CS 770/870
Assignment P2: Textures

February 22, 2017 (keyboard edits in red 2/27)

Due: Thursday, March 9 midnight; 3/10 -3, 3/11-19 -7, 3/20 -15

The primary purpose of this assignment is to practice with OpenGL texture mapping. You will need the same basic multiscene framework as P1. 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 title identifying the functionality you are trying to show and you need to turn in a text document with additional explanation.

Major tasks:

1. **Color/texture blend**: Must still support mixture of color/texture, but with blend that can be changed interactively. A single float ranging from 0 to 1 identifies the color percentage, c, that starts at 0; 1-c is the texture percentage. The “c” key should increment c by 0.1 with a maximum of 1. The “C” key should decrement c by 0.1 with a minimum of 0.

2. **Interactive viewing changes**. In order to see the effects of many of the texture parameter variations, it is often very helpful to be able to modify the viewing parameters. It’s relatively easy to provide some very effective interaction features by implementing a constrained set of modifications to the `lookAt` parameters: eye, at, and fovy using keyboard interactions. First let n = at – eye, the vector from eye to at.
   - Zoom In (“i”): move eye position dz units closer to at along n. Compute dz to be some fraction of the initial specification of the `lookAt` parameters for the scene. If dz is 1/10 the length of the initial n you’d get 9 key strokes before having an invalid `lookAt` – of course, you don’t want to allow the 10th “zoom in”. If you find a smaller or larger dz to be more effective, feel free to use it.
   - Zoom Out (“o”): move eye position dz units away from at along n.
   - Smaller fovy angle (“a”): subtract da from fovy. As with dz, make da about 1/10 the size of the initial perspective specification for the scene. The equivalent change to an orthographic projection is to narrow the left/right and top/bottom boundaries by about 1/10 th of the initial ortho parameters. For example, let dx = 0.1*(initialRight - initialLeft). Each “a” means left+ = dx/2 and right = dx/2. Similar changes occur with top and bottom. Feel free to use smaller steps, like 5% (0.05 factor).
   - Larger fovy angle (“A”). Same da or dx,dy as for “a”, but add the deltas rather than subtract.
   - Pan left/right (“l”, “r”): change the at x coordinate by −dx or +dx. Choose increment to be useful. This requires you to change “line” mode to a capital “L”.
   - Pan up/down (“u”, “d”): change the at y coordinate by −dy or +dy.
   - Restore original viewing specifications (`GLFW_KEY_BACKSPACE`): restore all view/projection parameters to original scene specifications.

3. **Cylinder** object with a variable number of vertical faces (a parameter to the constructor) and a single texture wrapping around it. Cylinders can have optional top and/or bottom faces that also have the texture mapped to them. All triangles can have the same underlying non-texture color.

4. **Mipmap** creation. Mipmaps of a texture can be automatically generated with the `glGenerateMipmap` function. Figure out how to do this and support it.

5. **Textures**. Your application should use at least two different textures that can be assigned to any `Quad`, `Box` or `Cylinder` object in the scene. The Java and C++ texture demos both have a basic `Texture` class with hard-coded texture parameters. You need to modify this class in some way to support 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 around 48 different possible combinations of values. You should support all choices in your code, but you only need to show a subset of the combinations in your test scenes:

a. You may always treat the two “wrap” parameters in the same way (both GL_CLAMP or both GL_REPEAT). That alone reduces the combinations to 24.

b. You do not need to show all combinations of MAG and MIN options. Show all MIN options with only 1 MAG option using multiple relatively small objects (of different sizes), where the MIN effects are needed and might be visible. Show all MAG options with the same MIN option using 1 or 2 large objects (also of different sizes), where only the MAG effects are needed.

c. It would be best to use the Quad object to show all your variations and have multiple objects in the same scene showing different options so you can see them near each other. You will have to experiment with the sizes of the objects you use in order to be able to see differences and some combinations may be difficult to see.

d. A regular pattern such as a checkerboard is likely to provide a better vehicle for showing variations.

6. Texture parameter specification

Texture parameters need to be specified before the texture is actually created. OpenGL does this with its “state” machine model, but this isn’t very friendly in an object-oriented world, where you would like to encapsulate the parameters and the texture creation in a single class. Two possible options to consider:

a. Change the Texture constructor method so that it just saves information passed to it; add parameter specification “set” methods to specify the various options, and add a makeTexture method that ends all specification and builds everything needed for the texture. In principle, later set and makeTexture calls to the same object could create a new texture, but that certainly isn’t necessary for this assignment.

b. You could consider creating a TextureParameters object that would have all the set methods for defining the different options. You could then pass a TextureParameters object to the constructor for a Texture, which could then immediately create the texture data.

c. Both these options are reasonable and there may be other good options, as well. However, do better than OpenGL’s state model!

7. readme.txt

The relatively short title available for each scene should be used as a section title in your readme.txt file, which should include a description of the features being demonstrated by the scene. This should be a plain text file named “readme.txt”, without the quotation marks. It should not be README.txt, nor readme, nor Readme.txt, nor any other variation. Do NOT submit a Word file

Tentative point allocation
20 Cylinder implementation with a single texture mapped to the vertical area.
10 Cylinder top/bottom faces; they’ll get textures if they have texture coordinates mapped.
25 Interactive parameters changes; texture weights and viewing parameters.
10 Mipmap generation. Read about glGenerateMipmap.
35 Texture mapping parameter options and general quality of your tests for showing completeness and correctness; the quality of your readme.txt will be a factor here. You can primarily use Quad and/or Box objects for the demonstrations; many effects are easier to see with simple objects. Similarly, many effects are almost impossible to see with very complex patterns. The checkerboard will be your best friend here!

Submission:
1. Submit as P2. Your main Java class should be P2.java, your C++ executable should be p2.
2. The correct Makefile with MAIN=P2 or PROG=p2.
3. Your texture image files, but not the images posted for demos; you can use them, but we’ll load them.
4. A readme.txt to add explanations about each scene.