Lecture 18

- Uniform Color
- Generic Attribute
- Texture (sampler2D)
- Plain 3D Renderer
- Rainbow Color based on the vertex position
- “discard” statement in the fragment shader
- Gouraud Shading
- Phong Shading

Uniform color

- If all primitives must be drawn in the same color, the color can be made a *uniform*.
- A renderer should cache an identifier of a uniform so that the C++ program can send an information to the GLSL program.

- In renderer.h, add:

```
class UniformColorRenderer : public RendererBase
{
public:
   GLuint attribVertexPos;
   GLuint uniformColorPos;
   virtual void CacheAttributeAndUniformIdent(void);
};
```

- In renderer.cpp, add:

```
void UniformColorRenderer::CacheAttributeAndUniformIdent(void)
{
   attribVertexPos=glGetAttribLocation(programIdent,"vertex");
   printf("Attribute Vertex Position=%d\n",attribVertexPos);
   uniformColorPos=glGetUniformLocation(programIdent,"color");
   printf("Uniform Color Position=%d\n",uniformColorPos);
}
```

- svn-copy:
  - vertex_shader.glsl to uniform_color_vertex_shader.glsl
  - fragment_shader.glsl to uniform_color_fragment_shader.glsl

- No change in the vertex shader.
- The fragment shader should copy uniform color to the final output color. Therefore it needs to be:

```
uniform vec4 color;
void main()
{
    gl_FragColor=color;
}
```

- Add a member variable:

```
UniformColorRenderer uniformColor;
```

- In Initialize(),

```
 uniformColor.CompileFile(
        "uniform_color_vertex_shader.glsl",
        "uniform_color_fragment_shader.glsl");
```

- In Draw(),

```
void FsLazyWindowApplication::Draw(void)
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

    glUseProgram(uniformColor.programIdent);
    const GLfloat vtx[12]=
    {
        -1,-1,0,
         1,-1,0,
         1, 1,0,
        -1, 1,0
    };
    const GLfloat col[4]={0,1,0,1};
    glUniform4fv(uniformColor.uniformColorPos,1,col);
    glEnableVertexAttribArray(uniformColor.attribVertexPos);
    glVertexAttribPointer(uniformColor.attribVertexPos,3,GL_FLOAT,GL_FALSE,0,vtx);
    glDrawArrays(GL_TRIANGLE_FAN,0,4);
    glDisableVertexAttribArray(uniformColor.attribVertexPos);

    FsSwapBuffers();

    needRedraw=false;
}
```

Wait, do we need to create separate shader programs for uniform color and variable color?

- It will be a problem if we need to write separate shaders for:
  - Color as uniform vs. color as attribute.
  - Using texture vs. no texture.
  - Position offset as uniform vs. position offset as attribute.
  - !@#$ as uniform vs. !@#$ as attribute.
- For the exactly same rendering program, if you can think of N possible attribute, that can also be a uniform, the combination will be 2^N.

Generic Attribute

- Solution: Generic Attribute.
- Just like glUniform??, you can use glVertexAttrib??.
- The attribute value specified by glVertexAttribute?? will be applied to all vertices.
- Strange limitation (known bug): In many (even popular) GPUs, vertex attribute 0 cannot be generic.
- Use glBindAttribLocation to force an attribute to have a non-zero identifier.

- I let you study yourself.

Where is this zero identifier bug coming from?

- Once you could use glVertex??, glColor??, glNormal??, with a GLSL program.
- glVertex?? was passing a position to the GLSL program as a variable called gl_Vertex.
- gl_Vertex was always bound to an attribute-identifier of 0.
- gl_Vertex cannot be generic, while gl_Normal, gl_Color, gl_TeCoord could be generic.
- Some GPUs are still inheriting this limitation as a known bug.

Using texture.

- A bitmap texture is called as sampler2D.
- A sampler2D must be given as a uniform.
- You can sample a color within a shader program by using a function called *texture*.
- If you want to give a texture coordinate per vertex (just like glTexCoord2), a texture coordinate must be given as an attribute.

