlport" 폴더에 있습니다.
Application.mk : native module 생성 파일로 ndk-r4b까지는 그다지 의미가 없었지만, ndk-r5b에서는 stlport 를 설정해 주기도 합니다.(APP_STL := stlport_static)

7. jni.GLES10 : OpenGL ES 1.0용 Native Renderer 이다.

8. jni.GLES20 : OpenGL ES 2.0용 Native Renderer 이다.

9. libs.armeabi : 빌드한 Shared Library 인 libnative.renderer.so 와 libnative.renderer20.so 가 있다.

기본적으로 Android 프로젝트는 위와 같이 구성 되어 있습니다.
그런데, Android 프로젝트에는 MFC 프로젝트에서 설명했던 Renderer 클래스인 RenderingEngine.ES1.cpp와 RenderingEngine.ES2.cpp 등이 보이지 않습니다.
또한 도우미 라이브러리인 Interfaces.hpp, Matrix.hpp, Quaternion.hpp, Vector.hpp 등도 보이지 않습니다.

이 프로젝트의 구성을 멀티 플렛폼에 맞춰서 구성 하다 보니, 소스의 위치를 Android Project 보다 상위 디렉토리 레벨에 배치 하였습니다.

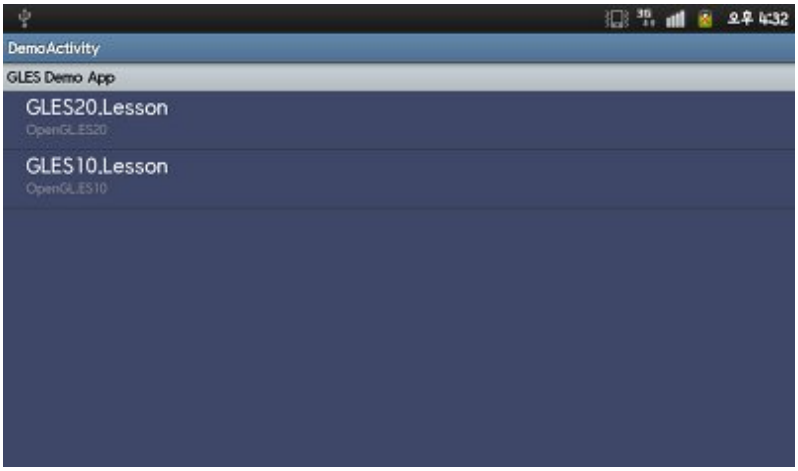
Eclipse 내에서 프로젝트내에 배치 할경우, jni 폴더 하위에 두면 되긴 하지만, mfc와 iOS등과 디렉토리 레벨이 틀어 지기 때문에 현재와 같이 상위 디렉토리 레벨에 배치 하였습니다.

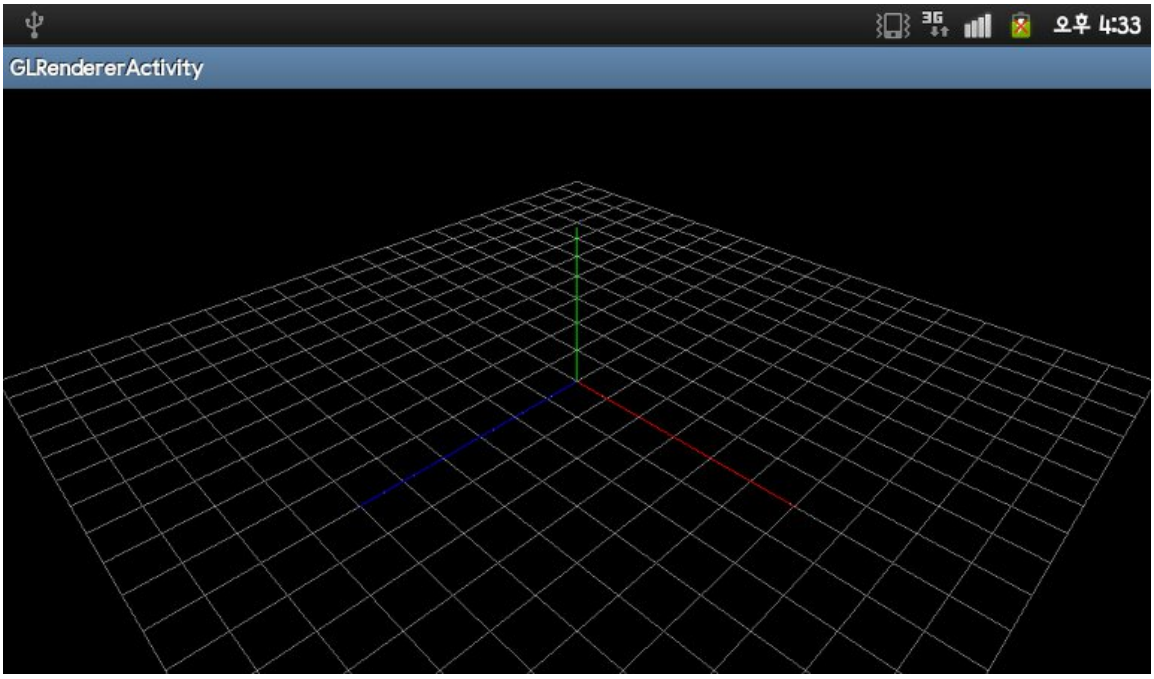
어차피 GDB을 이용해서 native 단에서의 android debugging이 쉽지 않기 때문에, mfc에서 디버깅 할수 있도록 배치 하였습니다.

(이 구성은 Imagination Technology사의 PowerVR SDK와 유사한 방식 입니다.)

2. 실행 결과

Android Galaxy Tab에서 실행하였으며, 결과는 다음과 같습니다.



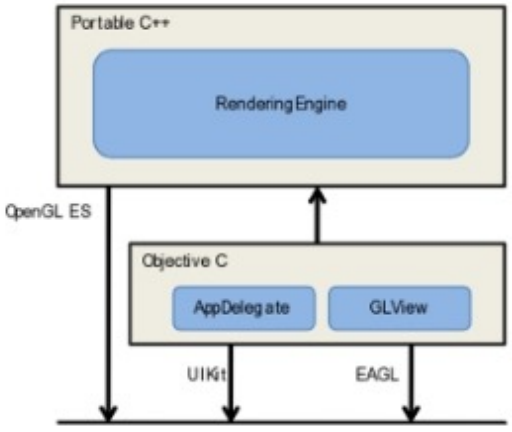


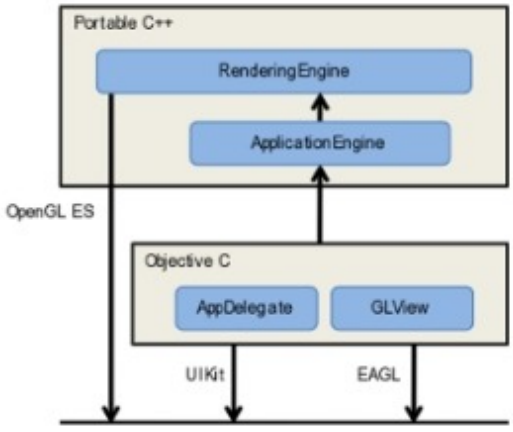
Perspective Projection 으로 X축 Red, Y축 Green, Z축 Blue 로 표현 되어 있다.

3. 렌더러 설명

제가 여기에 소개하는 Nativer Renderer 는 기본적으로 iPhone 3D Programming 책에 설명이 다 나와 있습니다.
그럼으로 이 책을 사서 보시는게 이해는 더 빠르실 거라 생각 됩니다.
또한 렌더러 설계에 대해서는 제 블로그 http://blog.daum.net/aero2k/41 에서 한번 소개 했었습니다.

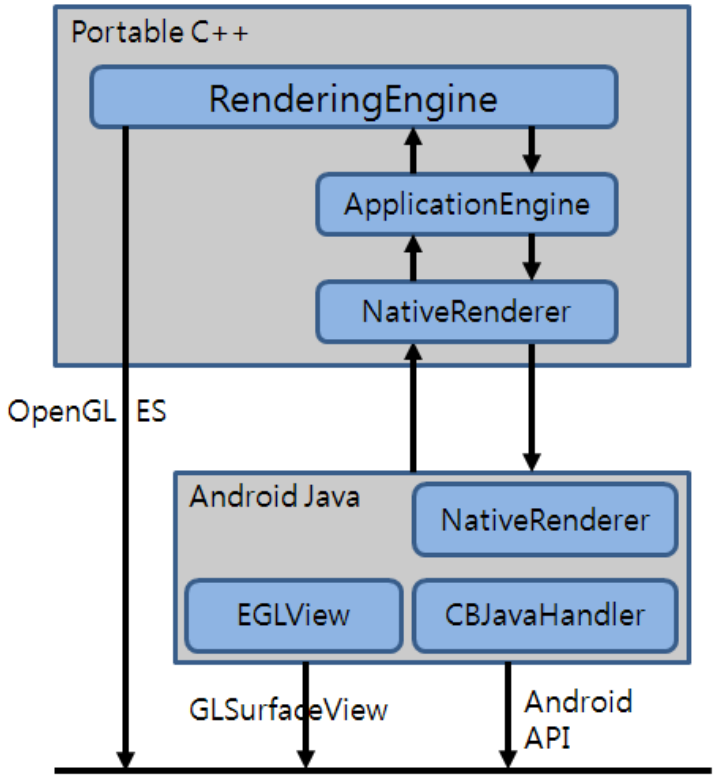
렌더링 엔진 구성은 다음과 같습니다.





제가 Android 3D로 포팅한 렌더러 엔진도 iOS 프레임워크 부분을 Android Java 프레임워크로 변경하였을 뿐 위에 있는 구조와 유

사 합니다.



Classes/RenderingEngine.ES2.cpp 에 있는 Render 함수의 구성에 대해서 간단하게 설명하도록 하겠습니다.

PROJECTION MATRIX

```
if (width > height)
    ratio = (GLfloat)width / (GLfloat)height;
else
    ratio = (GLfloat)height / (GLfloat)width;
mat4 projectionMatrix = mat4::Perspective(45.0f, ratio, 1.0f, 100.0f);
glUniformMatrix4fv(m_simple.Uniforms.Projection, 1, 0, projectionMatrix.Pointer());
```

CAMERA MATRIX

```
 /* Setting the camera */
 vec3 eye(10, 10, 10);
 vec3 target(0, 0, 0);
 vec3 up(0, 1, 0);
 mat4 camera = mat4::LookAt(eye, target, up);
```

틀릴수도 있지만, 저는 위의 코드를 아래와 같이 해석합니다.
카메라 설정 기능으로 머리위인 상향 벡터(up(0, 1,0)로 두었으며, 물체는 원점(target(0, 0, 0))으로 설정 되었고,
시점은 X, Y, Z 축으로 -10 만큼 떨어져 있다.
만약 3차원 좌표계가 아닌 2D 좌표계인 X-Y 기준으로 보고 싶다면, eye(0, 0, 10)을 하시면 X와 Y축만 남게 되고, Z축은 시점에서 정면이 됩니다.

VERTICES

```
 /* Draw Grid and Axis */
/* 20x20 의 격자형 바닥을 그린다. */
 GLfloat gridVertices[] = {
  -10.0f, 0.0f, 0.0f,
  10.0f, 0.0f, 0.0f,
  0.0f, 0.0f, -10.0f,
  0.0f, 0.0f, 10.0f,
 };

/* X(Red)-Y(Green)-Z(Blue) 축을 그린다. */
 GLfloat axisVertices[] = {
 // x
 0.0f, 0.0f, 0.0f,
 6.0f, 0.0f, 0.0f,
 // y
 0.0f, 0.0f, 0.0f,
 0.0f, 4.0f, 0.0f,
 // z
 0.0f, 0.0f, 0.0f,
 0.0f, 0.0f, 6.0f,
 };
```

MODELVIEW MATRIX

```
 // 배경화면 Shader인 Simple shader 프로그램 바인딩.
 glUseProgram(m_simple.Program);

 // GLES10에 있는 glEnableClientState(GL_VERTEX_ARRAY)를 대체 하는 함수.
 glEnableVertexAttribArray(m_simple.Attributes.Position);
 // Draw Grid.
 // Grid Vertices Buffer Array 등록
```

```
// GLES10에 있는 glVertexPointer(3, GL_FLOAT, 0, gridVertices);
glVertexAttribPointer(m_simple.Attributes.Position, 3, GL_FLOAT, GL_FALSE, 0, gridVertices);
glLineWidth(1.0f);
GLint grid_axis_diffuse = m_simple.Attributes.DiffuseMaterial;

// OGLES20에서는 색을 칠하는 Color Buffer Array 나 glColor4f등의 API는 지원하지 않는다.
// 이를 대체하는 방법으로 model에 재질을 입혀서 색을 표현해야 한다.
// diffuse material(난반사 재질) 색상을 회색으로 칠했다.
vec3 grid_axis_material_dif = vec3(0.5f, 0.5f, 0.5f);
glVertexAttrib3fv(grid_axis_diffuse, grid_axis_material_dif.Pointer());

mat4 grid_axis_modelview;
for (int i = -10; i <= 10; ++i) {
 grid_axis_modelview = mat4::Translate(0, 0, (GLfloat)i);
 grid_axis_modelview = grid_axis_modelview * camera;

 glUniformMatrix4fv(m_simple.Uniforms.Modelview, 1, 0, grid_axis_modelview.Pointer());
 glDrawArrays(GL_LINES, 0, 2);
}
for (int i = -10; i <= 10; ++i) {
 grid_axis_modelview = mat4::Translate((GLfloat)i, 0, 0);
 grid_axis_modelview = grid_axis_modelview * camera;

 glUniformMatrix4fv(m_simple.Uniforms.Modelview, 1, 0, grid_axis_modelview.Pointer());
 glDrawArrays(GL_LINES, 2, 2);
}

// Draw Axis.
// Axis Vertices Buffer Array 등록
// GLES10에 있는 glVertexPointer(3, GL_FLOAT, 0, axisVertices);를 대체 하는 함수.
glVertexAttribPointer(m_simple.Attributes.Position, 3, GL_FLOAT, GL_FALSE, 0, axisVertices);
glLineWidth(1.5f);

// Axis X
// 빨간색 난반사 재질 입히기
grid_axis_material_dif = vec3(1.0f, 0.0f, 0.0f);
glVertexAttrib3fv(grid_axis_diffuse, grid_axis_material_dif.Pointer());
// glTranslatef()를 대체하는 함수.
grid_axis_modelview = mat4::Translate(0.05f, 0.05f, 0.05f);
```

```
grid_axis_modelview = grid_axis_modelview * camera;
glUniformMatrix4fv(m_simple.Uniforms.Modelview, 1, 0, grid_axis_modelview.Pointer());
glDrawArrays(GL_LINES, 0, 2);

// Axis Y
// 녹색 난반사 재질 입히기
grid_axis_material_dif = vec3(0.0f, 1.0f, 0.0f);
glVertexAttrib3fv(grid_axis_diffuse, grid_axis_material_dif.Pointer());
glDrawArrays(GL_LINES, 2, 2);

// Axis Z
// 파란색 난반사 재질 입히기
grid_axis_material_dif = vec3(0.0f, 0.0f, 1.0f);
glVertexAttrib3fv(grid_axis_diffuse, grid_axis_material_dif.Pointer());
glDrawArrays(GL_LINES, 4, 2);


// GLES10에 있는 glDisableClientState(GL_VERTEX_ARRAY)를 대체 하는 함수.
glDisableVertexAttribArray(m_simple.Attributes.Position);
```

GLES20용 RenderingEngine.ES2.cpp와 GLES10용 RenderingEngine.ES1.cpp 파일 두개를 비교해서 보면,
양자 간에 API들의 구성이 어떻게 변경되었는지 이해하기 편할 것 같습니다.

4. Shader Language

바른생활님이 작성한 http://cafe.naver.com/gld3d/402 강좌를 보면, EGLSL과 달리 GLSL에서는 OpenGL 1.1 스펙에 있는 API들과 같이 사용할 수 잇는 것을 볼수 있습니다.
하지만, Embedded Graphic Library Shader Language 에서는 임베디드용으로 최적화 되면, OpenGL 1.1 스펙에서 Shader 언어로 처리할 수 있는 API 들에 대해서는 전부 제거 되었습니다.

이에 대한 스펙을 이해하기 위해서는 앞장에서 소개 했던 OpenGL ES 2.0 Programming Guide 책을 한번 보시는 것도 좋습니다.
그런데 이 책에 대한 서평들을 읽어 보시면 알겠지만, 너무 않좋습니다.
저같은 경우는 원서로 봤는데, 책이 출판된 시점이 크로노스 그룹에서 GLES2.0이 만들어 진지 얼마 되지 않은 시점에 나와서 그런지 예제는 아주 많이 부실하고 왠지 스펙 문서를 읽는 듯한 느낌이 듭니다 ^^;

그래도 한번 쭉 읽어 보는 것이 도움이 됨으로 추천은 합니다.
(솔직히 OpenGL ES 2.0 관련된 다른 책이 없어서 추천할 대체제가 없습니다.
전 돈주고 샀지만 인터넷에 PDF 파일이 많음으로 블랙 마켓에서 구해서 보는 것도... ㅠ ㅠ)

각설하고,.... 아래와 같이 배경화면 그리는 수준의 Shader 언어에서는 별 내용이 없습니다.
이 부분은 바른 생활님이 설명하신 http://cafe.naver.com/gld3d/402 장과 비교하시면서 한번 쭈욱 보시면,
이해하 실수 있을 겁니다.

Simple.vert : Vertex Shader

```
tatic const char* SimpleVertexShader = STRINGIFY(
precision highp float;
precision highp int;
// Attributes
attribute vec4 a_position;//Position;
attribute vec3 a_diffuseMaterial;
uniform mat4 u_modelViewProjectionMatrix;//Projection;
uniform mat4 u_modelViewMatrix;//Modelview;
varying vec3 v_diffuse;//Diffuse;
void main(void)
{
 v_diffuse  = a_diffuseMaterial;
 gl_Position  = u_modelViewProjectionMatrix * u_modelViewMatrix * a_position;
}
```

Simple.frag : Fragment Shader

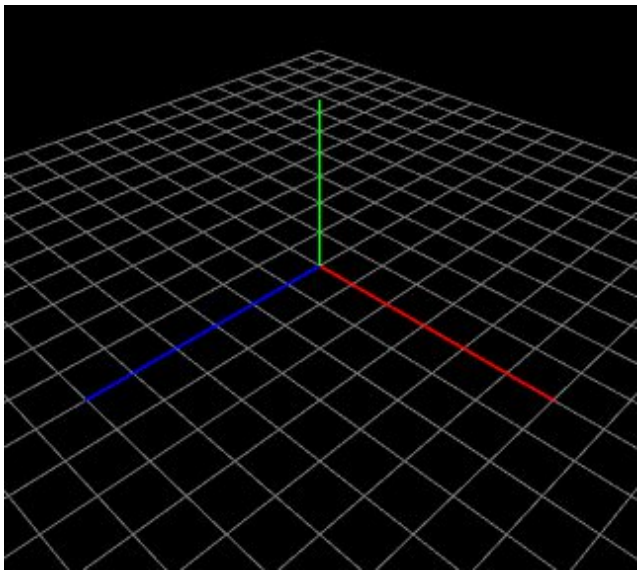
```
static const char* SimpleFragmentShader = STRINGIFY(
varying lowp vec3 v_diffuse;//Diffuse;

void main(void)
{
 gl_FragColor  = vec4(v_diffuse, 1.0);
}
```

이장은 프로젝트 설정 부분임으로 여기 까지만 소개하겠습니다. ^^;

다음 장에서는 Model에 Material 재질 값을 입히고 색상을 표현 하는 방법에 대해서 알아 보도록 하겠습니다.
(솔직히 이미 아시는 분들은 알겠지만, 이 글에서 소개한 정도만으로도 glColor4f를 대체하는 색상 그리는 방법은 이해하실수 있을 겁니다.)

OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list
http://code.google.com/p/myastro/downloads/detail?name=OGLES20Application.v10.zip&can=2&q=#makechanges



1. Color Shading

앞절에서 간략하게 설명한 RenderingEngine.ES2.cpp에 대해서 좀더 풀어서 설명하도록 하겠습니다.

Classes/RenderingEngine.ES2.cpp

1. Create the GLSL program

```
m_simple.Program = BuildProgram(SimpleVertexShader, SimpleFragmentShader);
```

BuildProgram 함수에서는 GLSL source 인 Vertex Shader text(simple.vert)와 Fragment Shader text(simple.frag) 파일을 아래의 GLES함수를 이용해서 컴파일 합니다.

Vertex Shader 생성

```
GLuint vertexShader= glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &source, 0);
```

```
glCompileShader(vertexShader);
```

Fragment Shader 생성

```
GLuint fragmentShader= glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &source, 0);
glCompileShader(fragmentShader);
```

Vertex Shader와 Fragment Shader 링크

```
GLuint programHandle = glCreateProgram();
glAttachShader(programHandle, vertexShader);
glAttachShader(programHandle, fragmentShader);
glLinkProgram(programHandle);
```

2. Program 선택

바탕화면이 하나 짜리를 사용할 경우에는 아래와 같이 생성한 Program 한개를 사용한다고 GLES 라이브러리에 알립니다.

```
glUseProgram(m_simple.Program);
```

만약 한개 이상의 Program을 사용할 경우에는 필요할 때마다 glUseProgram을 이용해서 변경 가능합니다.
이는 glPushMatrix()-glPopMatrix() 쌍으로 묶여있는 렌더러 구성과 유사하게 응용할 수 있습니다.

GLES10에서는 바탕화면을 그리고 상자를 그린다고 하면 다음과 같이 그릴수 있다.

```
glPushMatrix();
drawBackground();
glPopMatrix();
glPushMatrix();
drawCubebox();
glPopMatrix();
```

GLES20에서는 glPushMatrix()-glPopMatrix() 함수를 지원하지 않는다.

```
glUseProgram(m_simple.Program);
drawBackground();
glUseProgram(m_cubebox.Program);
drawCubebox();
```

와 같이 처리할 수 있다.

3. PROJECTION MATRIX

```
m_simple.Uniforms.Projection = glGetUniformLocation(program, "u_modelViewProjectionMatrix");
m_simple.Uniforms.Projection  = glGetUniformLocation(program, "u_projectionMatrix");

if (width > height)
    ratio = (GLfloat)width / (GLfloat)height;
else
    ratio = (GLfloat)height / (GLfloat)width;
mat4 projectionMatrix = mat4::Perspective(45.0f, ratio, 1.0f, 100.0f);
glUniformMatrix4fv(m_simple.Uniforms.Projection, 1, 0, projectionMatrix.Pointer());
```

m_simple.Uniforms.Projection은 glGetUniformLocation() 함수를 이용해서
simple.vert내에 선언된 uniform mat4 u_projectionMatrix;//Projection; 과 연결되는 ID를 얻을 수 있습니다.
이 ID에 Perspective, Frustum, Ortho 투영 행렬을 glUniformMatrix4fv() 함수를 이용해서 저장합니다.

4. MODELVIEW MATRIX

4.1 CAMERA MATRIX

```
/* Setting the camera */
vec3 eye(10, 10, 10);
vec3 target(0, 0, 0);
vec3 up(0, 1, 0);
mat4 camera = mat4::LookAt(eye, target, up);
```

틀릴수도 있지만, 저는 위의 코드를 아래와 같이 해석합니다.
카메라 설정 기능으로 머리위인 상향 벡터(up(0, 1,0)로 두었으며, 물체는 원점(target(0, 0, 0))으로 설정 되었고,
시점은 X, Y, Z 축으로 -10 만큼 떨어져 있다.
만약 3차원 좌표계가 아닌 2D 좌표계인 X-Y 기준으로 보고 싶다면, eye(0, 0, 10)을 하시면 X와 Y축만 남게 되고, Z축은 시점에서 정면이 됩니다.

camera 행렬을 modelview 행렬에 곱해서 사용합니다.
(camera 행렬은 Option으로서 없어도 3D를 표현할 수 있습니다. 하지만 명확하게 3D를 표현할 때는 Camera 행렬을 이용하는 것이 좋습니다.)
camera에 대해서는 바른생활님 기초강좌님 다음 페이지들을 참고하십시요.
http://cafe.naver.com/gld3d/366
http://cafe.naver.com/gld3d/368
http://cafe.naver.com/gld3d/371

4.2 VERTICES

```
/* Draw Grid and Axis */
/* 20x20 의 격자형 바닥을 그린다. */
GLfloat gridVertices[] = {
-10.0f, 0.0f, 0.0f,
10.0f, 0.0f, 0.0f,
0.0f, 0.0f, -10.0f,
0.0f, 0.0f, 10.0f,
};

/* X(Red)-Y(Green)-Z(Blue) 축을 그린다. */
GLfloat axisVertices[] = {
// x
0.0f, 0.0f, 0.0f,
6.0f, 0.0f, 0.0f,
// y
0.0f, 0.0f, 0.0f,
0.0f, 4.0f, 0.0f,
// z
0.0f, 0.0f, 0.0f,
0.0f, 0.0f, 6.0f,
};
```

20x20 격자형 바닥을 표현할 gridVertices 배열과 X-Y-Z 축을 표현할 axisVertices 배열 입니다.
이 정점들은 simple.vert 내의 attribute vec4 a_position; 에 값들을 저장 합니다.

```
m_simple.Attributes.Position = glGetAttribLocation(program, "a_position");
```

m_simple.Attributes.Position은 glGetAttribLocation(program, "a_position"); 함수를 이용해서
simple.vert내에 선언된 uniform mat4 attribute vec4 a_position; 과 연결되는 ID를 얻을 수 있습니다.

a_position에 정점 값을 저장할려면, glEnableVertexAttribArray()과 glVertexAttribPointer()함수를 이용합니다.
이 함수는 GLES1.0과 비교하면 다음과 같습니다.

GLES10	GLES20
glEnableClientState(GL_VERTEX_ARRAY); glDisableClientState(GL_VERTEX_ARRAY);	glEnableVertexAttribArray(m_simple.Attributes.Position) glDisableVertexAttribArray(m_simple.Attributes.Position

glVertexPointer(3, GL_FLOAT, 0, gridVertices);
 	
glVertexPointer(3, GL_FLOAT, 0, axisVertices);
	glVertexAttribPointer(m_simple.Attributes.Position, 3, GL_FALSE, 0, gridVertices);
	
	glVertexAttribPointer(m_simple.Attributes.Position, 3, GL_FALSE, 0, axisVertices);

4.3 MODELVIEW MATRIX

```
// 배경화면 Shader인 Simple shader 프로그램 바인딩.
glUseProgram(m_simple.Program);

// GLES10에 있는 glEnableClientState(GL_VERTEX_ARRAY)를 대체 하는 함수.
glEnableVertexAttribArray(m_simple.Attributes.Position);
// Draw Grid.
// Grid Vertices Buffer Array 등록
// GLES10에 있는 glVertexPointer(3, GL_FLOAT, 0, gridVertices);
glVertexAttribPointer(m_simple.Attributes.Position, 3, GL_FLOAT, GL_FALSE, 0, gridVertices);
glLineWidth(1.0f);
GLint grid_axis_diffuse = m_simple.Attributes.DiffuseMaterial;
// OGLES20에서는 색을 칠하는 Color Buffer Array 나 glColor4f등의 API는 지원하지 않는다.
// 이를 대체하는 방법으로 model에 재질을 입혀서 색을 표현해야 한다.
// diffuse material(난반사 재질) 색상을 회색으로 칠했다.
```

```
vec3 grid_axis_material_dif = vec3(0.5f, 0.5f, 0.5f);
glVertexAttrib3fv(grid_axis_diffuse, grid_axis_material_dif.Pointer());

mat4 grid_axis_modelview;
for (int i = -10; i <= 10; ++i) {
    grid_axis_modelview = mat4::Translate(0, 0, (GLfloat)i);
    grid_axis_modelview = grid_axis_modelview * camera;

    glUniformMatrix4fv(m_simple.Uniforms.Modelview, 1, 0, grid_axis_modelview.Pointer());
    glDrawArrays(GL_LINES, 0, 2);
}
for (int i = -10; i <= 10; ++i) {
    grid_axis_modelview = mat4::Translate((GLfloat)i, 0, 0);
    grid_axis_modelview = grid_axis_modelview * camera;

    glUniformMatrix4fv(m_simple.Uniforms.Modelview, 1, 0, grid_axis_modelview.Pointer());
    glDrawArrays(GL_LINES, 2, 2);
}



// Draw Axis.
// Axis Vertices Buffer Array 등록
// GLES10에 있는 glVertexPointer(3, GL_FLOAT, 0, axisVertices);를 대체 하는 함수.
glVertexAttribPointer(m_simple.Attributes.Position, 3, GL_FLOAT, GL_FALSE, 0, axisVertices);
glLineWidth(1.5f);

// Axis X
// 빨간색 난반사 재질 입히기
grid_axis_material_dif = vec3(1.0f, 0.0f, 0.0f);
glVertexAttrib3fv(grid_axis_diffuse, grid_axis_material_dif.Pointer());
// glTranslatef()를 대체하는 함수.
grid_axis_modelview = mat4::Translate(0.05f, 0.05f, 0.05f);
grid_axis_modelview = grid_axis_modelview * camera;
glUniformMatrix4fv(m_simple.Uniforms.Modelview, 1, 0, grid_axis_modelview.Pointer());
glDrawArrays(GL_LINES, 0, 2);

// Axis Y
// 녹색 난반사 재질 입히기
grid_axis_material_dif = vec3(0.0f, 1.0f, 0.0f);
```

```
glVertexAttrib3fv(grid_axis_diffuse, grid_axis_material_dif.Pointer());
glDrawArrays(GL_LINES, 2, 2);

// Axis Z
// 파란색 난반사 재질 입히기
grid_axis_material_dif = vec3(0.0f, 0.0f, 1.0f);
glVertexAttrib3fv(grid_axis_diffuse, grid_axis_material_dif.Pointer());
glDrawArrays(GL_LINES, 4, 2);


// GLES10에 있는 glDisableClientState(GL_VERTEX_ARRAY)를 대체 하는 함수.
glDisableVertexAttribArray(m_simple.Attributes.Position);
```

위에 코드을 이용해서 20x20 격자형 바닥을 그린후 X-Y-Z축을 그립니다.
여기서 색상 Color Shading 에 대해서 나옵니다.
GLES10에서는 물체에 색상을 입힐때, COLOR BUFFER ARRAY 또는 glColor3f, glColor4f 등을 이용합니다.
하지만, GLES20에서는 glColor를 지원하지 않습니다.
그럼으로 위에서 사용했던 방법과 같이 Position 정점을 구하는 방법과 마찬가지로, Color에 대해서 Attribute로 설정합니다.

저는 난반사 재질인 Diffuse material로 정하고 회색(0.5f, 0.5f, 0.5f)을 입혔습니다.
(보통 불투명 색상을 diffuse로 정한다고 합니다.)

```
m_simple.Attributes.DiffuseMaterial = glGetAttribLocation(program, "a_diffuseMaterial");
                                    .....

vec3 grid_axis_material_dif = vec3(0.5f, 0.5f, 0.5f);
glVertexAttrib3fv(grid_axis_diffuse, grid_axis_material_dif.Pointer());
```

Color Shading은 다음과 같이 vertex shader인 simple.vert에서는 a_diffuseMaterial 로 받은 색상 값을
varying keyword를 이용해서 fragment shader인 simple.frag에 전달 합니다.

simple.vert

```
varying vec3 v_diffuse;//Diffuse;

void main(void)
```

```
{
    v_diffuse = a_diffuseMaterial;
   gl_Position = u_modelViewProjectionMatrix * u_modelViewMatrix * a_position;
}
```

varying keyword로 전달받은 v_diffuse는 gl_FragColor 에 저장합니다.
(gl_FragColor는 (r, g, b, a) 임으로 vec4 형태로 변경해 줘야 합니다.
물론 varying 통해서 전달 받는 값 자체를 vec4 형으로 받아와도 됩니다.)

simple.frag

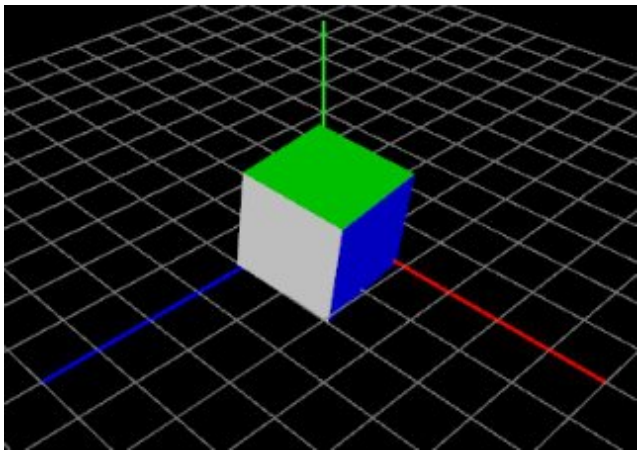
```
varying lowp vec3 v_diffuse;//Diffuse;

void main(void)
{
    gl_FragColor = vec4(v_diffuse, 1.0);
}
```

이 장에서는 간단하게 glColor4f를 이용해서 색상을 입히는 방법에 대해서 알아 봤습니다.

색상에 대한 상세한 정리는 Light 조명 효과로 유명한 Phong Shader를 다룰때 설명하도록 하겠습니다.

OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list
http://code.google.com/p/myastro/downloads/detail?name=OGLES20Application_v10.zip&can=2&q=#makechanges



기하 변환

이번 장에서는 CubeBox를 뛰운 후에 이를 확대, 회전, 이동 시키는 변환 방법에 대해서 알아 보도록 하겠습니다.

1. 회전 변환

GLES10에서는 모델에 대해서 회전 변환을 적용하기 위해서 glRotatef() 함수를 이용합니다.

glRotatef(90.0f, 1.0f, 0.0f, 0.0f) : X축을 기준으로 90도 회전
glRotatef(90.0f, 0.0f, 1.0f, 0.0f) : Y축을 기준으로 90도 회전
glRotatef(90.0f, 0.0f, 0.0f, 1.0f) : Z축을 기준으로 90도 회전

하지만, GLES20에서는 glRotatef() 함수를 지원하지 않습니다.

즉 이를 대체하는 Function Set를 직접 만들어야 합니다.

구현 방법은 예전에 정리했었던, "오일러 회전"장에 보시면 기본 원리가 있습니다.

(변환 으로 가는 길 #2 : 오일러 회전)

Utils/Matrix.hpp

-. 오일러 변환 X축

```
static Matrix4<T> RotateX(T degrees)
{
    T radians = degrees * 3.14159f / 180.0f;
    T s = std::sin(radians);
    T c = std::cos(radians);

    Matrix4 m;
    m.x.x =  1; m.x.y = 0; m.x.z = 0; m.x.w = 0;
    m.y.x =  0; m.y.y = c; m.y.z =-s; m.y.w = 0;
    m.z.x =  0; m.z.y = s; m.z.z = c; m.z.w = 0;
    m.w.x =  0; m.w.y = 0; m.w.z = 0; m.w.w = 1;
    return m;
}
```

-. 오일러 변환 Y축

```
static Matrix4<T> RotateY(T degrees)
{
    T radians = degrees * 3.14159f / 180.0f;
    T s = std::sin(radians);
    T c = std::cos(radians);

    Matrix4 m;
    m.x.x =  c; m.x.y = 0; m.x.z = s; m.x.w = 0;
    m.y.x =  0; m.y.y = 1; m.y.z = 0; m.y.w = 0;
    m.z.x = -s; m.z.y = 0; m.z.z = c; m.z.w = 0;
    m.w.x =  0; m.w.y = 0; m.w.z = 0; m.w.w = 1;
    return m;
}
```

-. 오일러 변환 Z축

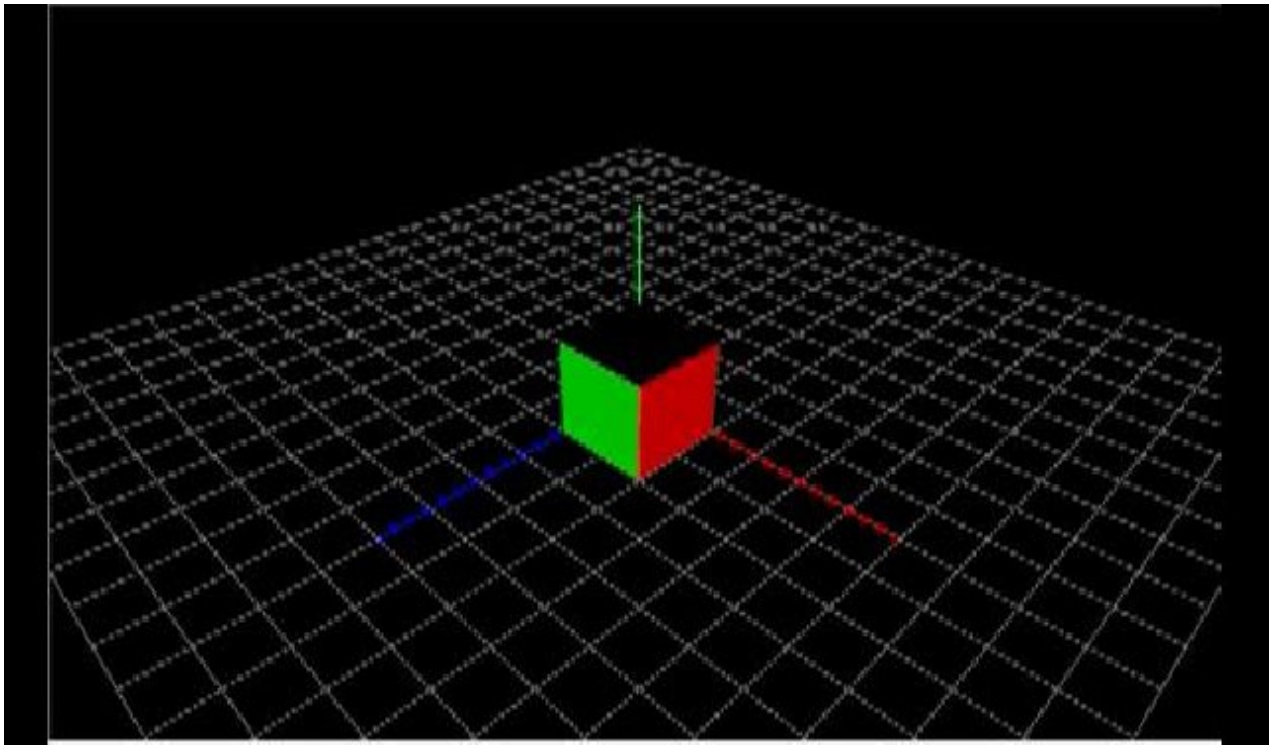
```
static Matrix4<T> RotateZ(T degrees)
{
    T radians = degrees * 3.14159f / 180.0f;
    T s = std::sin(radians);
```

```
        T c = std::cos(radians);

        Matrix4 m;
        m.x.x =  c; m.x.y = -s; m.x.z = 0; m.x.w = 0;
        m.y.x = s; m.y.y = c; m.y.z = 0; m.y.w = 0;
        m.z.x =  0; m.z.y = 0; m.z.z = 1; m.z.w = 0;
        m.w.x =  0; m.w.y = 0; m.w.z = 0; m.w.w = 1;
        return m;
    }
```

사용 방법은 다음과 같습니다.

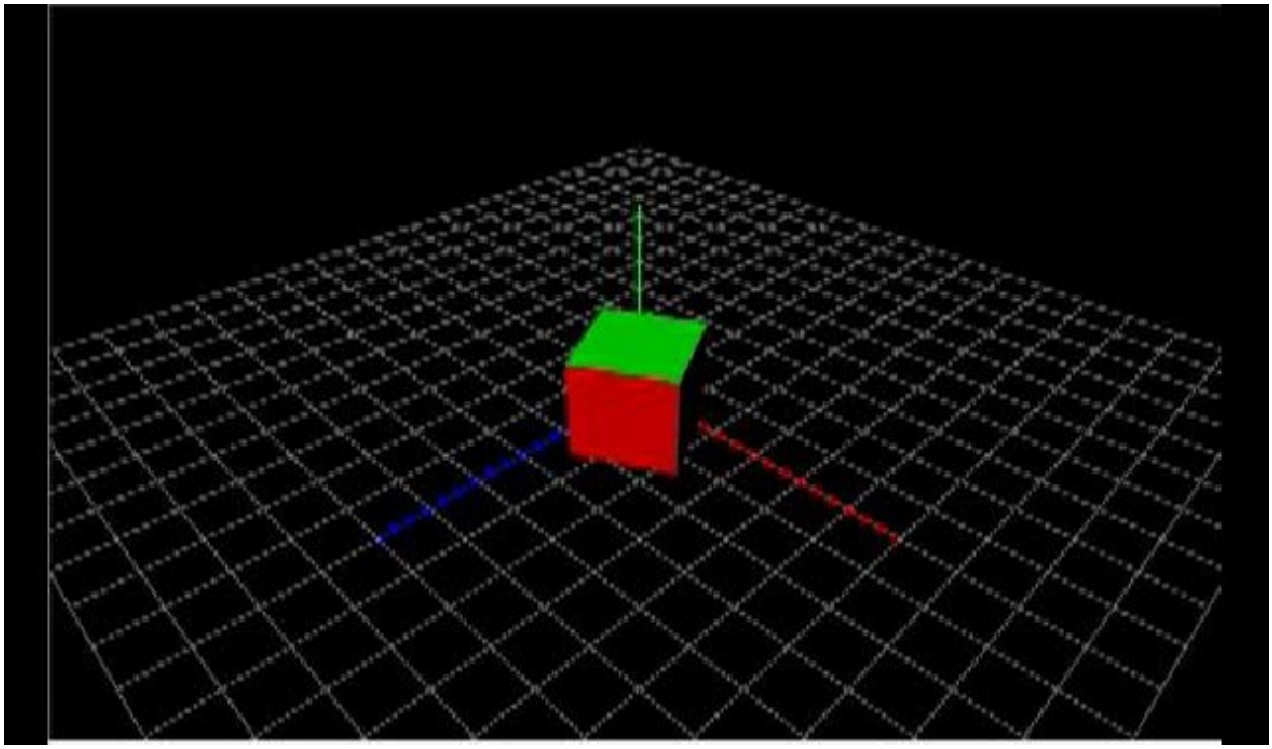
X축 회전

```
void RenderingEngine::drawCubebox(int textureMode)
{


 mat4 euler_x  = mat4::RotateX(xrot);
 mat4 euler_y  = mat4::RotateY(yrot);
 mat4 euler_z = mat4::RotateZ(yrot);

 mat4 rotation  = euler_x * m_orientation.ToMatrix();
 mat4 modelview  = scale * rotation * translation;

}
```



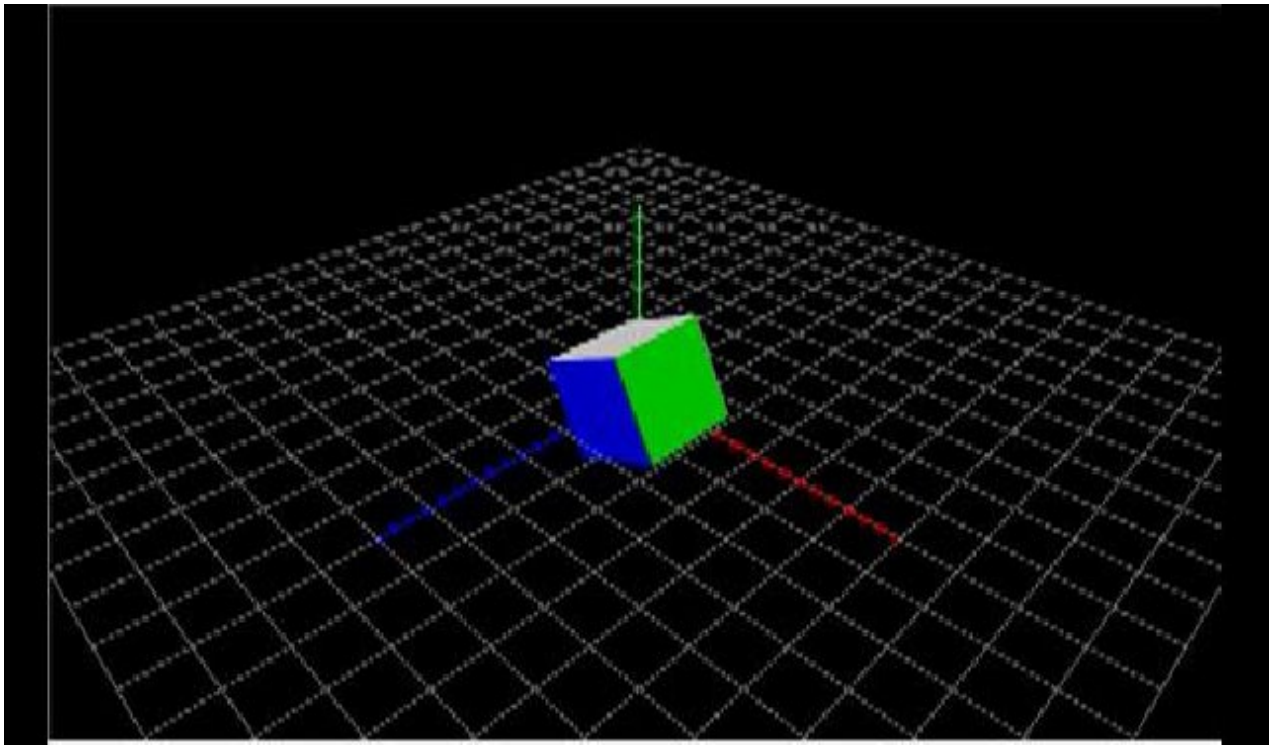
Y축 회전

```
void RenderingEngine::drawCubebox(int textureMode)
{


 mat4 euler_x  = mat4::RotateX(xrot);
 mat4 euler_y  = mat4::RotateY(yrot);
 mat4 euler_z = mat4::RotateZ(yrot);

 mat4 rotation  = euler_y * m_orientation.ToMatrix();
 mat4 modelview  = scale * rotation * translation;

}
```



Z축 회전

```
void RenderingEngine::drawCubebox(int textureMode)
{


 mat4 euler_x  = mat4::RotateX(xrot);
 mat4 euler_y  = mat4::RotateY(yrot);
 mat4 euler_z = mat4::RotateZ(yrot);

 mat4 rotation  = euler_z  * m_orientation.ToMatrix();
 mat4 modelview  = scale * rotation * translation;

}
```



2. 축척 변환

GLES10에서는 모델에 대해서 축척 변환을 적용하기 위해서 glScalef() 함수를 이용합니다.

