CS 770/870
P0: 2D Shapes

January 27, 2017

Due: Monday, 2/6 (by midnight 23:59:59)

Late penalties: Tuesday -3, Wed -7, Thursday -15

The goals of this assignment are to insure that you gain access to a working OpenGL/GLSL 3 environment right away, and to introduce you to the most basic output features of OpenGL and GLSL 3 within a simple object-oriented context for representing 2D objects that can be displayed. You have the choice of using C++ or Java with LWJGL3. Your task is to extend one of the *BasicLWJGL or basicGLSL* demo programs to add support for Rectangle and Polygon objects.

1. Preliminaries

1. Download either *BasicLWLJG.jar* or *basicGLSL.tar* (or *basicGLSL.tar.gz*) from the course web site (http://cs.unh.edu/~cs770). You can reach the files by clicking on the appropriate "demo" link or by going to the *CS770/870 agate public directory* link.
2. Rename *BasicLWJGL.java* to *P0.java* or *basicGLSL.cpp* to *p0.cpp*. **This is important since our test framework tries to run either the class P0 or the executable p0.** Be sure to edit the Makefile to set the MAIN or PROG variable to *P0* or *p0*.
3. Software installation: see section below.

2. Refactor Triangle/Shape2D code

Both the Java and C++ demos do all the rendering work in the *Triangle* class. Now that we are adding other children to the *Shape2D* hierarchy, this code should be shared as much as possible. It would be "safest", to refactor the existing code first before trying to generalize it to support the additional classes. In particular, the *makeBuffers* and *redraw* methods need to become the responsibility of *Shape2D*.

3. Rectangle

Add a *Rectangle* class that inherits from *Shape*. A *Rectangle* object should be specified by *location* and *width, height* parameters to the constructor. It should inherit the *set/get* methods from *Shape2D*. In addition, you should *refactor* the existing *Shape2D/Triangle* code so that the *makeBuffers* and *redraw* methods are moved to *Shape2D* and generalized (just a bit of generalization is needed)

4. Polygon

Add a *Polygon* class that extends the *Shape2D* class. A *Polygon* object constructor can take information defining any number of points. You should define at least 2 constructors for Polygon: one that takes an array of *x* and an array of *y* coordinates and one that uses a class that encapsulates *x* and *y* into one object.

5. Scene

Build a scene using multiple copies of all your classes using all available constructors for those classes. and using the *setLocation* and *setSize* methods that are inherited from *Shape2D.*

The *basic* demos are not appropriately designed for growing into real applications. In particular, there really should be a separate class for representing a *Scene* and most real applications will have multiple scene objects, or ways to modify the objects in a current scene. For this assignment, you just need to define a *Scene* class that will (for now) just encapsulate the collection of *Shape2D* objects that are in the scene. (Later we will add scene transformations and scene lights, etc.) You also should define a class that creates scenes (maybe called SceneCreator or SceneMaker or whatever??). For this assignment you will just move the code that creates the objects from the BasicLWJGL class or basicGLSL.cpp file to your maker class, which will return an instance of *Scene*.

6. Interaction

There will be only one simple interaction requirement for this assignment. If the user presses the "p" key, you should toggle your display from drawing "filled" triangles to drawing only the edges of the triangles. This is done with a single OpenGL call:

glPolygonMode(GL_FRONT_AND_BACK, GL_LINE); // Default 2nd parm: GL_FILL

7. Point allocation

60 Implementation and display of the *Triangle* and *Polygon* classes that inherit from *Shape* that draw many distinct shapes using multiple colors, locations, and scales.

20 Implement the *Scene* and scene maker classes.

20 Very clear demonstration from the visual results that you have implemented all the required features.

Points are earned by correctly implementing features robustly. Points are deducted for bugs, incorrect implementation, poor style, poor commenting, poor design decisions, non-functioning make file, etc.

Submission information will be available soon.

8. Software installation

Create a cs770 "home" directory in your file system, called HOME770 here. This directory is to store the libraries that you will need to run your programs. You'll probably want to keep your assignments there as well, but that isn't necessary. In your HOME770, Java users create a *jars* folder; C++ users create an *include* folder. Everyone should create a *lib* folder. You must edit the *HOME770* assignment (line 49 of the Makefile) to specify the location of your HOME770. Be sure to leave the "=?" in the assignment; if you don't, your code won't run when we try to test it.