Add checker.cpp and checker.h

checker.h

```
#ifndef CHECKER_IS_INCLUDED
#define CHECKER_IS_INCLUDED

// Note: OpenGL ES only takes 2^n times 2^n texture.
const int checker_pattern_wid=8;
const int checker_pattern_hei=8;
extern const unsigned char checker_pattern[checker_pattern_wid*checker_pattern_hei*4];

#endif
```

checker.cpp

```
#include "checker.h"

const unsigned char checker_pattern[checker_pattern_wid*checker_pattern_hei*4]=
{
255,  0,  0,255,    0,255,  0,255,    0,  0,255,255,  255,255,  0,255,  255,255,255,255,    0,  0,  0,255,  255,255,255,255,    0,  0,  0,255,
255,255,  0,255,  255,  0,  0,255,    0,255,  0,255,    0,  0,255,255,    0,  0,  0,255,  255,255,255,255,    0,  0,  0,255,  255,255,255,255,
  0,  0,255,255,  255,255,  0,255,  255,  0,  0,255,    0,255,  0,255,  255,255,255,255,    0,  0,  0,255,  255,255,255,255,    0,  0,  0,255,
  0,255,  0,255,    0,  0,255,255,  255,255,  0,255,  255,  0,  0,255,    0,  0,  0,255,  255,255,255,255,    0,  0,  0,255,  255,255,255,255,
255,255,255,255,    0,  0,  0,255,  255,255,255,255,    0,  0,  0,255,  255,  0,  0,255,    0,255,  0,255,    0,  0,255,255,  255,255,  0,255,
  0,  0,  0,255,  255,255,255,255,    0,  0,  0,255,  255,255,255,255,  255,255,  0,255,  255,  0,  0,255,    0,255,  0,255,    0,  0,255,255,
255,255,255,255,    0,  0,  0,255,  255,255,255,255,    0,  0,  0,255,    0,  0,255,255,  255,255,  0,255,  255,  0,  0,255,    0,255,  0,255,
  0,  0,  0,255,  255,255,255,255,    0,  0,  0,255,  255,255,255,255,    0,255,  0,255,    0,  0,255,255,  255,255,  0,255,  255,  0,  0,255,
};
```

- Add a member variable:

```
GLuint texIdent;
```

- In Initialize(), make a texture as:

```
glGenTextures(1,&texIdent);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D,texIdent);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexImage2D(
   GL_TEXTURE_2D,0,GL_RGBA,checker_pattern_wid,checker_pattern_hei,
              0,GL_RGBA,GL_UNSIGNED_BYTE,checker_pattern);
```

- Add sampler2d_vertex_shader.glsl

```
attribute vec2 texCoord;
attribute vec3 vertex;

varying vec2 texCoordOut;

void main()
{
    gl_Position=vec4(vertex,1.0);
    texCoordOut=texCoord;
}
```

Just pass incoming texCoord to texCoordOut

- Add sampler2d_fragment_shader.glsl

```
uniform sampler2D texIdent;
varying vec2 texCoordOut;

void main()
{
    gl_FragColor=texture2D(texIdent,texCoordOut);
}
```

A texture identifier.

This is how a texture can be sampled.

Note about texture2D function

- The function texture2D is deprecated at GLSL 1.3, and has been replaced with texture.
- Older GPUs don’t understand newer version GLSL.
- Obviously, GLSL is not as well designed as C/C++. Rather, I say it is a very poorly designed programming language.
- Their excuse is that graphics technology is growing rapidly.

Note about texture2D function

- So far, I haven’t seen any GPUs that does not recognize texture2D.
- If a GPU does not recognize it, you will need to put:

  ```
  #version 120
  ```

  and keep using texture2D, or change textu2D to texture.
- At this point, GLSL for OpenGL ES also only recognizes texture2D (tested on my Mac mini, first-generation iPad mini, and iPhone 5S).