하지만, GLES20에서는 glScalef() 함수를 지원하지 않습니다.

축척을 변환하는 방법은 위의 함수를 사용하지 않고도 간단하게 구현할 수 있습니다.

왜냐하면, 확대 축소는 수학적으로 벡터 량이 아닌 스칼라 량이기 때문에 곱하기만 하면 됩니다.

iPhone 3D Programming 책에서 제공하는 모듈에서는 이에 대한 함수를 제공함으로 이를 이용하면 됩니다.

Utils/Matrix.hpp

```
static Matrix4<T> Scale(T s)
{
    Matrix4 m;
    m.x.x = s; m.x.y = 0; m.x.z = 0; m.x.w = 0;
    m.y.x = 0; m.y.y = s; m.y.z = 0; m.y.w = 0;
```

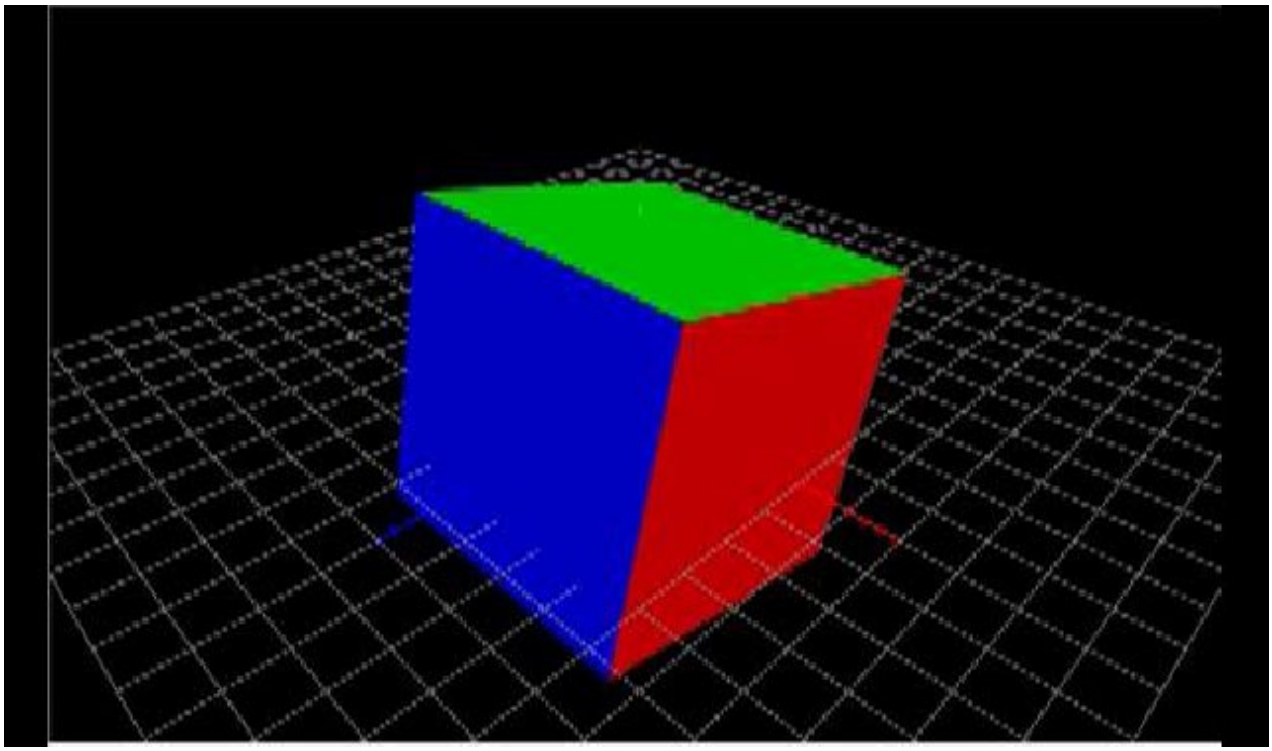
```
        m.z.x = 0; m.z.y = 0; m.z.z = s; m.z.w = 0;
        m.w.x = 0; m.w.y = 0; m.w.z = 0; m.w.w = 1;
        return m;
    }
    static Matrix4<T> Scale(T x, T y, T z)
    {
        Matrix4 m;
        m.x.x = x; m.x.y = 0; m.x.z = 0; m.x.w = 0;
        m.y.x = 0; m.y.y = y; m.y.z = 0; m.y.w = 0;
        m.z.x = 0; m.z.y = 0; m.z.z = z; m.z.w = 0;
        m.w.x = 0; m.w.y = 0; m.w.z = 0; m.w.w = 1;
        return m;
    }
```

사용 방법은 다음과 같습니다.

```
void RenderingEngine::drawCubebox(int textureMode)
{
    mat4 euler_x  = mat4::RotateX(xrot);
    mat4 euler_y  = mat4::RotateY(yrot);
    mat4 euler_z = mat4::RotateZ(yrot);

//    mat4 scale    = mat4::Scale(1.0f);
    mat4 scale    = mat4::Scale(3.0f*cos(interpolation_z));
    mat4 translation = mat4::Translate(x, y, z);

    mat4 rotation  = euler_x * m_orientation.ToMatrix();
    mat4 modelview  = scale * rotation * translation;
    interpolation_z += (2*Pi) * 0.0008f;
}
```

보간값을 계산할때 삼각함수인 cos을 이용했습니다.
cos에 치역은 -1.0f ~ 1.0f 사이 입니다.
위의 3.0f*cos(interpolation_z) 의 경우, interpolation_z가 0에서 부터 시작함으로, cos(0) = 1 입니다.
+3 ~ 0 ~ -3 ~ 0 ~ +3 의 값을 반복해서 반환하게 됩니다.

결국 아래의 동영상에서 볼수 있듯이 CubeBox가 원점을 기준으로 +3에 큰 크기에서 시작 했다가
원점으로 수렴해 가면서 작아집니다. 이후 -3에 점점 수렴해 가기 때문에 그 모양이 뒤집혀서 커지게 됩니다.
((2*Pi) * 0.0008f; 증가 수치는 PowerVR SDK 내의 보간 증가 수치를 가져왔습니다. ^^;)



3. 이동 변환

GLES10에서는 모델에 대해서 이동 변환을 적용하기 위해서 glTranslatef() 함수를 이용합니다.

하지만, GLES20에서는 glTranslatef() 함수를 지원하지 않습니다.

이동 변환의 원리는 4x4 행렬상에서 4번째 행에 값을 저장하면 됩니다.
예를 들어 아래와 같은 회전 행렬이 있다고 한다면,

```
| 1   0   0   0 |
| 0   1   0   0 |
| 0   0   1   0 |
| wx   wy   wz   1 |
```

wx, wy, wz 요소에 이동 변환 값을 저장 한후 model matrix에 곱하면 됩니다.

구현함수는 다음과 같습니다.

Utils/Matrix.hpp

```
    static Matrix4<T> Translate(T x, T y, T z)
    {
        Matrix4 m;
        m.x.x = 1; m.x.y = 0; m.x.z = 0; m.x.w = 0;
        m.y.x = 0; m.y.y = 1; m.y.z = 0; m.y.w = 0;
        m.z.x = 0; m.z.y = 0; m.z.z = 1; m.z.w = 0;
        m.w.x = x; m.w.y = y; m.w.z = z; m.w.w = 1;
        return m;
    }
```

사용 방법은 다음과 같습니다.

```
void RenderingEngine::drawCubebox(int textureMode)
{
    mat4 euler_x  = mat4::RotateX(xrot);
    mat4 euler_y  = mat4::RotateY(yrot);
    mat4 euler_z = mat4::RotateZ(yrot);

    mat4 scale    = mat4::Scale(1.0f);
//    mat4 translation = mat4::Translate(x, y, z);
    mat4 translation = mat4::Translate(x, y, 15.0f*cos(interpolation_z)-7.5f);

    mat4 rotation  = euler_x * m_orientation.ToMatrix();
    mat4 modelview  = scale * rotation * translation;
    interpolation_z += (2*Pi) * 0.0008f;
}
```

보간값은 축척 변환에 적용했던 삼각함수 cos을 이용했습니다.

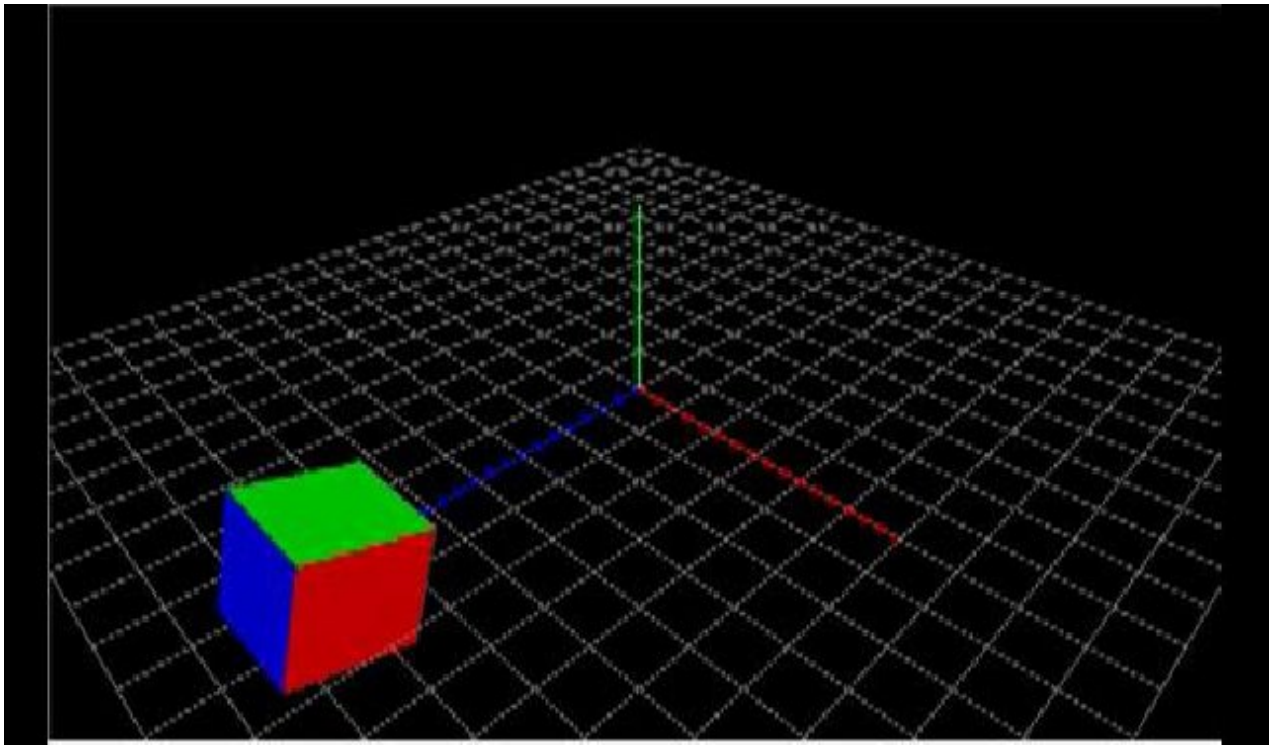
cos에 치역은 -1.0f ~ 1.0f 사이 입니다.

15.0f*cos(interpolation_z)-7.5f 의 경우, interpolation_z가 0에서 부터 시작함으로, cos(0) = 1 입니다.

+7.5 ~ -7.5 ~ -22.5 의 값을 반복해서 반환하게 됩니다.

특이 사항은 cos(theta)가 0 ~ -1일 경우, -7.5에 가중치를 받기 때문에, 아주 약간 EaseOut 효과 처럼 가속를 받게 됩니다.

((2*Pi) * 0.0008f; 증가 수치는 PowerVR SDK 내의 보간 증가 수치를 가져왔습니다. ^^;)



4. vertex shader 내에서 변환 적용 하기

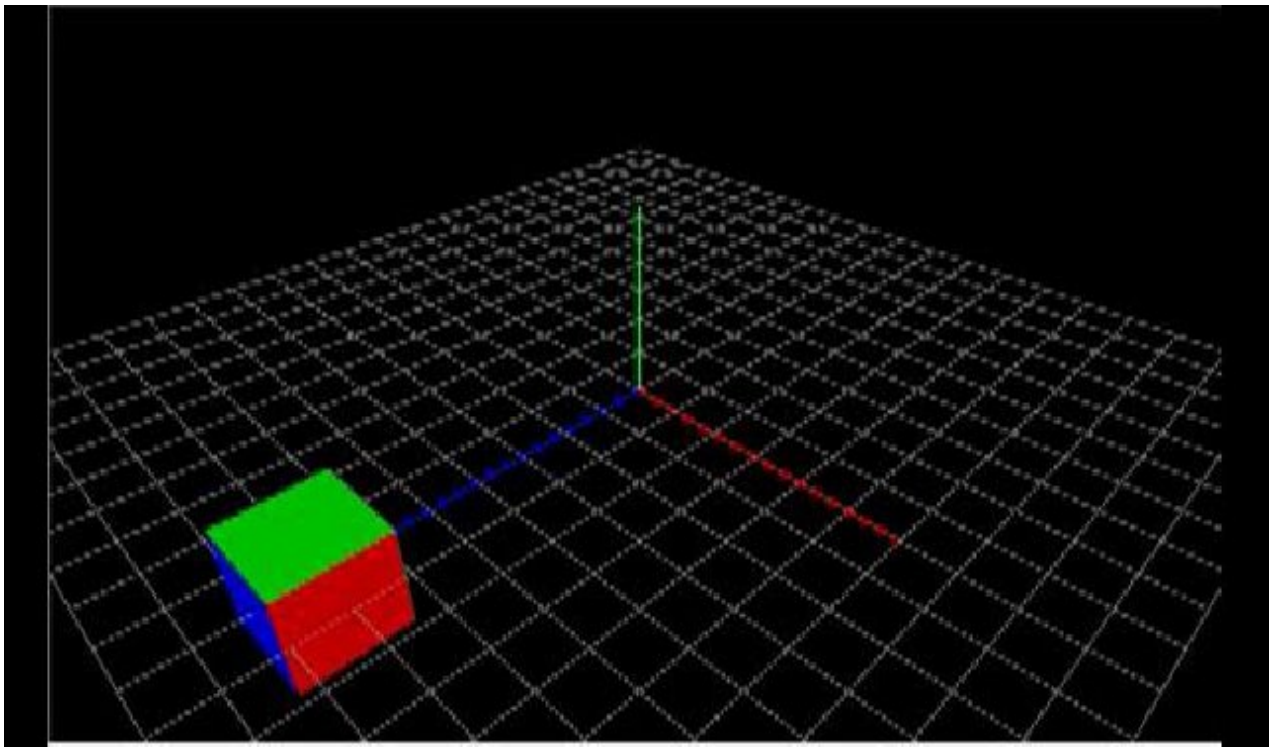
이와 같은 변환의 적용은 vertex shader 내에서 적용해도 됩니다.
shader language 내에서 처리할 경우, 그 연산 처리를 GPU에서 하기 때문에, CPU에 대한 부하를 줄일 수 있다고 합니다.
(저도 실제 측정해 보지는 않았습니다.)

Shaders/SimpleLighting.vert

```
// Attributes
attribute vec4 a_position;//Position;


// Uniforms
uniform mat4 u_projectionMatrix;//Projection;
uniform mat4 u_modelViewMatrix;//Modelview;
uniform mat3 u_normalMatrix;//NormalMatrix;
uniform float u_interpolation_z;
varying vec3 v_eyespaceNormal;//EyespaceNormal
varying vec3 v_diffuse;//Diffuse;
varying vec2 v_textureCoordOut;
```

```
void main(void)
{

 v_eyespaceNormal = u_normalMatrix * a_normal;
 v_diffuse = a_diffuseMaterial;
 v_textureCoordOut = a_textureCoordIn;

 // Vertex Position
 vec4 newPos = vec4(a_position);

 mat4 translation = mat4(1.0);
 mat4 modelview = mat4(u_modelViewMatrix);
 translation[3][2] = (15.0*cos(u_interpolation_z) - 7.5);
 modelview *= translation;

 gl_Position = u_projectionMatrix * modelview * newPos;
}
```

translation[3][2] 는 이동변환에서 Z축을 기준으로 변환시키는 요소 입니다.



저는 아직까지는 vertex shader 내에서 논리 연산을 많이 사용 하지는 않습니다.

튜닝적인 요소가 있을경우에는 적용할 가치가 있을것 같지만,
아직까지는 shader 코드 내에서의 helper 함수를 새로 구성해야 하는 점과 디버깅이 불편하다는 점 때문에 잘 사용하지는 않습니다.

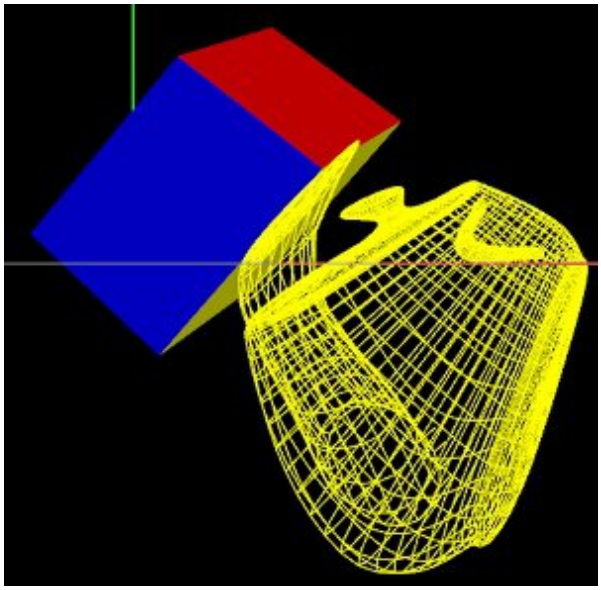
OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list

http://code.google.com/p/myastro/downloads/detail?name=01_GeometryTransform.v1.1.zip&can=2&q=

[GLES20] 05. Vertex Position 처리 (1)

2011.11.11 21:24



Vertex Shader 내에서 Vertex Position 조정

이번 장에서는 Vertex shader 내에서 정점 위치를 조정하는 방법에 대해서 정리해 보도록 하겠습니다.

정리에 앞서서 OpenGL GLSL로 만들어진 바른생활님 강좌를 한번 읽어 보시면 도움이 될 것 같습니다..
http://cafe.naver.com/gld3d/405
http://cafe.naver.com/gld3d/406

OpenGL ES에서는 OpenGL에서는 지원하는 즉시 모드(Immediate Mode)형태의 glVertex3f()를 지원하지 않습니다.
그럼으로 당연히 GLSL내에 있는 gl_Vertex도 지원하지 않습니다.

정점을 얻기위해서는 attribute keyword 형태로 선언해서 OpenGL App(이하 본App ^^) 에서 정점 값을 얻어와야 합니다.
(나중에 다루겠지만, Normal vector, Texture vector 값도 attribute으로 선언해서 본App으로 부터 얻어 옵니다.)

1. Cubebox 그리기

여기서 구현한 Cubebox는 GL_TRIANGLE_STRIP으로 구현하는 방법으로 정점을 구성하는 방법중 하나 일 뿐입니다.
예를 들어, vertices, normals, texcoords 배열을 초기화시점에 미리 다 설정해서 구성해도 됩니다.

```
GLfloat texcoords[4][2];
GLfloat vertices[4][3];
GLfloat normals[4][3];
GLubyte indices[4]={0, 1, 3, 2}; /* QUAD to TRIANGLE_STRIP conversion; */ -- (0)

glUseProgram(m_cubebox.Program);

// GLES10에 있는 glEnableClientState(GL_VERTEX_ARRAY)를 대체 하는 함수.  -- (1)
glEnableVertexAttribArray(m_cubebox.Attributes.Position);
glEnableVertexAttribArray(m_cubebox.Attributes.Normal);
glEnableVertexAttribArray(m_cubebox.Attributes.TextureCoord);

// GLES10에 있는 glVertexPointer(3, GL_FLOAT, 0, gridVertices);  ---------- (2)
glVertexAttrib3fv(diffuse, box_material_dif.Pointer());
glVertexAttribPointer(position, 3, GL_FLOAT, GL_FALSE, 0, vertices);
glVertexAttribPointer(normal, 3, GL_FLOAT, GL_FALSE, 0, normals);
glVertexAttribPointer(texCoord, 2, GL_FLOAT, GL_FALSE, 0, texcoords);

/* Front Face */    ------------------------------------------- (3)
/* Normal Pointing Towards Viewer */
// 법선 벡터를 Z축 방향으로 앞으로 나오는 면이므로, Z축 열에 +1.0을 넣는다.
normals[0][0] = normals[1][0] = normals[2][0] = normals[3][0] = 0.0f;
normals[0][1] = normals[1][1] = normals[2][1] = normals[3][1] = 0.0f;
normals[0][2] = normals[1][2] = normals[2][2] = normals[3][2] = 1.0f;

/* Point 1 (Front) */
texcoords[0][0] = 1.0f; texcoords[0][1] = 0.0f;
vertices[0][0] = -1.0f; vertices[0][1] = -1.0f; vertices[0][2] = 1.0f;
/* Point 2 (Front) */
texcoords[1][0] = 0.0f; texcoords[1][1] = 0.0f;
vertices[1][0] = 1.0f; vertices[1][1] = -1.0; vertices[1][2] = 1.0f;
/* Point 3 (Front) */
texcoords[2][0] = 0.0f; texcoords[2][1] = 1.0f;
vertices[2][0] = 1.0f; vertices[2][1] = 1.0; vertices[2][2] = 1.0f;
/* Point 4 (Front) */
texcoords[3][0] = 1.0f; texcoords[3][1] = 1.0f;
vertices[3][0] = -1.0f; vertices[3][1] = 1.0; vertices[3][2] = 1.0f;
```

```
/* Blue Color */  ------------------------------------------------  (4)
if (textureMode == 0) {
 box_material_dif = vec3(0.f, 0.f, 1.f) * 0.75f;
 glVertexAttrib3fv(diffuse, box_material_dif.Pointer());
}

/* Draw one textured plane using two stripped triangles */  ------------ (5)
glDrawElements(GL_TRIANGLE_STRIP, 4, GL_UNSIGNED_BYTE, indices);


/* Back Face */     ------------------------------------------------  (6)
/* Normal Pointing Away From Viewer */
// 법선 벡터를 Z축 방향으로 뒤로 들어가는 면이므로, Z축 열에 -1.0을 넣는다.
   normals[0][0] = normals[1][0] = normals[2][0] = normals[3][0] = 0.0f;
   normals[0][1] = normals[1][1] = normals[2][1] = normals[3][1] = 0.0f;
   normals[0][2] = normals[1][2] = normals[2][2] = normals[3][2] = -1.0f;

/* Point 1 (Back) */
texcoords[0][0] = 0.0f; texcoords[0][1] = 0.0f;
vertices[0][0] = -1.0f; vertices[0][1] = -1.0f; vertices[0][2] = -1.0f;
/* Point 2 (Back) */
texcoords[1][0] = 0.0f; texcoords[1][1] = 1.0f;
vertices[1][0] = -1.0f; vertices[1][1] = 1.0f; vertices[1][2] = -1.0f;
/* Point 3 (Back) */
texcoords[2][0] = 1.0f; texcoords[2][1] = 1.0f;
vertices[2][0] = 1.0f;  vertices[2][1] = 1.0f; vertices[2][2] = -1.0f;
/* Point 4 (Back) */
texcoords[3][0] = 1.0f; texcoords[3][1] = 0.0f;
vertices[3][0] = 1.0f; vertices[3][1] = -1.0; vertices[3][2] = -1.0f;

/* Black Color */
if (textureMode == 0) {
 box_material_dif = vec3(0.f, 0.f, 0.f) * 0.75f;
 glVertexAttrib3fv(diffuse, box_material_dif.Pointer());
}
/* Draw one textured plane using two stripped triangles */
glDrawElements(GL_TRIANGLE_STRIP, 4, GL_UNSIGNED_BYTE, indices);
```

```
/* Top Face */     ------------------------------------------------ (7)
/* Normal Pointing Up */
// 법선 벡터를 Y축 방향으로 위로 올라가는 면이므로, Y축 열에 +1.0을 넣는다.
   normals[0][0] = normals[1][0] = normals[2][0] = normals[3][0] = 0.0f;
   normals[0][1] = normals[1][1] = normals[2][1] = normals[3][1] = 1.0f;
   normals[0][2] = normals[1][2] = normals[2][2] = normals[3][2] = 0.0f;

/* Point 1 (Top) */
texcoords[0][0] = 1.0f; texcoords[0][1] = 1.0f;
vertices[0][0] = -1.0f; vertices[0][1] = 1.0f; vertices[0][2] = -1.0f;
/* Point 2 (Top) */
texcoords[1][0] = 1.0f; texcoords[1][1] = 0.0f;
vertices[1][0] = -1.0f; vertices[1][1] = 1.0f; vertices[1][2] = 1.0f;
/* Point 3 (Top) */
texcoords[2][0] = 0.0f; texcoords[2][1] = 0.0f;
vertices[2][0] = 1.0f;  vertices[2][1] = 1.0f; vertices[2][2] = 1.0f;
/* Point 4 (Top) */
texcoords[3][0] = 0.0f; texcoords[3][1] = 1.0f;
vertices[3][0] = 1.0f;  vertices[3][1] = 1.0f; vertices[3][2] = -1.0f;

/* Green Color */
if (textureMode == 0) {
 box_material_dif = vec3(0.f, 1.f, 0.f) * 0.75f;
 glVertexAttrib3fv(diffuse, box_material_dif.Pointer());
}
/* Draw one textured plane using two stripped triangles */
glDrawElements(GL_TRIANGLE_STRIP, 4, GL_UNSIGNED_BYTE, indices);


/* Bottom Face */     ------------------------------------------------ (8)
/* Normal Pointing Down */
// 법선 벡터를 Y축 방향으로 위로 올라가는 면이므로, Y축 열에 -1.0을 넣는다.
   normals[0][0] = normals[1][0] = normals[2][0] = normals[3][0] = 0.0f;
   normals[0][1] = normals[1][1] = normals[2][1] = normals[3][1] = -1.0f;
   normals[0][2] = normals[1][2] = normals[2][2] = normals[3][2] = 0.0f;

/* Point 1 (Bottom) */
texcoords[0][0] = 0.0f; texcoords[0][1] = 1.0f;
vertices[0][0] = -1.0f; vertices[0][1] = -1.0f; vertices[0][2] = -1.0f;
```

```
/* Point 2 (Bottom) */
texcoords[1][0] = 1.0f; texcoords[1][1] = 1.0f;
vertices[1][0] = 1.0f;  vertices[1][1] = -1.0f; vertices[1][2] = -1.0f;
/* Point 3 (Bottom) */
texcoords[2][0] = 1.0f; texcoords[2][1] = 0.0f;
vertices[2][0] = 1.0f;  vertices[2][1] = -1.0f; vertices[2][2] = 1.0f;
/* Point 4 (Bottom) */
texcoords[3][0] = 0.0f; texcoords[3][1] = 0.0f;
vertices[3][0] = -1.0f; vertices[3][1] = -1.0f; vertices[3][2] = 1.0f;

/* Yellow Color */
if (textureMode == 0) {
 box_material_dif = vec3(0.8f, 0.8f, 0.f) * 0.75f;
 glVertexAttrib3fv(diffuse, box_material_dif.Pointer());
}
/* Draw one textured plane using two stripped triangles */
glDrawElements(GL_TRIANGLE_STRIP, 4, GL_UNSIGNED_BYTE, indices);


/* Right face */     ---------------------------------------------  (9)
/* Normal Pointing Right */
// 법선 벡터를 X축 방향으로 오른쪽으로 이동하는 면이므로, X축 열에 +1.0을 넣는다.
   normals[0][0] = normals[1][0] = normals[2][0] = normals[3][0] = 1.0f;
   normals[0][1] = normals[1][1] = normals[2][1] = normals[3][1] = 0.0f;
   normals[0][2] = normals[1][2] = normals[2][2] = normals[3][2] = 0.0f;

/* Point 1 (Right) */
texcoords[0][0] = 0.0f; texcoords[0][1] = 0.0f;
vertices[0][0] = 1.0f;  vertices[0][1] = -1.0f; vertices[0][2] = -1.0f;
/* Point 2 (Right) */
texcoords[1][0] = 0.0f; texcoords[1][1] = 1.0f;
vertices[1][0] = 1.0f;  vertices[1][1] = 1.0f; vertices[1][2] = -1.0f;
/* Point 3 (Right) */
texcoords[2][0] = 1.0f; texcoords[2][1] = 1.0f;
vertices[2][0] = 1.0f;  vertices[2][1] = 1.0f; vertices[2][2] = 1.0f;
/* Point 4 (Right) */
texcoords[3][0] = 1.0f; texcoords[3][1] = 0.0f;
vertices[3][0] = 1.0f;  vertices[3][1] = -1.0f; vertices[3][2] = 1.0f;
```

```
/* Red Color */
if (textureMode == 0) {
 box_material_dif = vec3(1.0f, 0.f, 0.f) * 0.75f;
 glVertexAttrib3fv(diffuse, box_material_dif.Pointer());
}
/* Draw one textured plane using two stripped triangles */
glDrawElements(GL_TRIANGLE_STRIP, 4, GL_UNSIGNED_BYTE, indices);

/* Left Face*/     ---------------------------------------------  (10)
/* Normal Pointing Left */
// 법선 벡터를 X축 방향으로 왼쪽으로 이동하는 면이므로, X축 열에 -1.0을 넣는다.
   normals[0][0] = normals[1][0] = normals[2][0] = normals[3][0] = -1.0f;
   normals[0][1] = normals[1][1] = normals[2][1] = normals[3][1] = 0.0f;
   normals[0][2] = normals[1][2] = normals[2][2] = normals[3][2] = 0.0f;

/* Point 1 (Left) */
texcoords[0][0] = 1.0f; texcoords[0][1] = 0.0f;
vertices[0][0] = -1.0f; vertices[0][1] = -1.0f; vertices[0][2] = -1.0f;
/* Point 2 (Left) */
texcoords[1][0] = 0.0f; texcoords[1][1] = 0.0f;
vertices[1][0] = -1.0f; vertices[1][1] = -1.0f; vertices[1][2] = 1.0f;
/* Point 3 (Left) */
texcoords[2][0] = 0.0f; texcoords[2][1] = 1.0f;
vertices[2][0] = -1.0f; vertices[2][1] = 1.0f; vertices[2][2] = 1.0f;
/* Point 4 (Left) */
texcoords[3][0] = 1.0f; texcoords[3][1] = 1.0f;
vertices[3][0] = -1.0f; vertices[3][1] = 1.0f; vertices[3][2] = -1.0f;

/* White Color */
if (textureMode == 0) {
 box_material_dif = vec3(1.f, 1.f, 1.f) * 0.75f;
 glVertexAttrib3fv(diffuse, box_material_dif.Pointer());
}
/* Draw one textured plane using two stripped triangles */
glDrawElements(GL_TRIANGLE_STRIP, 4, GL_UNSIGNED_BYTE, indices);


// GLES10에 있는 glDisableClientState(GL_VERTEX_ARRAY)를 대체 하는 함수.  -- (11)
 glDisableVertexAttribArray(position);
```

```
glDisableVertexAttribArray(normal);
glDisableVertexAttribArray(texCoord);
```

---

(0) : 사각형을 glDrawArrays가 아닌 glDrawElements로 그리기 위한 indices 순서 입니다.

(1) : glEnableVertexAttribArray()는 vertex shader에 선언된 attribute keyword를 활성화 합니다.
이는 GLES10의 glEnableClientState()를 대체합니다.

(2) : glVertexAttribPointer()를 이용해서 vertex shader에 선언된 attribte keyword 벡터 변수에 정정, 법선 또는 텍셀 정보를 저장 합니다.
이는 GLES10의 glVertexPointer(), glNormalPointer(), glTexCoordPointer()를 대체합니다.
이는 OpenGL의 glVertex3f(), glNormal3f(), glTexCoord3f() 를 대체 합니다.

(3) : Front Face 즉 앞 부분 사각 면을 그리는 부분 입니다.
파란색으로 칠하고 있으며, 아직 normals와 texcoords는 사용하지는 않습니다.

(4) : 파란색으로 칠하기 위해서 본App에서 vertex shader로 접근하는 부분입니다.
정점당 색상값인 Color Buffer Array을 대체하는 부분니다.

(5) : (0)에서 설정한 색인 값인 indices 순서로 사각형을 그립니다.

(6) : Back Face 즉 뒷 부분 사각 면을 그리는 부분 입니다.
검정색으로 칠하고 있으며, 아직 normals와 texcoords는 사용하지는 않습니다.

(7) : Top Face 즉 윗 부분 사각 면을 그리는 부분 입니다.
녹색으로 칠하고 있으며, 아직 normals와 texcoords는 사용하지는 않습니다.

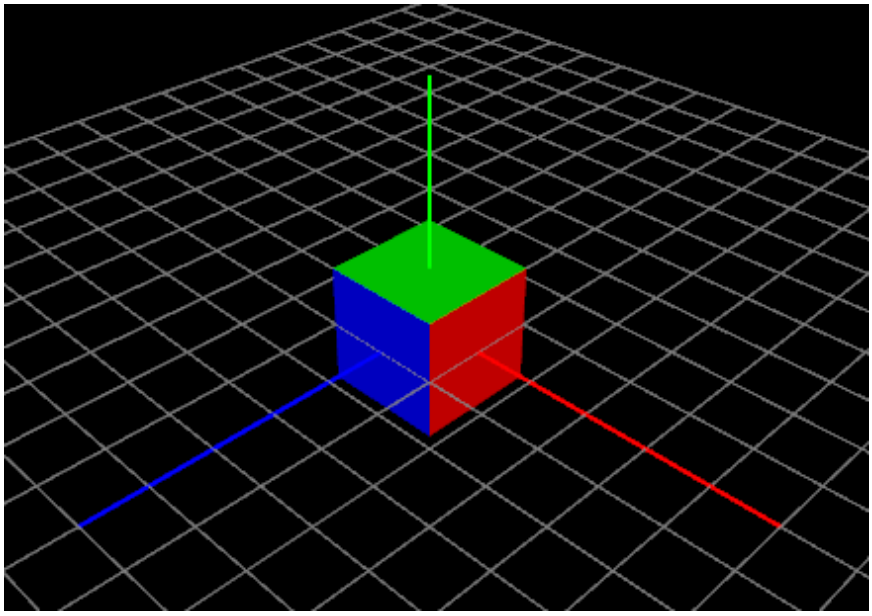
(8) : Bottom Face 즉 아랫 부분 사각 면을 그리는 부분 입니다.
노란색으로 칠하고 있으며, 아직 normals와 texcoords는 사용하지는 않습니다.

(9) : Right Face 즉 우측 부분 사각 면을 그리는 부분 입니다.
빨간색으로 칠하고 있으며, 아직 normals와 texcoords는 사용하지는 않습니다.

(10) : Left Face 즉 뒷 좌측 부분 사격 면을 그리는 부분 입니다.
흰색으로 칠하고 있으며, 아직 normals와 texcoords는 사용하지는 않습니다.

(11) : (1)에서 활성화한 정점, 법선, 텍셀을 비활성화 합니다.
이는 GLES10의 glDisableClientState()를 대체 합니다.

결과는 다음과 같습니다.



2. Vertex Position

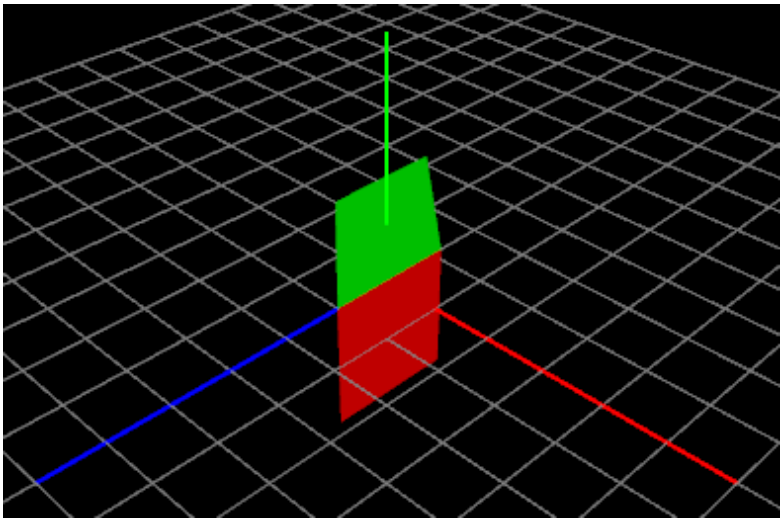
Vertex shader는 본App으로 부터 Position 정점 정보를 a_position 으로 전달 받게 됩니다.

attribute와 uniform으로 선언된 변수들은 shader내에서는 constant(상수?)로 동작합니다.
즉 shader 내에서는 data를 읽기(read)는 가능하지만 쓰기(write)를 할수는 없습니다.

그럼으로 vertex 위치를 임의로 변경하기 위해서는 다음과 같이 vec4 newPos 변수를 만들어 주고 이에 저장합니다.

/Shaders/SimpleLighting.vert

```
// Vertex Position
vec4 newPos = vec4(a_position);
newPos.z = newPos.z + sin(2.0 * newPos.x);

gl_Position = u_projectionMatrix * u_modelViewMatrix* newPos;
```



3. 결론

Cubebox의 정점이 변경하기 전과 약간 변화된 것을 볼수 있습니다.
변형 시키는 sin 함수내의 가중치 값 2.0을 더 크게 하면 변화가 눈에 "뛰게 보이지만, 주전자 만큼 크게 찌그러지지는 않습니다.

```
newPos.z = newPos.z + sin(2.0 * newPos.x);
```

이는 상자의 경우 정점의 갯수가 많지 않아서 그런것으로 보입니다.
바른생활님 예제(http://cafe.naver.com/gld3d/406)와 같이 주전자로 그림을 그리면, 얼추 비슷한 결과가 나올 겁니다.

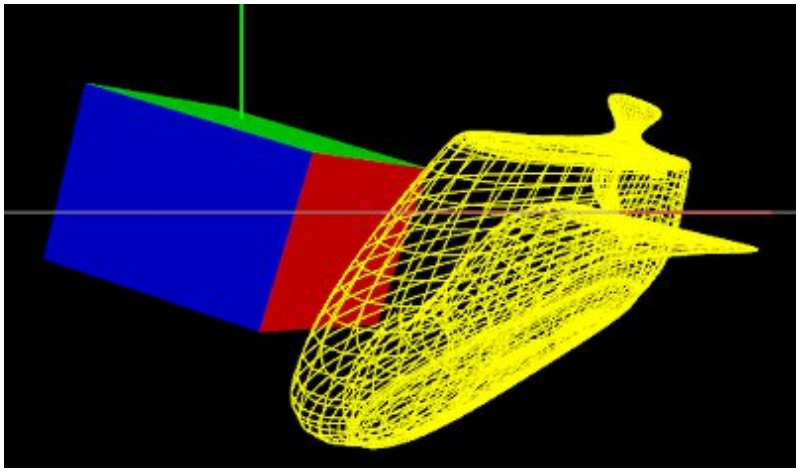
그런데, OpenGL ES에서는 glut lib 를 지원하지 않기 때문에,
OpenGL과 같이 glutSolidTeapot() 함수를 통해서 쉽게 주전자를 그릴수는 없습니다. ㅠ ㅠ

다음 장에서는 주전자를 그리기 위해서 wavefront obj 파일을 header로 추출한 파일을 이용해서,
주전자를 그리고 Vertex Position 보간 값을 적용해 보도록 하겠습니다.

