CS 770/870
Assignment 3: Textures, Materials, and a little Light

October 13, 2014

Due: Friday, 10/23 by 23:59:59
Late: 10/24-25 -3; 10/26 -7; 10/27 -15

The primary purpose of this assignment is to give you practice with OpenGL texture mapping and materials facilities. You will need the same basic multiscene framework as P2, but you won't need to use composite *Object3D* classes.

As in P2, the user must be able to step through a set of predefined scenes that clearly show that the **new** features of your program work correctly. Each scene must have a corresponding brief description and you must have the scene transformation functionality working. Each scene must include objects that approximate curved surfaces so that they can illustrate lighting and material properties well. These objects must also have correct normals to effect good lighting. A good choice for this is the *gluQuadric* functionality. Major tasks:

1. **Box** object. Redefine the *Object3D Box* class by replacing its call to *glutSolidCube* with OpenGL calls that define the vertices *with normals* for the 6 faces of a cube centered at the origin with sides of length *length* (the parameter to the constructor). The normals to the vertices should be the normal to each face of the cube. Note that each vertex will be replicated 3 times, once for each of the faces that share that vertex, but the normals will be different. All faces have the same color or material parameters and if a texture is assigned to a Box object, the texture must be mapped to each face of the Box.
2. **RoundedBox** object. Make this a child of the **Box** class. The only difference in behavior is that the normals to the vertices in this class should be the *average* of the normals of the 3 faces that share the vertex.
3. **Quadric** class(es). Create subclass(es) of *Object3D* that draw 2 different **gluQuadric** objects. You can either 1) create a single child class that accepts a **gluQuadric** object as a constructor parameter and use that class with **two** different *gluQuadric* objects, or 2) you can create **two** different children of *Object3D* that hard-code two different *gluQuadric* objects as data members. **Note:** The simple *glutSolid* objects do not provide access to the coordinates of the points generated to represent them, nor do they allow you to specify how to map a texture to their surface. Consequently, you cannot readily associate a texture with them. However, the **gluQuadric** objects allow you to enable texturing and have internal code to do the mapping and to compute the normals. Google "gluquadric" for useful sites; one that looks particularly useful is: http://www.cs.csustan.edu/~rsc/sdsu/Modeling.GLU.GLUT.pdf
4. **Object3D** extension. Add material properties and texture features to the *Object3D* abstract class as defined below. You only need to support this functionality in the 2 *box* and (1 or 2) *Quadric* classes you create.
5. **Textures.** Your application should use at least two different textures that can be assigned to any of the *Box* or *Quadric* objects in the scene. JOGL already contains a *Texture* class (used in the JOGL texture demo) that should be adequate as a data member of *Object3D*. This class encapsulates the texture and the various parameters that are used to apply the texture. The current parameter settings are used when the primitive is defined, so you can use the same texture with different parameters for different primitives. The C++ texture demo also contains a primitive *Texture* class, but you may need to add features to this class to provide all the desired functionality, such as the various texture mapping options provided by OpenGL. Regardless of your language choice, you need to support the *glTexParameteri()* functionality defined by the options:
   GL_TEXTURE_MAG_FILTER,
   GL_TEXTURE_MIN_FILTER,
   GL_TEXTURE_WRAP_S
   GL_TEXTURE_WRAP_T.
   You do not need to implement GL_TEXTURE_BORDER_COLOR or GL_TEXTURE_PRIORITY. Even without these 2, there are 48 different possible combinations of values. There is no reason not to support all choices in your code, but you only need to show a few sets in your test scenes. For example, you may always treat the two "wrap" parameters in the same way (both GL_CLAMP or both GL_REPEAT). For the "min filter", you really should show a mipmap parameter along with a non-mipmap option (GL_NEAREST or GL_LINEAR). For the "mag filter", you could show the GL_NEAREST vs GL_LINEAR. This requires

a minimum of 3 scenes, each with at least 2 objects, each showing the results of varying just 1 parameter between 2 choices.

6. **Material Properties.** Your application should define a class to represent material properties, and the *Object3D* abstract class should accept and save an instance of this class. Your scenes should clearly show multiple instances of the same object class with significantly different material properties, like shiny vs. dull. A user should be able to see many variations of material properties showing that you support all the material parameters. *It can be extremely hard to see the effects of different material property settings if you also have textures mapped to the same surface, so the scenes you build to show material property functionality should not use textures, except as an "extra".*
7. **Lighting.** Your application should have two lights of different types, each of which can be independently turned on and off with a check box.
   - Point source light. One light source should be a point light source defined at a specific location.
   - Directional light. One light source should be directional light, similar to the sun, located at infinity. You get this by specifying its "location" with a *w* parameter of 0.

   The two light sources should light the scene from clearly separate directions; e.g. one could be placed so it is seen more on the left and bottom sides of an object at the origin and the other more on the right and top sides, but some portions of the object should be seen by both lights. Make the initial colors of the lights different, but all aspects (ambient, diffuse, specular) of each light should have the same color.

   Define 3 sliders for the R,G,B components of the light and 2 radio buttons to select which light's color is being changed.

Point allocation

10	Revised *Box* class shows flat shading.
5	*RoundedBox* class shows shading across face.
10	One *gluQuadric* object implemented within the *Object3D* framework.
5	A second *gluQuadric* object implemented within the *Object3D* framework.
10	Point lighting is implemented (3); user can turn it on/off (2) and modify the color (5).
10	Directional lighting is implemented (3); user can turn it on/off (2) and modify the color (5).
20	Material properties class implemented and clearly shown to be working by test scenes
30	Parameterized texture mapping implemented and shown to be working by test scenes

Please **do not** use the animation framework from the texture demos; it is too hard to show what you want to be able to show us.

jpeglib: The C++ texture demo and C++ submissions for this assignment must use the *jpeglib* utility (http://www.ijg.org). This comes in Fedora Linux and I think also in other linux distributions. (Check in your system for /usr/include/jpeglib.h and /usr/lib64/libjpeg.so to be sure.) **Mac** users can copy *libmac.tar.gz* and *incl-mac.tar.gz* from the *public* link on the course web page or from ~cs770/public on agate. The content of these tarballs should be moved to your *cs770/lib* and *cs770/include* folders respectively. If you do that, the standard *Makefile* will find them on your machine and will also work when we run your code. **Windows** users will need to download the Windows code from http://www.ijg.org. If you aren't using *make* on Windows, just be sure your code compiles and executes on a CS linux machine before submitting.

Submission:

1. Submit as *P3*. Even though the assignment is *P3*, your class name need not be *P3.java* or *P3.cpp*. If your C++ *main* program is **not** a class, it is ok call it *p3.cpp*, but it should **not** be *P3.cpp*. Class names and only class names should be capitalized.
2. Don't forget to submit:
   a. The correct *Makefile* (**not** *Makefile.java*, **not** *Makefile.cpp*) **in which you have explicitly set the PROG or MAIN variable to the correct name of your executable C++ file or the main Java class.**
   b. Your texture image files.
3. **Do not** submit your Java code as a *package*.
4. All of your files, including your texture files, must be in a single directory