- In renderer.h, add:

```
class Sampler2dRenderer : public RendererBase
{
public:
   GLuint attribVertexPos;
   GLuint attribTexCoordPos;
   GLuint uniformTexIdentPos;
   virtual void CacheAttributeAndUniformIdent(void);
};
```

- In renderer.cpp, add:

```
void Sampler2dRenderer::CacheAttributeAndUniformIdent(void)
{
   attribVertexPos=glGetAttribLocation(programIdent,"vertex");
   printf("Attribute Vertex Position=%d\n",attribVertexPos);
   attribTexCoordPos=glGetAttribLocation(programIdent,"texCoord");
   printf("Attribute TexCoord Position=%d\n",attribTexCoordPos);
   uniformTexIdentPos=glGetUniformLocation(programIdent,"texIdent");
   printf("Uniform TexIdent Position=%d\n",uniformTexIdentPos);
}
```

- Add a member variable:

```
Sampler2dRenderer sampler2d;
```

- In Initialize(),

```
sampler2d.CompileFile(
   "sampler2d_vertex_shader.glsl",
   "sampler2d_fragment_shader.glsl");
```

- In Draw(),

```
void FsLazyWindowApplication::Draw(void)
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

    glUseProgram(sampler2d.programIdent);
    const GLfloat vtx[12]=
    {
        -1,-1,0,
         1,-1,0,
         1, 1,0,
        -1, 1,0
    };
    const GLfloat texCoord[8]=
    {
        0,0, 0,1, 1,1, 1,0
    };

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D,texIdent);

    // GL_TEXTURE0 -> 0.  Don't use GL_TEXTURE0! (Frequent confusion).
    glUniform1i(sampler2d.uniformTexIdentPos,0);
    glEnableVertexAttribArray(sampler2d.attribVertexPos);
    glVertexAttribPointer(sampler2d.attribVertexPos,3,GL_FLOAT,GL_FALSE,0,vtx);
    glEnableVertexAttribArray(sampler2d.attribTexCoordPos);
    glVertexAttribPointer(sampler2d.attribTexCoordPos,2,GL_FLOAT,GL_FALSE,0,texCoord);
    glDrawArrays(GL_TRIANGLE_FAN,0,4);
    glDisableVertexAttribArray(sampler2d.attribVertexPos);
    glDisableVertexAttribArray(sampler2d.attribTexCoordPos);

    FsSwapBuffers();

    needRedraw=false;
}
```

Setting up texture number 0.
In this case, making texture number 0 as a 2D texture, which is identified by texIdent.

Give 0 to use GL_TEXTURE0. What’s confusing is GL_TEXTURE0 is not actually numeric value of 0 in OpenGL ES. To make it work both in ES and full-spec OpenGL, use 0.

Plain 3D renderer

- Vertex attributes
  - Position (vec3)
  - Color (vec4)
- Uniforms
  - Projection matrix (mat4)
  - ModelView matrix (mat4)
- Varying
  - Color (Just pass color attribute to color varying in the vertex shader, and use it as is in the fragment shader.)

From uniform_color example

- Add plain3d_vertex_shader.glsl as:

```
attribute vec3 vertex;
attribute vec4 color;

uniform mat4 projection,modelView;

varying vec4 colorOut;

void main()
{
   colorOut=color;
   gl_Position=projection*modelView*vec4(vertex,1.0);
}
```

- Add plain3d_fragment_shader.glsl as:

```
varying vec4 colorOut;
void main()
{
   gl_FragColor=colorOut;
}
```

- Add Plain3dRenderer class in renderer.h

```
class Plain3dRenderer : public RendererBase
{
public:
    GLuint attribVertexPos;
    GLuint attribColorPos;
    GLuint uniformProjectionPos;
    GLuint uniformModelViewPos;
    virtual void CacheAttributeAndUniformIdent(void);
};
```

- In renderer.cpp, implement CacheAttributeAndUniformIdent

```
void Plain3dRenderer::CacheAttributeAndUniformIdent(void)
{
    attribVertexPos=glGetAttribLocation(programIdent,"vertex");
    printf("Attribute Vertex Position=%d\n",attribVertexPos);
    attribColorPos=glGetAttribLocation(programIdent,"color");
    printf("Attribute Color Position=%d\n",attribColorPos);

    uniformProjectionPos=glGetUniformLocation(programIdent,"projection");
    printf("Uniform Projection Position=%d\n",uniformProjectionPos);
    uniformModelViewPos=glGetUniformLocation(programIdent,"modelView");
    printf("Uniform ModelView Position=%d\n",uniformModelViewPos);

}
```

- In Initialize function,

```
 plain3d.CompileFile(
    "plain3d_vertex_shader.glsl",
    "plain3d_fragment_shader.glsl");
```