(wavefront obj 파일은 3DMax나 Blender와 같은 3D 그래픽 도구를 이용해서 그린 후 export를 obj 형태로 하면 됩니다. ^^;)

[GLES20] 06. Vertex Position 처리 – Obj 와 VBO (2)

2011.11.15 19:46



2011.11.18 : teapot을 그릴때, normal 값에 대한 offset이 빠져 있었습니다. ^^;
Wireframe 형태일때는 잘 못느껴지겠지만, 면으로 그려보면 느껴지네요.
참고하십시요.
(이전에 Render To Texture 로 그릴때 뺀 이후로 계속 빠져있었던듯 싶네요 ㅎㅎ)

Vertex Shader 내에서 Vertex Position 조정

앞장에 이어서 이번 장에서는 Wavefront Obj 포맷으로 구성된 Obj파일에서 mesh data를 추출 한 후, 이를 Vertex Buffer Object 형으로 구성한 뒤, Wireframe 형식의 주전자(Teapot)를 그려보도록 하겠습니다.

그리고 이 OpenGL App(이하 본App)의 결과를 Vertex Shader 내에서 Vertex Position을 조정해 보겠습니다.

1. Wavefront Obj mesh data

주전자 Teapot를 그리는 방법은 구글링 하면 많이 나옵니다.

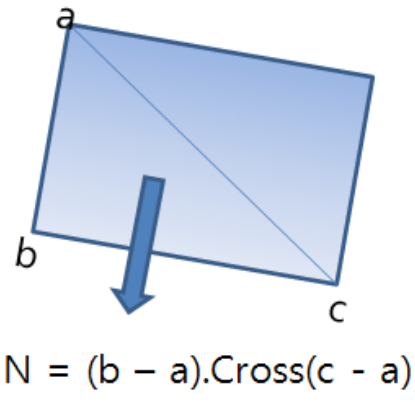
예를 들어, 애플에서 발표한 WWDC 2010 에서 발표했던 CorMotionTeapot 를 참조해도 되고,

Wavefront Obj 모델 파일을 구해서, header로 변환해서 이용해도 됩니다.

Obj스펙은 구글링하면 손쉽게 얻을 수 있으며, iPhone3D Programming 책의 4장에도 설명은 나와 있습니다.
그런데, iPhone3D Programming 책의 경우 OBJ 파일 포맷이 3dsMax나 Blender와는 다른 형태의 mesh 구조 입니다.

작명에도 나와 있지만, Insanely Simple ObJ File(직역하면, "제정신이 아니게 간단한 OBJ File) -ㅁ-; 입니다.
또한 vertex와 면만을 구하고 normal과 texture coordinate 은 고려하지 않고 만들어진 파서 입니다.

책에서는 normal 값을 Triangle 를 구성하는 3개의 정점이 한개의 면을 이루게 됩니다.
같은 평면상에 있는 두개의 벡터의 외적을 구하면 법선을 계산할 수 있습니다.
즉, 면을 구성하는 3개의 정점이 a, b, c라고 한다면, N = (b - a).Cross(c - a)가 됩니다.



N = (b – a).Cross(c - a)

혹 이 책을 보시고 Obj 포맷 파싱을 책과 같이 하면 되겠구나 오해 하시면 안되며,
Example 4.23과 같이 f(face) 값 같이 숫자 사이에 '/' 으로 구분자를 인식하게금 만드셔야 합니다.

Example 4.22. Insanely Simple OBJ File

```
# This is a comment.

v 0.0 1.0 1.0
v 0.0 -1.0 1.0
v 0.0 -1.0 -1.0
v -1.0 1.0 1.0

f 1 2 3
f 2 3 4
```

Example 4.23. An OBJ File with Vertex Normals

```
v 0.0 1.0 1.0
v 0.0 -1.0 1.0
v 0.0 -1.0 -1.0
vn 1 0 0
f 1//1 2//1 3//1
```

3ds Max나 Blender 로 만든 Obj 파일은 Example 4.23 과 같은 포맷을 얻을 수 있습니다.

그럼으로 이책에서 소개하는 OBJ 파서는 이런 포맷도 있다고 이해하시고 다른 파서를 구하시는 것이 좋습니다.

저는 다음과 같이 prebuilt header로 이미 파싱된 데이타를 이용합니다.
http://heikobehrens.net/2009/08/27/obj2opengl/
obj2opengl 이라고 perl script 형태이며, 손쉽게 헤더 파일을 생성해 줍니다.
(Material 까지 파싱은 하지 못하기 때문에, 이 값은 수작업으로 설정 해야 합니다. ^^;)

주전자 하나 그리는데 너무 삼천포로 빠져 버렸네요 ㅠ ㅠ
그만큼 Opengl es는 opengl 보다 뭔가를 그리기가 쉽지가 않습니다.
자기만의 utility function 등을 구현해서 Module화 해놓는 것이 좋습니다.

그럼 다시 Teapot 주전자 그리기로 돌아가도록 하겠습니다.

Mesh data는 정점(teapot2Verts)과 법선정보(teapot2Normals) 두가지만 담겨 있습니다.(텍스쳐 정보는 없음)

Models/teapot2.h

```
/*   ------------------------------------ (1)
created with obj2opengl.pl
source file    : ./teapot2.obj
vertices       : 3644
faces          : 6320
normals        : 5995
texture coords : 0

// include generated arrays
#import! "./teapot2.h"
// set input data to arrays
glVertexPointer(3f, GL_FLOAT, 0, teapot2Verts);
glNormalPointer(GL_FLOAT, 0, teapot2Normals);
// draw data
glDrawArrays(GL_TRIANGLES, 0, teapot2NumVerts);
*/

unsigned int teapot2NumVerts = 18960;   // ------- (2)

float teapot2Verts [] = {   // -------------------- (3)
  // f 2909//1 2921//1 2939//1
  0.204248774988595f, 0.110553208769824f, -0.0353058869814484f,
  0.206408240328974f, 0.105045437554406f, -0.0356647617094558f,
  0.209210851457354f, 0.105045437554406f, 3.80670129563477e-05f,
            ......
};

float teapot2Normals [] = {   // ------------------ (4)
  // f 2909//1 2921//1 2939//1
  -0.926910681623383f, -0.368160873543573f, 0.0727609750079555f,
  -0.926910681623383f, -0.368160873543573f, 0.0727609750079555f,
  -0.926910681623383f, -0.368160873543573f, 0.0727609750079555f,
            ......
```

};

---

(1) : obj2opengl.pl 스크립트를 이용해서 생성된 Obj 파일에 대한 정보 입니다.
GLES 1.0 에서 사용하는 Example을 보여 줍니다.

(2) : 주전자의 정점 갯수를 나타냅니다. teapot2NumVerts (18960)
이 값은 면의 갯수가 6320개임으로, 한개의 면당 3개의 정점이 있기 때문에 18960개가 됩니다.

(3) : 주전자 정점의 위치 입니다. = teapot2Verts
배열 정보는 1차원 배열이지만, 한줄당 면 한개를 표현하는데 필요한 정점 3개를 나타냅니다.
그럼으로, 면의 갯수가 6320임으로 전체 줄수는 6320줄이며 한 줄당 3개의 정점을 포함하고 있기 때문에,
18960 요소가 있습니다.
이는 일차원 배열로는 [18960]이며 이차원 배열로는 [6320][3] 으로 정의할 수 있습니다.

아래의 정보는 (1)번의 "obj2opengl.pl" 로 생성한 정점 좌표 입니다.
// f 2909//1 2921//1 2939//1
0.204248774988595f, 0.110553208769824f, -0.0353058869814484f,
0.206408240328974f, 0.105045437554406f, -0.0356647617094558f,
0.209210851457354f, 0.105045437554406f, 3.80670129563477e-05f,

간단하게 살펴 보면, "// f v/vt/vn" 한개의 면을 구성하는데 필요한 vertex/texture/normal 정보를 나타냅니다.
예를 들어 "//f 2909//1" 값을 보면 teapot2.obj 에는 texture 정보는 없다는 것을 알수 있습니다.
2909 : 면을 표현하는데 필요한 vertices 정보는 2909번째 줄에 있는 면 정점
VOID : slash 사이에 0을 채우는 게 아니라 비워둡니다.
1 : 면을 표현하는데 필요한 normal 정보는 1번째 줄에 있는 면의 법선을 나타냅니다.

다시 teapot2.h를 보면 teapot2Verts의 경우,
첫 째줄의 정점 x, y, z 정보는 teapot2.obj의 2909번째 줄의 정점값을 나타내야합니다.

그런데, teapot2.obj의 2909줄의 vertex 정보는 "v 1.368074 2.435437 -0.227403" 으로 그 값이 다릅니다.

이는 obj2opengl.pl 스크립트를 만드신 분의 정점에 대한 정보를 최적화? 시키면서 변경된 사항입니다.
대략적인 알고리즘은 각각의 정점 x, y, z 중 중간값 center(x, y, z) 정보를 구하고,
scale factor 값을 x, y, z 정점중 각각의 차중 가장 큰값을 1로 나눠서 정규화 scale factor를 구합니다.
이 center 정보와 scalefac에 의해서 계산 되었기 때문에 다른겁니다.

teapot2.obj나 위의 obj2opengl.pl 패키지에 포함된 banana.obj 는 정점이 많은 모델이기 때문에 이해가 어려울수 있습니다.

obj2opengl.pl 패키지 내에 같이 포함된 cube.obj와 이를 컨버팅한 cube.h를 비교해서 보시면 좀더 쉽게 해석할 수 있을 것 같습니다.

```
cubeVerts[] = {
  // f  1//2  7//2  5//2
  -0.5, -0.5, -0.5,
  0.5, 0.5, -0.5,
  0.5, -0.5, -0.5,
};
```

cube.obj

```
v  0.0  0.0  0.0
v  0.0  0.0  1.0
v  0.0  1.0  0.0
v  0.0  1.0  1.0
v  1.0  0.0  0.0
v  1.0  0.0  1.0
v  1.0  1.0  0.0
v  1.0  1.0  1.0
```

Cube의 경우, 정점의 range는 0 ~ 1.0 임으로 center(x, y, z)는 (0.5, 0.5, 0.5) 입니다.
scale factor는 xmax - min = xdiff 가 1임으로 1.0 / 1.0이 되서 1.0입니다.
(x_token - center.x) / 1.0 = (0.0 - 0.5) / 1.0 = -0.5
(y_token - center.y) / 1.0 = (0.0 - 0.5) / 1.0 = -0.5
(z_token - center.z) / 1.0 = (0.0 - 0.5) / 1.0 = -0.5
첫 번째 줄의 값이 됩니다.
일곱 번째 줄의 face 값은 x(1.0), y(1.0), z(0.0)임으로 컨버팅 값은 vx(0.5), vy(0.5), vz(-0.5) 가 됩니다.
다섯 번째 줄의 face 값은 x(1.0), y(0.0), z(0.0)임으로 컨버팅 값은 vx(0.5), vy(-0.5), vz(-0.5)가 됩니다.

(4) : 주전자 법선 정보 입니다. = teapot2Normals
앞의 (3) 정점 정보와 같은 방식으로 계산할 수 있습니다.

```
  // f 2909//1 2921//1 2939//1
  -0.9269106816233831f, -0.3681608735435731f, 0.07276097500795551f,
  -0.9269106816233831f, -0.3681608735435731f, 0.07276097500795551f,
  -0.9269106816233831f, -0.3681608735435731f, 0.07276097500795551f,
```

엄하게 Vertex Position을 다룰려고 했는데 Obj Parser를 다루게 되어 버렸네요 -ㅁ-;

책 수준에서 공부할 때는 이 부분을 넘겨도 되지만, 차후 상용 프로젝트을 진행할 경우에는 3D 디자이너와 협업을 하게 되는데, 이때 3D 디자이너는 3ds Max(유료)나 Blender(무료) 툴을 이용해서 Model을 그려서 Obj 포맷으로 배포할 수 있습니다.

그럼으로 Obj spec(http://www.martinreddy.net/gfx/3d/OBJ.spec)을 읽어 보시고 직접 Uitility module을 개발 하시는 것도 좋은 공부가 될 수 있습니다.

다음에 Texture 장을 다룰때는 텍스쳐 정보가 같이 있는 Obj파일인 banana.obj와 banana.jpg을 다뤄 보도록 하겠습니다.

2. Vertex Buffer Object

앞절에서 모델을 header로 변환하는 방법에 대해서 설명하였습니다.
이번 절은 주전자 정보를 stl의 vector 클래스에 보관한 후 이를 VBO에 저장하는 방법에 대해서 다루겠습니다.
iPhone 3D Programming 예제중 ModelViewer.ObjViewer 프로젝트가 있습니다.
ISurface interface를 상속 받아서 ObjSurface라는 모듈을 구현 하고 있습니다.
앞절에서 소개 했었던 Example 4.22 Insanely Simple Obj 파일로부터 데이타를 추출하고,
vertices 정보를 저장하고 normals 값은 계산해서 저장하고 있습니다.

저는 이 파서를 사용하지 않고 다음과 같은 ObjSurfaceFromData 모듈을 만들어서 사용합니다.

Utils/ObjSurfaceFromData.cpp

이 모듈은 Obj 파일로 부터 추출한 vertices, textures, normals의 정보를 stl의 vector 클래스를 이용해서 저장합니다.

특이사항은 obj2opengl.pl 이 indices 색인 정보를 이용해서 삼각형을 그리는 glDrawElements() 방식이 아닌,
정점으로 삼각형을 그리는 glDrawArrays() 방식 임으로 indices 정보는 저장하지 않습니다.

ObjSurfaceFromData(int numVerts, float* vertices, float* normals);
정점과 법선 정보 한 묶음(stride)으로 만들어서 stl의 vector 클래스에 저장 합니다.
vertices(x, y, z)와 normal(x, y, z) 임으로 stride 는 float형으로 6개가 배치 됩니다.

ObjSurfaceFromData(int numVerts, float* vertices, float* normals, float* texCoords);
정점, 법선 그리고 텍스쳐 정보를 한 묶음(stride)으로 만들어서 stl의 vector 클래스에 저장 합니다.
vertices(x, y, z), normal(x, y, z), texCoords(s, t) 임으로 stride 는 float형으로 8개가 배치 됩니다.

void GenerateVertices(vector<float>& vertices, unsigned char flags) const;

저장된 정점(x, y, z), 법선(x, y, z), 텍스쳐(s, t) 정보를 포함하는 vector 클래스의 포인터를 반환 합니다.

Vertex Buffer Object로 Data를 옮기기 위해서는 CreateDrawable() 함수를 이용합니다.
VBO에 대해서는 iPhone 3D Programming 책의 "3장 정점과 터치점, 3-3 정점 버퍼 객체를 사용하여 성능 향상 시키기" 장에서 설명이 되어 있으며, 바른생활님이 정리하신 다음 포스트도 읽어 보시면 도움이 될것 같습니다.
http://cafe.naver.com/gld3d/129

```
 // Create the VBO for the vertices.   ------------------------- (1)
struct Drawable {
 GLuint VertexBuffer;
 GLuint IndexBuffer;
 int IndexCount;
 int VertexCount;
 int Flags;
};

 // Create the VBO for the vertices.   ------------------------- (2)
struct Drawables {
 Drawable teapot;
 //
};

void RenderingEngine::Initialize(int width, int height)
{
 // Create the teapot drawable   ----------------------------- (3)
 int flags = VertexFlagsNormals;//VertexFlagsNormals | VertexFlagsTexCoords;
 m_drawables.teapot = CreateDrawable(ObjSurfaceFromData(teapot2NumVerts, teapot2Verts, teapot2Normals),
flags);
      ......
}

Drawable RenderingEngine::CreateDrawable(const ObjSurfaceFromData& surface, int flags)
{
 // Create the VBO for the vertices.   ------------------------- (4)
 vector<float> vertices;
 surface.GenerateVertices(vertices, flags);
 GLuint vertexBuffer;
```

```
glGenBuffers(1, &vertexBuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
glBufferData(GL_ARRAY_BUFFER,
    vertices.size() * sizeof(vertices[0]),
    &vertices[0],
    GL_STATIC_DRAW);

// Create a new VBO for the indices if needed.   ---------------- (5)
int vertexCount = surface.GetVertexCount();
int indexCount = surface.GetTriangleIndexCount();
GLuint indexBuffer;

// 인덱스는 사용하지 않도록 예외 처리.    ------------------------ (6)
if (indexCount < 0) {
 indexBuffer = 0;
 indexCount = -1;
} else {
 // 이 아래 부분은 이 장에서는 사용 되지 않는 처리 임.   ------------ (7)
 vector<GLushort> indices(indexCount);
 surface.GenerateTriangleIndices(indices);
 glGenBuffers(1, &indexBuffer);
 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexBuffer);
 glBufferData(GL_ELEMENT_ARRAY_BUFFER,
     indexCount * sizeof(GLushort),
     &indices[0],
     GL_STATIC_DRAW);
}

// Fill in the data into the drawable structure.   ----------------- (8)
Drawable drawable = {0};
drawable.VertexBuffer = vertexBuffer;
drawable.IndexBuffer = indexBuffer;
drawable.VertexCount = vertexCount;
drawable.IndexCount = indexCount;
drawable.Flags = flags;

// GL_ARRAY_BUFFER에 binding 했던 것을 해제하자. --------------- (9)
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

```
 return drawable;
}
```

---

(1) : Drawable 구조체 입니다.
VBO로 생성한 VertexBuffer ID, IndexBuffer ID, Index 갯수, Vertex 갯수,
그리고 Buffer에 저장된 데이타가 Normal이 포함되어 있는지 TexCoord 가 포함 되어 있는지 알려주는 Flag 값이 있습니다.

(2) : 여러개의 Drawable 포함하는 Drawables 구조체 입니다.
현재는 teapot 하나이지만, 여러개의 모델이 있을 경우 여기에 추가할 예정 입니다.

(3) : teapot drawable를 생성 하는 CreateDrawable() 함수를 호출 합니다.
이 함수는 Vertex Buffer ID, Index Buffer ID 등이 포함된 Drawable 객체를 반환해 줍니다.
또한 ObjSurfaceFromData가 ISurface를 상속 받기 때문에,
추후에 iPhone 3D Programming에서 이용하는 Parametric Surface 형으로 만든 기하 구조와 호환 가능 합니다.
(Parametric Surface는 매개변수 방정식을 이용해서 만든 Geometry Sample(구, 원뿔, 뫼비우스 띠 등) 입니다.)

(4) : 정점 정보를 저장하는 VBO을 생성합니다.

(5) : Vertex의 갯수를 반환 합니다.
Index 의 갯수는 Parametric Surface에서 이용되며, 여기서는 glDrawArrays()로 그리기 때문에 사용 되지 않습니다.

(6) : IndexBuffer와 IndexCount를 사용하지 않는 값으로 설정 합니다.

(7) : VBO의 Index Buffer 를 등록하는 부분입니다.
glDrawElements()로 그릴때 필요 합니다. 주전자를 그릴때는 사용하지 않습니다.
나중에 Parametric Surface를 이용해서 기하 구조를 그릴때 사용할 예정 입니다.

(8) : Drawable 구조체에 VBO로 생성한 ID, Vertex count 등을 저장 합니다.

(9) : VBO의 Binding을 해제 합니다.
여기는 초기화 부분이지 실제 Rendering 하는 부분이 아님으로, VBO를 사용하는 시점에 Binding 해서 사용합니다.
만약 Vertex Buffer Array(VBA)와 번갈아 가면서 모델을 그릴 경우에는 VBO을 UnBind해 놔야 충돌이 발생하지 않습니다.

3. Teapot 그리기

이제 드디어 주전자를 그릴 준비가 다 되었습니다.
OpenGL 에서는 glutSolidTeapot() 함수 하나로 해결이 되지만, OpenGL ES에서는 많은 준비가 필요 합니다. ^^;

주전자를 그리는 방법은 앞장에서 배운 Cubebox 그리기와 유사 하며,
Vertex Shader도 Cubebox에서 사용했던 SimpleLighting.vert를 재 사용 하겠습니다.

차이가 나는 부분은 바탕 화면인 Grid Axis와 1차 모델인 Cubebox는 VBA 형태로 그렸지만,
Teapot은 VBO로 그릴 것 임으로 이 부분에 대해서만 정리하도록 하겠습니다.

```
void RenderingEngine::drawTeapot(float size, int textureMode)
{
                    ......

 /* stride 설정 */   ------------------------------------------ (1)
 int stride = sizeof(vec3);
 if (drawable.Flags & VertexFlagsNormals) {
    stride += sizeof(vec3);
 }
 if (drawable.Flags & VertexFlagsTexCoords) {
    stride += sizeof(vec2);
 }
 const GLvoid* offset = (const GLvoid*) sizeof(vec3);


 /* Load teapot object as VBO */   ------------------------------ (2)
 glBindBuffer(GL_ARRAY_BUFFER, drawable.VertexBuffer);
 glVertexAttribPointer(position, 3, GL_FLOAT, GL_FALSE, stride, 0);
 glVertexAttribPointer(normal, 3, GL_FLOAT, GL_FALSE, stride, 0);
```

```
 glVertexAttribPointer(normal, 3, GL_FLOAT, GL_FALSE, stride, offset);


 /* Select A Texture Based On filter */   ---------------------------- (3)
 glBindTexture(GL_TEXTURE_2D, 0);
 glUniform1i(m_cubebox.Uniforms.TextureMode, 0);

 /* Draw teapot */                  -------------------------------- (4)
 //glDrawArrays(GL_TRIANGLES, 0, drawable.VertexCount);
 glLineWidth(1);
 glDrawArrays(GL_LINES, 0, drawable.VertexCount);
 //glDrawArrays(GL_POINTS, 0, drawable.VertexCount);

                    ......
}
```

---

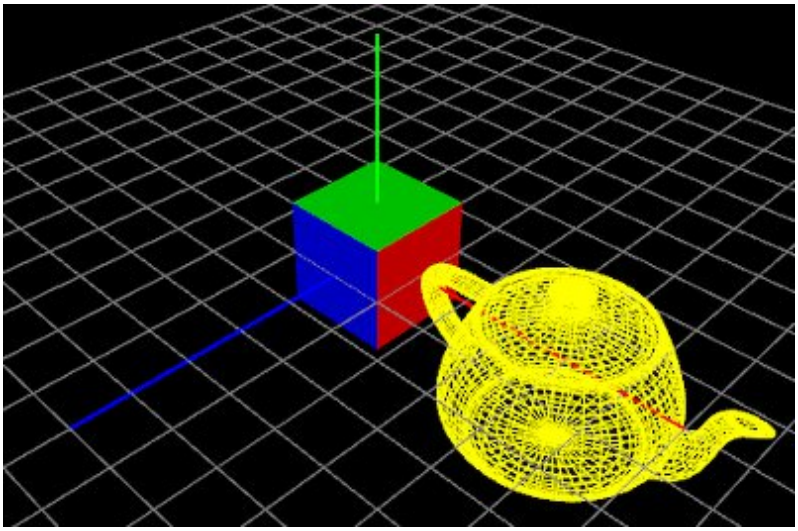
(1) : stride를 설정 합니다.
size(vec3) 는 float x, y, z 임으로 3 * 4byte = 12 입니다. 주전자는 normal 값만 있음으로 stride는 24 입니다.
offset은 normal 값에 대한 간격 입니다. vertices 위치 다음임으로 sizeof(vec3) 으로 설정 합니다.

(2) : 앞에서 생성했던 Teapot VBO 객체를 Bind 합니다.
VBA에서는 vertices와 normals의 배열 포인터를 넘기지만,
VBO는 stride 형태로 Vertex Buffer 의 묶음의 크기를 정하고, 주소는 NULL(0)을 넣습니다.
normal 값에 대해서는 (1)에서 계산한 offset을 설정해 줍니다.

```
  glVertexAttribPointer(position, 3, GL_FLOAT, GL_FALSE, stride, 0);
 glVertexAttribPointer(normal, 3, GL_FLOAT, GL_FALSE, stride, 0);
 glVertexAttribPointer(normal, 3, GL_FLOAT, GL_FALSE, stride, offset);
```

(3) : 텍스쳐는 사용하지 않을 것임으로 UnBind 시킵니다.

(4) : Wireframe 형태로 주전자를 그립니다.
Face 면으로 그릴려면, GL_TRIANGLES 부분의 주석을 해제해 줍니다.
Point 점으로 그릴려면, GL_POINTS 부분의 주석을 해제해 줍니다.

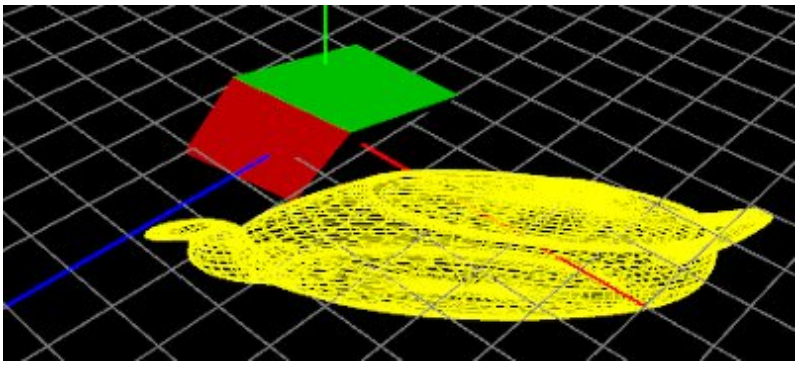


4. Vertex Position

이제 앞 장에서 적용 했던 Vertex Position을 적용해 보겠습니다.

Shaders/SimpleLighting.vert

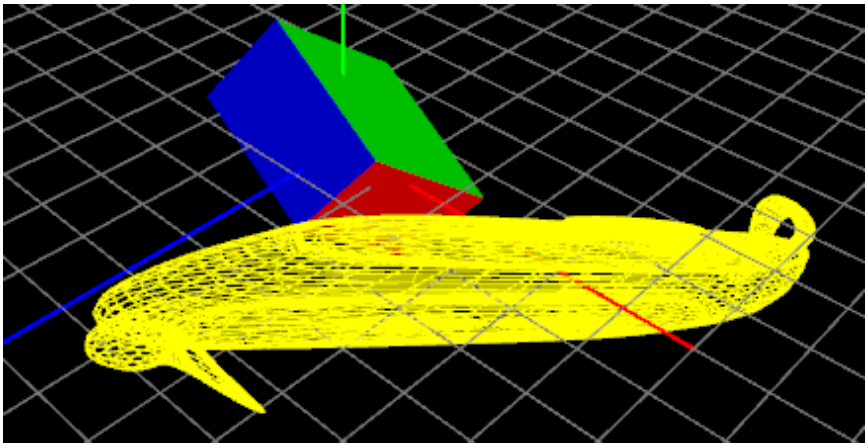
```
// Vertex Position
vec4 newPos = vec4(a_position);
newPos.z = newPos.z + sin(2.0 * newPos.x);

gl_Position = u_projectionMatrix * u_modelViewMatrix* newPos;
```



변화가 생기기는 했지만, 바른생활님 예제 만큼 크게 느껴지지는 않습니다.

가중치 값을 2.0 에서 4.0으로 올려서 할경우,

```
newPos.z = newPos.z + sin(4.0 * newPos.x);
```



Cubebox에 비해서 확연히 찌브러지는 것을 확인할 수 있습니다.

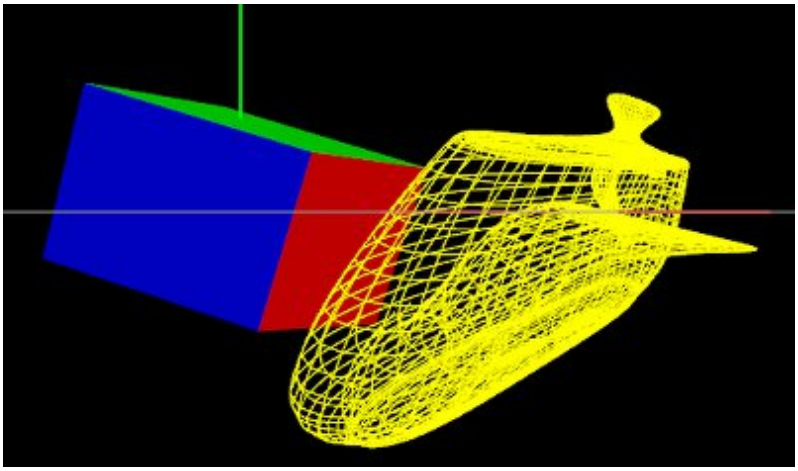
카메라 방향을 다음과 같이 변경해서 바라 볼 경우, 좀더 효과를 잘 볼수 있는 것 같습니다.

```
 vec3 eye(0, 0, 10); //   ----------- (1)
 vec3 target(0, 0, 0); //   ------------ (2)
 vec3 up(0, 1, 0); //  ---------------- (3)

m_camera_0 = mat4::LookAt(eye, target, up);

-------------------------------------------------
```

(1) : eye vector 로 camera의 Z축을 World Coordinate의 원점을 바라보게 설정한다.

(2) : target(center) vector로 camera의 중심을 World Coordinate 기준으로 원점으로 설정한다.

(3) : up vector로 camera의 Y축은 World Coordinate의 X-Y 평면 위에 있게 한다.



5. 결론

RenderingEngine::Render() 함수에서 다음과 같은 순서로 호출하면 됩니다.

```
// VBO을 UnBind 해야 VBA와 충돌이 없다.
 glBindBuffer(GL_ARRAY_BUFFER, 0);
 glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

 /* ---------------------------------------------------------------- */
 // 배경화면 Shader인 Simple shader 프로그램 바인딩.
 drawGridAndAxis();

 /* ---------------------------------------------------------------- */
 /* Use cubebox shader */
 drawCubebox(1.0f);

 /* ---------------------------------------------------------------- */
 /* Use cubebox shader */
 drawTeapot(6.0f);
```

이상으로 Wavefront Obj 파일을 파싱하고, 이 데이타를 VBO에 저장해서 Renderer에서는 VBA와 VBO를 섞어서 사용하는 방법에 대해서 알아 보았습니다.

또한 Shaders/SimpleLighting 쉐이더를 재 사용해서 사용할 수 있음을 확인할 수 있었습니다.

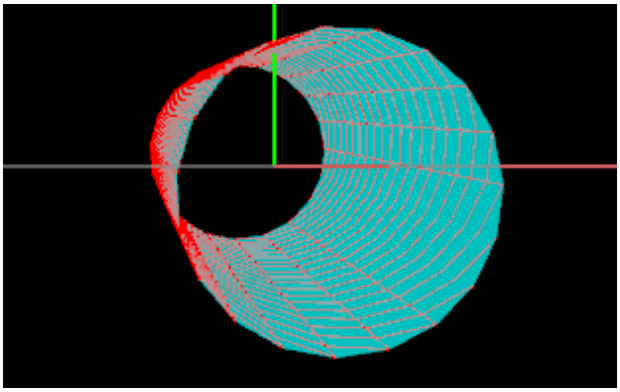
다음 장에는 Pixel Lighting를 에 정리하기 앞서서 Parametric Surface로
"구(Sphere), 원뿔(Cone), 토러스(Torus), 클레인 병(Klein bottle), 뮈비우스 띠(Mobius strip)" 모델을 그려 보도록 하겠습니다.

OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list
http://code.google.com/p/myastro/downloads/detail?name=01_GeometryTransform.v1.1.1.zip&can=2&q=#makechanges

[GLES20] 07. Parametric Surface

2011.11.16 22:19



Parametric Surface

iPhone 3D Programming 책의 4장에 소개된 Parametric Surface는 Parametric representation(매개변수 방정식 표현 법)을 이용해서 기하 구조를 그리는 방법이 서술되어 있습니다.

저는 처음 iPhone 3D Programming 4장의 매개변수 방정식을 봤을때, 이런 수식법이 있었나 가물 가물 했었습니다.
구(Sphere)에 대해서 x^2 + y^2 + z^2 = r^2 인 것은 기억하지만, cosine과 sine을 써서 표현 하는 방법은 생소했었습니다.

그런데, 역시나 고등학교 정석책을 찾아 보니 -ㅁ-; 매개변수 방정식으로된 문제가 있었습니다. ㅠ ㅠ

매개변수 방정식에 대해서는 선형 보간법 Lerp(Linear Interpolation) 장에서 소개한 직선과 원을 표현하는 방법을 읽으면,
혹 도움이 될지도 모르겠네요.

매개 변수 방정식에 대해서 더 궁금하신분은 아래의 싸이트의 수식을 참고하십시요.

http://www.euclideanspace.com/maths/geometry/surfaces/parameterisation/index.htm
http://tutorial.math.lamar.edu/Classes/CalcIII/ParametricSurfaces.aspx
http://www.econym.demon.co.uk/isotut/parametric.htm

위의 수식을 iPhone 3D Programming 책의 예제인 ParametricEquations.hpp에 실제 적용해 보는 것도
좋은 공부될 것 같습니다.

제가 원본 소스에서 수정한 사항은 다음과 같습니다.

1. Cylinder 관련 Parametric Surface 추가
2. Teapot, Banana, Cubebox Obj Surface 추가

나머지 부분은 책(한글 번역서도 있음)의 4장 내용과 동일 함으로 따로 설명을 하지는 않겠습니다. ^^;
(http://ofps.oreilly.com/titles/9780596804824/chrealism.html)

1. Cylinder 관련 Parametric Surface 추가

아쉽게도 iPhone 3D Programming 책에는 실린더를 그리는 예제는 없습니다.
이에 실린더를 매개변수 방정식에 맞춰서 그려 봤습니다.
(http://tutorial.math.lamar.edu/Classes/CalcIII/ParametricSurfaces.aspx 참조)

Utils/ParametricEquations.hpp

```
class Cylinder : public ParametricSurface {
public:
 Cylinder(float radius) : m_radius(radius)
 {
  ParametricInterval interval = { ivec2(20, 20), vec2(Pi, TwoPi), vec2(20, 20) };  // ----------------- (1)
  SetInterval(interval);
 }
 vec3 Eval!uate(const vec2& domain) const
 {
  float u = domain.x, v = domain.y;    // ------------------------------------------------ (2)
  float x = u - Pi / 2; // 중앙에 오도록 조정하기 위해서 Pi / 2 를 빼주었다. //  ---------------------- (3)
  float y = m_radius * sin(v);
  float z = m_radius * cos(v);
  return vec3(x, y, z);
 }
private:
 float m_radius;
};
```

기존의 매개 변수 방정식에 맞춰서 구현하였습니다.
실린더의 수식은 원뿔인 Cone 방정식과 유사합니다.

즉 원 뿔 모양에서 위와 아래 둘다 같은 반지름을 갖고 있으면, 실린더 입니다.

(1) : 매개변수의 간격을 정합니다.
첫 번째 ivec2(20, 20)중 앞에 20은 실린더를 몇조각으로 나눌 지 결정하며, 뒤에 20은 19등분한 도형(원)을 결정 합니다.
만약 뒤에 20을 5로 한다면 4각형이 됨으로 앞-뒤가 열려 있는 Cube box가 그려 집니다.

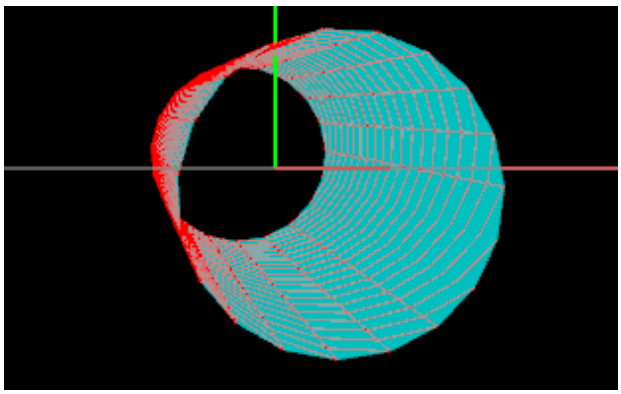
두 번째 vec2(Pi, TwoPi)중 앞에 Pi 는 실린더의 길이를 결정하며, 뒤에 TwoPi는 호의 길이를 결정 합니다.
앞에 Pi는 3.14159f의 길이를 뜻하며, 2.0f 여도 상관 없습니다.
뒤에 인자가 Pi이면 원형의 반만 그려 집니다.

세 번째 vec2(20, 20) 텍셀 좌표를 구할 때 사용 되며 가로 세로 조각 수를 결정합니다.

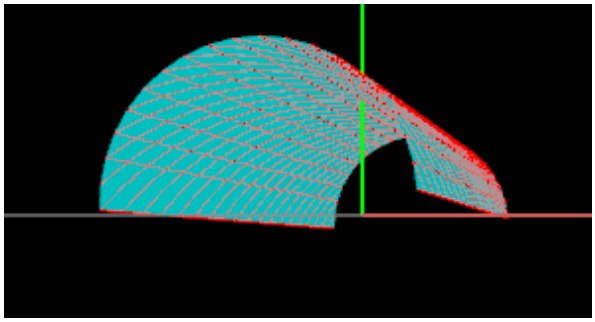
(2) : 정의역 값 이며 Parametric Surface에서 정해 집니다.

(3) : 실린더의 위치를 중앙에서 부터 시작하기 위해서 전체 길이(Pi;3.14159f)의 반을 빼 주었습니다.

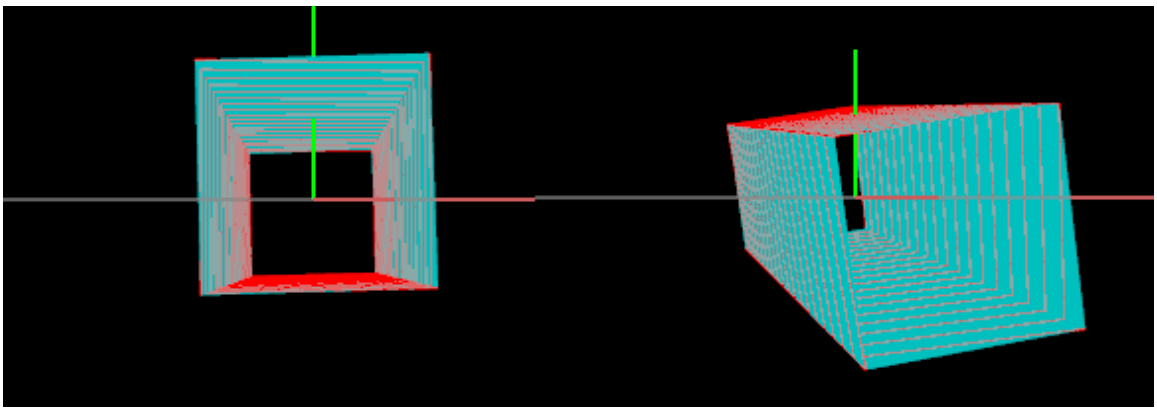
매개변수 값이 "ParametricInterval interval = { ivec2(20, 20), vec2(Pi, TwoPi), vec2(20, 20) };" 일 경우,



매개변수 값이 "ParametricInterval interval = { ivec2(20, 20), vec2(Pi, Pi), vec2(20, 20) };" 일 경우,



매개변수 값이 "ParametricInterval interval = { ivec2(20, 5), vec2(Pi, TwoPi), vec2(20, 20) };" 일 경우,



2. Teapot, Banana, Cubebox Obj Surface 추가

매개 변수 방정식은 재미난 공부는 될수 있겠지만, 한계가 분명히 있습니다.
아무래도 수식을 아무리 적용해도 한계가 있는 부분이 분명히 있으며, 디자이너가 그린 것보다 예쁘지도 않습니다. ㅠ ㅠ

거의 대부분의 실무에서는 디자이너가 3ds Max나 Blender 등으로 그려서 추출해서 준 3ds나 obj 파일을 임포트 해서 사용하게 됩니다.
(물론 Obj등의 파일을 그대로 이용하지 않고 이미지 용량을 줄이기 위해서 압축해서 사용하기도 합니다.
또한 리소스를 외부에서 사용하지 못하도록 막기위해 인코딩을 해서 사용하기도 합니다. ^^;)

이전 장에서 주전자 Teapot Obj 파일을 해석해서 사용하는 방법에 대해서 다뤘음으로 앞장을 참고하십시요.
앞장과의 차이점은 RenderingEngine.CH02.ES2.cpp 이 아닌, ApplicationEngine.CH02.cpp 에 구현 되어 있습니다.

Classes/ApplicationEngine.CH02.cpp

```
void ApplicationEngine::Initialize(int width, int height)
{
    m_textureIndex = 3;
//    LoadTexture();
    vector<ISurface*> surfaces(SurfaceCount);
    surfaces[0] = new Sphere(1.4f);
    surfaces[1] = new Cone(3, 1);
    surfaces[2] = new Torus(1.4f, 0.3f);
    surfaces[3] = new TrefoilKnot(1.8f);
    surfaces[4] = new KleinBottle(0.2f);
    surfaces[5] = new MobiusStrip(1);
    surfaces[6] = new Quad(1, 1);
    surfaces[7] = new Cylinder(.5f);
    surfaces[8] = new ObjSurfaceFromData(CH02::teapot2NumVerts, CH02::teapot2Verts, CH02::teapot2Normals);
    surfaces[9] = new ObjSurfaceFromData(CH02::bananaNumVerts, CH02::bananaVerts, CH02::bananaNormals);
    surfaces[10] = new ObjSurfaceFromData(CH02::cubeNumVerts, CH02::cubeVerts, CH02::cubeNormals);

    m_renderingEngine->Initialize(surfaces);
    for (int i = 0; i < SurfaceCount; i++)
        delete surfaces[i];

    ResizeWindow(width, height);
}
```

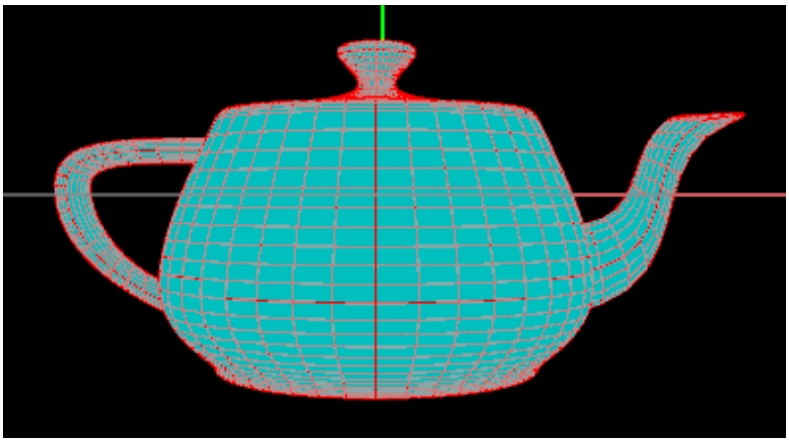
0~7 번까지는 Parametric Surface이며, 8 ~ 10 번까지는 Obj Surface 입니다.

Classes/RenderingEngine.CH02.ES2.cpp

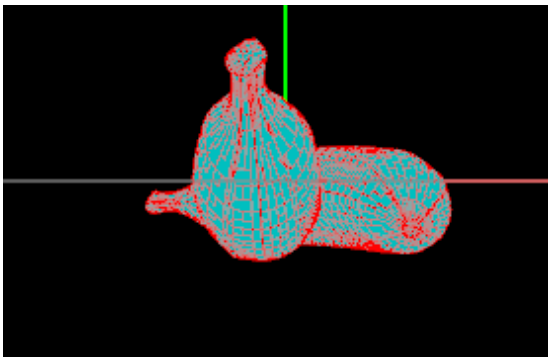
```
void RenderingEngine::Render(const vector<Visual>& visuals)
{
            ......