8.1. Java

Download one (or more) of *lwjgl-{macosx,windows,linux}.jar* from the *CS770/870 agate public directory* link. After un-jarring, you'll find *lwjgl770.jar* and a *macosx* or *windows* or *linux* folder. All platform folders contain another folder, *lwjgl-native.* Move the *lwjgl-native* folder to the *lib* folder you created in HOME770. This directory is referenced in the Makefile in a java option for the JVM.

8.2. C++ Mac OSX

1. Install GLFW 3.2.1 on your machine (http://www.glfw.org). If your linux has a package for it, use that; it will install it in /usr/include and /usr/lib, which should make access to it automatic. The MacOSX installation took 3 terminal commands

```
cmake -DBUILD_SHARED_LIBS=ON .
make
sudo make install
```

   It installs the code into /usr/local/include and /usr/local/lib. These are generally searched automatically, but not necessarily by your IDE. So, you may have to explicitly tell your IDE to look for includes in /usr/local/include and libraries in /usr/local/lib.

2. Download GLEW source package from http://glew.sourceforge.net. Execute

```
make
sudo make install
make clean
```

   Installation comments are the same as for GLFW.

8.3. C++ Windows

You can download binaries for both GLFW http://www.glfw.org/download.html and GLEW http://glew.sourceforge.net.

8.4. C++ Linux

1. See C++ Mac OSX for installation of GLFW.
2. Verify that GLEW is installed. Do a "locate GLEW" from a terminal. You'll probably see: `/usr/lib64/libGLEW.so` If it isn't there read Mac OSX section below about installing GLEW.

Get a working OpenGL environment on your machine

1. **Windows**: if you have an older version of Windows, verify that it supports OpenGL 4.1 or higher. You really should install *make* on Windows and learn to use it from the windows command line. If you don't, be sure you've tested the *make* on a CS workstation before submitting **each assignment**. This will be hard to do just before a deadline because you must be sitting at the workstation; remote execution doesn't work with this level of OpenGL.
2. **Linux** (your machine): make sure you have Mesa 11 or newer. Mesa is the public domain version of OpenGL that is used by most Linux systems. Mesa 11 supports OpenGL 4.1. Mesa 12 and 13 are available.
3. **Mac OS X:** You may have trouble with the OpenGL level if you don't have either OS 10.9 (Mavericks) or 10.10 (Yosemite). I have 10.11. You also need to 1) Install XCode and 2) **From XCode Preferences, install "Command Line Tools"**. Even if you don't use XCode as your IDE, you'll need the command line tools for *make* to work correctly; it sets up all the hidden Mac OSX libraries to be available in the "standard" unix locations.
4. **Java**: install LWJGL from http://cs.unh.edu/~cs770/public as described in P0 description.
5. **C++**: Make sure you can access OpenGL, GLEW, and GLFW and *make* on both agate and whatever machine you plan to use for most of your code development. OpenGL and (I think) GLEW are already part of any Linux installation; GLFW is on all CS public machines. OpenGL is also included in MacOS X as a framework. Copy ~cs770/public/c++/basicGLSL.tar **and** ~cs770/public/include.tar to your agate file system. Or, you can download both files from the course website: http://cs.unh.edu/~cs770/public These will unpack into directories named *basicGLSL* and *include*. You should move the *include* directory to your *cs770* home ($HOME/cs770/include), which is where the C++ Makefile will be looking for it.
6. **Make: Be sure that *make* works on the demo before you start extending the code so we have time to address problems.** In a terminal/console window, you need to cd to the directory with your source and data, and then run:

   make

   By default the *make* program looks for a file named *Makefile* for its input. With no arguments and this *Makefile*, make will compile any files that have changed since the last make, will link the compiled code (if C++), and execute the file defined by the MAIN (Java) or PROG (C++) variables. This make has 4 alternate entry points defined by the first argument to make:

   ```
   make clean   # deletes all compiled output and executables
   make new   # deletes all compiled output and executables and then compiles, links and runs.
   make build   # compiles/links (if needed), but does not execute
   make run   # executes without compiling or linking – unless something needs to be compiled!
   ```

   Hopefully, you will not need to make any changes to the Makefile, except for the MAIN/PROG variable, and HOME770 variable assignments as described above and in the Makefiles. If something doesn't work for you, send me email. It should be possible to tune the make file to work on your configuration without resulting in failure on ours.