- In CMakeLists.txt,

```
set(LIB_DEPENDENCY fslazywindow ysclass ysgl ysport geblkernel)
```

- Add headers:

```
#include <ysshellext.h>
#include <ysport.h>
#include <ysshellextio.h>
```

- Bunch of cutting & pasting from the STL viewer example
- Member variables and functions:

```
// Cut & Pasted from stl viewer >>
Ys3DDrawingEnvironment drawEnv;

int prevMx,prevMy;
YsShellExt shl;
std::vector <float> vtx,nom;
std::vector <float> col;
YsVec3 min,max;

void LoadModel(const char fn[]);
void CacheBoundingBox(void);
static void YsShellToVtxNom(
    std::vector <float> &vtx,
    std::vector <float> &nom,
    std::vector <float> &col,
    const YsShellExt &shl);
// Cut & Pasted from stl viewer <<
```

- Copy LoadModel, CacheBoundingBox, and YsShellToVtxNom functions
- (In STL reading, make default color as YsBlue())
- In the constructor of FsLazyWindowApplication,

```
drawEnv.SetProjectionMode(YsProjectionTransformation::PERSPECTIVE);
drawEnv.SetViewTarget(YsVec3::Origin());
drawEnv.SetViewDistance(20.0);
drawEnv.SetViewAttitude(YsAtt3(YsPi/4.0,YsPi/5.0,0.0));

prevMx=0;
prevMy=0;

needRedraw=false;
```

- In BeforeEveryting,

```
if(2<=argc)
{
    LoadModel(argv[1]);
}
```

- In Interval,

```
int wid,hei;
FsGetWindowSize(wid,hei);

int lb,mb,rb,mx,my;
auto evt=FsGetMouseEvent(lb,mb,rb,mx,my);
if(0!=lb && (mx!=prevMx || my!=prevMy))
{
   double denom=(double)YsGreater(wid,hei);
   double dx=2.0*(double)(prevMx-mx)/denom;
   double dy=2.0*(double)(prevMy-my)/denom;
   drawEnv.RotateView(dx,dy);
}
prevMx=mx;
prevMy=my;
```

- Finally, Draw function should look like:

```
int wid,hei;
FsGetWindowSize(wid,hei);

glClear(GL_COLOR_BUFFER_BITIGL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);

drawEnv.SetProjectionMode(YsProjectionTransformation::PERSPECTIVE);
drawEnv.SetAspectRatio((double)wid/(double)hei);
drawEnv.SetFOVY(YsPi/4.0);
drawEnv.SetNearFar(0.1,100.0);
drawEnv.SetViewTarget((min+max)/2.0);
drawEnv.SetViewDistance((max-min).GetLength()/2.0);

GLfloat projMat[16];
drawEnv.GetProjectionMatrix().GetOpenGlCompatibleMatrix(projMat);

GLfloat viewMat[16];
drawEnv.GetViewMatrix().GetOpenGlCompatibleMatrix(viewMat);

glUseProgram(plain3d.programIdent);
glUniformMatrix4fv(plain3d.uniformProjectionPos,1,GL_FALSE,projMat);
glUniformMatrix4fv(plain3d.uniformModelViewPos,1,GL_FALSE,viewMat);

glEnableVertexAttribArray(plain3d.attribVertexPos);
glVertexAttribPointer(plain3d.attribVertexPos,3,GL_FLOAT,GL_FALSE,0,vtx.data());
glEnableVertexAttribArray(plain3d.attribColorPos);
glVertexAttribPointer(plain3d.attribColorPos,4,GL_FLOAT,GL_FALSE,0,col.data());

glDrawArrays(GL_TRIANGLES,0,vtx.size()/3);

glDisableVertexAttribArray(plain3d.attribVertexPos);
glDisableVertexAttribArray(plain3d.attribColorPos);

FsSwapBuffers();
needRedraw=false;
```

Color based on the location

- Fragment shader can make the color as a function of raw-input xyz coordinate.