ProgramHandles handler = m_pixelLight;
handler = m_vertexLight;        // ------------------------------------ (1)
```

```
// Set the light Mode : 0(Diffuse Mat Color), 1(DIFFUSE), 2(AMBIENT_DIFFUSE), 3(AMBIENT_DIFFUSE_SPECULAR)
glUniform1i(handler.Uniforms.LightMode, 0);    // ------------------------- (2)



-----------------------------------------------------------------------

(1) : VertexLighting 으로 실행 합니다.
Diffuse Material 만으로 볼 것임으로 Pixel Light 이든 Vertex Light 이든 크게 상관은 없습니다.

(2) : 확산광 Diffuse Material 재질 값만으로 그립니다.
이는 Light Off 상태 입니다.

            ......
}
```

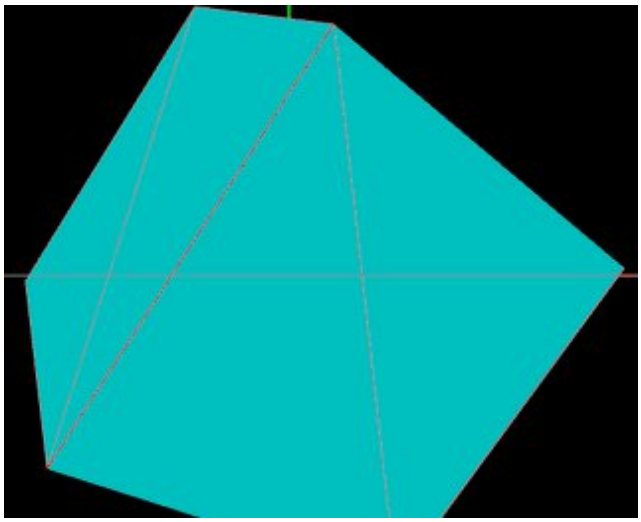
Teapot Obj 결과



Banana Obj 결과

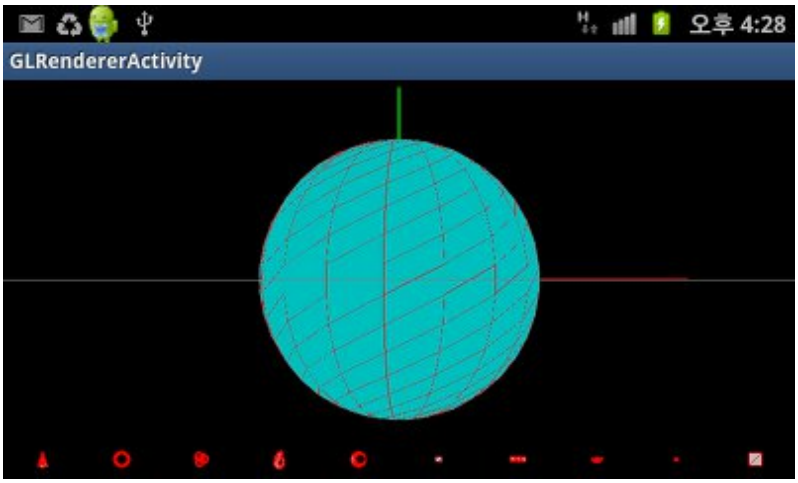


Cubebox Obj 결과



3. 결론

iPhone 3D Programming 책의 4장에서 예제로 제공된 Parametric Surface를 mfc와 android 에 변경해서 작업해 보았습니다.



조명을 적용하기에 앞서서 조명을 투영할 다양한 기하 구조를 그려 보았습니다.

다음 장에서는 Ambient Light(주변 광), Diffuse Light(확산 광), Specular Light(경면 광) 등에 대해서 다루도록 하겠습니다.

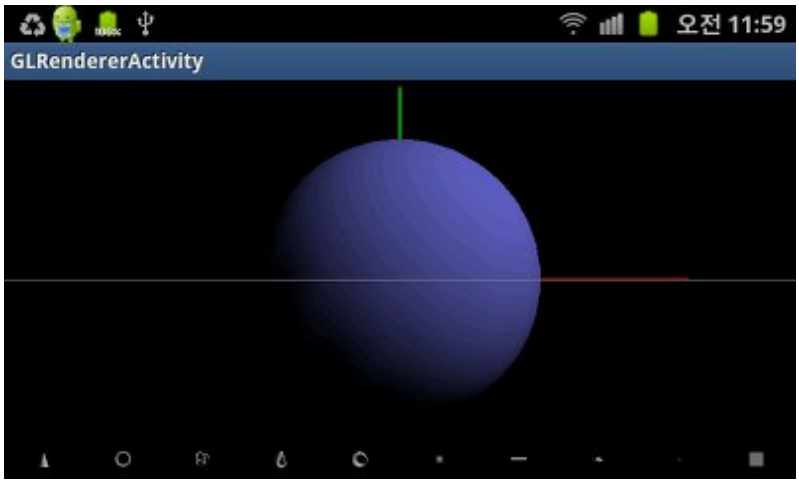
OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list

http://code.google.com/p/myastro/downloads/detail?name=02_ParametricSurface_v.1.2.0.zip&can=2&q=#makechanges

[GLES20] 08. 조명 적용하기 – 확산광, Diffuse Vertex Light

2011.11.20 17:04



조명 적용하기 - Vertex Light

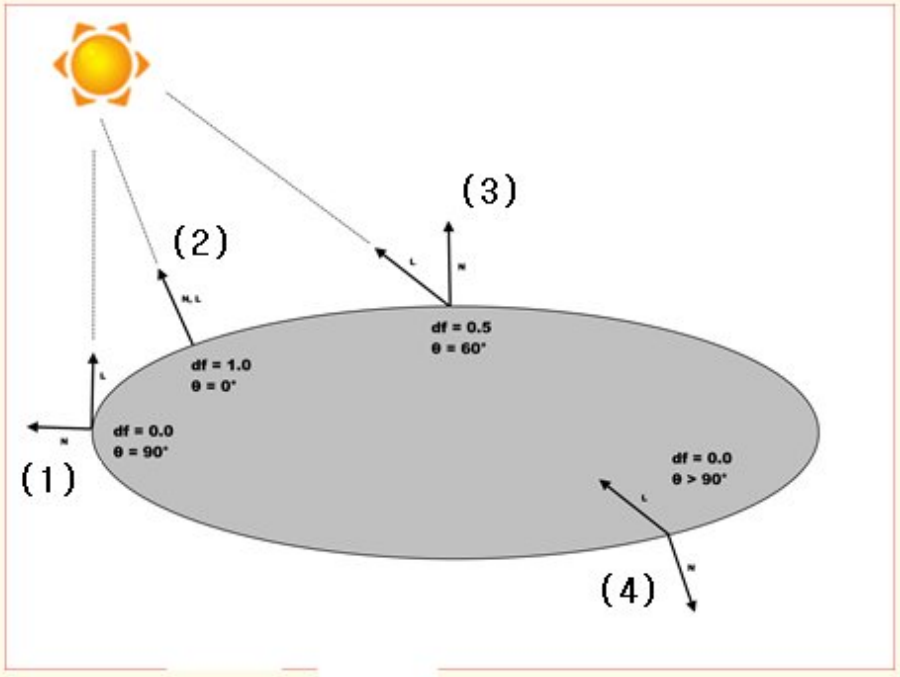
여기서 다루는 조명 적용하기는 iPhone 3D Programming 책의 "4-5장 조명 적용하기"의 Vertex Light 부분을 다룹니다.
상세한 설명은 책을 보시면 이해가 잘 될 겁니다.
또한 바른생활님의 정리해 놓으신 GLSL 부분 강좌와 비교해서 보시면 GLES20과의 변경 사항을 확인하실 수 있을 것 같습니다.

1. Diffuse Light : 확산광 적용

확산광은 실시간 조명을 나타내는 가장 일반적인 형태라고 하며, 람베르트 반사(Lambertian Reflection)를 이용합니다.

책에도 이미 다 잘 나와있고, 바른생활님도 정리한 내용을 다시 정리하는 것은 중복 작업이 될것 같아서
구체적인 기술은 생략하도록 하겠습니다. ^^;
(바른생활님 GLSL 17강 : Diffuse 조명의 구현(http://cafe.naver.com/gld3d/435))

Figure 4.9. Diffuse Lighting



확산율(Diffuse Factor) 은 위의 그림에서 보이는 df 값입니다.

df = max(0, N dot L)

정점을 통해서 구성된 면(face)의 법선 벡터(N)와 광원의 위치벡터(L) 간에 사잇각이 df 입니다.
(dot 은 사잇각을 구하는 내적)

(1) 법선과 광원의 위치가 직각(90)을 이루면 cos(90)이 됨으로 df는 0.0 이 됩니다.
df = |N|*|L|*cos(90) = 0.0

(2) 법선과 광원의 위치가 수평(0)을 이루면 cos(0)이 됨으로 df는 1.0 이 됩니다.
df = |N|*|L|*cos(0) = 1.0 (N과 L은 normalize 되어 있기 때문에 단위 값으로 볼수 있다.)

(3) 법선과 광원의 위치가 60도를 이루면 cos(60)이 됨으로 df는 0.5가 됩니다.

(4) 법선과 광원의 위치가 90도 보다 크다면, 뒷 면에 있다고 볼수 있음으로 df는 0.0으로 합니다.

이에 대한 Diffuse Light Shader 구현 코드는 다음과 같습니다.

```
const lowp float INTENSITY = 1.0;
```

```
// diffuse light
vec3 calcLightDiffuse()
{
 vec3 N = normalize(eyespaceNormal);    // -------------------- (5)
 vec3 L = normalize(directionLight - vec3(positionLight));    // --- (6)
 float df = max(0.0, dot(N, L));               // -------------------- (7)

return INTENSITY * a_diffuseMaterial * df;   // -------------------- (8)

}
```

---

(5) N : eyespaceNormal 은 관측자 관점에서의 법선벡터 입니다.
이 값은 면당 법선 벡터값 a_normal 과 본App 에서 이미 설정되어 있는 NormalMatrix값을 곱한 값입니다.
본App에서 보면 NormalMatrix 값은 modelView matrix 값이 그대로 적용되어 있습니다.
이는 이 예제에서의 좌표계가 직교 정규화 좌표계이 이기 때문에 modelview에 대한 역 전치가 따로 필요하지 않습니다.

(6) L : 광원의 위치 입니다.
Directional Light 와 Positional Light 의 개념이 들어가는 부분 입니다.

Directional Light : 방향성 조명으로 실 세계를 예로 들어서 태양이 광원이라고 할 경우,
모델 위치로부터 광원이 멀리 떨어져 있기 때문에, 모델의 정점값인 Vertex Position은 원점(0, 0, 0)으로 볼수 있습니다.
즉 광원의 빛은 모델 전체의 Vertex에 균등하게 뿌려주게 됩니다.
이런 상태가 될려면, "- vec3(positionLight)" 부분이 0 벡터가 되어야 합니다.
책이나 기타 다른 소스를 보면, 광원의 위치 값 L를 directionLight 값만 사용하고 vertex postion 값을 사용하지 않는 것은
이 때문입니다.

directionLight는 main()함수에서 구현하였으며, 본App으로 부터 전달받은 u_lightPosition에서 값을 받아 옵니다.
기본 값은 (10.0, 10.0, 10.0, 0.0) 임으로 오른쪽-위-전면에 있게 됩니다.
(참고로 바른생활님 예제는 광원이 (1, 1, 1, 0)인데, 제 예제는 (10, 10, 10, 0)인 이유는
바른생활님 예제는 1.0 단위의 직교좌표계 프로젝션를 사용하고 있으며,
저는 Perspective Projection에 카메라를 사용해서 10만큼 뒤로 땡겼기 때문에
광원도 같이 10만큼 보정해 주었습니다. <-- 혹 이 부분이 틀렸다면 댓글로 지적해 주세요 ^^;)

Positional Light : 위치기준 조명으로 모든 Vertices 정점들에 균등하게 뿌려주는것이 아니라,

방향과 위치 기준으로 해당하는 부분만 뿌려 줍니다.
(해당하는 기준은 "directionLight - vec3(positionLight)" 와 같이 벡터의 차로 구하고 있음으로
directionLight 벡터와 postionLight 벡터(정점) 을 이어주는 선이 되겠습니다.)

더 상세한 내용은 바른생활님의 정리하신, "21강 Positional Light에 대한 이해" 장을 읽어 보시면 도움이 될겁니다. ^^;

(5), (6)번에 normalize()함수를 처리하고 있는데 중요한 부분입니다.
바른생활님 강좌중 "14강 Fragment shader를 이용한 pixel별 연산" 부분에 설명이 나와 있으며,
제 블로그에서도 "선형 보간 법 Lerp"의 단위 사원수를 Normalize 하는 부분에 왜? 정규화가 필요한 지 설명이 있습니다.
참고하십시요.

(7) Diffuse Factor 확산율을 구합니다.

(8) INTENSITY : 빛의 강도를 뜻합니다.
더 밝게 또는 약하게 할 수 있습니다.
여기서는 1.0으로 설정해서 적용하지는 않고 있습니다.

Diffuse Light main() 함수

Shaders/VertexLighting.vert

```
void main(void)
{
  eyespaceNormal = u_normalMatrix * a_normal;

  // Calculate Color
  vec3 color = a_diffuseMaterial;

  // Calculate Light
  // Directional Light     // ---------------------------- (1)
  positionLight = vec4(0, 0, 0, 0);
  directionLight = vec3(u_lightPosition);

  // Positional Light      // ---------------------------- (2)
  //positionLight = u_modelViewMatrix * a_position;
  //directionLight = vec3(u_lightPosition);
```

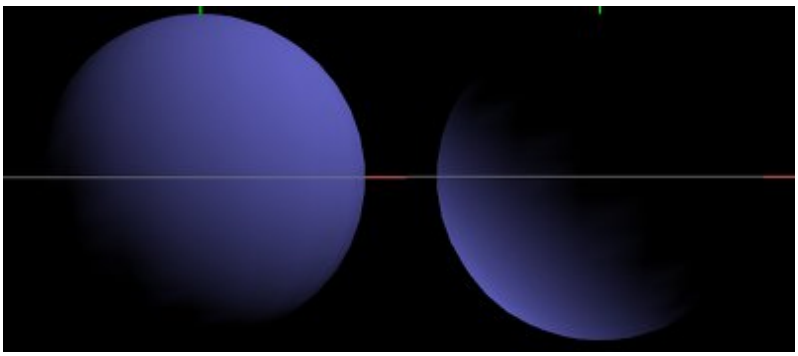
```
  //directionLight *= u_interpolation_z;

  if (u_lightMode == DIFFUSE_LIGHT)  // ------------------ (3)
     color = calcLightDiffuse();
  else if (u_lightMode == AMBIENT_DIFFUSE_LIGHT)
     color = calcLightAmbDif();
   else if (u_lightMode == AMBIENT_DIFFUSE_SPECULAR_LIGHT)
     color = calcLightAmbDifSpec();

   v_destinationColor = vec4(color, 1.0) * u_color;

   // Vertex Position
   gl_Position = u_projectionMatrix * u_modelViewMatrix * a_position;
}
```

---

(1) : Directional Light를 설정해 줍니다.
positionalLight는 0 벡터여야 하며, Positional Light (2) 번 부분은 주석으로 막아 주셔야 합니다.

(2) : Positional Light를 설정해 줍니다.
positionalLight 의 설정 부분입니다. 적용을 위해서는 주석을 풀어 주면 됩니다.

(3) : 확산광 계산을 실행 시킵니다.
u_lightMode 정보는 본App으로 부터 얻어 옵니다.



〈좌측은 Directional Light 상태이며, 우측은 Positional Light 상태의 결과〉

2. 본App 에서 Diffuse Light 실행 하기 (Vertex Lighting)

```
// Classes/RenderingEngine.CH02.ES2.cpp

void RenderingEngine::Initialize(int width, int height)
{
    ......
 // Set up some default material parameters.
 vec4 diffuseLight = vec4(1, 1, 1, 1); // 백색광
 vec4 ambientLight = vec4(0.05f, 0.05f, 0.05f, 1.0f); // 흑색광
 vec3 diffuseMaterial = vec3(1, 1, 1); // 백색 재질
 vec3 ambientMaterial = vec3(1.0f, 0.f, 0.f); // 적색 재질
// vec3 ambientMaterial = vec3(0.0f, 0.f, 0.f); // 주변광 끄기
 vec3 specularMaterial = vec3(1, 1, 1); // 백색 재질
    ......
}


void RenderingEngine::Render(const vector<Visual>& visuals)
{
    ......

ProgramHandles handler = m_pixelLight;
handler = m_vertexLight;        // ------------------------------------ (1)

GLfloat weight = cos(interpolation_z) * 1.5f;    // ------------------------- (2)
glUniform1f(handler.Uniforms.Interpolation_z, (GLfloat) weight);
LOG_PRINT("interpolation_z:%f, cos(%f)", interpolation_z, weight);
interpolation_z += (2*Pi) * 0.0008f;


// Set the light Mode : 0(Diffuse Mat Color), 1(DIFFUSE), 2(AMBIENT_DIFFUSE), 3(AMBIENT_DIFFUSE_SPECULAR)
glUniform1i(handler.Uniforms.LightMode, 1);    // ------------------------- (3)
```

```
    ......
}
```

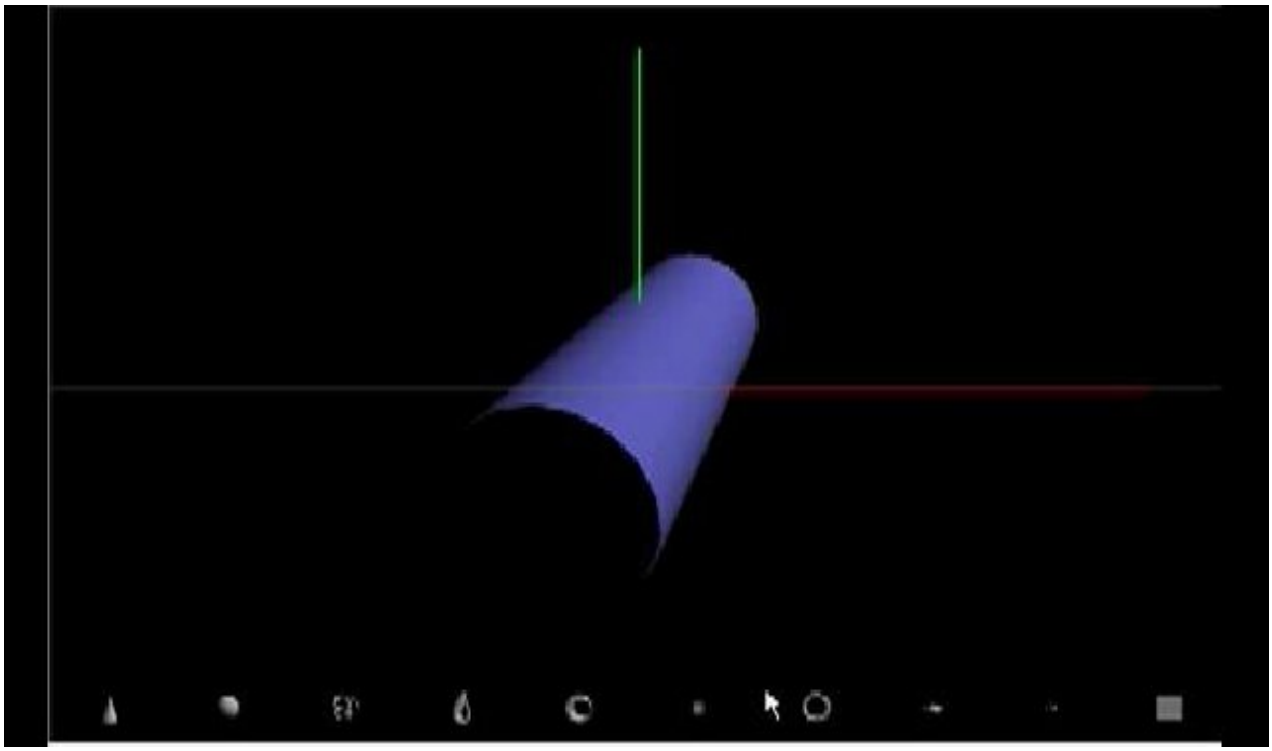
---

(1) : VertexLighting 으로 실행 합니다.
pixelLight 보다 성능이 떨어지지만 Diffuse 만으로 볼 경우에는 크게 차이가 나지는 않습니다. ^^;

(2) : Postional Light 시에 광원의 위치를 이동 시키기 위해서 사용되는 가중치 값입니다.

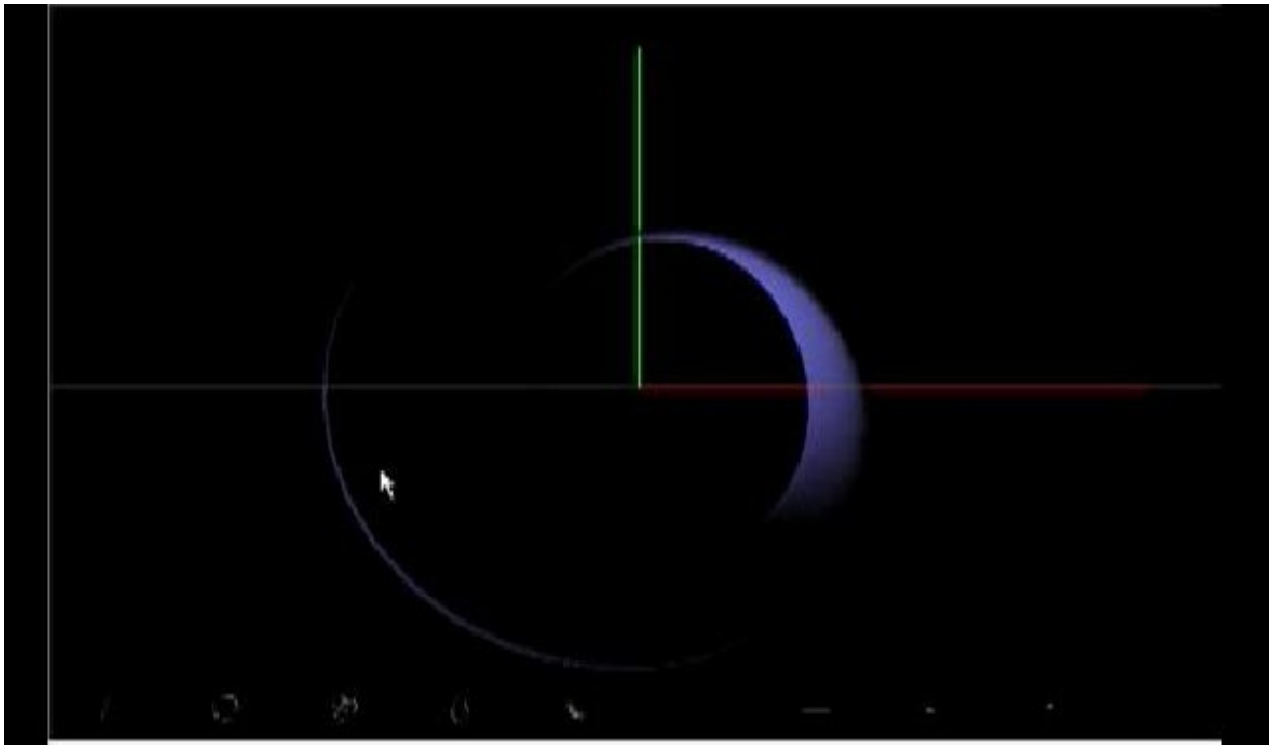
(3) : 확산광 Diffuse Light으로 실행 합니다.

〈Directional Light〉



(용량 1.3MB)

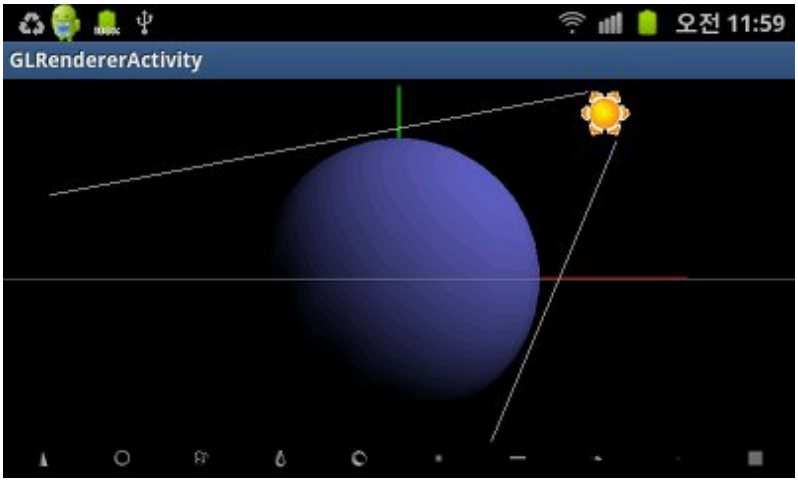
〈Positional Light〉



(용량 1.6MB)

3. 결론

mfc와 android app으로 Vertex Light 확산광을 적용해 보았습니다.
광원에 위치는 아래와 같으며 ^^; Directional Light 적용시에는 아래와 같습니다.



Positional Light 적용시에는 반대 방향으로 이동해서 어둡게 나오게 됩니다 ^^;

다음 시간에는 Ambient Light에 대해서 다뤄보겠습니다. ^^;

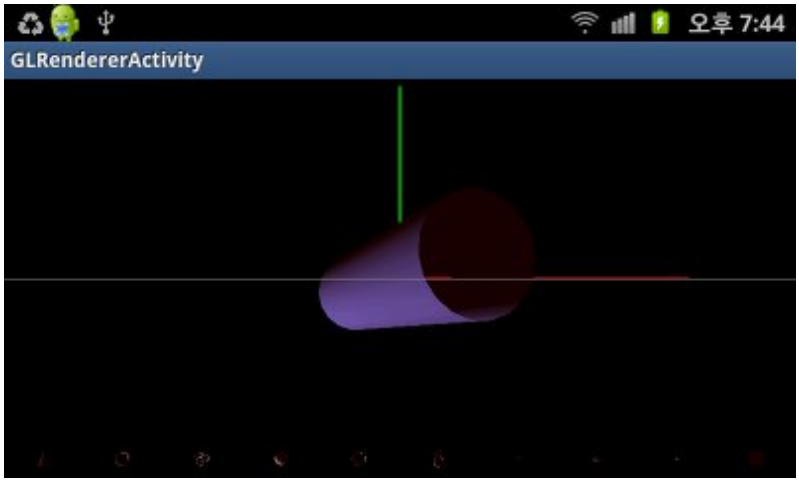
OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list

http://code.google.com/p/myastro/downloads/detail?name=02_ParametricSurface_v.1.2.0.zip&can=2&q=#makechanges

[GLES20] 09. 조명 적용하기 – 주변광, Ambient Vertex Light

2011.11.20 17:09



조명 적용하기 - Ambient Light

1. Ambient Light : 주변광 적용

주변광은 깊이 있게 들어가면 실세계의 주변(환경?)을 그대로 표현 해야하는 조명 표현임으로 무지 어렸다고 합니다. -ㅁ-;

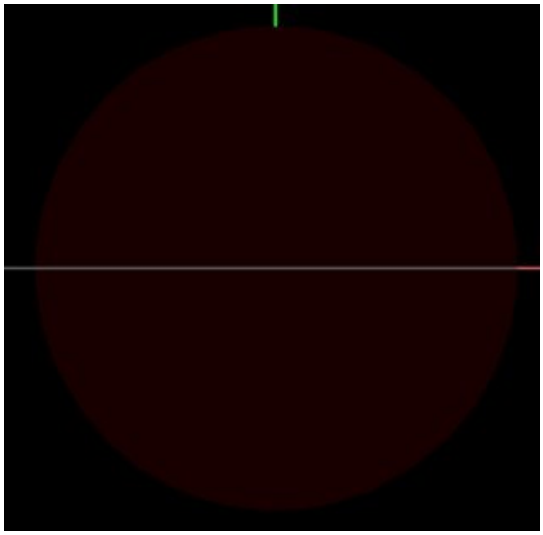
하지만, 일반적으로 OpenGL에서는 '균일한 단색'을 나타낸다고 하니 쉽게 처리할 수 있습니다. ^^;

빛의 강도, 광원의 위치(Light Source Position), 표면의 방향에 의해서 그 색상이 변하지 않기 때문에,
단지 대상 물체의 주변 색상을 나타낸다고 말할 수 있습니다.

그런데, 아무래도 주변광만 사용할 경우, 주변 색상과 물체의 색상이 구분 되지 않기 때문에, 그림자 효과를 넣지 않는한,..
물체와 주변이 구분되지 않게 됩니다.

주변광은 확산광과 경면광 등과 결합해서 조명 효과를 더욱 현실감 있게 만드는데 사용 됩니다.

만약 주변광만 사용해서 그리면 다음과 같이 단색으로만 나오게 됩니다.



(어두운 적색 원이 그려 집니다. 않보이면 뚫어지게 보십시요 ^^;)

위에 그림에서 보듯이 주변광만을 사용하는 경우는 거의 없으며, 다음과 같이 확산광(Diffuse), 경면광(Ambient) 등과 결합해서 사용합니다.

주변광에 대해서 iPhone 3D Programming 책에서는 크게 다루지는 않습니다.
이 부분은 바른생활님 강좌중 "18강 Ambient 조명의 구현" 을 참고하시면 도움이 될것 같습니다.

수식은 다음과 같습니다.

$$I_a = G_a * M_a + L_a * M_a$$

제 예제를 기준으로 보면 주변광은 바탕화면의 색인 검정색이 주변색이 됩니다.
하지만, 검정색을 유지하면, 주변광이 잘 안보임으로 적색으로 변경해서 시험해 보도록하겠습니다.

Shaders/VertexLighting.vert

```
// ambient + diffuse light
vec3 calcLightAmbDif()
{
 vec3 N = normalize(eyespaceNormal);
 vec3 L = normalize(directionLight - vec3(positionLight));
 float df = max(0.0, dot(N, L));

 //  (1)
 vec3 globalAmbient = u_ambientMaterial * vec3(0.05); // * gl_LightSource[0].ambient
 vec3 ambient = u_ambientMaterial * vec3(u_ambientLight);    // ------ (2)
```

```
 return (globalAmbient + ambient) + (INTENSITY * a_diffuseMaterial * df);  // ----- (3)
}
```

---

(1) : 전역 주변광(Ga) 색을 구합니다.

Ma * Ga

globalAmbient는 Ambient Material * gl_LightSource[0].ambient와 결합해서 구합니다.

하지만, GLES20에서는 멀티 광원 처리시 사용되는 gl_LightSource 를 지원하지 않습니다.

따로 uniform 형태로 만들어서 사용해야 하며 여기서는 u_ambientLight를 사용해도 되고,

바탕화면색이 검정색임으로 vec3(0.05)로 처리 하였습니다.

(2) : 주변광(La) 색을 구합니다.

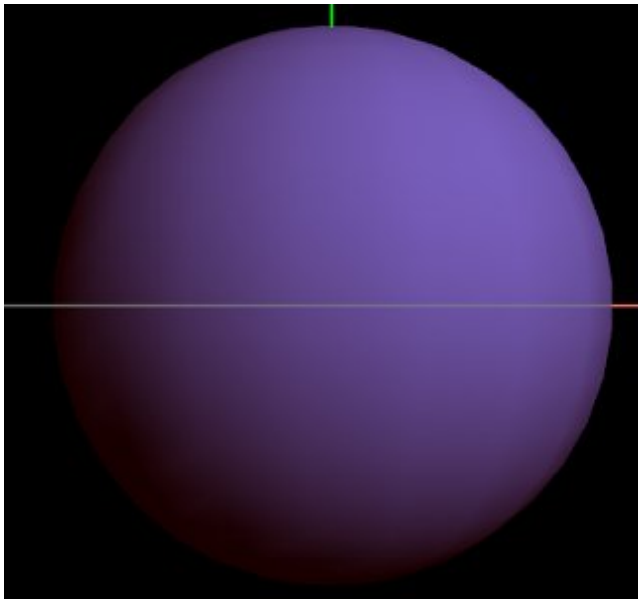
Ma * La

ambient는 Ambient Material과 Ambient Light를 곱해주면 됩니다.

(3) : 전역 주변광과 주변광을 더해주면 됩니다.

주변광은 확산광과 더해 지기 때문에, 그 값이 0 벡터여도 확산광 값에 영향을 미치지 않습니다.

즉, 주변광을 꺼도 확산광으로 표현 되게 됩니다.



(바탕이 검정색이기 때문에 주변광을 검정색이 아닌 적색으로 변경하였습니다.)

2. 본App 에서 Ambient Light 실행 하기 (Vertex Lighting)

Classes/RenderingEngine.CH02.ES2.cpp

```
void RenderingEngine::Initialize(int width, int height)
{
    ......
 // Set up some default material parameters.
 vec4 diffuseLight = vec4(1, 1, 1, 1); // 백색광
 vec4 ambientLight = vec4(0.05f, 0.05f, 0.05f, 1.0f); // 흑색광
 vec3 diffuseMaterial = vec3(1, 1, 1); // 백색 재질
 vec3 ambientMaterial = vec3(1.0f, 0.f, 0.f); // 적색 재질
// vec3 ambientMaterial = vec3(0.0f, 0.f, 0.f); // 주변광 끄기
 vec3 specularMaterial = vec3(1, 1, 1); // 백색 재질
    ......
}



void RenderingEngine::Render(const vector<Visual>& visuals)
{
    ......

ProgramHandles handler = m_pixelLight;
handler = m_vertexLight;        // ------------------------------------- (1)

GLfloat weight = cos(interpolation_z) * 1.5f;    // ------------------------- (2)
glUniform1f(handler.Uniforms.Interpolation_z, (GLfloat) weight);
LOG_PRINT("interpolation_z:%f, cos(%f)", interpolation_z, weight);
interpolation_z += (2*Pi) * 0.0008f;



// Set the light Mode : 0(Diffuse Mat Color), 1(DIFFUSE), 2(AMBIENT_DIFFUSE), 3(AMBIENT_DIFFUSE_SPECULAR)
glUniform1i(handler.Uniforms.LightMode, 2);    // ------------------------- (3)

    ......
}
```

---

(1) : VertexLighting 으로 실행 합니다.
pixelLight 보다 성능이 떨어지지만 Diffuse 만으로 볼 경우에는 크게 차이가 나지는 않습니다. ^^;

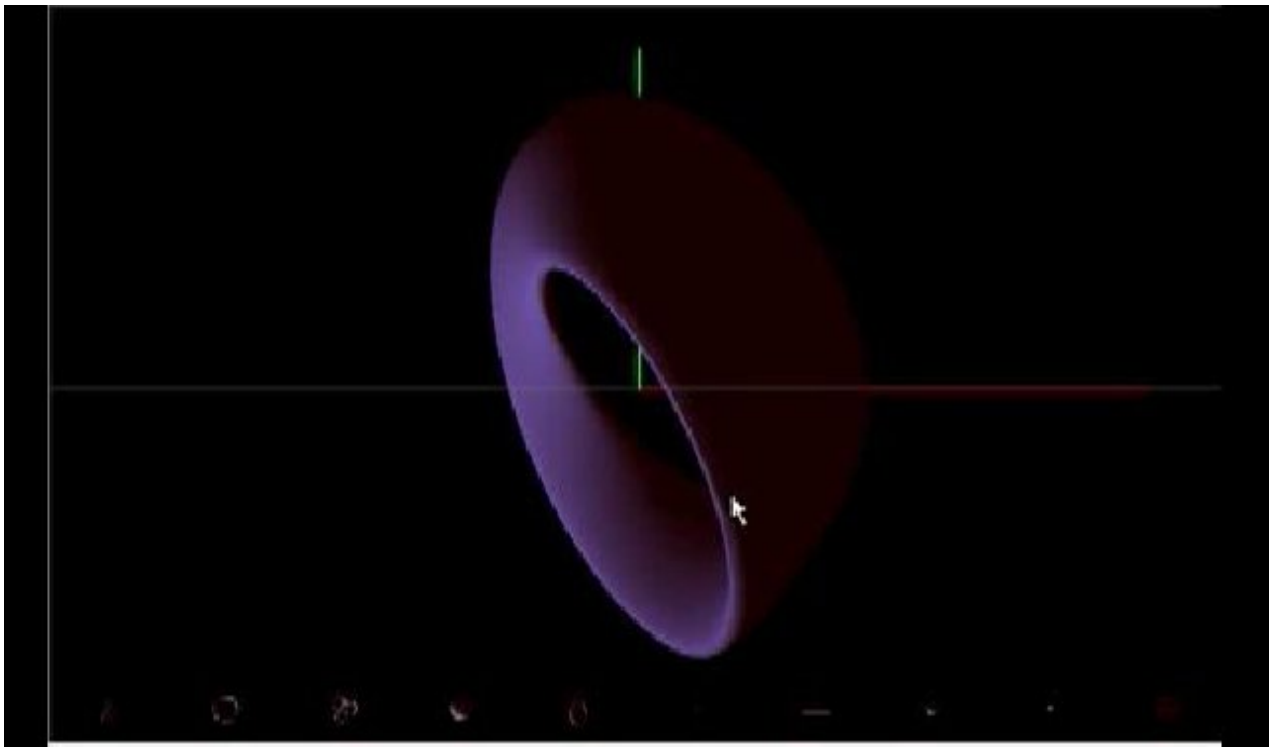
(2) : Postional Light 시에 광원의 위치를 이동 시키기 위해서 사용되는 가중치 값입니다.

(3) : 주변광과 확산광 결합으로 실행 합니다.
원래 주변광은 검정색이지만, 물체와 바탕화면을 구분하기 위해서 적색 재질을 사용하였습니다.
vec3 ambientMaterial = vec3(1.0f, 0.f, 0.f);

계산 수식을 적용할때 주의할 점은 검정색이 (0, 0, 0) 이라고 해서 0으로만 전부 곱하면, 곱셈 법칙에 의해서 대상 값도 0 이 되어 버립니다.
그럼으로, 주변 바탕이 검정색이더라도 0에 가까운 색(0.05 ?)으로 설정해야 합니다. ^^;

만약 주변광을 끄고 싶다면, AmbientMaterial 재질 값을 (0, 0, 0)으로 설정하면 됩니다.

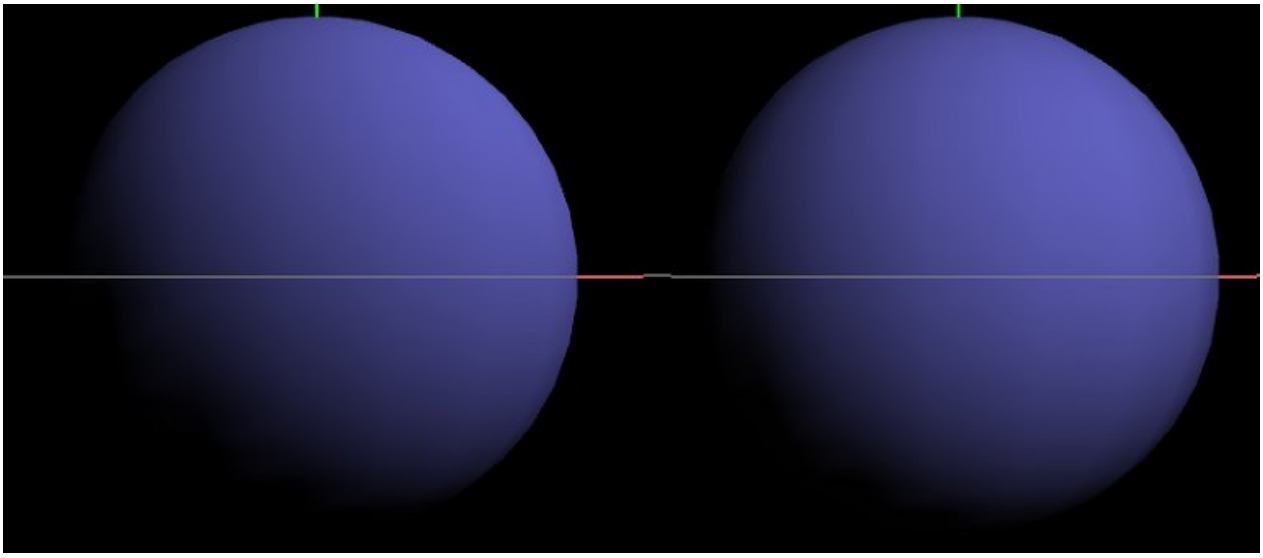
〈Positional Light〉



(용량 1.0MB)

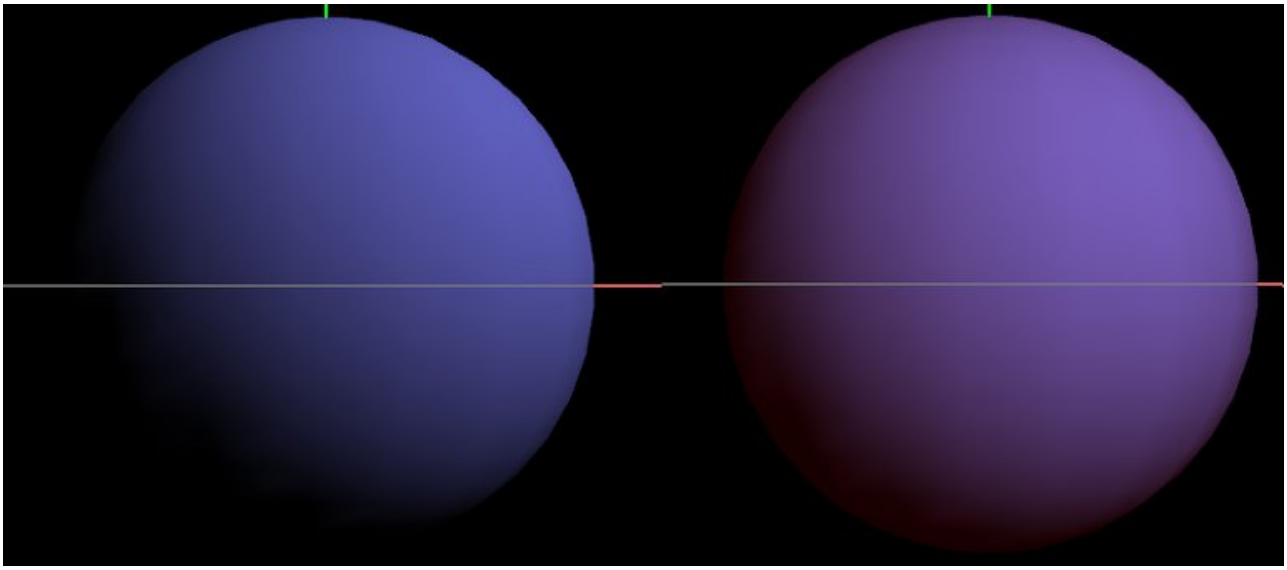
3. 결론

mfc와 android app으로 Vertex Light 주변광 확산광 합성을 적용해 보았습니다.
광원에 위치는 앞장과 같으며, Directional Light 는 생략하고 Positional Light 만 적용하였습니다.



〈좌측은 Diffuse Only 이며, 우측은 Ambient + Diffuse 상태〉

주변광의 재질을 바탕색인 검정색으로 하면 Diffuse 로 했을때와 크게 차이가 느껴지지는 않습니다.
다음과 같이 주변광 재질을 적색으로 할 경우에는 어두운 적색 테두리가 만들어 지게 됩니다.



〈좌측은 Diffuse Only 이며, 우측은 Ambient + Diffuse 상태〉

다음 시간에는 Specular Light에 대해서 다뤄보겠습니다. ^^;

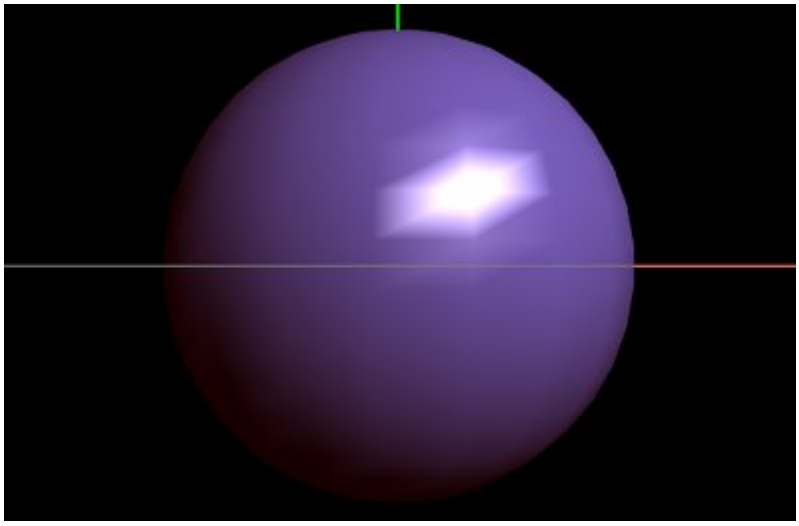
OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list

http://code.google.com/p/myastro/downloads/detail?name=02_ParametricSurface_v.1.2.0.zip&can=2&q=#makechanges

[GLES20] 10. 조명 적용하기 – 경면광, Specular Vertex Light

2011.11.21 23:22



조명 적용하기 – Specular Light

1. Specular Light : 경면광 적용

앞에서 다뤘던 확산광과 주변광 가는 달리 Specular 경면광의 경우, 벡터의 사칙연산, 정사영(Projetion Vector),
반사벡터(Reflection Vector), 반 벡터(Half Vector) 에 대한 선행 학습이 필요합니다.
다음의 블로그를 참고하십시요.

> 벡터의 사칙연산과 정사영은 제 블로그의 "변환 으로 가는 길 #1 : 벡터와 행렬" 에 설명이 되어 있습니다.
> 반사벡터의 경우에는 바른생활님의 "쉐이더로 구현하는 specular 조명" 과
> 감자님의 "반사벡터" 블로그를 참고하시면 도움이 될 것같습니다.

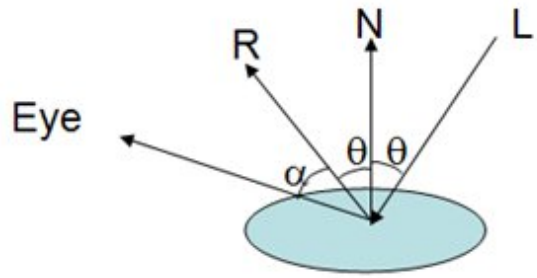
Specular Light 경면광에 대한 자료를 찾아보면 무지 많기 때문에, 이론에 대한 원류 보다는 해석에 좀더 집중하겠습니다.
(바른생활님의 "GLSL 19 장 쉐이더로 구현하는 specular 조명" 를 먼저 보시면 더 좋을 것 같습니다.)

일반적으로 Specular 을 설명할때 Phong Model과 Blinn Model 두가지를 많이 소개 합니다.

1.1 퐁 모델

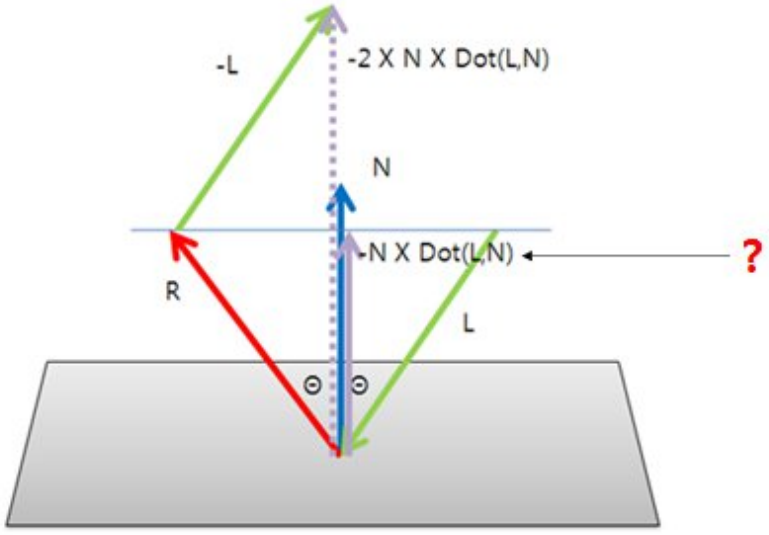
specular 수식은 다음과 같으며, R은 반사 벡터를 의미 합니다.

$$Spec = (R \cdot Eye)^{Shininess} \times L_s \times M_s$$

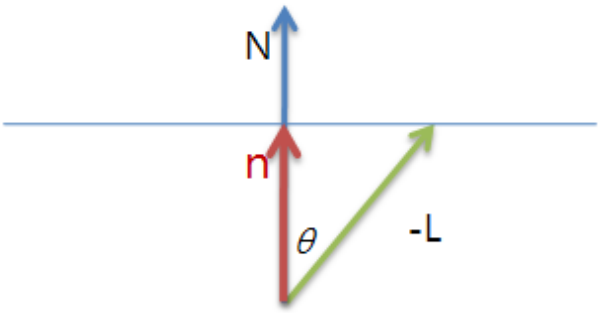


관측자의 시점(Eye), 조명(L)과 법선 벡터(N)에 대해서는 알고 있으며, 이를 기준으로 반사 벡터 R을 유도 할수 있습니다.

반사벡터 R을 바른생활님이 그려 놓으신 그림을 기준으로 해석하면 다음과 같습니다.



반산벡터 R을 구하는데 유의할 점은 "?" 부분 입니다. -N*Dot(L, N)이 유도 되기 위해서는
다음과 같은 정사영(투영 벡터)을 이해하고 있어야 합니다.



조명을 -L 벡터로 만든 후 법선벡터 N에 투영할 경우, 투영 벡터 n을 구할 수 있습니다.
(아래의 수식이 이해가 가지 않는 다면, "변환 으로 가는 길 #1 : 벡터와 행렬" 의 7 페이지를 다시 한번 살펴 보십시요 ^^;)

$$\vec{b} = |b| \times N = \left(A \cdot B / |B|\right) \times \left(B / |B|\right) = \left(A \cdot B / |B| \times |B|\right) \times B$$

$$\vec{b} = \left(A \cdot B / B \cdot B\right) \times B \qquad \boxed{|B||B| \times \cos 0 = B \cdot B}$$

위와 같은 정사영 식에서 A와 B 벡터를 -L과 N벡터로 교체해서 보면 -Dot(L, N)*N을 구할 수 있습니다.
(분모였던, Dot(N, N)의 경우 법선벡터의 크기는 1이기 때문에 1*1이 되어서 생략되게 됩니다.)

분모를 Dot(N, N)을 만들어 주어야 함으로 내적 부분의 순서는 조명 L이 앞에 있고 법선 N은 뒤에 있어야 합니다.

이제 반사 벡터 R을 유도하기 위해서 투영 벡터 n에 2를 곱하고 조명 L벡터와 합해주면 ^^;

$$R = -2N(L \cdot N) + L$$

이를 통해서 Specular 의 Phong 모델인 다음의 수식에 적용할 수 있습니다.

$$Spec = (R \cdot Eye)^{Shininess} \times L_s \times M_s$$

(경면 지수 Shininess 는 바른생활님 강좌 "GLSL 19 장 쉐이더로 구현하는 specular 조명" 을 참고하십시요 ^^;)

참고로 반사 벡터 R은 OpenGL ES 20의 shader 함수에서 지원을 하고 있으며
다음과 같이 "vec3 R = reflect(-L, N);" 함수를 이용해도 됩니다. ^^;

1.2 Blinn 모델

블린 모델은 퐁 모델에 비해서 수식이 복잡하지 않습니다. 그런데 효과는 퐁 모델 만큼 나기 때문에 거의 대부분 블린 모델을 이용

합니다.

사람에게 복잡한 수식인 만큼 컴퓨터에게도 똑같이 연산 프로세스를 높이기 때문에 성능을 하락 시키게 됩니다.

(iPHone 3D Programming, OpenGL ES 2.0 Programming Guide 책에서도 퐁 모델이 아닌

블린 모델로 Specular 를 설명 하고 있습니다.

〈OpenGL ES 2.0 Programming Guide 책에서의 블린 모델 Specular〉

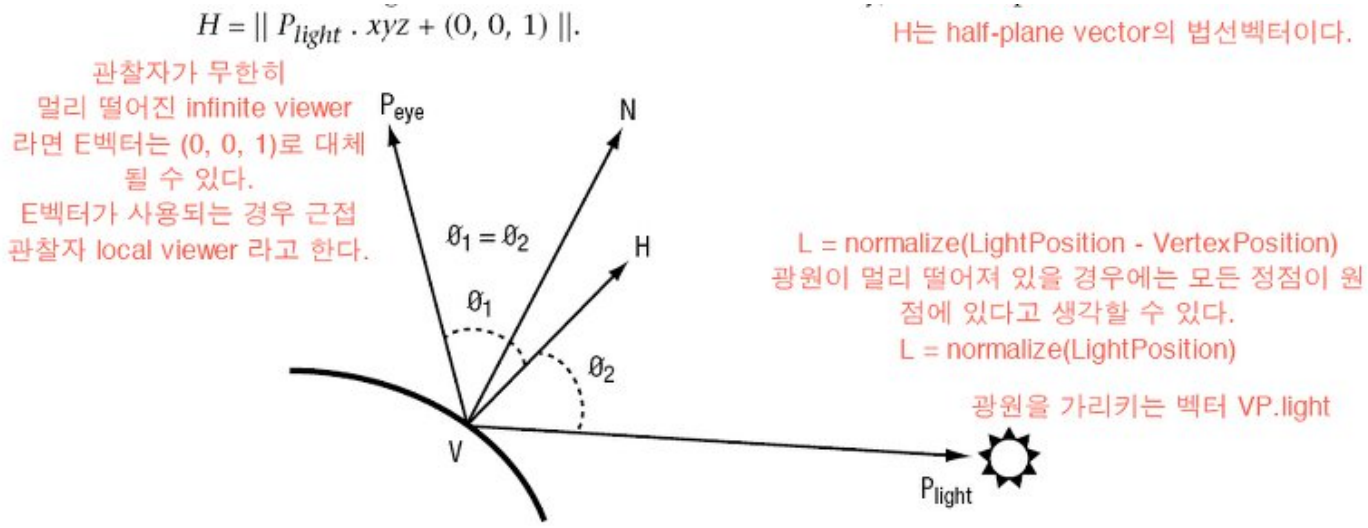
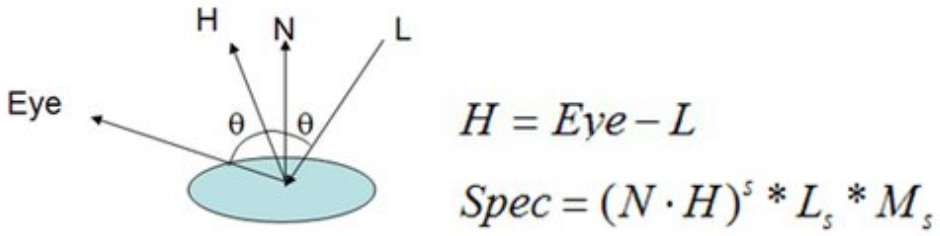


Figure 8-3 Geometric Factors in Computing Lighting Equation for a Directional Light

〈바른생활님의 그린 블린 모델 Specular〉



블린 모델이 입력 값중 조명과 관측자의 시점으로 풀이 합니다.

앞의 퐁 모델에서는 반사 벡터 R을 구하는 것이 핵심이라면, 블린 모델에서는 반각 벡터(Half Vector) 를 구하는 것이 핵심입니다.

반각 벡터는 다음과 같이 조명의 위치와 관측자의 시점의 벡터 합으로 구할 수 있습니다. 아주 간단하죠 ^^;

$$H = Eye + (-L)$$

$$Spec = (N \cdot H)^{Shininess} \times L_s \times M_s$$

제 예제는 iPhone 3D Programming 과 OpenGL ES 2.0 Programming Guide를 참고 하였음으로,

블린 모델로 구현 되어 있습니다.

Shaders/VertexLighting.vert

```
// ambient + diffuse + specular light
vec3 calcLightAmbDifSpec()
{
 vec3 N = normalize(eyespaceNormal);
 vec3 L = normalize(directionLight - vec3(positionLight));  // ---------- (1)
 //vec3 L = -normalize(vec3(positionLight) - directionLight);

 vec3 E = vec3(0, 0, 1);     // ------------------------------------ (2)
 vec3 H = normalize(L + E);  // ------------------------------------ (3)

 float df = max(0.0, dot(N, L));
 float sf = max(0.0, dot(N, H));    // ---------------------------------- (4)
 sf = pow(sf, u_shininess);    // ---------------------------------- (5)

 vec3 globalAmbient = u_ambientMaterial * vec3(0.05); // * gl_LightSource[0].ambient
 vec3 ambient = u_ambientMaterial * vec3(u_ambientLight);

// ------------------------------------------------------------------ (6)
 return (globalAmbient + ambient) + (INTENSITY * a_diffuseMaterial * df) + (u_specularMaterial * sf);
}
```

---

(1) : Positional Light 구하기

수식를 보면, normalize(directionLight - vec(positionLight)); 인데 directionLight는 광원의 위치이고, positionLight는 광원이 조사되는 정점 입니다.

이를 벡터로 보면, world space를 기준으로 벡터의 차로 본다면, (D - P) 라 하면, P --〉 D로 연결하는 벡터가 생성 됩니다.

즉 위의 그림에서 보면 L 벡터는 정점을 향해서 조사되는데 이와 반대 방향인 -L 벡터가 구해 집니다.

(2) : 관측자의 시점(시선) 또는 관찰자

여기서는 vec(0, 0, 1)로 했는데 무한히 멀리 떨어져 있는 관측자(infinite viewer) 입니다.

만약 E값을 사용할 경우에는 근접 관측자(local viewer)라 합니다.

왠지 앞장에서 Positional Light를 구하는 방법을 이용해서 하면 근접 관측자가 가능할 것 같은데,

근접 관측자에 대한 예제는 한번 구해봐야 할 것 같네요.

(3) : 반각 벡터 구하기

OpenGL ES 20에서는 OpenGL과 달리 반각 벡터 H를 직접 구해 줘야 합니다.

조명과 관찰자간의 벡터의 합을 통해서 쉽게 구할 수 있습니다. ^^;

(4) : 경면 율(Specular Factor) 구하기

확산율 df는 법선(N)과 조명(L)간의 사잇각을 이용해서 구하지만,

경면율 sf는 법선(N)과 반각 벡터(H)간의 사잇각을 이용해서 구합니다.

(5) : 경면 지수 구하기

경면 지수는 상수 값이며, 본App에서 uniform을 이용해서 저장 하였습니다.

경면 지수 Shininess 값이 크면 Specular 가 비춰지는 백색광 점의 반경은 작아지지만 색은 환하게 보입니다.

값이 작으면 백색광 점의 반경이 넓게 퍼집니다.

제 예제에서는 128을 사용 하였습니다.

(6) : 주변광, 확산광, 경면광 반영

2. 본App 에서 Specular Light 실행 하기 (Vertex Lighting)

Classes/RenderingEngine.CH02.ES2.cpp

```
void RenderingEngine::Initialize(int width, int height)
{
......
// Set up some default material parameters.
vec4 diffuseLight = vec4(1, 1, 1, 1); // 백색광
vec4 ambientLight = vec4(0.05f, 0.05f, 0.05f, 1.0f); // 흑색광
vec3 diffuseMaterial = vec3(1, 1, 1); // 백색 재질
vec3 ambientMaterial = vec3(1.0f, 0.f, 0.f); // 적색 재질
// vec3 ambientMaterial = vec3(0.0f, 0.f, 0.f); // 주변광 끄기
vec3 specularMaterial = vec3(1, 1, 1); // 백색 재질
```

```
glUniform4fv(m_vertexLight.Uniforms.DiffuseLight, 1, diffuseLight.Pointer());
glUniform4fv(m_vertexLight.Uniforms.AmbientLight, 1, ambientLight.Pointer());
glUniform3fv(m_vertexLight.Uniforms.AmbientMaterial, 1, ambientMaterial.Pointer());
glUniform3fv(m_vertexLight.Uniforms.SpecularMaterial, 1, specularMaterial.Pointer());
glUniform1f(m_vertexLight.Uniforms.Shininess, 128);
.....
}


void RenderingEngine::Render(const vector<Visual>& visuals)
{
.....

ProgramHandles handler = m_pixelLight;
handler = m_vertexLight; // ------------------------------------- (1)

GLfloat weight = cos(interpolation_z) * 1.5f; // -------------------------- (2)
glUniform1f(handler.Uniforms.Interpolation_z, (GLfloat) weight);
LOG_PRINT("interpolation_z:%f, cos(%f)", interpolation_z, weight);
interpolation_z += (2*Pi) * 0.0008f;


// Set the light Mode : 0(Diffuse Mat Color), 1(DIFFUSE), 2,(AMBIENT_DIFFUSE), 3(AMBIENT_DIFFUSE_SPECULAR)
glUniform1i(handler.Uniforms.LightMode, 3); // ------------------------- (3)

.....
}
```

---

(1) : VertexLighting 으로 실행 합니다.
pixelLight 보다 성능이 떨어져서 나옵니다. ^^;
나중에 Pixel Light(Fragment Light)를 다룰 때 비교를 해보겠습니다.

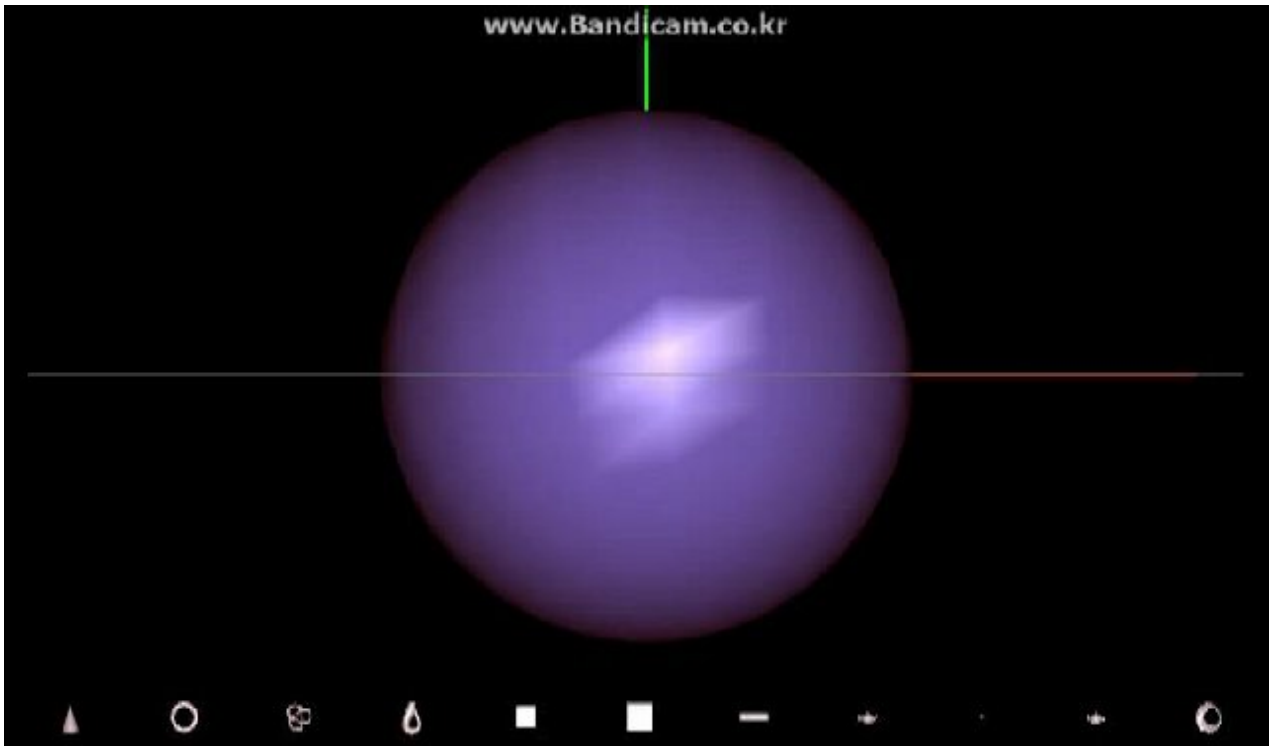
(2) : Postional Light 시에 광원의 위치를 이동 시키기 위해서 사용되는 가중치 값입니다.

(3) : 주변광, 확산광, 경면광 결합으로 실행 합니다.
원래 주변광은 검정색이지만, 물체와 바탕화면을 구분하기 위해서 적색 재질을 사용하였습니다.
경면광은 백색으로 처리 되어 있습니다.

```
vec3 ambientMaterial = vec3(1.0f, 0.f, 0.f);
vec3 specularMaterial = vec3(1, 1, 1); // 백색 재질
```

만약 주변광을 끄고 싶다면, AmbientMaterial 재질 값을 (0, 0, 0)으로 설정하면 됩니다.

〈Specular Light〉

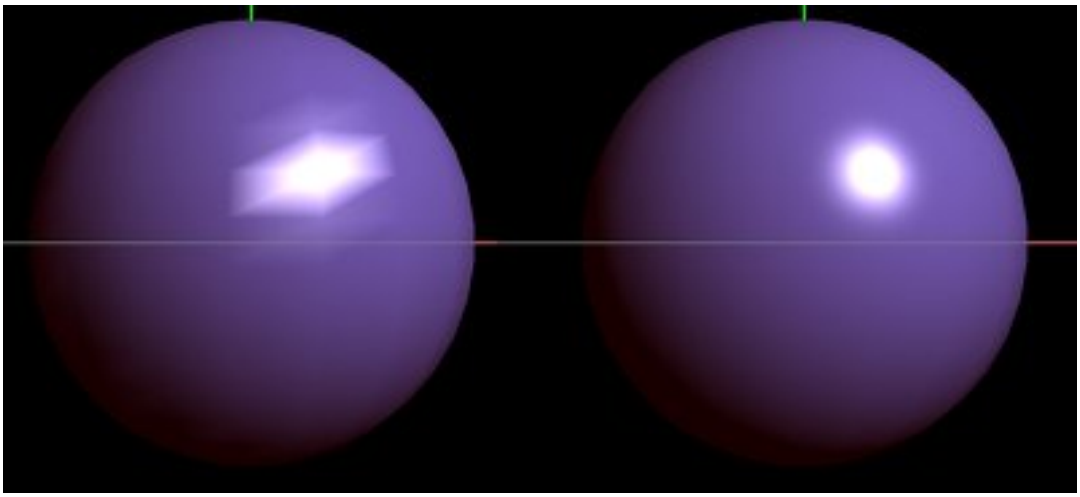


3. 결론

mfc와 android app으로 Vertex Light 주변광 확산광 경면광 합성을 적용해 보았습니다.

Vertex Light 로 경면광을 표현할 경우, 바른 생활님 예제인 "GLSL 19 장 쉐이더로 구현하는 specular 조명"과 마찬가지로 표현이 뭔가 이상하게 보입니다.

Pixel Light로 표현할 경우 다음과 같이 정상적이 화면이 나옵니다.



다음장에는 Pixel Light로 표현 해서 살펴 보도록 하겠습니다.

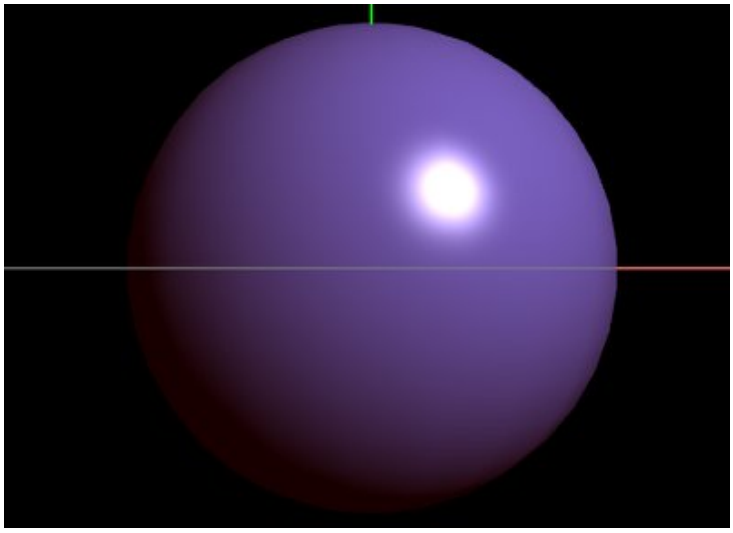
OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list

http://code.google.com/p/myastro/downloads/detail?name=02_ParametricSurface_v.1.2.0.zip&can=2&q=#makechanges

[GLES20] 11. 조명 적용하기 - Pixel Light

2011.11.26 13:23



조명 적용하기 - Pixel Light

1. Pixel Light : 주변광, 확산광, 경면광 적용

Pixel Light는 픽셀 단위 조명(Per Pixel Light)을 의미 합니다. 또한 Fragment Shader에서 처리 됨으로 Fragment Light 라고도 불립니다. 일반적으로 Fragment Light 보다는 Pixel Light라고 더 많이 불리는것 같네요.
(Direct X의 경우, Fragment Shader를 Pixel Shader라고 부릅니다.)

픽셀 단위 조명은 앞장에서 배웠던 정점 단위 조명(Vertex Shader에서 처리하는 Vertex Light)을 그대로 Pixel Shader에서 구현하면 됩니다.
이때 Vertex(정점)처리와 관련된 부분만 Vertex Shader에서 varying 형 변수로 선언해서 Fragment Shader로 넘겨주면 됩니다.

Specular Light 를 기준으로 보면 다음과 같습니다.

Shaders/PixelLighting.vert

```
// Varying      // ------------------------------------ (1)
varying vec3 v_eyespaceNormal;//EyespaceNormal
varying vec3 v_lightPosition;//EyespaceNormal
varying vec3 v_diffuse;//Diffuse;


void main(void)
{
// ------------------------------------------------- (2)
 v_eyespaceNormal = u_normalMatrix * a_normal;
 v_diffuse = a_diffuseMaterial;
 v_textureCoordOut = a_textureCoordIn;

 // ------------------------------------ (3)
 // Directional Light
 vec4 positionLight = vec4(0, 0, 0, 0);
 vec3 directionLight = vec3(u_lightPosition);
 // Positional Light
 positionLight = u_modelViewMatrix * a_position;
 directionLight = vec3(u_lightPosition);
 directionLight *= u_interpolation_z;
 // Varying
 v_lightPosition = directionLight - vec3(positionLight);

 // Vertex Position
 gl_Position = u_projectionMatrix * u_modelViewMatrix * a_position;
}
```

---

(1) : varying 형 변수 선언

Vertex Shader에서 정점 관련된 정보를 이용해서 계산 한후, Fragment Shader로 전달할 예정인 변수 입니다.

v_eyespaceNormal, v_lightPosition, v_diffuse 등이 있습니다.

색상 정보중 diffuse color를 빼고, ambient, diffuse, specular material 과 color는

Fragment Shader에서 uiform 형으로 본App으로 부터 전달 받았습니다.

(2) : 법선 벡터, Vertex Color

정점 정보와 관련되어서 계산 되는 부분 입니다.

(3) : 방향성 또는 위치성 조명 벡터

Light Vector를 계산합니다.

이 정보를 이용해서 Vertex Shader에서 Half Vector를 생성한후 varying 형으로 Fragment Shader로 전달해도 됩니다.

제 예제에서는 Half Vector등은 Fragment Shader에서 처리 하였습니다.

Shaders/PixelLighting.frag

```
// ambient + diffuse + specular light
vec3 calcLightAmbDifSpec()
{
 vec3 N = normalize(v_eyespaceNormal);    // ---------- (1)
 vec3 L = normalize(v_lightPosition);    // ---------- (2)
 vec3 E = vec3(0, 0, 1);    // ---------- (3)
 vec3 H = normalize(L + E);    // ---------- (4)

 float df = max(0.0, dot(N, L));
 float sf = max(0.0, dot(N, H));    // ---------- (5)
 sf = pow(sf, u_shininess);       // ---------- (6)

// ---------- (7)
 vec3 globalAmbient = u_ambientMaterial * vec3(0.05); // * gl_LightSource[0].ambient
 vec3 ambient = u_ambientMaterial * vec3(u_ambientLight);

 return (globalAmbient + ambient) + (INTENSITY * v_diffuse * df) + (u_specularMaterial * sf);
}
```

FragmentShader의 Pixel Light 계산은 앞장에서 처리했던 Vertex Light 부분을 그대로 옮겨 오기만 하면 됩니다.

(1) : 법선 벡터 구하기

(2) : Positional Light 구하기

수식를 보면, normalize(directionLight - vec(positionLight)); 인데 directionLight는 광원의 위치이고, positionLight는 광원이 조사되는 정점 입니다.

이를 벡터로 보면, world space를 기준으로 벡터의 차로 본다면, (D - P) 라 하면, P --〉 D로 연결하는 벡터가 생성 됩니다.
즉 위의 그림에서 보면 L 벡터는 정점을 향해서 조사되는데 이와 반대 방향인 -L 벡터가 구해 집니다.

(3) : 관측자의 시점(시선) 또는 관찰자

여기서는 vec(0, 0, 1)로 했는데 무한히 멀리 떨어져 있는 관측자(infinite viewer) 입니다.
만약 E값을 사용할 경우에는 근접 관측자(local viewer)라 합니다.
왠지 앞장에서 Positional Light를 구하는 방법을 이용해서 하면 근접 관측자가 가능할 것 같은데,
근접 관측자에 대한 예제는 한번 구해봐야 할 것 같네요.

(4) : 반각 벡터 구하기

OpenGL ES 20에서는 OpenGL과 달리 반각 벡터 H를 직접 구해 줘야 합니다.
조명과 관찰자간의 벡터의 합을 통해서 쉽게 구할 수 있습니다. ^^;

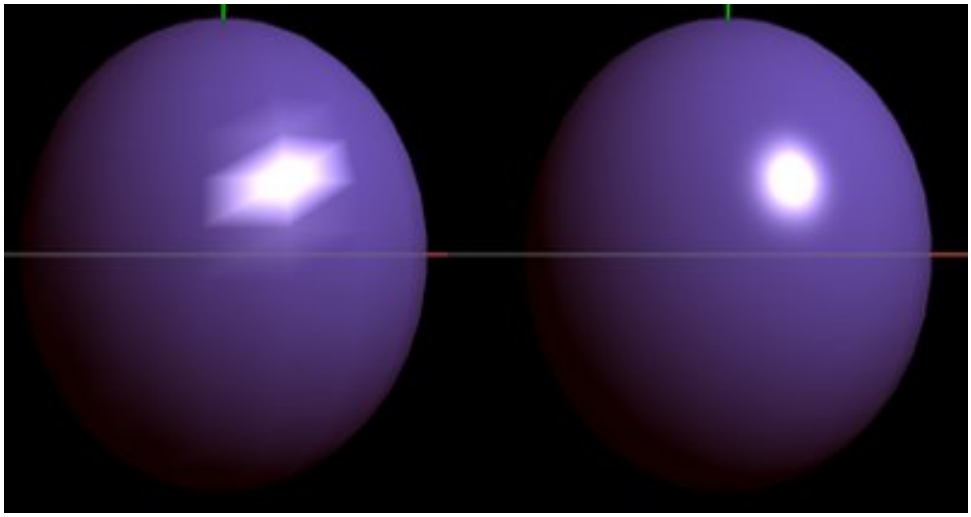
(5) : 경면 율(Specular Factor) 구하기

확산율 df는 법선(N)과 조명(L)간의 사잇각을 이용해서 구하지만,
경면율 sf는 법선(N)과 반각 벡터(H)간의 사잇각을 이용해서 구합니다.

(6) : 경면 지수 구하기

경면 지수는 상수 값이며, 본App에서 uniform을 이용해서 저장 하였습니다.
경면 지수 Shininess 값이 크면 Specular 가 비춰지는 백색광 점의 반경은 작아지지만 색은 환하게 보입니다.
값이 작으면 백색광 점의 반경이 넓게 퍼집니다.
제 예제에서는 128을 사용 하였습니다.

(7) : 주변광, 확산광, 경면광 반영



2. 본App 에서 Pixel Light 실행 하기

Classes/RenderingEngine.CH02.ES2.cpp

```
void RenderingEngine::Initialize(int width, int height)
{
......
// Set up some default material parameters.
vec4 diffuseLight = vec4(1, 1, 1, 1); // 백색광
vec4 ambientLight = vec4(0.05f, 0.05f, 0.05f, 1.0f); // 흑색광
vec3 diffuseMaterial = vec3(1, 1, 1); // 백색 재질
vec3 ambientMaterial = vec3(1.0f, 0.f, 0.f); // 적색 재질
// vec3 ambientMaterial = vec3(0.0f, 0.f, 0.f); // 주변광 끄기
vec3 specularMaterial = vec3(1, 1, 1); // 백색 재질

glUniform4fv(m_vertexLight.Uniforms.DiffuseLight, 1, diffuseLight.Pointer());
glUniform4fv(m_vertexLight.Uniforms.AmbientLight, 1, ambientLight.Pointer());
glUniform3fv(m_vertexLight.Uniforms.AmbientMaterial, 1, ambientMaterial.Pointer());
glUniform3fv(m_vertexLight.Uniforms.SpecularMaterial, 1, specularMaterial.Pointer());
glUniform1f(m_vertexLight.Uniforms.Shininess, 128);
......
}
```

```
void RenderingEngine::Render(const vector<Visual>& visuals)
{
......

ProgramHandles handler = m_pixelLight;
//handler = m_vertexLight; // ------------------------------------ (1)

GLfloat weight = cos(interpolation_z) * 1.5f; // ------------------------- (2)
glUniform1f(handler.Uniforms.Interpolation_z, (GLfloat) weight);
LOG_PRINT("interpolation_z:%f, cos(%f)", interpolation_z, weight);
interpolation_z += (2*Pi) * 0.0008f;


// Set the light Mode : 0(Diffuse Mat Color), 1(DIFFUSE), 2,(AMBIENT_DIFFUSE), 3(AMBIENT_DIFFUSE_SPECULAR)
glUniform1i(handler.Uniforms.LightMode, 3); // ------------------------- (3)

......
}
```

---

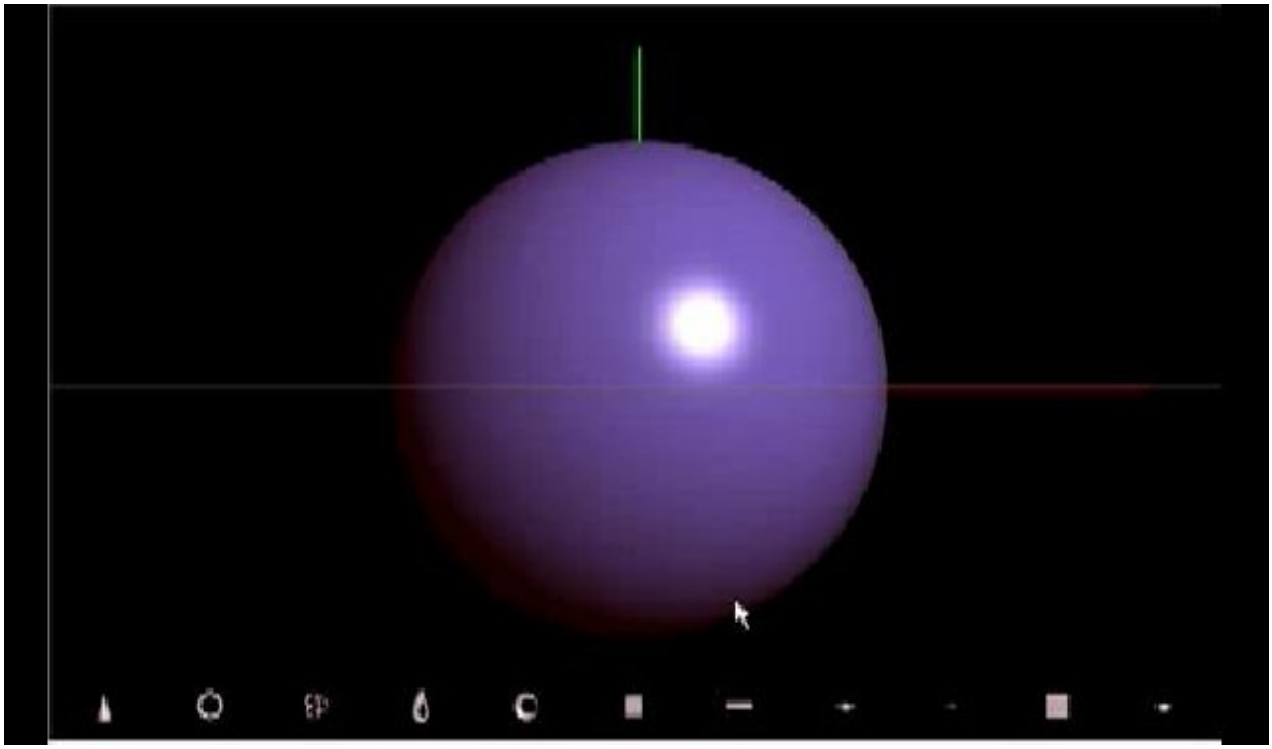
(1) : PixelLighting 으로 실행 합니다.
Pixel Light로 동작하도록 vertexLight는 주석으로 막아 줍니다.

(2) : Postional Light 시에 광원의 위치를 이동 시키기 위해서 사용되는 가중치 값입니다.

(3) : 주변광, 확산광, 경면광 결합으로 실행 합니다.
원래 주변광은 검정색이지만, 물체와 바탕화면을 구분하기 위해서 적색 재질을 사용하였습니다.
경면광은 백색으로 처리 되어 있습니다.