Shader programs

- rainbow3d_vertex_shader.glsl

```
attribute vec3 vertex;

uniform mat4 projection,modelView;

varying vec3 vertexOut;

void main()
{
   vertexOut=vertex;
   gl_Position=projection*modelView*vec4(vertex,1.0);
}
```

- rainbow3d_fragment_shader.glsl

```
varying vec3 vertexOut;
void main()
{
   float pi=3.1415927;
   gl_FragColor=vec4(sin(vertexOut.x),sin(pi/2.0+vertexOut.y),sin(pi+vertexOut.z),1.0);
}
```

Rainbow3dRenderer class

- In renderer.h

```
class Rainbow3dRenderer : public RendererBase
{
public:
    GLuint attribVertexPos;
    GLuint uniformProjectionPos;
    GLuint uniformModelViewPos;
    virtual void CacheAttributeAndUniformIdent(void);
};
```

- In renderer.cpp

```
void Rainbow3dRenderer::CacheAttributeAndUniformIdent(void)
{
    attribVertexPos=glGetAttribLocation(programIdent,"vertex");
    printf("Attribute Vertex Position=%d\n",attribVertexPos);

    uniformProjectionPos=glGetUniformLocation(programIdent,"projection");
    printf("Uniform Projection Position=%d\n",uniformProjectionPos);
    uniformModelViewPos=glGetUniformLocation(programIdent,"modelView");
    printf("Uniform ModelView Position=%d\n",uniformModelViewPos);
}
```

- In Initialize()

```
rainbow3d.CompileFile(
    "rainbow3d_vertex_shader.glsl",
    "rainbow3d_fragment_shader.glsl");
```

Experiment

- What if I do:

```
if(length(vertexOut)<3.0)
{
   discard;
}
```

in the fragment shader?

- You can stop OpenGL from writing a color to the frame buffer by discarding a fragment.

Gouraud Shading

- Light intensity is calculated per vertex and interpolated in the screen coordinate.
- Three light components (reflections):
  - Ambient light
  - Diffuse reflection
  - Specular reflection

Gouraud Shading

- Additional vertex attribute:
  - Normal vector
- Additional uniform:
  - Direction to the light in the camera coordinate system.

Gouraud Shading

- Vertex Shader

```
attribute vec3 vertex;
attribute vec3 normal;
attribute vec4 color;
uniform mat4 projection,modelView;
uniform vec3 lightDir;
uniform float ambient;
uniform float specularIntensity;
uniform float specularExponent;
varying vec4 colorOut;

void main()
{
   vec3 nom=normalize((modelView*vec4(normal,0.0)).xyz);
   vec3 lit=normalize(lightDir);

   float diffuse=dot(nom,lit);

   vec4 posInView=modelView*vec4(vertex,1.0);
   vec3 viewDir=-normalize(posInView.xyz);
   vec3 midDir=normalize(viewDir+lightDir);
   float specular=specularIntensity*pow(dot(midDir,nom),specularExponent);

   colorOut=vec4(color.rgb*(ambient+diffuse),color.a)
         +vec4(specular,specular,specular,0.0);

   gl_Position=projection*modelView*vec4(vertex,1.0);
}
```

Gouraud Shading

- Fragment Shader - Since all calculations are done in the vertex shader, the fragment shader just need to pass color value from left to right.

```
varying vec4 colorOut;
void main()
{
    gl_FragColor=colorOut;
}
```

Gouraud Shading

- In renderer.h

```
class Gouraud3dRenderer : public RendererBase
{
public:
   GLuint attribVertexPos;
   GLuint attribNormalPos;
   GLuint attribColorPos;

   GLuint uniformProjectionPos;
   GLuint uniformModelViewPos;
   GLuint uniformLightDirPos;
   GLuint uniformAmbientPos;
   GLuint uniformSpecularIntensityPos;
   GLuint uniformSpecularExponentPos;

   virtual void CacheAttributeAndUniformIdent(void);
};
```

Gouraud Shading

- In renderer.cpp

```
void Gouraud3dRenderer::CacheAttributeAndUniformIdent(void)
{
    attribVertexPos=glGetAttribLocation(programIdent,"vertex");
    attribNormalPos=glGetAttribLocation(programIdent,"normal");
    attribColorPos=glGetAttribLocation(programIdent,"color");

    uniformProjectionPos=glGetUniformLocation(programIdent,"projection");
    uniformModelViewPos=glGetUniformLocation(programIdent,"modelView");
    uniformLightDirPos=glGetUniformLocation(programIdent,"lightDir");
    uniformAmbientPos=glGetUniformLocation(programIdent,"ambient");
    uniformSpecularIntensityPos=glGetUniformLocation(programIdent,"specularIntensity");
    uniformSpecularExponentPos=glGetUniformLocation(programIdent,"specularExponent");
}
```