```
vec3 ambientMaterial = vec3(1.0f, 0.f, 0.f);
vec3 specularMaterial = vec3(1, 1, 1); // 백색 재질
```

만약 주변광을 끄고 싶다면, AmbientMaterial 재질 값을 (0, 0, 0)으로 설정하면 됩니다.



3. 결론

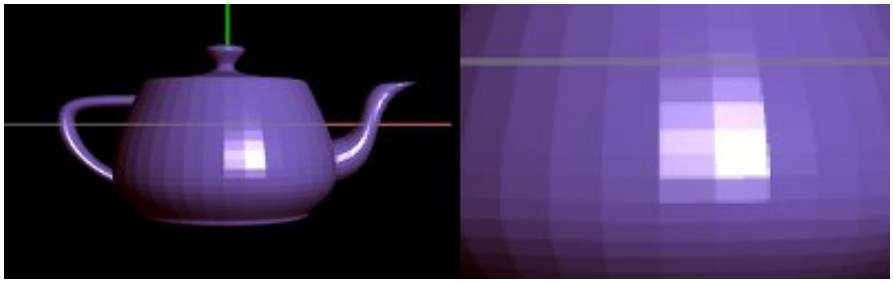
mfc와 android app으로 Pixel Light 주변광 확산광 경면광 합성을 적용해 보았습니다.

Pixel Light 로 경면광을 표현하는 것이 Vertex Light 보다 훨씬 예쁘게 나오는 것을 확인 할 수 있었습니다.
결국 확산광, 주변광을 처리할 때는 Vertex Light 로 처리 해도 좋지만, Specular Light 경면광을 처리할 때는 Pixel Light로 처리 해야 훨씬 예쁘게 나오는 것을 볼수 있었습니다.

다음에 다룰 Spot Light나 Toon Light도 Vertex Light가 아닌 Pixel Light로 처리 해야 훨씬 예쁘게 나옵니다.

그런데 Specular Light로 돌려 보시면 알겠지만, 두가지 이상한 점이 발견됩니다.

첫 번째로 다른 물체들은 다 예쁘게 나오는데 유독 주전자 Teapot 만 이상하게도
평평한 Flat Model 로만 나오는 것을 볼수 있을 겁니다.

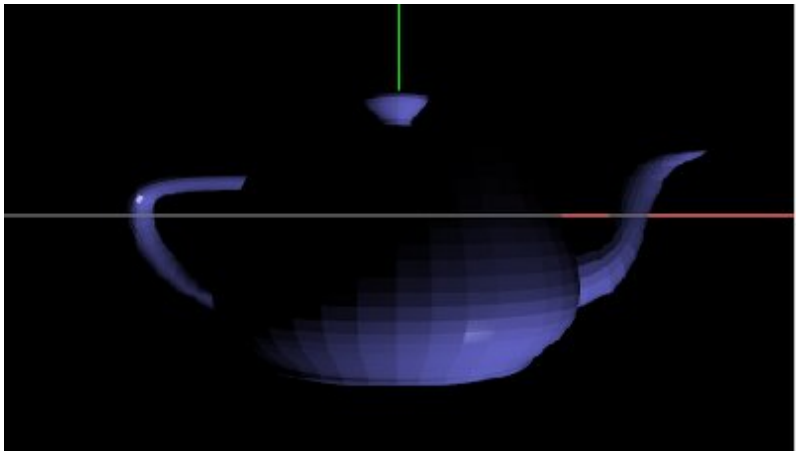


두 번째로는 현재 적용한 조명 방식이 Directional Light 가 아닌 Positional Light 임에도 불구하고, Light의 위치를 가까이해도, 전체적으로 뿌려지는 것 처럼 보입니다.
이는 감쇠 수식(attenuation)를 반영하지 않고 있기 때문에 발생한 현상 입니다.

다음 장에서는 이 부분들에 대해서 해결하는 방안에 대해서 알아보도록 하겠습니다.

OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list
http://code.google.com/p/myastro/downloads/detail?name=02_ParametricSurface_v.1.2.0.zip&can=2&q=#makechanges



조명 적용하기 - Flat Shading과 Smooth Shading

1. Flat Shading

OpenGL ES 2.0에서는 glShadeModel() API를 지원하지 않습니다.

이때문에 현재 제가 사용하고 있는 주전자 teapot2.bj 파일을 이용해서 처리할 경우에는 원본 자체가 Flat 형으로 평평하기 때문에, Phong을 적용하든 Blinn을 적용하든 Smooth 효과가 먹지 않습니다.

기본적으로 퐁과 블린은 Normal 값인 법선 벡터, Light 벡터, 관측자 시선 벡터 등이 주요 인자 입니다.
그런데, Flat Shade Model의 경우에는 한 면을 이루는 세 정점의 Normal 값이 같은 방향을 바라 보고 있습니다.
이때문에 법선에 의한 조명이 먹긴 하지만 평평한 모습이 보이는 겁니다.

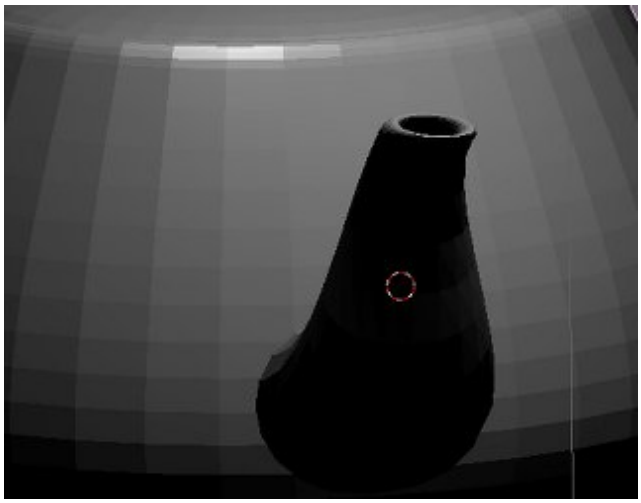
Models/teapot2.h

```
double teapot2Normals [] = {
  // f 2909//1 2921//1 2939//1
  -0.9269106816233833f, -0.3681608735435733f, 0.07276097500795555f,
```

```
-0.926910681623383f, -0.368160873543573f, 0.0727609750079555f,
-0.926910681623383f, -0.368160873543573f, 0.0727609750079555f,
// f 2939//2 2931//2 2909//2
-0.926913284595327f, -0.368154113036399f, 0.0727620223405273f,
-0.926913284595327f, -0.368154113036399f, 0.0727620223405273f,
-0.926913284595327f, -0.368154113036399f, 0.0727620223405273f,
// f 2869//3 2877//3 2921//3
-0.903724329229544f, -0.368523134254108f, 0.217883079375474f,
-0.903724329229544f, -0.368523134254108f, 0.217883079375474f,
-0.903724329229544f, -0.368523134254108f, 0.217883079375474f,
// f 2921//4 2909//4 2869//4
-0.903723860047075f, -0.36852494292931f, 0.217881966258257f,
-0.903723860047075f, -0.36852494292931f, 0.217881966258257f,
-0.903723860047075f, -0.36852494292931f, 0.217881966258257f,
```

법선 벡터의 데이타를 보면 한 면을 이루는 Set가 동일한 방향을 바라보고 있음을 알수 있습니다.
이와 같은 플랫 쉐이딩은 표면이 굴곡이 보이지 않습니다.

대신 플랫 쉐이딩을 이용할 경우 비현실 적인 화면처럼 보이지만, 연산 처리가 Smooth에 비해서 많이 들지 않기 때문에 많이 사용된다고 합니다.
또한 다각형 간에 윤곽선이 보이는 것처럼 보이기도 합니다. ^ㅁ^;



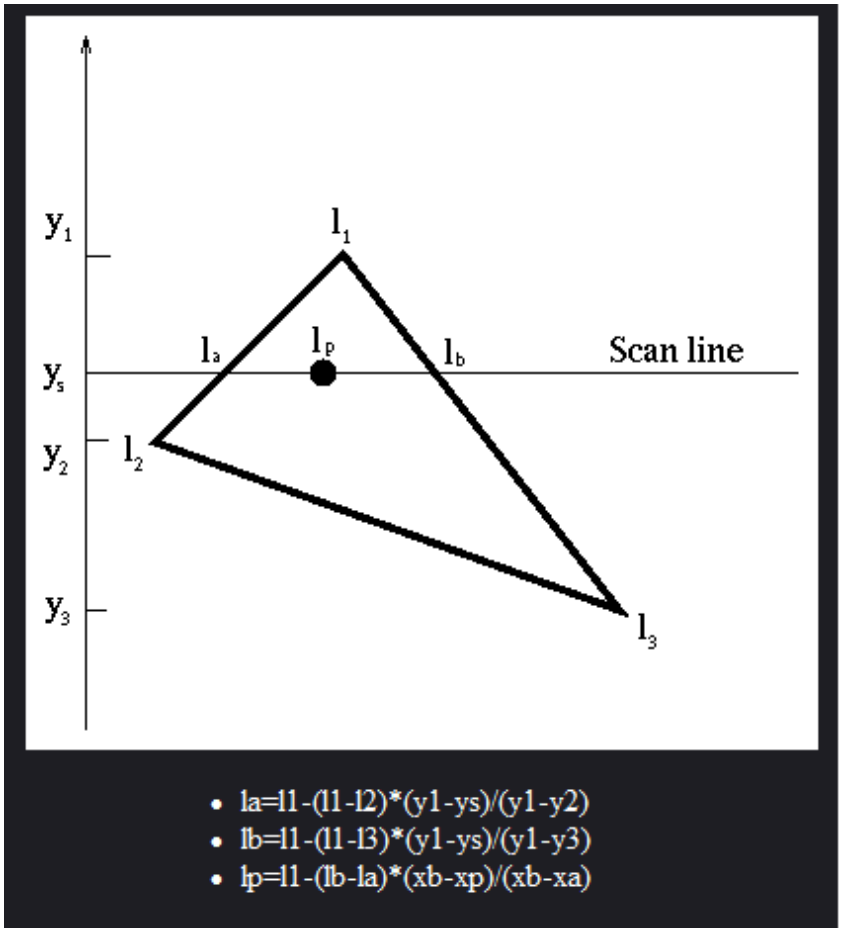
OpenGL ES 1.0의 경우에는 glShadeModel(GL_SMOOTH) 를 이용해서 표면을 부드럽게 표현할 수 있습니다.
이는 고정 파이프 라인 기능으로 CPU 연산에 부담을 많이 준다고 합니다.

이에 OpenGL ES 2.0에서는 빠졌습니다. -ㅁ-;
다시 말하면 예제의 teapot2.obj를 그대로 이용하기 위해서는 GL_SMOOTH 을 직접 구현해 줘야 합니다.

2. Smooth Shading - Gouraud Shading

OpenGL ES 2.0에서 Smooth Shading을 구현 하기 위해서는 Gouraud Shading 기법(고라우드? 고로? 구로? 등의 발음을 함)을 이용합니다.

다각형 내부를 서로 다른 색으로 채우는 방법으로 표면위에 한 정점을 잡아서 이를 둘러 싸고 있는 면의 법선 벡터를 계산한후 평균값을 구한다고 합니다. -ㅁ-;
표면위에 있는 임의의 한 정점 Ip를 구하는 방법 입니다.

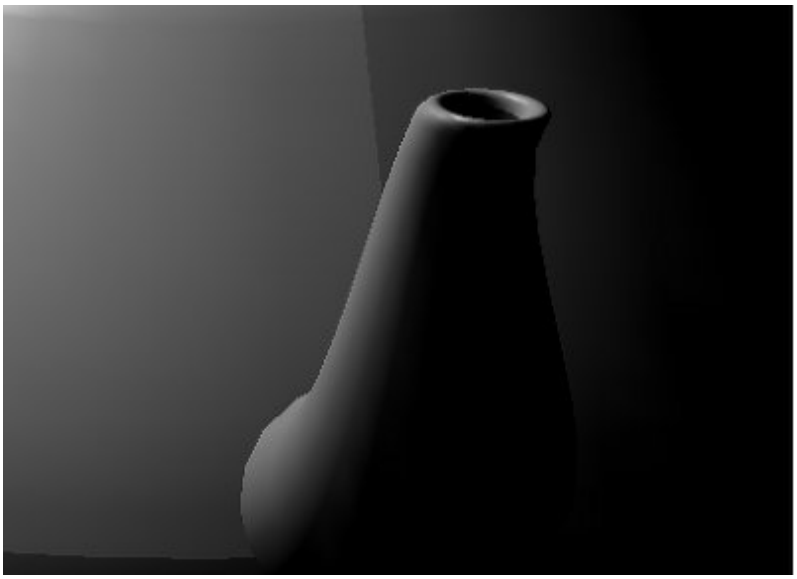


의외로 이론은 많지만, OpenGL ES 2.0에서는 구현 코드가 잘 보이지 않습니다.
아마도 정점이 아주 많을 경우에는 이와 같은 정점에 대한 보간 처리가 CPU에 부담을 많이 주기 때문이 아닌가 생각 되네요.
(공부 차원에서 구현을 한번 해 보는 것도 좋을 것 같네요. ^^;)

참고 페이지

http://m.blog.naver.com/PostView.nhn?blogId=darkedge81&logNo=120040318294
http://www.stack.nl/~dimitri/3dsview/gouraud.html
http://omega.di.unipi.it/web/IUM/Waterloo/node84.html#SECTION00152000000000000000

예제의 teapot2.obj 파일을 블렌더를 이용해서 Smooth를 걸어서 변경해 보겠습니다.

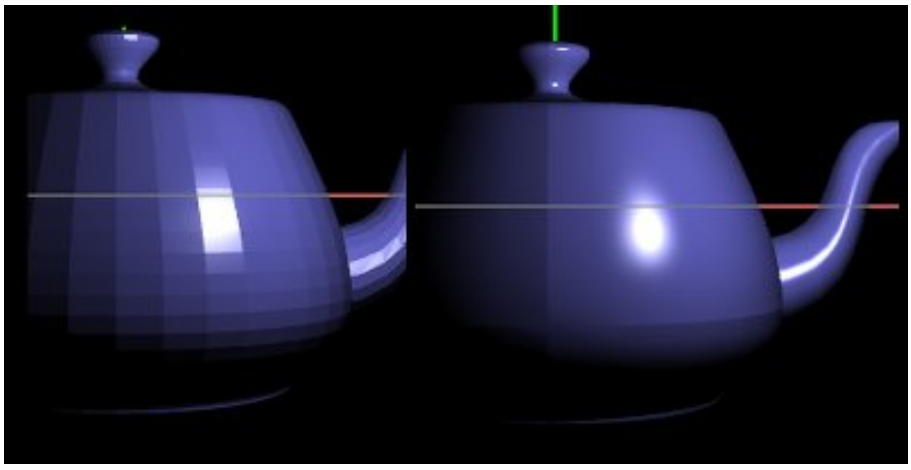


Models/teapot2_smooth.h

```
double teapot2_smoothNormals [] = {
  // f 2909//1 2921//2 2939//3
  -0.94681898769934, -0.284130897478523, 0.151008071406948,
  -0.91365422609069, -0.369282579203481, 0.169900947152883,
  -0.926916056314328, -0.368148199769382, 0.0727566323661089,
  // f 2939//3 2931//4 2909//1
  -0.926916056314328, -0.368148199769382, 0.0727566323661089,
  -0.96417026254053, -0.254257716714598, 0.0756882971377034,
  -0.94681898769934, -0.284130897478523, 0.151008071406948,
  // f 2869//5 2877//6 2921//2
  -0.911147515387562, -0.284468468461901, 0.298140731289795,
  -0.875375967008971, -0.369676898357678, 0.311537970725489,
  -0.91365422609069, -0.369282579203481, 0.169900947152883,
  // f 2921//2 2909//1 2869//5
```

```
-0.91365422609069, -0.369282579203481, 0.169900947152883,
-0.94681898769934, -0.284130897478523, 0.151008071406948,
-0.911147515387562, -0.284468468461901, 0.298140731289795,
```

법선 벡터의 데이타를 보면 한 면을 이루는 Set내의 법선의 방향이 다른것을 볼수 있습니다.



왼쪽은 원본이며, 오른쪽은 블렌더를 이용해서 Smooth 를 적용한 화면입니다.

3. 결론

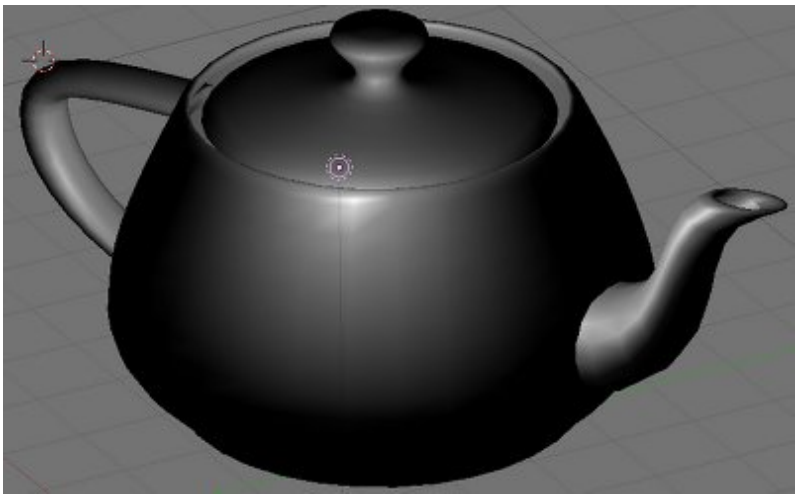
이번 장에서는 Flat Shading과 Smooth Shading 처리에 사용되는 glShadeModel() API가 OpenGL ES 2.0에서는 지원하지 않는다는 것을 확인하였습니다.

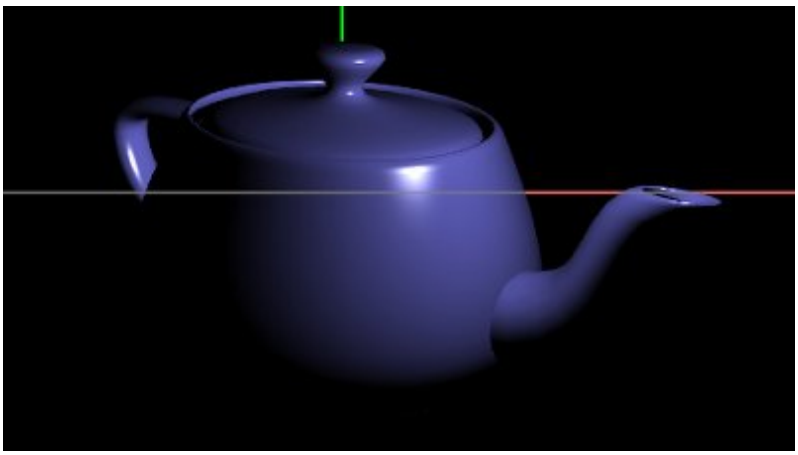
또한 인터넷 또는 디자이너로 부터 얻은 이미지 데이타가 Flat model일 경우 이를 수정 하는 방법에 대해서 간단하게 알아 보았습니다.

그런데, 이 예제의 teapot2.obj로는 smooth로 처리해도 위-경도 사이에 경계선이 그려져 있는 문제점과 Texture 좌표가 Obj 내에 없다는 문제가 있습니다.

가장 좋은 방법은 인터넷에서 더 좋은 Source를 구하거나 디자이너로 부터 예쁜 주전자 이미지를 받는 것이 가장 좋을것 같습니다.

참고로 3ds Max에는 기본으로 Teapot이 있으며 이를 이용해서 Smooth를 정해주고 단일 Group으로 선택해서 그리면 다음과 같은 더욱 괜찮은 주전자가 탄생합니다.(회사 디자이너가 그려줌 ^^;)





Gouraud Shading에 대해서는 시간이 나면 구현을 한번 해봐야할 것 같네요. ^^;
우선은 조명과 텍스쳐의 기초 과정을 다르고 있으니 이 진행이 끝나고 나서 시작해 봐야할 것 같네요. ㅎㅎ

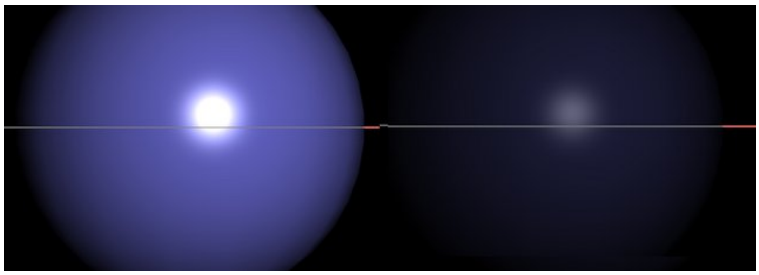
OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list

http://code.google.com/p/myastro/downloads/detail?name=02_ParametricSurface_v.1.2.1.zip&can=2&q=#makechanges

[GLES20] 13. 조명 적용하기 – Point Light와 감쇠 방정식

2011.11.28 01:09



조명 적용하기 - Positional Light 와 감쇠 효과

앞에 장에서는 방향성 조명을 Directional Light라고 하고 위치성 조명은 Positional Light라고 하였습니다.
그런데, 3D 소스나 책등을 보면 Directional Light는 Direction Light라고 하고, Positional Light는 Point Light라고도 부릅니다.
같은 의미 이기 때문에 혼동이 없으시길 바랍니다 .^^;

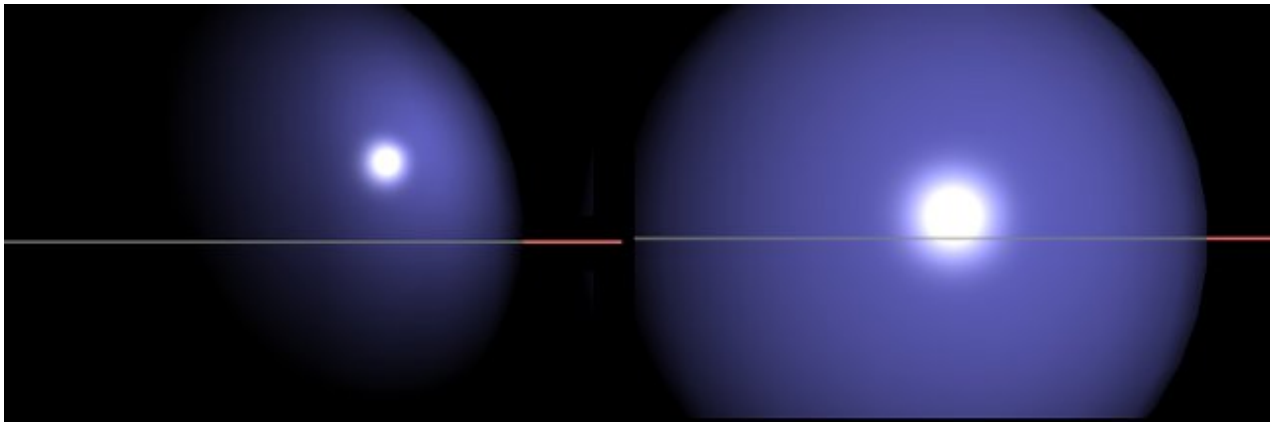
또한 Positional Light에 대한 선행 지식을 쌓고 싶으시다면,
바른 생활님의 설명하신 "[GLSL] 21. Positional Light에 대한 이해" 장을 한번 읽어 보시면 도움이 될겁니다.

1. Positional Light

"[GLES20] 08. 조명 적용하기 - 확산광, Diffuse Vertex Light" 장에서 광원의 위치에 따라서 확산과 경면광의 조사 되는 위치가 변경되는 것을 보았습니다.

그런데, 이상하게도 광원이 모델로 부터 멀어지면 빛이 모델에 닿는 거리가 멀어지기 때문에 모델이 어두어 져야 하는데 앞의 예제에서는 그렇지가 않습니다.

다음 화면을 보면, 기존의 Specular를 Pixel Light 구현했을때의 화면입니다.



왼쪽의 광원 위치는 (2, 0, -7)인 상태로 물체에 가까이 다가 갔을 때 화면 입니다.
(카메라의 eye vector를 (0. 0. 10)으로 설정 한 상태임으로 Z가 -7이면 물체에 가까이 다가간 것입니다.)

반대로 오른쪽의 광원 위치는 (2, 0, 0)인 상태로 물체에 멀어졌을 때 화면 입니다.
그런데, 오른쪽 구의 광원을 받은 정도를 보면 더 많은 광량을 보여주고 있습니다.
물론 광원이 모델로 부터 떨어졌기 때문에 방사되는 영역이 넓어져서 그렇다고도 볼수 있지만,
Z값을 0~100으로 올려도 점점 더 밝아 지게 됩니다.

이는 마치 Directional Light을 (1, 1, 1, 0)으로 놓고 Positional Light를 적용하지 않음으로서, 무한히 멀리 떨어진 태양 처럼 광원에 의해서 모델 전체에 비춰지는 현상과 유사 합니다.

아무튼 광원이 계속 뒤로 떨어져 가는데도 물체가 더 밝아지는 것은 이상한 상태 입니다. -ㅁ-;

이는 08~10 장에서 다룬 Diffuse, Ambient, Specular Light에 광원의 위치 이동에 따른 방향 정보(Directional Light)는 정상적으로 적용이 되었지만, 광원과 모델간의 **거리**에 따른 정보(Positional Light)는 정상적으로 적용 되지 않았기 때문에 나타나는 현상입니다.

이와 같이 **거리 정보**에 따른 Positional Light 적용을 위해서는 감쇠(attenuation) 방정식을 적용 해야 합니다.

2. Attenuation - 감쇠 방정식

$$attenuation\ factor = \frac{1}{K_c + K_l d + K_q d^2}$$

감쇠율 방정식

Kc = GL_CONSTANT_ATTENUATION, 감쇠 상수

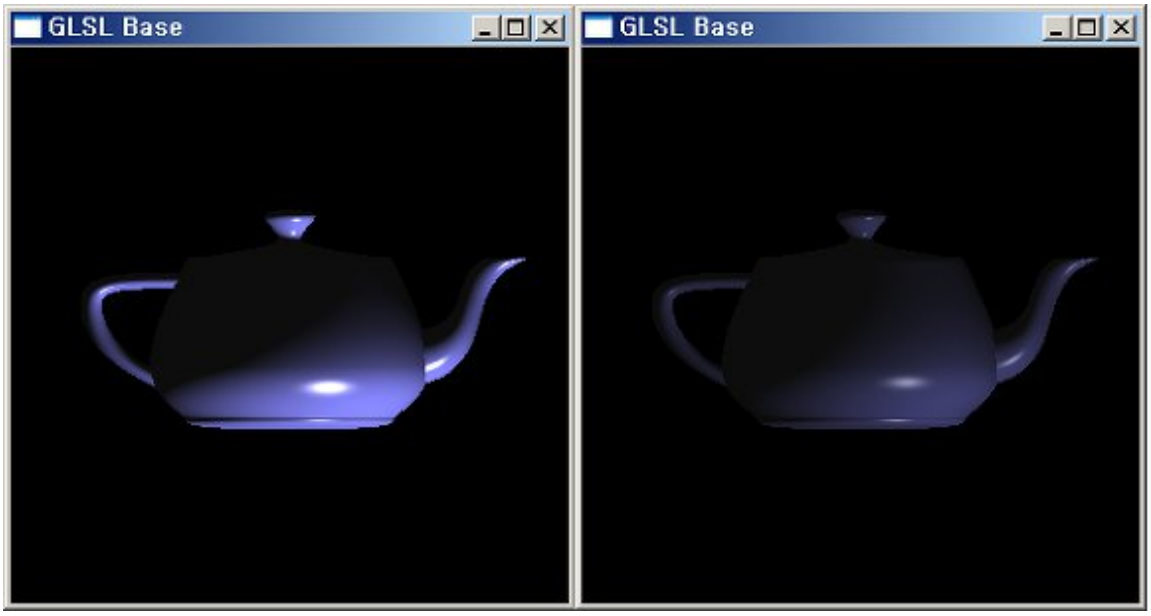
Kl = GL_LINEAR_ATTENUATION, 감쇠 일차 방정식

Kq = GL_QUDRATID_ATTENUATION, 강쇠 이차 방정식

d = 광원과 표면의 정점 사이 거리

OpenGL ES 2.0에서는 OpenGL 에서 지원하는 다음과 같은 함수을 지원하지 않습니다.

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.5);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.2);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```



왼쪽은 감쇠율은 (1.0, 0.0, 0.0) 이며, 오른쪽은 (1.5. 0.5, 0.2) 인 상태 입니다.

기본 Default 감쇠율은 Kc만 1.0이고 Kl과 Kq는 0.0 이기 때문에 attenuation factor는 1이 되며 감쇠 효과를 사용하지 않게 됩니다.

하지만, 오른쪽 Kc(1.5), Kl(0.5), Kq(0.2) 의 경우에는, 거리 값 d가 1을 기준으로 할 경우, 0.4545 가 되며, 전체적인 Light 값에서 반 정도 빠지게 됩니다.

물론 광원과 정점 사이의 거리인 d가 0 ~ 1 사이라면 더 밝아질 것이고, 1 ~ ? 사이 라면 점점 어두어 지게 됩니다.

결국 Kc, Kl, Kq 에 의해서 계산된 결과 값인 attenuation factor 는 그 값이 1.0에 가까운 값이라면, 감쇠를 적용하지 않는 것이며, 0.0에 가깝게 되면 감쇠율에 영향을 크게 받게 되며 Light가 어두워 지게 됩니다.

결국 다음의 방정식은

$$attenuation\ factor = \frac{1}{K_c + K_l d + K_q d^2}$$

분모가 1보다 큰 값이면, 0.0에 가까워 지며, 분모가 1보다 작은 값이면 오히려 감쇠 효과는 사라지고 더 밝아 지게 됩니다.

Kc, Kl, Kq는 상수 이기 때문에 광원과 표면의 정점 사이의 거리인 d가 작아서 서로 가까이 있다면, att 값은 0.0 보다 커지게 되며, 거리가 멀어져서 d값이 커지면, att 값은 0.0에 가까워 집니다.

위와 같은 감쇠율을 계산 하기 위한 인자들은 uniform 형으로 본App으로 부터 전달 받아서 사용해야 합니다.

또는 다음의 Post와 같이 수식으로 정리할 수도 있습니다.

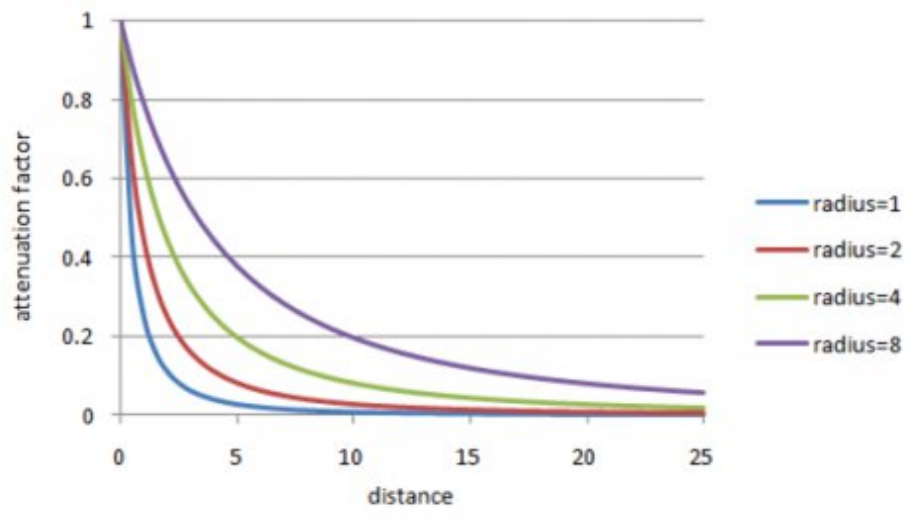
Light attenuation

http://imdoingitwrong.wordpress.com/2011/01/31/light-attenuation/

이 분이 정리한 방식은 광원의 반지름을 임의로 설정할 경우, 조사 되는 광량을 조절하고 있습니다. 대단 ^^;

$$attenuation\ factor = \frac{1}{(\frac{d}{r} + 1)^2}$$

위와 같은 수식에서 광원과 표면의 정점 사이의 거리인 d는 기존 정보로 계산할 수 있으며,
여기에 임의의 반지름 r을 1, 2, 4, 8 로 놓고 계산 할 경우 다음과 같은 그래프가 나옵니다.



이 수식을 넣어서 풀어 보면 반지름이 1일 경우에는 거리가 1.5만 넘어가도 att가 0.2가 되어 버리기 때문에 아주 어두어져 버립니다.
이 경우에는 광원이 표면에 가까이 있을 경우에만 밝게 비춰집니다.
반면 반지름을 8로 할 경우에는 거리가 10정도는 떨어졌을경우에 att가 0.2에 도달 함으로 어두어지는 현상이 천천히 발생합니다.

또한 위의 수식을 2차 방정식으로 풀면 다음과 같이 나옵니다.

$$attenuation\ factor = \frac{1}{(1 + \frac{2}{r}d + \frac{1}{r^2}d^2)}$$

이제 방정식을 OpenGL 의 attenuation factor로 유도 하면, Kc = 1, Kl = 2/r, Kq = 1/(r^2) 을 얻을 수 있습니다. ^^;

구현 코드는 기존의 Specular 부분에 attenuation factor를 추가 하였습니다.

```
// ambient + diffuse + specular + attenuation light
vec3 calcLightAmbDifSpecAtt()
{
 float dist = v_distance;
 //vec3 N = normalize(2.0 * v_eyespaceNormal - c_one);  // 보정 및 정규화 : from ShaderX2
 vec3 N = normalize(v_eyespaceNormal);
 vec3 L = normalize(v_lightPosition);
 vec3 E = vec3(0, 0, 1);

 vec3 H = normalize(L + E);
 vec3 R = reflect(-L, N);
 //vec3 R = -normalize(-2.0 * N * dot(L, N) + L); //-2*N*(dot(L, N)) + L;
 //vec3 R = normalize(2.0 * N * dot(N, L) - L); // from ShaderX2
 float df = max(c_zero, dot(N, L));
 float sf = max(c_zero, dot(N, H)); // Blinn model
 //float sf = max(c_zero, dot(R, E));  // Phong model
 sf = pow(sf, u_shininess);
 vec3 globalAmbient = u_ambientMaterial * vec3(0.05); // * gl_LightSource[0].ambient
 vec3 ambient = u_ambientMaterial * vec3(u_ambientLight);
 vec3 color = globalAmbient;
 // GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION
 float r = 4.0; //lightRadius;
 float d = max(dist - r, c_zero);
 //L = normalize(L / dist);

 // calculate basic attenuation
 float denom = d/r + c_one;
 float att;

 float Kc = c_one;
 float Kl = 2.0/r;
 float Kq = c_one/(r * r);
```

```
 // r이 4.0일때...
 //Kc = 1.0;
 //Kl = 0.5;
 //Kq = 0.0625;

 //Kc = 1.5;
 //Kl = 0.5;
 //Kq = 0.2;

 // Default
 //Kc = 1.0;
 //Kl = 0.0;
 //Kq = 0.0;

 if (df > c_zero) {
  // att = c_one / (denom*denom);
  att = c_one / (Kc + (Kl * d) + (Kq * d * d));
  color += att * (v_diffuse * df + ambient);
  color += att * (u_specularMaterial * sf);
 }

 return color;
}
```

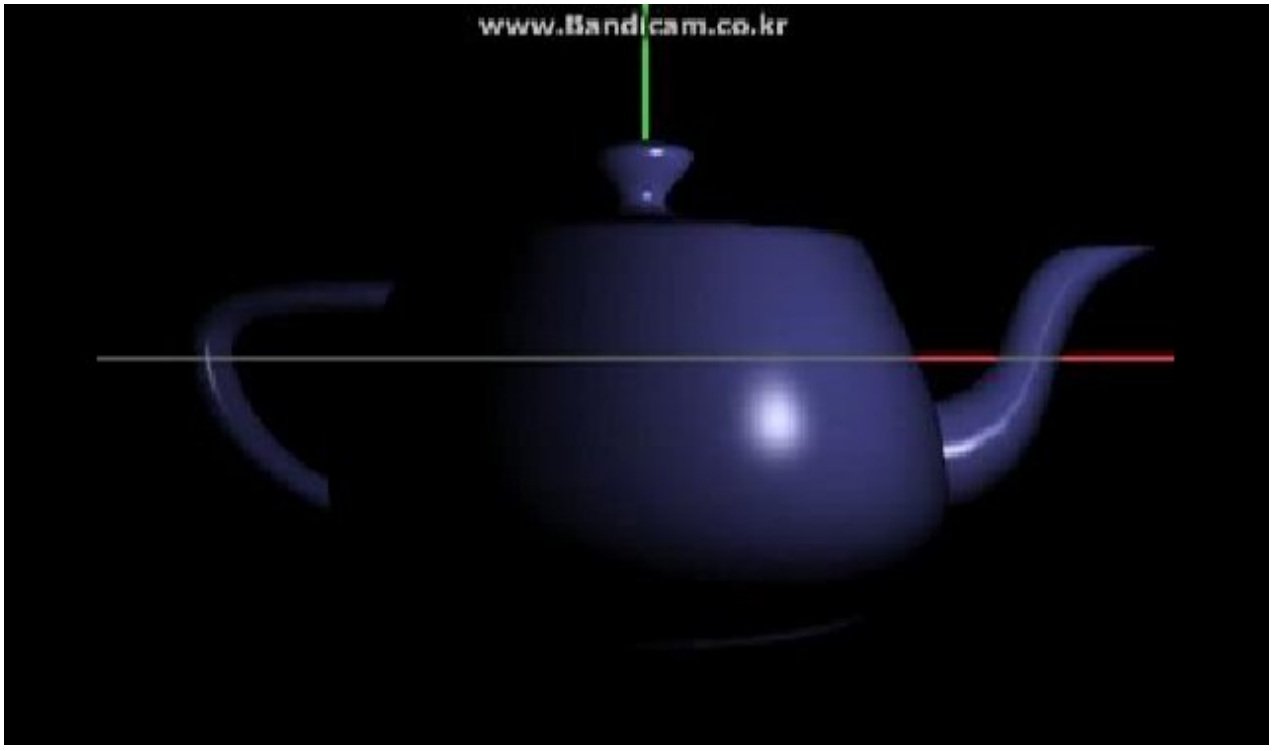
수식중 아래와 같은 주석 부분을 이용할 경우 좀더 최적화가 될것 같습니다.

attenuation factor를 Kc, Kl, Kq로 구분하지 않고 바로 아래와 같이 분모를 (d/r +1)로 구해서 계산하고 있습니다.

```
 // att = c_one / (denom*denom);
```

해가 동치 임으로 결과도 동일 합니다.

〈radius 4로 세팅한 결과〉



〈Kc=1.0, Kl=0.5, Kq=0.2〉

3. 결론

이번장은 Spot Light에 들어가기 앞서서 필요한 감쇠율 Attenuation factor를 적용하는 법에 대해서 알아 보았습니다.

OpenGL ES 2.0 Programming Guide 책의 "8장 Vertex Shaders"에서 Spot Light 부분에 attenuation factor 가 나오는데, K0, K1, K2 만 언급하고 생략 되어 있습니다.
혹 이책은 기본적으로 OpenGL Super Bible 이나 Red Book 을 숙지하고 있다는 전제로 시작하고 있기 때문에 세세한 부분을 설명하지는 않습니다. -ㅁ-;

결국 OpenGL ES 2.0에서는 다음의 glLightf()를 지원하지 않기 때문에 직접 구현해 주셔야 합니다.

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.5);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.2);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

3가지 정보다 Fixed Pipe line 기준으로 본다면 3개다 상수 임으로 uniform 형으로 전달 받아서 처리해 줘도 됩니다.
또는 "light attenuation" 을 구하는 방정식을 이용해서 직접 shader 내에서 구현해도 됩니다.
향상된 버전인 "Improved light attenuation"도 제공하고 있음으로 추가적인 공부가 필요하신 분은 참고하십시요.

참고로 OGL ES 2.0 Programming Guide 책에 대해서 좀더 설명한다면 이책에서 제공하는 예제는 아주 많이 부실 합니다.
하지만, Windows, iOS, Android, WebGL 까지 두루 예제로 제공하면서 MIT License 임으로 뜯어 고쳐서 실무에 응용해도 됨으로 분석은 하는 것이 좋습니다.
이 코드와 더불어 동작 가능한 소스(?)는 아니지만, iOS 용으로 OpenGL 과 호환 가능하도록 API Set을 만들어 놓은 작업이 있습니다.(http://code.google.com/p/gles2-bc/)

이 소스도 같이 겸해서 보시는 것을 추천 합니다. ^ㅁ^;

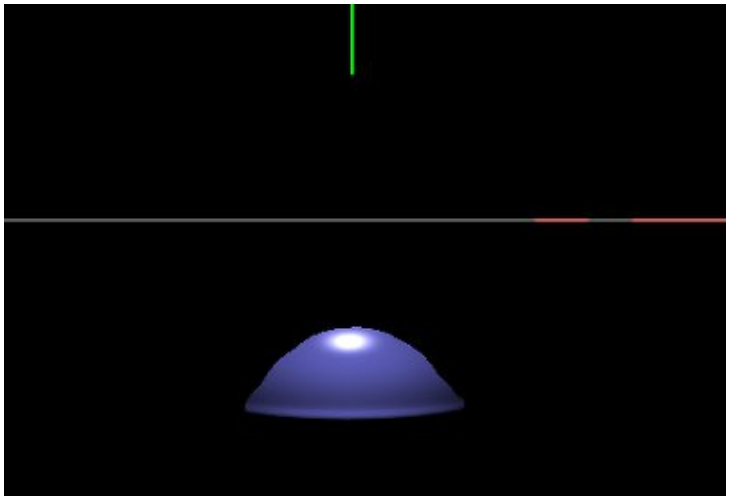
또한 이 장을 통해서 "I'm doing it wrong" 블로그의 저자 님이 적용한 Light Attenuation 으로도 비슷한 효과가 나는 것을 확인하였습니다.

OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list
http://code.google.com/p/myastro/downloads/detail?name=02_ParametricSurface_v.1.2.1.zip&can=2&q=#makechanges

[GLES20] 14. 조명 적용하기 - Spot Light

2011.11.28 23:31



조명 적용하기 - Spot Light

이번장은 기초 조명 과정중 마지막 장이라 할 수 있는 Spot Light 입니다.

Spot Light에 대해서는 iPhone 3D Programming 책에서는 다루지 않고 있습니다. (attenuation 감쇠율 부분도 빠져 있음)
이 부분에 대해서는 OpenGL ES 2.0 Programming Guide 책의 "8장 Vertex Shader"을 참고하는 것이 좋습니다.

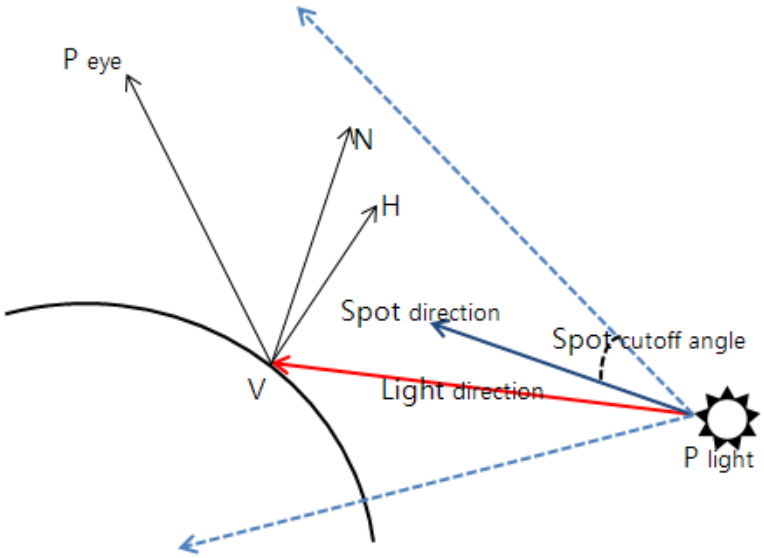
물론 OpenGL ES2.0 책은 독자가 OpenGL 조명에 대해서 어느 정도 이해한 상태에서 설명을 합니다.
그렇기 때문에 OpenGL ES 부터 시작 하는 개발자들은 구현 함수인 spot_light() 함수만 봐서는 이런 수식을 이용하면 되겠구나 할 수는 있지만 실제 동작 시켜 볼수는 없습니다.

하지만, 13장 까지 잘 이해하고 따라 오셨다면 14장의 Spot Light도 책내의 spot_light()함수만 보고도 적용하실 수 있을 것이라고 생각합니다.
추가적인 참조 자료로는 바른 생활님의 "[GLSL]23 Spot Light 의 구현" 장과 PowerVR SDK Training Course 내에 있는 "ComplexLighting" 코드를 분석해 보시면 도움이 될 겁니다.

여기서는 Spot Light의 수식과 관련된 핵심 부분만 정리하도록 하겠습니다.

1. Spot Light

기본적으로 Spot Light는 다음과 같은 그림으로 설명할 수 있습니다.



1. 광원으로 부터 Spot(지점) 방향으로 향하는 벡터 Spot direction 과 광원의 표면의 정점으로 향하는 Light direction 간의 사잇각을 구한다.

2. 그 사잇각이 Spot cutoff angle 보다 작다면 Spot Effect를 계산하고, 크다면 감쇠율을 0.0으로 둔다.

3. Spot factor가 Cutoff angle 보다 작다면, 아래와 같은 Spot Effect 수식에 따라서 계산한다.

$$spotEffect = (spotDirection \cdot lightDir)^{spotExp}$$

위의 수식을 완성하기 위해서는 다음과 같은 값들을 본App으로 부터 uniform 형으로 받아와야 합니다.

매개변수	기본값	설명
GL_SPOT_DIRECTION	0,0,-1	방향
GL_SPOT_EXPONENT	0	스포트라이트 지수
GL_SPOT_CUTOFF	180	스포트라이트 절단각

(spotExp의 기본값은 0으로 이 값이 높아지면 spotEffect가 활성화 되는 영역과 그 밖에 감쇠되는 영역이 부드러워 집니다.)

13장까지 정리한 것을 기준으로 14장을 보면 Spot Light는 쉽게 해결할 수 있습니다.

Spot Light에 대한 구현 코드는 다음과 같습니다.

Shaders/PixelLighting.frag

```
 // ambient + diffuse + specular + attenuation + Spot light
vec3 calcLightAmbDifSpecAttSpot()
{
      ......
  if (u_spotCutOffAngle < 180.0)
  {
    vec3 S = (vec3(u_spotDirection));
    float spot_factor = dot(-L, S);    // ------------------------------ (1)

    if (spot_factor >= cos(radians(u_spotCutOffAngle))) {    // -------- (2)
      spot_factor = pow(spot_factor, u_spotExponent);        // -------- (3)
    } else {
      spot_factor = c_zero;      // ---------------------------------- (4)
    }
    att *= spot_factor;      // ------------------------------------- (5)
 }
      ......
}
```

---

(1) Spot direction과 Light direction 사잇각 구하기

지점 방향을 나타내는 Spot direction 벡터와 광원 방향을 나타내는 Light Direction 간의 사잇각을 구합니다.

Spot direction은 본App에서 기본값인 (0, 0, -1)로 설정 되어 있으며,

Light Direction은 vertex shader에서 광원과 정점의 벡터 차를 통해서 구한 Positional Light벡터를 이용합니다.

dot(-L, S)에서 -L인 이유는 광원과 정점의 벡터 차를 구했음으로 그 벡터 방향은 "정점->광원"으로 향합니다.

저희는 "광원->정점" 방향으로 구해야 함으로 -L을 해줘야 합니다.

(2) Spot factor와 cut off angle 비교

Spot Light는 원뿔형으로 빛이 비춰집니다.

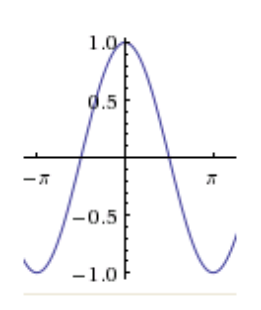
이때 원뿔의 영역을 정하는 기준이 Spot direction을 중심으로 spot cut off angle(절단각)을 통해서 구해 집니다.

즉 이 cut off angle 내에만 Spot Effect를 계산 해주고, 나머지 영역은 감쇠율 값에 0.0을 해줌으로서 빛을 전부 빼줘야 합니다.

이때 cos() 함수가 이용되는데 이는 spot_factor가 -L과 S간의 내적을 통해서 구해 짐으로 그 결과는 cos(theta) 입니다.

그런데 절닥각보다 작아야 한다고 하는데, 비교식은 "크거나 같다" 입니다.

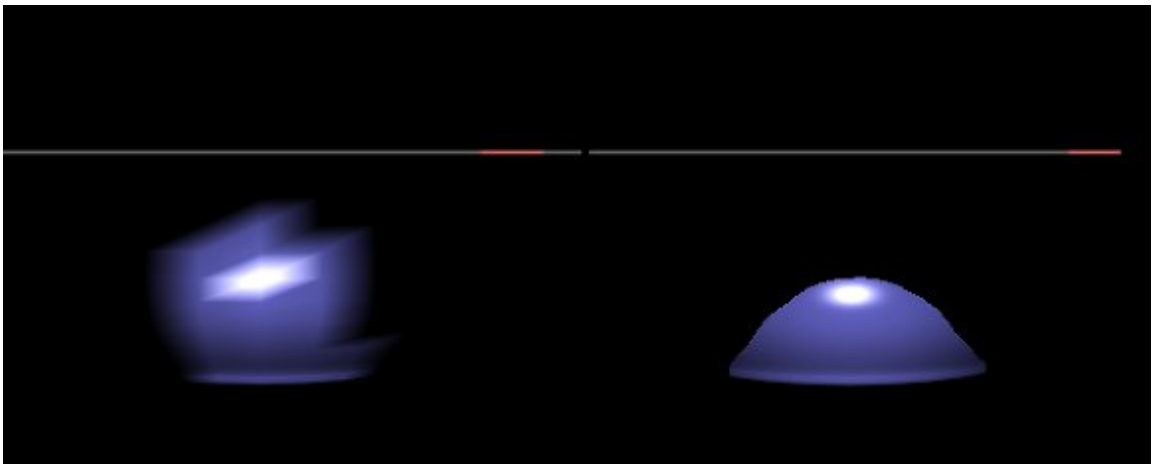
이는 아래와 같이 Cosine함수가 Theta 각도(Range)가 커질수록 그 결과값(치역)은 작아지기 때문에 그렇습니다.



결국 내적의 결과인 spot_factor가 cos(CutOffAngle)의 결과 보다 크다면, 그 사잇각은 작다는 것을 의미합니다.
(이 부분은 제가 이렇게 이해 했다는 의미 입니다. -ㅁ-; 혹 다른 의미가 있다면 댓글 달아주세요 ^^;)

(4) 절단각에서 벗어나는 영역은 감쇠율을 최대치로 높인다.

(5) 감쇠율에 Spot factor을 곱해줘서 뒤에 오는 주변광, 확산광, 경면광에 적용해 준다.



〈왼쪽은 Vertex Light를 이용한 Spot Light 이며, 오른쪽은 Pixel Light를 이용한 Spot Light〉

2. 본App 에서 Spot Light 실행 하기

Shader 내의 코드는 수학적 논리가 단순해서 어렸지는 않습니다.
그런데, GLES20 책은 응용 App 단 부분은 생략되어 있기 때문에, 도대체 Spot direction은 왜? (0, 0, -1)을 넣어야 하고,
Spot Exponent는 몇으로 잡아줘야 하는지? Cut Off Angle 절단각은 180도를 넘어가면 어떻게 되는지?
Light direction은 몇으로 잡아야 하는지? 등에 대해서는 책만 봐서는 알수 없습니다.

그럼 실제 응용 App에서는 어떻게 호출 해 줘야 하는지 보도록 하겠습니다.

Classes/RenderingEngine.CH02.ES2.cpp

```
void RenderingEngine::Initialize(int width, int height)

{
......
 m_pixelLight.Uniforms.AttenuationFactor = glGetUniformLocation(program, "u_attenuationFactor");
 m_pixelLight.Uniforms.IsAttenuation = glGetUniformLocation(program, "u_computeAttenuation");
 m_pixelLight.Uniforms.LightRadius = glGetUniformLocation(program, "u_lightRadius");
 m_pixelLight.Uniforms.SpotDirection = glGetUniformLocation(program, "u_spotDirection");
 m_pixelLight.Uniforms.SpotExponent = glGetUniformLocation(program, "u_spotExponent");
 m_pixelLight.Uniforms.SpotCutOffAngle = glGetUniformLocation(program, "u_spotCutOffAngle");
......
}

-------------------------------------------------------------------------------
Attenuation과 Spot Factor에 대한 신규 uniform 형들이 추가 되었음으로 이를 초기화 함수에 추가해 줍니다.
-------------------------------------------------------------------------------


void RenderingEngine::Render(const vector<Visual>& visuals)
{
......

 // ----------------------------------------
 // PixelLight
 bool bIsPixelLight = true;    // ------------------------------------ (1)
 ProgramHandles handler;
 if (bIsPixelLight)
   handler = m_pixelLight;
 else
   handler = m_vertexLight;
 glUseProgram(handler.Program);
```

```
// Set the light Mode            // ------------------------------------ (2)
// 0 : MENU_DIFFUSE_MATERIAL -> DIFFUSE_MATERIAL
// 1 : MENU_DIFFUSE_LIGHT -> DIFFUSE_LIGHT
// 2 : MENU_AMBIENT_DIFFUSE_LIGHT -> AMBIENT_DIFFUSE_LIGHT
// 3 : MENU_AMBIENT_DIFFUSE_SPECULAR_LIGHT -> AMBIENT_DIFFUSE_SPECULAR_LIGHT
// 4 : MENU_AMBIENT_DIFFUSE_SPECULAR_ATT_LIGHT -> AMBIENT_DIFFUSE_SPECULAR_ATT_LIGHT
// 5 : MENU_AMBIENT_DIFFUSE_SPECULAR_ATT_SPOT_LIGHT -> AMBIENT_DIFFUSE_SPECULAR_ATT_SPOT_LIGHT
glUniform1i(handler.Uniforms.LightMode, MENU_AMBIENT_DIFFUSE_SPECULAR_ATT_SPOT_LIGHT);
//glUniform4f(handler.Uniforms.AmbientLight, 0, 0, 0, 0);

// Initialize various state.
glEnableVertexAttribArray(handler.Attributes.Position);
glEnableVertexAttribArray(handler.Attributes.Normal);

GLfloat weight = sin(interpolation_z) * 3;//12.0f;//1.5f;    // ------------------- (3)
//weight = 0.0f;
//glUniform1f(handler.Uniforms.Interpolation_z, (GLfloat) weight);
//LOG_PRINT("interpolation_z:%f, cos(%f)", interpolation_z, weight);
interpolation_z += (2*Pi) * 0.0015f;

// Set the attenuation factor.                    // -------------------- (4)
//vec3 attenuationFactor(1.5f, 0.5f, 0.2f);
vec3 attenuationFactor(1.0f, 0.0f, 0.0f);
glUniform3fv(handler.Uniforms.AttenuationFactor, 1, attenuationFactor.Pointer());
glUniform1i(handler.Uniforms.IsAttenuation, 1);
glUniform1f(handler.Uniforms.LightRadius, 4.0f);
  // Set the light position.
//vec4 lightPosition(0.f, 0.f, -1.f, 1.0f); // for ortho
vec4 lightPosition(0.f, 0.f, -6.f, 1.0f);       // --------------------------------- (5)

//lightPosition.x += weight;
lightPosition.z = max(-6.5f, lightPosition.z+weight*2.f);  // --------------- (6)
lightPosition.y += weight;
//LOG_PRINT("lightPosition:(%f, %f, %f) + weight(%f)", lightPosition.x, lightPosition.y, lightPosition.z, weight);
glUniform4fv(handler.Uniforms.LightPosition, 1, lightPosition.Pointer());


// Set the spot direction.
vec4 spotDirection(0, 0, -1, 1);                    // ---------------- (7)
```

```
 glUniform4fv(handler.Uniforms.SpotDirection, 1, spotDirection.Pointer());
 glUniform1f(handler.Uniforms.SpotExponent, 0.0f);
 glUniform1f(handler.Uniforms.SpotCutOffAngle, 25.0f);
 //glUniform1f(handler.Uniforms.SpotExponent, 40.0f);
 //glUniform1f(handler.Uniforms.SpotCutOffAngle, 51);

.....
}
```

---

(1) : bIsPixelLight 변수 설정합니다.

Pixel Light로 동작할지 Vertex Light로 동작할지 선택할 수 있습니다.

(2) : Spot Light가 동작하도록 설정합니다. ; MENU_AMBIENT_DIFFUSE_SPECULAR_ATT_SPOT_LIGHT

(3) : Light Position의 위치를 변경 시켜주기 위한 가중치 값입니다.

이 값은 sin()함수로 동작 함으로 0~1~-1~0 를 반복합니다.

여기에 세 배를 가해 줌으로 0 ~ 3 ~ 0 ~ -3 ~ 0 값이 가중치 값으로 이용합니다.

(4) : Attenuation factor인 감쇠율 값을 설정 합니다.

기본값인 (1.0, 0.0, 0.0)으로 설정 되어 있지만, 앞장에서 배웠던 광원의 반지름 공식에 의해서 감쇠율이 적용 됩니다.

(5) : Light Position을 설정 합니다.

Light Position 설정은 (0, 0, -6)으로 시작합니다.

예제는 Camera를 통해서 Z축을 정면으로 보는 상태에서 카메라를 10만큼 뒤로 뺀 상태 입니다. (eye(0, 0, 10))

다시 말하면, 모델 객체는 월드 좌표계를 기준으로 Z축을 기준으로 -10 만큰 뒤에 있는 상태입니다.

그럼으로 광원 위치를 모델 객체(로컬 좌표)에 가까이 다가가게 하기 위해서 -6 만큼 빼주었습니다.

(6) : (3)번에 계산한 weight값으로 광원 위치를 변경해 줍니다.

y와 z값을 변경해 줍니다.

Z값의 경우, 시작 설정값이 -6인데 모델 객체(로컬 좌표)의 원점은 -10입니다. 그런데 Z값이 -6.5 이하로 더 내려가면,

광원이 모델 객체와 충돌을 일으키는지 광원이 사라져 버립니다. ^^

그래서 Z값의 최대한 멀어지는 위치는 -6.5로 한정 시켰습니다.

(7) : Spot Direction을 설정 합니다.
기본 값인 (0, 0, -1, 1)로 설정합니다.
w값 1은 vertex positon의 w와 동일하다고 본다고 합니다.
이 예제에서는 w값은 실제 shading 중에 사용되지는 않습니다.
Spot Exponent는 기본값인 0으로 하였으며, 이 값을 높이면 Spot Light 경계지점이 부드러 지네요 ^^;
Spot Cut Off Angle은 25도로 설정 하였으며, 이 값을 높이면 Spot Light 가 커지게 됩니다.
(180도 이상 올리면 Spot Light가 동작하지 않게 됩니다.)

----- 여기서 부터는 제 주관적인 관점 이기 때문에 틀릴수도 있습니다. 저도 독학 이어서 ^ㅁ^; -----
여기서 중요한 값은 -1 입니다. 스팟 라이트가 정상적으로 동작하기 위해서는 이 값은 음수여야 합니다.
광원의 위치인 Light Positon은 음수 방향으로 보고 있습니다.
Spot Light도 같은 방향인 음수 방향으로 보고 있어야 L과 S간의 내적을 통해서 구하는 사잇 각이 예각 내에서 구해 지게 됩니다.
만약 Spot Light가 (0, 0, 1) 이면 L과 S의 벡터 방향이 반대가 되어 버려서 내적을 통해서 계산 되는 사잇 각은 예각이 아닌 둔각이 되어 버립니다.
그럼으로 이 부분은 (0, 0, -1)이 되어야 하는것 같습니다.
(아닐수도 있습니다. 어쨋든 양수로 하면 효과가 원하는 바대로 나오지 않습니다.)

〈Vertex shader로 구현된 Spot Light〉

〈Pixel shader로 구현된 Spot Light, Exponent 0〉

〈Pixel shader로 구현된 Spot Light, Exponent 36〉

3. 결론

여기서 설명한 부분들은 iPhone 3D Programming 책과 OpenGL ES 2.0 Programming Guide 책에 이미 나와 있는 내용 입니다.
책을 우선 보시고 여기서의 내용은 레퍼런스로 정도로 이용하시면 될것 같습니다.

위의 두 책에서 아쉬운 부분은 그림자 효과에 대해서는 다루지 않았습니다.
그림자 효과에 대해서는 Shader X2:Direct X9 쉐이더 프로그래밍 책에 설명이 나와 있던데,
궁금하신 분은 X2 책을 참고하셔도 좋을 것 같네요.

이상으로 조명과 관련한 기초 과정을 마무리 하도록 하겠습니다.
어느새 강좌를 쓰기 시작한지도 한달이 되었습니다.
보통 한달에 4개정도 포스팅 했는데, 14개를 하다니 엄청나게 폭풍 집필^^;을 했네요. 큭

앞으로는 속도 조절을 하면서 글을 써야할 것 같네요. -ㅁ-;

지금까지 여기서 설명한 방법들은 책을 보고 실행해 봐서 이렇게 되겠구나 하고 생각하는 점들을 정리한 것입니다.
실제 강의를 듣거나 마스터 분한테 사사 받은것이 아니어서 오류가 있을 수도 있습니다. ^^;
혹 논리에 문제가 있거나 정정이 필요한 부분이 있다면 댓글로 지적해 주십시요.

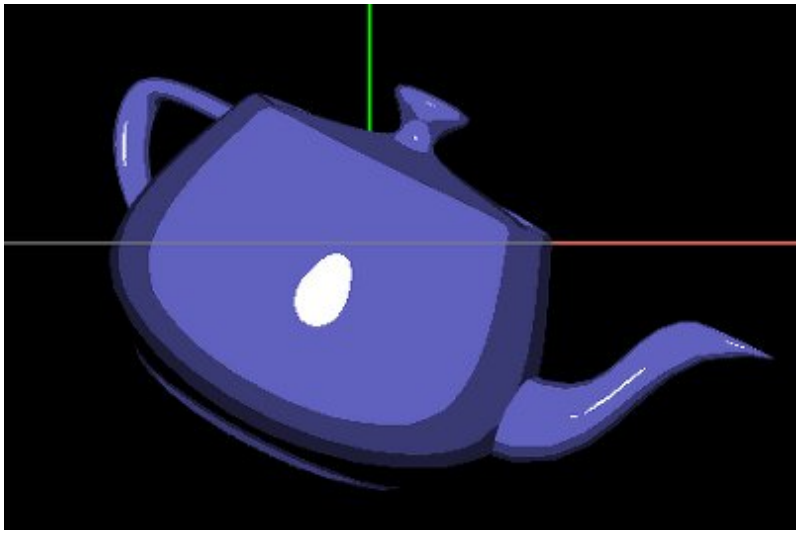
다음 장 부터는 조명 기법중 Cartoon 효과를 주는 Toon Shading을 간단하게 정리한 후에 Texture 처리로 넘어가도록 하겠습니다.

OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list
http://code.google.com/p/myastro/downloads/detail?name=02_ParametricSurface.v.1.2.1.zip&can=2&q=#makechanges

[GLES20] 15. 조명 적용하기 – Toon Shading

2011.11.30 23:51



조명 적용하기 – Toon Shading (불연속적인 쉐이딩 표현)

툰 쉐이딩은 다른 말로 Cell Shading 이라고도 부릅니다.
다음의 게시물이 있는데 정의가 잘 표현되어 있습니다.
http://bbs2.ruliweb.daum.net/gaia/do/ruliweb/default/101/read?bbsId=G005&articleId=8749477&itemId=421
에 보시면, cartoon rendering 에 대해서 "불연속적인 쉐이딩 표현" 또는 "실루엣 외곽선 묘사" 둘 중에 한가지가 적용되어 있다면, Toon Shading이라고 정의하고 있습니다.

iPhone 3D Programming 책에서는 "불연속적인 쉐이딩 표현" 방식으로 구현을 하고 있습니다.

1. Toon Shading 이란?

3D Programming의 목적은 화면에 비춰지는 것이 마치 실 세계의 그것과 비슷하게 표현하기 위한 방법입니다.
볼륨감이 느껴지고 깊이 감이 느껴지는 사람의 시각적으로 느끼는 것입니다.
그런데, 만화 처럼 느껴지게 처리하면, 비 현실 적인 느낌이 되어 버립니다.
Toon Shading의 렌더링 기법을 Non photo realistic rendering(NPR) 비사실적인 렌더링 방식 이라 합니다.

iPhone 3D Programming 책의 경우 "4장 깊이와 현실감 향상시키기"장에서 Toon Shading 을 설명하고 있습니다.

이책에서는 코드는 있지만, 설명은 없습니다. 정말 멋지게 처리 된다? 정도 -ㅁ-;
Toon Shading에 대해서 좀더 설명해 줬으면 하는 부분이긴 하지만, 이렇게 처리 하면 되는 구나 정도로도 괜찮을 것 같네요.

저는 저자가 설명하지 않고 넘어간 부분에 대해서 부연 설명을 하도록 하겠습니다.

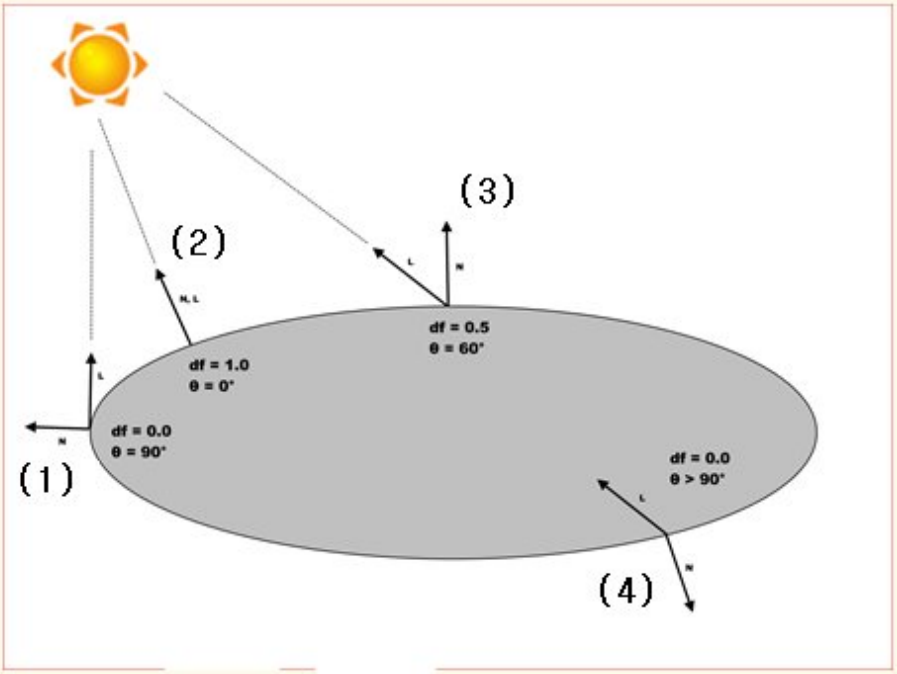
2. 불연속 적인 쉐이딩 표현

14장까지 저희가 배운 조명 처리 방법은 주변광 Ambient, 확산광 Diffuse, 경면광 Specular, 지점광 Spot 등이 있었습니다.
이중 확산광과 경면광에는 Diffuse factor와 Specular factor가 있었습니다.

Diffuse factor는 빛의 각도와 표면의 각도의 의해서 표현되는 값이며, Specular factor는 신선의 각도와 빛의 각도, 표면의 각도에 의해 표현되는 하일라이트 값입니다.

조명을 표현할때 중요한 factor는 Diffuse factor 입니다.

Figure 4.9. Diffuse Lighting



빛이 표면에 100% 밝기로 보여주기 위해서는 빛의 각도와 표면의 각도의 사잇 각이 0도를 이루어야 합니다.(acos(1) = 0도)
빛이 표면에 50% 밝기로 보여주기 위해서는 빛의 각도와 표면의 각도의 사잇 각이 60도를 이루어야 합니다. (acos(0.5) = 60도)
빛이 표면에 0% 밝기로 보여주기 위해서는 빛의 각도와 표면의 각도의 사잇 각이 90도 이상을 이루어야 합니다. (acos(0) = 90도)
확산광은 위와 같이 consine 함수를 통해서 구한 diffuse factor을 이용해서 쉐이딩을 스무스하게 표현 합니다.

iPhone 3D Programming 책의 예제에서는 다음과 같이 4단계로 이루어져 있습니다.

Shaders/PixelLighting.frag

```
vec3 calcLightToon() {
      ......
  float df = max(c_zero, dot(N, L));
  if (df < 0.1) df = 0.0;
  else if (df < 0.3) df = 0.3;
  else if (df < 0.6) df = 0.6;
  else df = 1.0;
      ......
}
```

-. 확산율 결과가 100% ~ 60% 면 100%로만 확산율을 적용한다.
-. 확산율 결과가 59% ~ 30% 면 60%로만 확산율을 적용한다.
-. 확산율 결과가 29% ~ 10% 면 30%로만 확산율을 적용한다.
-. 확산율 결과가 9% ~ 0% 면 0%로만 확산율을 적용한다.

4단계 영역으로 끊겨서 표현 됨으로 불연속 적인 쉐이딩 표현 입니다.

확산율과 더불어 하일라이트를 표현하는 Specular factor에도 두단계로 제약을 가해 줍니다.

Shaders/PixelLighting.frag

```
vec3 calcLightToon() {
      ......
  float sf = max(c_zero, dot(N, H)); // Blinn model
  sf = pow(sf, u_shininess);
  sf = step(0.5, sf);
      ......
}
```

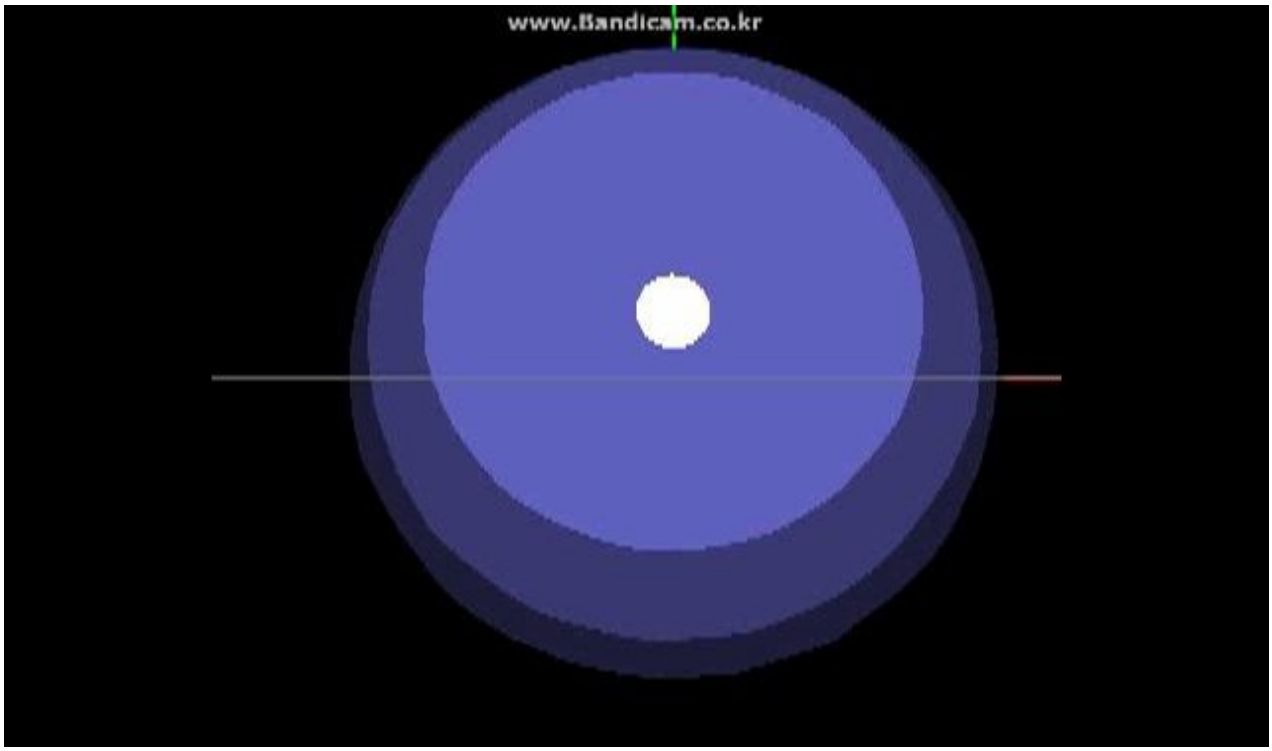
---

step(a, b) 함수는 다음과 같은 함수입니다.

```
float step(float edge, float x)
{
    return x < edge ? 0.0 : 1.0;
}
```

다시 말하면, step(0.5, sf)는 0.5보다 크면 Specular Light를 100% 반영하고, 0.5보다 작다면 0% 반영하라는 의미 입니다.

결과는 다음과 같이 비 현실 적인 예쁜 영상이 됩니다. ^ㅁ^;



3. 결론

Toon Shading은 이 방법 말고도 더 예쁘게 표현되는 많은 방법들이 있습니다.
먹물(Ink) 효과, 외곽선(Outline), 실루엣 외곽선, Hatching 등 정말 공부가 많이 있네요 ^^;

OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list
http://code.google.com/p/myastro/downloads/detail?name=02_ParametricSurface.v.1.2.1.zip&can=2&q=#makechanges

[GLES20] 16. Q/A 몇가지와 FBO 사용시 유의점 정리

2012.01.01 15:43

제가 정리한 Android 3D Programming 강좌에서 궁금한 질문 사항이 있어서 정리를 해 보았습니다.

1. Q&A

[질문1]
왜 안드로이드는 glgen glbind 안시키죠?

[답변]
glGen과 glBind는 어떤 부분을 말씀하시는지 잘 모르겠네요.
FBO, VBO, Texture 등 glGen과 glBind를 접두사로 취하는 것들이 많아서.
우선 제가 여기에 올린 예제는 Android와 Mfc를 혼합해 놓았습니다.
제 프로젝트에서는 Eclipse로 Android project만 열었을 경우에는 NDK(jni)쪽 파일들은 build용 Android.mk와 JNI용 NativeRenderer.cpp만 있지 실제 Renderer 구현 부분은 보이지 않습니다.
MFC쪽 프로젝트를 여서서 보시면 관련 파일들을 볼수 있으며 이쪽에 glGenxxx, glBindxxx 관련 부분이 있습니다.

[질문2]
단말기에서 가로 모드 / 세로 모드로 변경될 경우, 해상도 처리는 어디서 하나요?

[답변]

안드로이드의 경우, Java 단에서 처리 해주셔야 합니다.

안드로이드에서는 EGLView를 GLSurfaceView로 구현 합니다.

그리고 이 GLSurfaceView를 Android Activie의 Main View에 add 시킵니다.

이부분은 안드로이드 기본서적를 참고하시는게 좋을 것 같습니다.

(제 소스 코드상에서는 GLRenderActivity.java에 보시면 setContentViw(mGLView) 부분에서 추가 합니다.)

GLSurfaceView가 정상적으로 생성 되었다면, 단말기의 Orientation이 변경 될때 마다,

onSurfaceChanged() 함수가 호출 됩니다.

저는 NativeRenderer.java에서 이 부분을 처리하고 있으며, nativeResize()함수를 호출합니다.

이 함수의 구현 부분은 JNI로 구성되어 있으며, NativeRenderer.cpp의

Java_com_myastro_main_jni_NativeRenderer_nativeResize 와 메칭 됩니다.

참고로 제가 올린 예제에서는 ResizeWindow()내에 glGen과 glBind가 있는데, 이를 해제 하는 부분이 빠져 있습니다. 또한 단말기으 방향이 변경될때마다 계속 생성되게 됩니다.

최적화를 시킨 예제는 아니기 때문에 조정이 필요합니다.

예를 들어 아래와 같이...

```
void RenderingEngine::ResizeWindow(int width, int height)
{
   if (m_screenSize.x != width && m_screenSize.y != height) {

   // Create Off-screen FBO
   glGenFramebuffers(1, &m_offscreen_framebuffer);
   glBindFramebuffer(GL_FRAMEBUFFER, m_offscreen_framebuffer);
   // Create Off-screen FBO
   }
}
```

물론 이 방법도 이전 fbo를 삭제는 안하고 있기 때문에 완전히 최적화 되어 있지는 않습니다. ^^;.

[질문3]

GLSurfaceView는 더블버퍼링을 자기가 알아서 해주는가?

[답변]

더블버퍼링을 해주지는 않습니다.

제가 알기로는 아이폰도 더블버퍼링을 지원하지 않는 것으로 알고 있습니다.

일반적으로 OGLES2.0에서는 더블버퍼링을 지원한다고도 하는데(책에서는), 아직 저는 성공하지 못햇습니다.

(물론 아직 시도도 안하고 있고요 ^^;)
하지만 이미 아실지 모르겠는데, 더블버퍼링을 대체하는 기술로 Framebuffer Object를 이용합니다.
FBO를 OnScreen 한개, OffScreen 한개씩.. 만들어서 사용하는 방법입니다.
아이폰 3D programming 책에도 설명이 나와 있으니 이미 접해보셨을것 같네요.
(제 블로그에도 RenderToTexture 장(http://blog.daum.net/aero2k/53)에 기본적인 설명은 되어 있습니다.)

[질문4]
아이폰에서는 glDraw 하고 나서 [m_context presentRenderbuffer:GL_RENDERBUFFER]; 함수(iOS 기준으로는 message라고도 부르죠.)를 호출 하는데, 안드로이드는 GLSurfaceView만 등록하면 되나요?

[답변]
아이폰에서는 OpenGLES를 처리하는데 EGL를 사용하지 않고 EAGL이라고 Apple 용으로 만든 버전을 사용합니다.
다음의 [m_context presentRenderbuffer:GL_RENDERBUFFER]; 이 부분은 아이폰에만 해당하는 사항임으로 처리해줄 필요는 없습니다.

물론 안드로이드나 윈도우즈 시스템에서 OpenGL 를 이용하기 위해서 EGL 객체를 만들어야 합니다.
이 EGL를 만드는 위치가 안드로이드의 경우, GLSurfaceView임으로 이 객체를 활성화 해서 처리해줘야 합니다.
(윈도우즈의 경우, MFC 기준으로는 CView 클래스를 상속받아서 CEGLView 를 만들어야 합니다. EGL 에 대해서는 관련 자료들이 많으며, 제 소스와 비교해서 보시면 이해가 되시리라 생각 됩니다.)

GLSurfaceView에서 호출되어 지는 onDrawFrame() 함수(Listener method)에 제작한 Renderer를 호출해주시면 됩니다.
(이 렌더러는 Java로 작성할 수도 있으며, 제 예제와 같이 Native C++로 작성할 수도 있습니다.)
제 예제에서는 NDK단의 nativeRender() -> Java_com_myastro_main_jni_NativeRenderer_nativeRender함수를 호출 합니다.
Java Renderer와 Native Renderer의 사용 법에 대해서는 이미 안드로이드 서적에 많이 나와 있습니다.
책을 참고해서 제 예제를 보시면 도움이 더 잘될것 같네요 ^^;

2. FBO 사용시 유의점 정리

마지막으로 질문에는 없지만 첨언을 달자면, 아이폰과 안드로이드(Windows MFC)에서의 Framebuffer Object의 사용 방법에 있어서 차이가 있습니다.

Framebuffer란 아래의 그림과 같이 맨 마지막 단에 위치한 버퍼로 OpenGL에서 LCD Buffer 단으로 넘어가기 직전 위치 입니다.
(좀더 상세한 설명은 바른생활님이 정리하신 http://cafe.naver.com/gld3d/130 을 읽어 보십시요 ^^;)

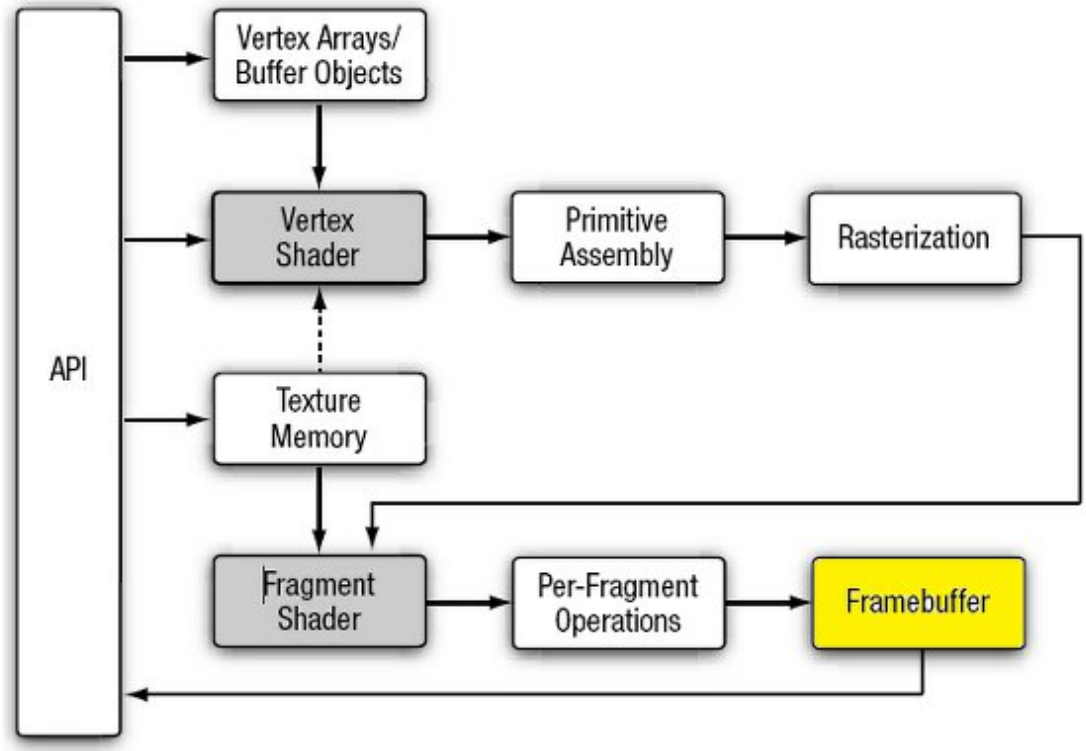


Figure 1-1 OpenGL ES 2.0 Graphics Pipeline

아이폰 소스를 안드로이드 소스로 포팅하는 중에 가장 첫번째로 부딪히는 부분이 아마도 Framebuffer Object 사용법이 아닐까 생각 합니다.

제가 올린 Android 3D예제에서도 이 문제와 연관된 논리상 오류가 있습니다.

(물론 동작하는데는 문제가 없습니다. ^^)

논리상 문제가 되는 부분은 렌더러를 초기화 하는 Initialize 함수에서 발생합니다.

```
void RenderingEngine::Initialize(int width, int height) {

    // Create the depth buffer.
    glGenRenderbuffers(1, &m_depthRenderbuffer);
    glBindRenderbuffer(GL_RENDERBUFFER, m_depthRenderbuffer);
    glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT16, width, height);
    // Create the framebuffer object.
    glGenFramebuffers(1, &m_framebuffer);
    glBindFramebuffer(GL_FRAMEBUFFER, m_framebuffer);
    glFramebufferRenderbuffer(GL_FRAMEBUFFER,
```

```
                        GL_COLOR_ATTACHMENT0,
                        GL_RENDERBUFFER,
                        m_colorRenderbuffer);
    glFramebufferRenderbuffer(GL_FRAMEBUFFER,
                        GL_DEPTH_ATTACHMENT,
                        GL_RENDERBUFFER,
                        m_depthRenderbuffer);

    // Bind the color buffer for rendering.
    glBindRenderbuffer(GL_RENDERBUFFER, m_colorRenderbuffer);

------------------------

#ifndef PLATFORM_IOS
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
#endif
}
```

초기화 함수를 FBO로 m_framebuffer 를 생성했는데, 안드로이드와 Windows에서는 이를 사용하지 않고 있습니다.

glBindFramebuffer(GL_FRAMEBUFFER, 0);

호출해서 m_framebuffer를 사용하지 않고 "0" 번 ID를 사용 합니다.

아이폰의 경우에는 On Screen Buffer 쪽 FBO ID의 경우, System Frame buffer Object 인 0번이 아닌, 개발자가 glGen한 ID를 사용해야 합니다.(아이폰 3D 프로그래밍 책에서도 언급이 되어 있습니다.)
하지만, 안드로이드와 Windows 플랫폼에서는 System Frame buffer Object는 0번으로 이미 정해져 있습니다.
그럼으로 On Screen Buffer 값은 0번 이어야 합니다.

안드로이드와 윈도우즈 플랫폼에서는 System Frame buffer Object를 확인하기 위해서 이용되는
glGetIntegerv(GL_FRAMEBUFFER_BINDING, &m_framebuffer);
이용해서 얻을수도 있습니다. 이 값은 0번이 나옵니다.

정리하면, 아이폰 소스를 안드로이드 소스로 포팅할 경우에는 On Screen Buffer 즉 LCD Buffer에 그리는 객체를 처리할 경우에는 FBO ID가 0번 이어야 정상적으로 화면이 디스플레이 됩니다.

-- 추가 설명 1 --
On Screen Buffer 라는 용어가 잘 이해가 안가신다면, 화면에 렌더링한 3D 객체를 그려주기 위해서,
glDrawArray 또는 glDrawElement 함수를 호출합니다. 이 함수를 호출하기 직전에 지정된 FBO ID가 On Screen Buffer 를 가리키는 ID 입니다.

-- 추가 설명 2 --
그렇다면 FBO를 이용해서 Render To Texture를 구현할 경우에는 더블 버퍼링과 같이 On Screen Buffer 뿐만 아니라 Off Screen Buffer를 FBO로 생성합니다.
이때는 On Screen Buffer에 해당하는 FBO ID는 0번이 되어야 하고, 더블 버퍼리을 위해 생성한 Off Screen Buffer용 FBO ID는 glGenFramebuffers(1, &m_offscreen_framebuffer);와 같이 만들어서,
glBindFramebuffer(GL_FRAMEBUFFER, m_offscreen_framebuffer); 함수로 바인딩해서 그림을 그리면 됩니다.

좀더 상세한 설명은 Render To Texture 장을 다룰때 다시 정리해 보도록 하겠습니다.

[GLES20] 17. 텍스쳐 적용하기 앞서서 준비 사항 정리

2012.01.16 13:18

16장 부터는 Texture 관련해서 정리해 보도록 하겠습니다.
기본적인 목차는 다음과 같습니다.

1. 이미지 파일 로딩하기
2. Single Texture
3. Multi Texture
4. Bumpmap Texture
5. Cubemap Texture
6. Render To Texture
7. Blur(Bruteforce blur and Gaussian blur)

(물론 위의 목차는 제 개인적인 사정 또는 갑작스럽게 재미 분야가 생기면 변경 될 수도 있습니다. ^^;)

텍스쳐 적용하기 앞서서 준비 사항 정리

3D 에서 Texture를 적용하기 위해서는 PNG, JPG, BMP, TGA, PVR 파일등과 같은 2D image 파일들을 decoding 해서, RGBA 형태의 byte array 로 변환해 줘야 합니다.

예를 들어 바나나를 3D로 그리기 위해서는 다음과 같은 2D 이미지 파일이 필요합니다.
(이 파일은 "http://heikobehrens.net/2009/08/27/obj2opengl/"에서 배포된 바나나 파일 입니다.)

이미지 자체는 바나나의 앞-뒷 면을 보여주는 화면 입니다.

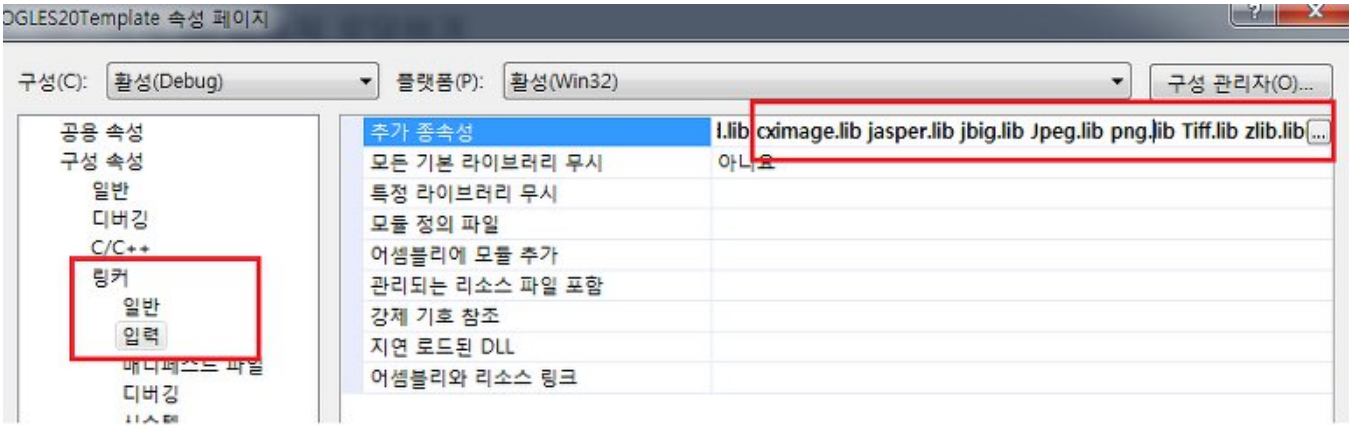
1. MFC에서 2D 이미지 로딩하기

MFC에서 이미지 파일을 로딩하는 인터넷에서 검색하면 다양한 방법을 찾을 수 있습니다.
BMP, TGA 등의 Decoder는 제 블로그에서도 소개를 했었지만, PNG나 JPG등을 Decoding 하기 위해서는 추가 라이브러리가 있어야 합니다.

여기서 작성하는 예제들은 테스트 용임으로 CxImage 라이브러리를 이용해서 작업하도록 하겠습니다.(Version 5.99c)
CxImage 라이브러리가 zlib 라이센스를 따르는 open source 임으로 저작자에 대한 라이센스 명기만 해준다면 상용화에 응용해도 크게 문제가 되지는 않기 때문에 선택했습니다.

기본적으로 사용법도 간담합니다.
다운로드 받은 CxImage 프로젝트를 Release로 빌드하면, 다음과 같은 라이브러리를 얻을 수 있습니다.