Gouraud Shading

- In Initialize() in main.cpp

```
gouraud3d.CompileFile(
    "gouraud_vertex_shader.glsl",
    "gouraud_fragment_shader.glsl");
```

Gouraud Shading

- In Draw function:

```
glUseProgram(gouraud3d.programIdent);
glUniformMatrix4fv(gouraud3d.uniformProjectionPos,1,GL_FALSE,projMat);
glUniformMatrix4fv(gouraud3d.uniformModelViewPos,1,GL_FALSE,viewMat);
glUniform3f(gouraud3d.uniformLightDirPos,0,0,1);
glUniform1f(gouraud3d.uniformAmbientPos,0.3f);
glUniform1f(gouraud3d.uniformSpecularIntensityPos,1);
glUniform1f(gouraud3d.uniformSpecularExponentPos,100.0f);

glEnableVertexAttribArray(gouraud3d.attribVertexPos);
glEnableVertexAttribArray(gouraud3d.attribNormalPos);
glEnableVertexAttribArray(gouraud3d.attribColorPos);

glVertexAttribPointer(gouraud3d.attribVertexPos,3,GL_FLOAT,GL_FALSE,0,vtx.data());
glVertexAttribPointer(gouraud3d.attribNormalPos,3,GL_FLOAT,GL_FALSE,0,nom.data());
glVertexAttribPointer(gouraud3d.attribColorPos,4,GL_FLOAT,GL_FALSE,0,col.data());

glDrawArrays(GL_TRIANGLES,0,vtx.size()/3);

glDisableVertexAttribArray(gouraud3d.attribVertexPos);
glDisableVertexAttribArray(gouraud3d.attribNormalPos);
glDisableVertexAttribArray(gouraud3d.attribColorPos);
```

Phong shading

- Same lighting model, but the light intensity is calculated per pixel, not per vertex.
- This really has been one thing that the OpenGL's fixed-function pipeline could not (did not) do.

- Additional varying:
  - Normal vector

Phong Shading

- Vertex Shader

```
attribute vec3 vertex;
attribute vec3 normal;
attribute vec4 color;

uniform mat4 projection,modelView;
uniform vec3 lightDir;
uniform float ambient;
uniform float specularIntensity;
uniform float specularExponent;

varying vec4 colorOut;
varying vec3 normalOut;
varying vec3 viewDirOut;

void main()
{
    vec4 posInView=modelView*vec4(vertex,1.0);
    viewDirOut=-normalize(posInView.xyz);
    normalOut=normalize((modelView*vec4(normal,0.0)).xyz);
    colorOut=color;
    gl_Position=projection*modelView*vec4(vertex,1.0);
}
```

Phong Shading

- Fragment Shader

```
varying vec4 colorOut;
varying vec3 normalOut;
varying vec3 viewDirOut;

uniform vec3 lightDir;
uniform float ambient;
uniform float specularIntensity;
uniform float specularExponent;

void main()
{
   vec3 lit=normalize(lightDir);

   float diffuse=dot(normalOut,lit);

   vec3 midDir=normalize(viewDirOut+lightDir);
   float specular=specularIntensity*pow(dot(midDir,normalOut),specularExponent);

   gl_FragColor=vec4(colorOut.rgb*(ambient+diffuse),colorOut.a)
         +vec4(specular,specular,specular,0.0);
}
```

Phong Shading

- In renderer.h

```
class Phong3dRenderer : public Gouraud3dRenderer
{
};
```

Phong Shading

- Add a member variable:

```
Phong3dRenderer phong3d;
```

- In Initialize function in main.cpp

```
phong3d.CompileFile(
    "phong_vertex_shader.glsl",
    "phong_fragment_shader.glsl");
```

- In Draw function in main.cpp

  Just replace gouraud3d with phong3d.

Phong vs Gouraud shading

- Difference is most visible if the object has a large or long polygon. (Like the bottom face of a cone.stl)