```
cximage.lib jasper.lib jbig.lib Jpeg.lib png.lib Tiff.lib zlib.lib
```



이미지 파일을 로딩하는 방법도 간단합니다.

/OGLES2/Platforms/Windows/NativeInterface.cpp

-. CxImage 라이브러리 호출 부분

```
ImageData CBLoadImage(char* filename, int type)
{
  ImageData imgData;
  CxImage image;
  char resourcePath[256];
  sprintf(resourcePath, "../../Textures/%s", filename);
  image.Load(resourcePath, type);              // ---- (1)
  if (image.IsValid()) {
    int length = (int)image.GetSize();
    long size = 0;
    BYTE* buffer = 0;
    image.Encode2RGBA(buffer, size);           // ---- (2)
    //CxMemFile memfile;
    //memfile.Open();
    //image.Encode(&memfile, CXIMAGE_FORMAT_BMP);
    //BYTE* buffer = memfile.GetBuffer();
    //long size = memfile.Size();
    m_pImageData = new BYTE[size];
    memcpy(&m_pImageData[0], buffer, sizeof(BYTE)*size);
    imgData.Size = ivec2((int)image.GetWidth(), (int)image.GetHeight());
    imgData.length = size;
```

```
    imgData.image_data.resize(size);
    vector<unsigned char>::iterator data = imgData.image_data.begin();
    for (int i=0; i<size; ++i) {
      *data++ = (unsigned char)m_pImageData[i];  // ---- (3)
    }
    image.FreeMemory(buffer);
  }
  return imgData;
}
```

(1) : CXIMAGE_FORMAT_BMP, CXIMAGE_FORMAT_JPG, CXIMAGE_FORMAT_PNG, CXIMAGE_FORMAT_TGA 와같은 type에 따른 이미지 파일 열기

(2) : 로딩한 이미지 파일을 RGBA 형태로 변환

(3) : RGBA 데이타를 Open GL 에서 사용할 byte array 형으로 변환한다.

-. BMP 파일 로딩

```
ImageData CBLoadImageBmp(string filename)
{
  ImageData imgData;
  int len = strlen(filename.c_str());
  char* text_utf = new char[len*2+1];
  strcpy(text_utf, filename.c_str());

  imgData = CBLoadImage(text_utf, CXIMAGE_FORMAT_BMP);
  memset(text_utf, 0x00, len*2+1);
  delete[] text_utf;
  return imgData;
}
```

-. JPG 파일 로딩

```
ImageData CBLoadImageJpg(string filename)
{
......
  imgData = CBLoadImage(text_utf, CXIMAGE_FORMAT_JPG);
......
```

```
}


-. PNG 파일 로딩
ImageData CBLoadImagePng(string filename)
{
......
  imgData = CBLoadImage(text_utf, CXIMAGE_FORMAT_PNG);
......
}


-. TGA 파일 로딩
ImageData CBLoadImageTga(string filename)
{
......
  imgData = CBLoadImage(text_utf, CXIMAGE_FORMAT_TGA);
......
}
```

CxImage에 대해서 더 궁금하신 분은 네이버 블로그 중 다음 글이 잘 설명 되어 있습니다. ^^;
http://blog.naver.com/kdha83?Redirect=Log&logNo=98673605
사용 법은 다음의 페이지를 참고하십시요.
http://www.codeproject.com/KB/graphics/cximage.aspx

2. Android에서 2D 이미지 로딩하기

아마도 이 강좌를 보시는 분들은 기본적으로 안드로이드에서 BMP나 PNG 이미지 로딩은 이미 경험해 보셨을 것으로 보입니다. 그럼으로 2D 이미지 로딩에 대한 구체적인 내용은 생략하도록 하겠습니다. ^^;

단지 Java가 아닌 NDK에서 Open GL를 처리하고 있음으로, Android API인 Drawable 이나 InputStream 을 이용해 얻은 Bitmap 객체를 생성한 후 RGBA byte array buffer로 변경 한후 NDK 단으로 내려 보내야 합니다.

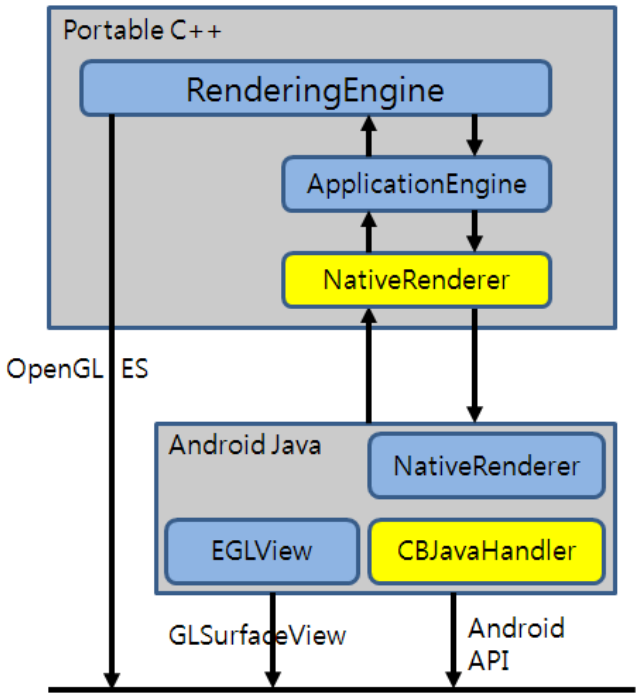
제 엔진(?)에서는 com.myastro.gles.ResourceManager 에서 bmp, png, tga, jpg등의 이미지 파일을 디코딩하도록 구현 되어

있습니다.

RGBA byte array buffer의 경우 Java Activity의 초기화 시에 생성해 둘 수도 있으며, 제가 구현한 방법과 같이 Activity 초기화 시점이 아닌, NDK단에서 Open GL이 초기화 되는 시점에 처리해 줄수도 있습니다.

즉 NDK와의 통신은 Activity main thread에서 호출 되는 형태가 아닌,
Native 단에 Activity 초기화 시점에 저장한 g_VM->AttachCurrentThread(&env, NULL)에서 Activity main thread로 호출 하는 방식으로 구현 되어 있습니다.
(일반적으로 JNI을 통한 통신이 Java -> JNI -> Native layer 로 통신 되지만, g_VM를 이용해서, Native layer -> JNI -> Java 로도 호출도 가능합니다. 이 부분에 대해서는 구글링을 하시거나 제 엔진에서 NativeRenderer20.cpp <--> CBJavaHander.java 간의 관계를 참고하십시요 ^^;)

이미지 파일이 로딩을 위한 순서는 다음과 같이 전개 됩니다.



jni/OGLES20/NativeRenderer20.cpp

```
ImageData CBLoadImage(string filename)
{
.....
   JNIEnv* env;
   g_VM->AttachCurrentThread(&env, NULL);  // --------------- (1)
   jMainCls = (*env).FindClass(CB_CLASS_MAIN);
   mLoadImage =
    (*env).GetStaticMethodID(jMainCls, CB_CLASS_LOADIMAGE_FUNC, CB_CLASS_LOADIMAGE_SIG);
```

```
 jobject res_texture_description =
   (jobject)(*env).CallStaticIntMethod(jMainCls, mLoadImage, (*env).NewStringUTF(text_utf)); // -- (2)

.....

 // ---- (3)
 int width = (int)env->GetIntField(res_texture_description, tex_description_field_id.width);
 int height = (int)env->GetIntField(res_texture_description, tex_description_field_id.height);
 int length = (int)env->GetIntField(res_texture_description, tex_description_field_id.length);
 jobject array_obj = (jobject)env->GetObjectField(res_texture_description, tex_description_field_id.image_data);
 jboolean iscopy;
 jbyte* array_element = env->GetByteArrayElements((jbyteArray)array_obj, &iscopy);

 imgData.Size = ivec2(width, height);
 imgData.length = length;
 imgData.image_data.resize(length);
 vector<unsigned char>::iterator data = imgData.image_data.begin();
 for (int i=0; i < length; i++) {
  *data++ = (unsigned char)array_element[i];  // ------------------- (4)
 }
 int len2 = sizeof(array_element);
 env->ReleaseByteArrayElements((jbyteArray)array_obj, array_element, 0);
 return imgData;
}
```

---

(1) g_VM->AttachCurrentThread(&env, NULL); 를 통한 Native thread 객체 정보 획득

(2) CallStaticIntMethod() 함수를 통해서 Java 단의 CBJavaHandler에서 구현된 _loadImage()함수 호출
이 지점에서 Native thread는 _loadImage()함수를 통해서 해당 이미지 파일을 로딩하기 전까지 Lock 된다.
NativeRenderer20.CBLoadImage.CallStaticInMethod() --> com.myastro.main.jni.CBJavaHandler._loadImage() --> com.myastro.gles.ResourceManager.ResourceManager() --> _loadImage() --> CBLoadImage()
로 이동 합니다.

(3) 로딩한 이미지에 대한 width, height, byte array buffer 얻기
com.myastro.main.jni.CBJavaHandler._loadImage() 함수를 통해서 얻은 tex_description_field_id 로 부터 정보를 얻습니다.

(4) RGBA 데이타를 Open GL 에서 사용할 byte array 형으로 변환

이제 Open GL 로 Texture 를 그릴때 필요한 width, height, RGBA byte array buffer 얻었습니다.

3. 결론

NDK 단에 OpenGL를 구성할 경우에는 이미지 프로세스 모듈을 Android.mk 내에 구현해서 사용하는 경우도 많습니다.
저는 공부 차원에서 Android API 이용해서 구현해 봤으며, 효율성으로 따진다면 그다지 추천하지는 못하겠네요 ^^;
굳이 JNI에 오버로드를 줄 필요는 없을 테니깐요 ㅎㅎ

추가로 아이폰과 달리 안드로이드의 경우에는 단말마다 GPU 칩이 다 다릅니다.
(아이폰은 아직까지 PowerVR 칩만 사용)
이때문에 NDK에 Open GL를 구성할 경우, A 단말에서는 동작하는 앱이 B단말에서는 동작하지 않는 문제가 발생합니다.

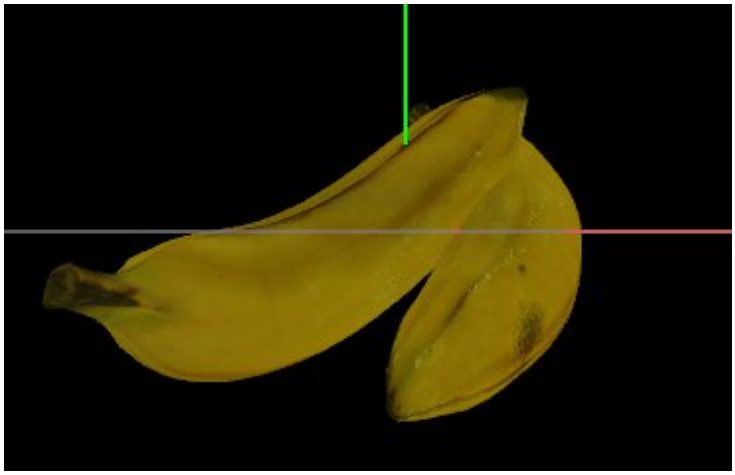
대표적인 예가 갤럭시 S와 갤럭시 탭의 경우 Power VR 칩을 사용하지만, 갤럭시 S2의 경우 ARM Mali 칩을 사용합니다.
Power VR사의 GPU의 경우 아이폰에서도 사용하기 때문에 아무래도 아이폰에서 구현한 코드가 수월하게 포팅이 되지만, 갤럭시 S2의 경우에는 FBO가 정상적으로 동작하지 않는 경우가 발생합니다.

반대로 갤럭시 S2에서는 수월하게 구현한 사항이 갤럭시 S나 탭에서는 비정상 동작하는 경우도 많습니다. -ㅁ-;
예를들어 RTT 구현시 Offscreen size를 NPOT 크기로 설정할 경우, 갤럭시 S2에서는 정상 동작하던 것이
갤럭시 탭에서는 검정 화면만 뜨는 경우가 발생합니다.
이는 Power VR 칩의 경우 이미지의 가로-세로 크기가 POT(Power Of Two, 2의 거듭제곱, 2x2, 4x4, 64x128) Size여야 합니다.
하지만 갤럭시 S2의 경우 NPOT Size도 정상적으로 지원합니다.

갤S2에서 POT에 신경 쓰지 않고 구현하다가 갤탭에서 텍스쳐 나오지 않는다면, POT를 첫번째로 의심해 보는 것도 좋습니다.

[GLES20] 18. 텍스쳐 적용하기 - 2D Texture

2012.01.18 00:41



텍스쳐 적용하기 - 2D Texture

아마도 이글을 읽는 분들의 경우, 기본적으로 OpenGL 또는 OpenGL ES 1.x에서 텍스쳐 구현을 해보셨을 것이라 생각 됩니다.
OpenGL 책을 사면 텍스쳐에 대한 설명은 필수 적이기 때문에 관련 책을 읽어 보는 것을 추천합니다.
바른생활님의 설명하신 "[iOS GLView 만들기]13. Texturing" 장에서도 설명이 잘 되어 있습니다.
All about OpenGL ES 2.x □ (part 2/3)
(http://db-in.com/blog/2011/02/all-about-opengl-es-2-x-part-23/)

여기서는 OpenGL ES 2.0에서 2D Texture 을 사용하기 위해서는 어떻게 처리해야 하는지에 대해서 좀더 집중하도록 하겠습니다.

1. Texture 그리기전 준비사항

텍스쳐의 종류에는 2D Texture, Multi Texture, Cubemap Texture, Bumpmap Texture 등등 다양한 방식이 있습니다.
(3D Texture도 있지만 이는 GL_TEXTURE_3D_OES 인 extension 임으로 생략...)
일반적으로 가장 많이 사용하는 2D Texture에 대해서 알아 보도록 하겠습니다.

텍스쳐를 그리기 위해서는 일차적으로는 해당 이미지 파일을 로딩한 후 buffer array로 변경해야 하며,
그 다음에는 Texture ID를 생성 한후 이 ID에 buffer array를 저장해야 합니다.

텍스쳐 ID 생성 및 이미지 RGBA 버퍼 저장

```
GLuint RenderingEngine::createTexture(const string& file, bool bPot, int wrapMode)
{
  GLuint tex;
  glGenTextures(1, &tex);
  glBindTexture(GL_TEXTURE_2D, tex);

  loadImage(m_resourceManager->LoadImage(file));

  GLenum minFilter = (bPot ? GL_LINEAR : GL_LINEAR_MIPMAP_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, minFilter);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

  if (bPot) {
    if (wrapMode == 1) {
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    } else {
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    }
  } else {
    //glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP, GL_TRUE);
    glGenerateMipmap(GL_TEXTURE_2D);
  }
  return tex;
}
```

코드의 구성은 OpenGL ES 1.x와 유사합니다.

(1) glGenTexture() 함수를 이용해서 Texture ID를 생성하고,
(2) glBindTexture() 함수를 이용해서 생성한 Texture ID를 방인딩하고,
(3) glTexImage2D() 함수를 이용해서 "16장 " 에서 얻은 RGBA byte array을 저장하고,
(4) glTexParameteri() 함수를 이용해서 축소필터 GL_TEXTURE_MIN_FILTER 과 확대 필터 GL_TEXTURE_MAG_FILTER 를 설정할 수 있습니다.
(5) glTexParameteri() 함수를 이용해서 GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T 를 이용해서 Vertex 점정의 경계 지점을 부드럽게 보간 시켜주는 GL_CLAMP_TO_EDGE 와 텍스쳐를 계속 반복 시킬 GL_REPEAT를 설정할 수 있습니다.
밉맵을 사용할 경우에는 glGenerateMipmap(GL_TEXTURE_2D) 를 이용해 주고요

축소 / 확대 필터

GL_NEAREST : 인접 축소 필터로, 가장 근접한 텍셀이 색상을 사용한다. 단순하고 거칠게 표현된다.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

GL_LINEAR : 양방향 선형(bilinear) 필터링 알고리즘으로, 텍셀에서 인접한 2x2 점을 샘플링하여 가중치 평균값을 구한다.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

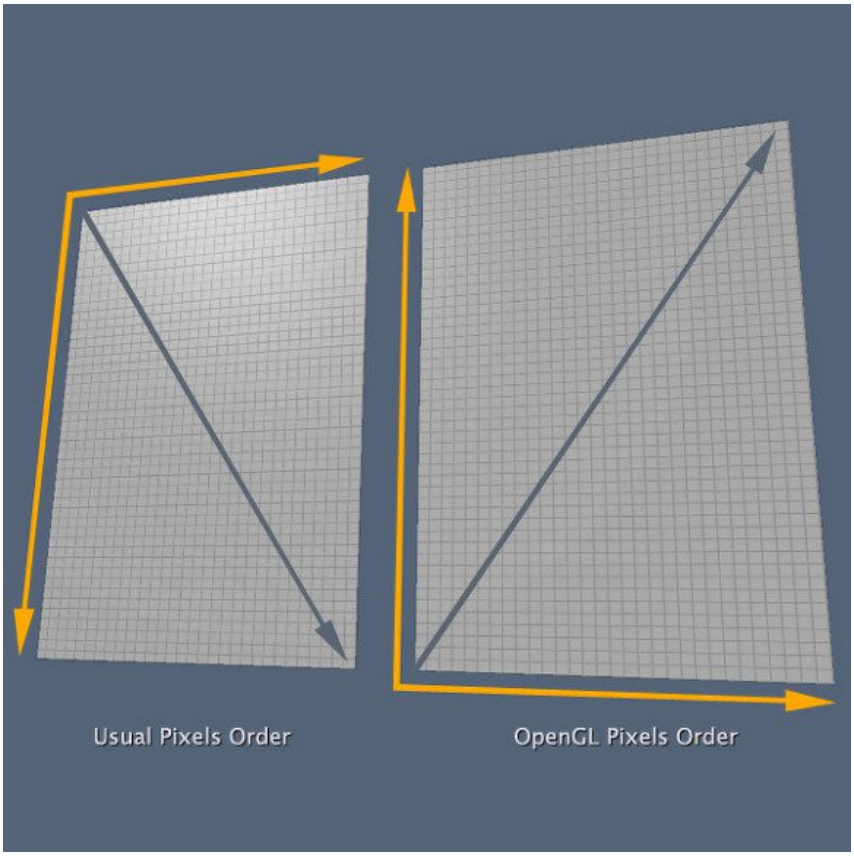
GL_LINEAR_MIPMAP_LINEAR : 삼중 선형(TRILINEAR) 필터링으로 가장 근접한 밉맵 두개를 찾아서 각각 GL_LINEAR 필터링 한한 결과 값을 섞는다. OpenGL은 이 필터링을 위해서 8개의 샘플을 추출하므로, 가장 높은 품질의 값을 생성하는 필터이다.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
밉맵은 축소필터에만 영향을 미치며, OpenGL ES 2.0에서 사용할 경우에는 자동 밉맵 함수인
glGenerateMipmap(GL_TEXTURE_2D); 이용해서 처리 가능하다.

이제 Texture ID를 생성하였음으로, 생성한 Texture ID를 이용해서 화면에 어떻게 렌더링 하는지에 대해서 알아보도록 하겠습니다.

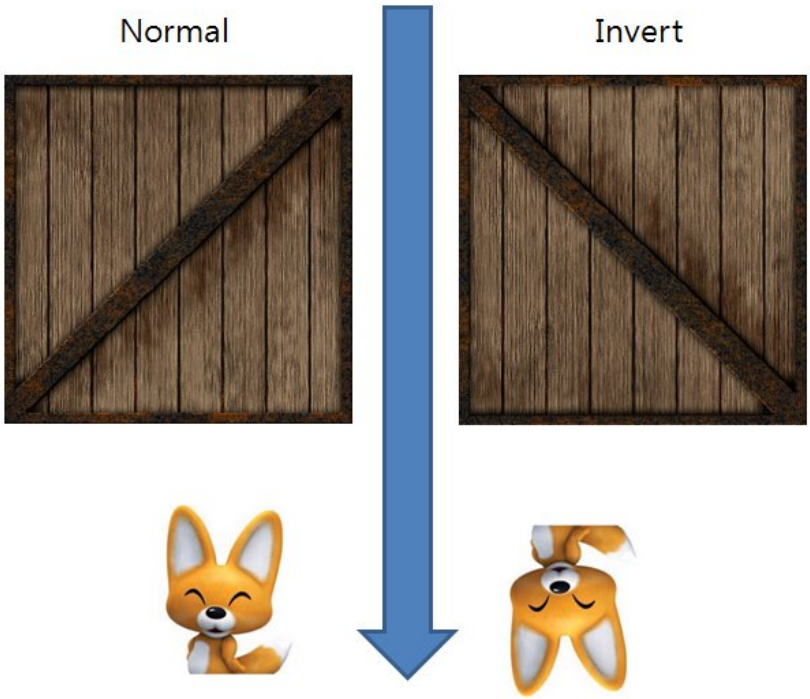
2. 2D Texture

앞장까지의 강좌에서는 Vertex Coordination(정점 좌표)와 Normal Coordination(법선 좌표)를 다뤘었습니다.
정점 좌표를 이용해서 삼각형, 큐브, 원뿔, 구, 뫼뷔우스 띠 등과 같은 Geometry(기하)를 그렸으며,
법선 좌표를 이용해서 삼각형의 각 면등에 조명(Light) 효과를 보여줬었습니다.
마찬가지로 텍스쳐를 Geometry에 표현하기 위해서는 텍스쳐 좌표가 필요합니다.
(텍스쳐 좌표를 Texture Element라 부르며 짧게 텍셀(texel)이라고 합니다.)

OpenGL에서 texel은 기존 pixel과 다릅니다.
다음의 그림을 보시면, 픽셀은 Left-Top이 원점(base point)입니다.
하지만, Open GL에서의 원점은 Left-Bottom 입니다.
(원점을 Anchor point 라고도 부르기도 합니다.)



상자와 에디 이미지를 예로 들어 보면 정상(normal) 이미지(pixel 순서)의 경우 왼쪽과 같으며, 이 이미지를 OpenGL에 저장하면, 오른쪽과 같이 뒤집혀서 들어가기 때문에 OpenGL로 화면을 그리면 뒤집혀 있게 됩니다.



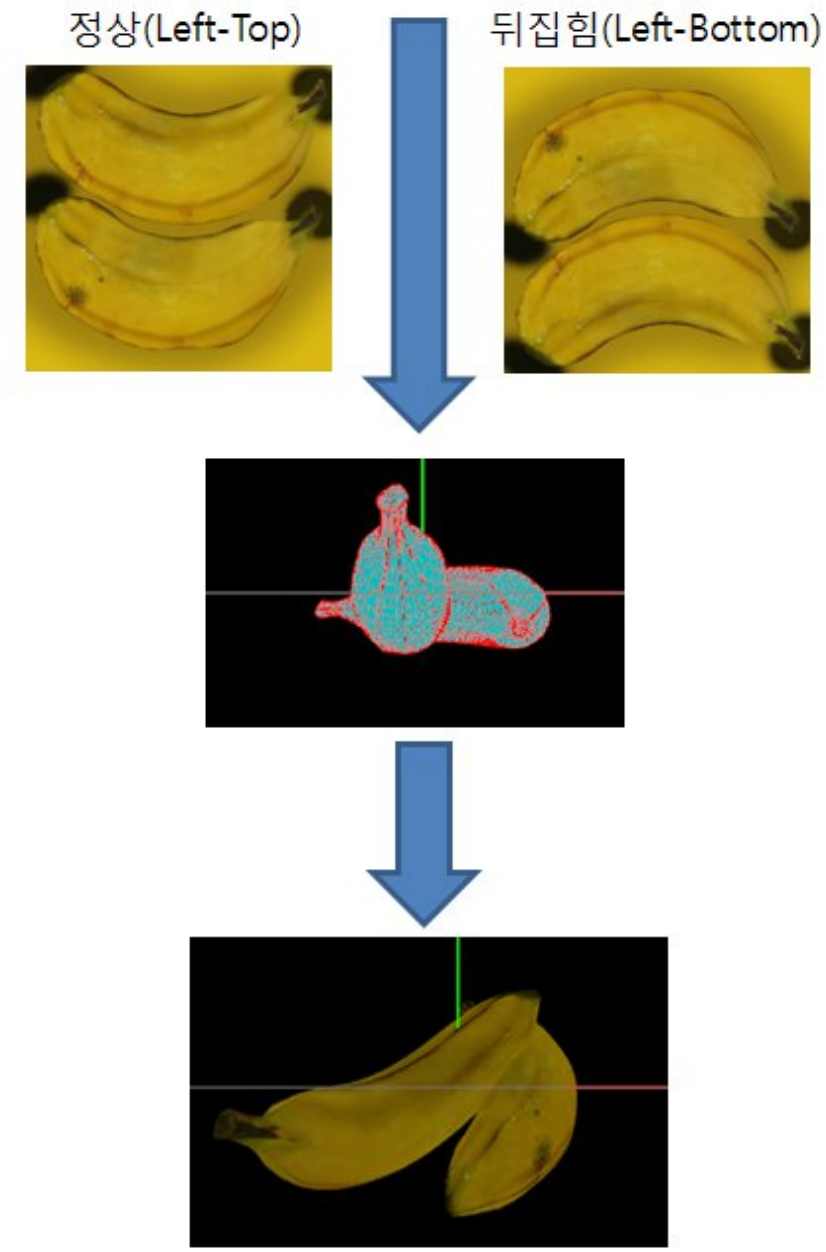
해결 방법은 여러가지가 있습니다.

(1) 이미지 자체를 수직(세로) 방향으로 뒤집어서 저장해 둔다.
(2) 이미지를 읽은 후 Affine coordination(아핀 좌표계) 를 api 등을 이용해서 뒤집어서 OpenGL에 전달한다.
(3) OpenGL에서 이용할 texel의 y 좌표에 -1.0f 를 곱해 줌으로서 항상 뒤집어 지게 한다.
(4) Vertex Shader에서 vertex position에 해당하는 값("gl_Vertex.y" to "a_position.y")에 -1.0f를 곱해 줌으로서
항상 뒤집어 지게 한다. ("a_position"은 제가 이용하는 attribute vec4 a_position; 변수로서 변수명은 바꿔서 사용해도 됩니다. ^^;)

여기서 주의할 점은 (3)번과 (4)번의 경우 Render To Texture를 이용해서 화면을 표현해 줄때 RTT한 이미지가 Left-Top으로 그려진
상태에서 텍셀의 y축에 음수가 곱해져서 Left-Bottom 순서로 다시 뒤집어지는 문제가 발생할 수도 있음으로,
제 개인적인 의견으로는 (1)번과 (2)번 방법이 적당할 것 같습니다.

3. Draw Banana

이제 바나나를 기준으로 실제 Shader을 이용해서 Texture Rendering 하는 부분에 대해서 보겠습니다.
렌더링의 절차를 그림으로 표현하면 다음과 같습니다.



(소스코드 분석에 앞서서 다음 장에서 다룰 MultiTexture와 혼합되어 있기 때문에 일반적인 방법과는 좀 다를 수 있습니다.)

Classes/RenderingEngine.CH03.ES20.cpp

렌더링 하기전 초기화 부분 정리

```
void RenderingEngine::Initialize(const vector<ISurface*>& surfaces)
{
...
  m_textures.BANANA = createTexture(TextureFiles[BANANA], true);  // ---- (1)
```

```
    // Initialize OpenGL ES STATE
    Initialize(width, height);  // ------------------- (4)
...
}

// ------------------- (2)
GLuint RenderingEngine::createTexture(const string& file, bool bPot, int wrapMode)
{
  GLuint tex;
  glGenTextures(1, &tex);
  glBindTexture(GL_TEXTURE_2D, tex);
  loadImage(m_resourceManager->LoadImage(file));  // ------------------- (3)
  GLenum minFilter = (bPot ? GL_LINEAR : GL_LINEAR_MIPMAP_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, minFilter);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  if (bPot) {
    if (wrapMode == 1) {
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    } else {
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    }
  } else {
    //glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP, GL_TRUE);
    glGenerateMipmap(GL_TEXTURE_2D);
  }
  return tex;
}
```

---

(1) : 바나나 텍스쳐 생성

createTexture() 함수를 이용해서 "banana.jpg" 파일을 읽은 후 Texture ID를 생성합니다.

이 생성한 Texture ID로 바나나를 3D로 렌더링 하기 전에 glBindTexture(GL_TEXTURE_2D, tex_id) 를 해주면 됩니다.

(렌더링 하기 전이란.. glDrawArrays나 glDrawElements 를 호출하기 전 이라고 보면 될것 같습니다.)

(2) : createTexture() 함수

위에서도 설명했었지만 텍스쳐 파일을 로딩한 후 Texture ID를 생성하는 함수 입니다.

주의할 실 점은 바른생활님 블로그에도 언급되었다 싶이, 밉맵 사용시에는 이미지 파일을 로딩한 후

OpenGL에 저장하는 함수인 glTexImage2D() 함수 처리를 앞에서 먼저 해 줘야 정상적으로 동작합니다.

(3) : loadImage() 함수

파일을 로딩하고 array buffer를 glTexImage2D()함수를 이용해서 OpenGL에 저장하는 역활을 합니다.

(4) : Initialize() 함수

OpenGL ES20과 관련한 VERTEX SHADER, FRAGMENT SHADER STATE를 초기화는 부분입니다.
이전 장과 다른 점은 texture를 사용할 것임으로 다음과 같이 두가지가 추가 됩니다.
VERTEX SHADER에 선언된 texel 과 관련한 attribute 형인 "attribute vec2 a_textureCoordIn;"와 연결되는
m_pixelLight.Attributes.TextureCoord = glGetAttribLocation(program, "a_textureCoordIn");
부분과 FRAGMENT SHADER에 선언된 sampler2D 형인 "uniform sampler2D Sampler0;"와 연결되는
glUniform1i(m_pixelLight.Uniforms.Sampler0, 0);
를 추가 되었습니다.

(Texture를 다룰 때 sampler2D형인 Sampler0에 0값으로 초기화 해주지 않는 책이나 코드가 간혹 있습니다.
Single Texture일 경우에는 0으로 초기화 해주지 않아도 정상적으로 동작은 합니다.
하지만 Multi Texture 까지 고려해서 프로그램을 짤 것임으로 Sampler0에는 0으로 초기화 해주는 것이 좋습니다.)

Classes/RenderingEngine.CH03.ES20.cpp

렌더링 부분 정리

```
void RenderingEngine::Render(const vector<Visual>& visuals)
{
...


  // PixelLight
  bool blsPixelLight = true;     // ---- (1)
  ProgramHandles handler;
  handler = m_pixelLight;
  glUseProgram(handler.Program);

  // Initialize various state.
  glEnableVertexAttribArray(handler.Attributes.Position);
  glEnableVertexAttribArray(handler.Attributes.Normal);
  glEnableVertexAttribArray(handler.Attributes.TextureCoord);    // ---- (2)

...
```

```
    // Set the Texture Mode
    if (bIsPixelLight) {
      glActiveTexture(GL_TEXTURE0 + TEXTURE_0);
      if (visualIndex == 9) {
        glBindTexture(GL_TEXTURE_2D, m_textures.BANANA);    // --------- (3)
        glUniform1i(handler.Uniforms.Sampler0, TEXTURE_0);
      } else {
        glBindTexture(GL_TEXTURE_2D, m_textures.CRATE);
        glUniform1i(handler.Uniforms.Sampler0, TEXTURE_0);
      }
    }

...

  // Draw the surface.
  int stride = 2 * sizeof(vec3) + sizeof(vec2);    // --------- (4)
  GLvoid* offset = (GLvoid*) sizeof(vec3);       // --------- (5)
  GLint position = handler.Attributes.Position;
  GLint normal = handler.Attributes.Normal;
  GLint texture = handler.Attributes.TextureCoord;  // --------- (6)
  vec4 white(1, 1, 1, 1);
  if (drawable.IndexCount < 0) {
    glBindBuffer(GL_ARRAY_BUFFER, drawable.VertexBuffer);
    glVertexAttribPointer(position, 3, GL_FLOAT, GL_FALSE, stride, 0);
    glVertexAttribPointer(normal, 3, GL_FLOAT, GL_FALSE, stride, offset);
    offset = (GLvoid*) (sizeof(vec3) * 2);              // --------- (7)
    glVertexAttribPointer(texture, 2, GL_FLOAT, GL_FALSE, stride, offset);  // --------- (8)
    glDrawArrays(GL_TRIANGLES, 0, drawable.VertexCount);  // --------- (9)
  } else {
    // --------- (10)
    glBindBuffer(GL_ARRAY_BUFFER, drawable.VertexBuffer);
    glVertexAttribPointer(position, 3, GL_FLOAT, GL_FALSE, stride, 0);
    glVertexAttribPointer(normal, 3, GL_FLOAT, GL_FALSE, stride, offset);
    offset = (GLvoid*) (sizeof(vec3) * 2);
    glVertexAttribPointer(texture, 2, GL_FLOAT, GL_FALSE, stride, offset);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, drawable.IndexBuffer);
    glDrawElements(GL_TRIANGLES, drawable.IndexCount, GL_UNSIGNED_SHORT, 0);
  }

  glDisableVertexAttribArray(handler.Attributes.Position);
  glDisableVertexAttribArray(handler.Attributes.Normal);
  glDisableVertexAttribArray(handler.Attributes.TextureCoord);  // --------- (11)
  // VBO을 UnBind 해야 VBA와 충돌이 없다.  // --------- (12)
  glBindBuffer(GL_ARRAY_BUFFER, 0);
```

```
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
  /* Flush all drawings */
  glFinish();
  error = checkGLError();

  if (!m_spinning) {
    xrot += xspeed; /* Add xspeed To xrot */
    yrot += yspeed; /* Add yspeed To yrot */
  }
  glBindTexture(GL_TEXTURE_2D, 0);          // ---------- (13)
  glUseProgram(0);                           // ---------- (14)

}
```

---

(1) : bIsPixelLight 변수 설정합니다.

```
bool bIsPixelLight = true;
handler = m_pixelLight;
glUseProgram(handler.Program);
```

(2) Texture Coordination Array 를 활성화 해줍니다.

"glEnableVertexAttribArray(handler.Attributes.TextureCoord);"

기존 GLES1.1 기준으로는 "glEnableClientState(GL_TEXTURE_COORD_ARRAY);" 함수를 대체하는 부분입니다.

활성화와 반대되는 비 활성화는 (11)번 함수" glDisableVertexAttribArray(handler.Attributes.TextureCoord);"가 쌍으로 있어야 합니다.

(3) Banana texture id를 바인딩 해줍니다.

생성한 텍스쳐를 3D 렌더링 객체에 적용하기 위해서는 해당 texture id를 바인딩 해줘야 합니다.

```
glBindTexture(GL_TEXTURE_2D, m_textures.BANANA);
glUniform1i(handler.Uniforms.Sampler0, TEXTURE_0);
```

(Sampler0에 대해서 0번으로 재 설정 해주는 부분은 없어도 상관은 없지만, 명시적 표현으로 나뒀습니다.)

(간혹 Shader에 정의된 sampler2d 변수에 glGenTexture() 함수를 이용해서 생성한 texture id에 -1를 한 후 넣어주는 코드가 있는 경우가 있습니다. 이는 glGen()시에 처음 생성된 ID가 1이기 때문에 그렇게 하는 경우입니다. 문제는 Galaxy Tab과 같은 PowerVR GPU의 경우, 처음 생성시 1이 아닌 3xxxxx 번 등과 같은 높은 숫자가 나오는 경우가 있습니다. 그럼으로 명시적으로 그냥 첫 번째 sampler2d 변수에는 0을 두 번째는 1를 넣어주는 것이 좋을 것 같습니다.)

(4) Vertex, Normal, Texture coord 에 대한 memory chunk 영역을 설정 해 줍니다.

banana.obj를 해석한 후 Vertex Buffer Object(VBO)에 저장해 놓았음으로, OpenGL에게 메모리가 한 묶음에 해당하는 크기를 알려줘야 합니다. 이 역할을 하는 부분이 stride 입니다.

```
int stride = 2 * sizeof(vec3) + sizeof(vec2);
```

"VertexCoord(12) + NormalCoord(12) + TextureCoord(8) = 32" 가 됩니다.
OpenGL ES 2.0 Programming Guide 책에서는 이부분을 Array Of Structures(Page 104) 라고 표현 합니다.

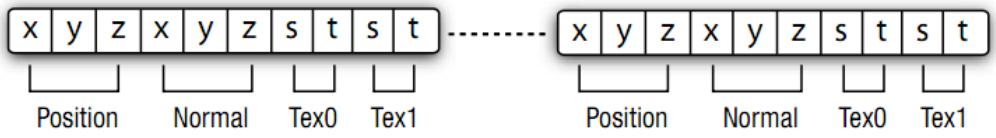


Figure 6-2 Position, Normal, and Two Texture Coordinates Stored As an Array

(5) Array Of Structures 에서 Normal Coord에 대한 offset 을 설정 합니다.
GLvoid* offset = (GLvoid*) sizeof(vec3);
VertexCoord 에 대한 크기 만큼 이동시켜줄 것임으로 12 bytes 입니다.
전달은 GLvoid* 형태로 캐스팅해서 주소로 넘겨 줍니다.

(6) VERTEX SHADER내의 Texture Attrbute와 연결된 ID 입니다.
GLint texture = handler.Attributes.TextureCoord;
PixelLighting.vert 내에 선언된 "a_attribute vec2 a_textureCoordIn;" 연결 가능한 Attribute ID 입니다.
이 ID를 이용해서 (8)번 항목에서 Vertex Attribute Pointer 등록 시 사용하게 됩니다.

(7) Array Of Structures 에서 Texture Coord에 대한 offset 을 설정 합니다.
offset = (GLvoid*) (sizeof(vec3) * 2);
VertexCoord 와 NormalCoord에 대한 크기 만큼 이동시켜줄 것임으로 24 bytes 입니다.
전달은 GLvoid* 형태로 캐스팅해서 주소로 넘겨 줍니다.

(8) 텍셀 정보를 저장 합니다.
텍셀 좌표에 대한 정보를 "glVertexAttribPointer(texture, 2, GL_FLOAT, GL_FALSE, stride, offset);"
함수를 이용해서 저장합니다.
기존 GLES1.1 기준으로는 "glTexCoordPointer(2, GL_FLOAT, 0, texcoords);"함수를 대체하는 부분입니다.

(9) Vertices 방식으로 렌더링을 합니다.
glDrawArrays(GL_TRIANGLES, 0, drawable.VertexCount);
banana.obj는 glDrawArray 방식으로 표출되게금 header 형태로 만들었기 때문에 이 부분에서 그려지게 됩니다.

(10) Indices 방식으로 렌더링을 합니다.
glDrawElements(GL_TRIANGLES, drawable.IndexCount, GL_UNSIGNED_SHORT, 0);
ParametricSurface 로 만들어진 Geometry 들은 Indices 로 만들어 졌음으로 이 부분에서 그려지게 됩니다.

(11) Texture Coordination Array 를 비 활성화 해줍니다.
"glDisableVertexAttribArray(handler.Attributes.TextureCoord);"

함수는 기존 GLES1.1 기준으로는 "glDisableClientState(GL_TEXTURE_COORD_ARRAY);" 함수를 대체하는 부분입니다.
비 활성화와 반대되는 활성화는 (2)번 함수"glEnableVertexAttribArray(handler.Attributes.TextureCoord);"가
쌍으로 있어야 합니다.

(12) VBO 객체를 UnBind 합니다.
Vertex Buffer Array 형태로 그려지는 렌더러와 충돌 하지 않도록 VBO를 해제해 줍니다.

(13) 바인딩 했던 텍스쳐를 해제해 줍니다.
glBindTexture(GL_TEXTURE_2D, 0);

(14) PixelLighting 쉐이더를 해제해 줍니다.
glUseProgram(0);

4. 좀더 파고들기

추가적으로 이미지 뒤집기 기능을 좀더 정리해 보도록 하겠습니다.

이미지 뒤집기 API 정리

앞에서 언급했지만, 이미지 파일를 로딩한 후 raw data를 뒤집을 필요가 있습니다.

이때 이용하는 것이 Affine Transformation API 입니다.

일반적으로 아이폰에서는 다음과 같이 적용하면 됩니다.

```
TextureDescription LoadImage(const string& file)
{
    NSString* basePath = [NSString stringWithUTF8String:file.c_str()];
    NSString* resourcePath = [[NSBundle mainBundle] resourcePath];
    NSString* fullPath = [resourcePath stringByAppendingPathComponent:basePath];
    UIImage* uiImage = [UIImage imageWithContentsOfFile:fullPath];
    TextureDescription description;
    description.Size.x = CGImageGetWidth(uiImage.CGImage);
    description.Size.y = CGImageGetHeight(uiImage.CGImage);
    description.BitsPerComponent = 8;
    description.Format = TextureFormatRgba;
    description.MipCount = 1;
    int bpp = description.BitsPerComponent / 2;
    int byteCount = description.Size.x * description.Size.y * bpp;
    unsigned char* data = (unsigned char*)calloc(byteCount, 1);

    CGAffineTransform     transform  = CGAffineTransformIdentity;
    //if (flipImage)
    {
      transform = CGAffineTransformTranslate(transform, 0,  description.Size.y);
      transform = CGAffineTransformScale(transform, 1.0, -1.0);
    }
    CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
    CGBitmapInfo bitmapInfo = kCGImageAlphaPremultipliedLast | kCGBitmapByteOrder32Big;
    CGContextRef context = CGBitmapContextCreate(data,
        description.Size.x, description.Size.y, description.BitsPerComponent, bpp * description.Size.x,
        colorSpace, bitmapInfo);
    CGContextConcatCTM(context, transform);
    CGColorSpaceRelease(colorSpace);
    CGRect rect = CGRectMake(0, 0, description.Size.x, description.Size.y);
    CGContextDrawImage(context, rect, uiImage.CGImage);
    CGContextRelease(context);

    m_imageData = [NSData dataWithBytesNoCopy:data length:byteCount freeWhenDone:YES];
    return description;
}
```

안드로이드에서는 다음과 같이 적용할 수 있습니다.

com.myastro.gles.ResourceManager.java

```
public ResourceManager(Context context, int res)
{
    // Get the texture from the Android resource directory
    Bitmap bitmap = null;
    InputStream is = context.getResources().openRawResource(res);
    try {
      // BitmapFactory is an Android graphics utility for images
      bitmap = BitmapFactory.decodeStream(is);
    } finally {
      //Always clear and close
      try {
       is.close();
       is = null;
      } catch (IOException e) {
     }
  }
  // 비트맵 변환을 위한 행렬을 만든다.
  Matrix mirrorMatrix = new Matrix();
  // 행렬 좌표를 Y축으로 뒤집는다.
  mirrorMatrix.preScale(1.0f, -1.0f);
  // 생성된 변환행렬을 이용해서 새로 bitmap을 생성한다. 속도는???
  Bitmap tranformedBitmap = Bitmap.createBitmap(bitmap, 0, 0, bitmap.getWidth(), bitmap.getHeight(), mirrorMatrix,
false);
  bitmap.recycle();
  if (tranformedBitmap != null) {
    initBitmap(tranformedBitmap);
  }
}
```

5. 결론

텍스쳐 적용하기를 다루면서 어떻게 정리할까 고민했는데, 대략적인 정리가 된것 같네요.

텍스쳐 장은 책에서 이미 많이 소개하고 있는 부분이어서 OpenGL 책을 보시는게 이해가 더 빠를 수도 있습니다.
여기에 정리된 사항은 책을 보고 만들어 보시고 이런 방식으로 ES2.0으로 적용도 가능 하다는 관점으로 보시면 좋을 것 같습니다.
또한 제가 정리한 방식은 최적화 되어 있지 않음으로 참조만 하시기 바랍니다. ^^;

OGLES20 Template Application ported to mfc and android

http://code.google.com/p/myastro/downloads/list
http://code.google.com/p/myastro/downloads/detail?
name=OGLES20Application.release.v.1.3.0.zip&can=2&q=#makechanges

OpenGL ES2.0 기초 강좌

블로그 나의별 http://blog.daum.net/aero2k

저자 나의별

발행일 2012.03.25 20:26:37

Daum 블로그