CS 770/870
Assignment 2: Basic 3D Viewing and Simple Hierarchical Modeling

September 26, 2014

Due: Friday, 10/10 by 23:59:59
Late: Sat-Sun -3, Mon -7, Tues -15

The purpose of this assignment is to give you practice using basic OpenGL 3D modeling and viewing features. You can base your assignment on the *threeD/ThreeD* demo programs.

3D objects

The *Object3D* class is used to represent simple 3D solid objects in the *threeD/ThreeD* demos, which define 2 children of *Object3D* that are *wrappers* for objects defined by one of the *glutSolidXXX* functions. The wrapper encapsulates the object type itself along with the transformations necessary to place the object in its parent frame.

1. Create at least 2 more one-object children of *Object3D.* In addition to the *glutSolids,* there are also some basic solids in the *glu* utility library that can be "wrapped" in such a class. You may also build your own object by correctly generating polygon primitives. This should look like a single object even though it is created using multiple primitives; such an object is **not** a hierarchical *composite* object as described in the next paragraph since its parts are primitives rather than other *Object3D* objects.
2. Create a child of *Object3D* that can define a composite object containing simple objects and other composite objects (and maybe some primitives). The *Object3D* class allows the user to specify the location, size, and orientation of the object in the coordinate *frame* within which it is being placed and to specify whatever colors should be used for a particular instance of the *Object3D* class. In other words, you'll need (at least) methods for the following actions:
   a. redraw the object
   b. set the scale along the x, y and z axes to size this object in the caller's coordinate frame
   c. set the object's orientation in the caller's frame based on an axis and angle of rotation
   d. set the location of the object in 3D relative to the caller's coordinate frame.
   e. set the "default" color of the object.
   f. set the *i*-th color of the object; this is an abstract mechanism for allowing instances of the same class to have multiple objects within them each with a different color. The 0^{th} color is the same as the "default" color.
   g. you'll probably find it convenient to have *get* methods for some/all of the *set* methods.

   The transformations will always be applied in the order TRS. That is the Scale will be applied to the vector coordinates first, then the Rotation and finally the Translation.
3. Create at least 3 children of *Object3D* that define composite objects composed of multiple instances of other *Object3D* classes. At least one of these must include multiple copies of a composite *Object3D* class. The *redraw* methods of these classes need to use the appropriate openGL matrix operations on the *ModelView* matrix stack to insure that you properly implement multilevel hierarchical object definitions.

Scenes

Your user interface should present and cycle through a series of pre-defined *scenes* controlled by the *Next Scene* button. The *Scene* class in the *threeD/ThreeD* demo encapsulates a collection of *Object3D* objects and a set of viewing and projection parameters. (You might consider encapsulating the viewing/projection parameters in a *View* class.) You should extend the Scene class to add a *scene transformation*, one final TRS composite that can be used (normally, interactively) to transform the all components of the scene uniformly.

Interactive Scene transformation

Build a *glui* or Swing/AWT window allowing the user to modify the scene transformation T, R, and S parameters interactively. One set of 3 sliders can be reused for multiple sets of input parameters as long as you have radio buttons to identify what they are currently controlling. Be conscious of user-friendliness.

Testing

As always you need to show us that you have tested your program well. The main goal is to show convincingly that your code implements the desired non-interactive functionality. For example, you should have one scene that shows an instance of each class you have implemented viewed in *its* own local coordinate system and then show multiple copies of that object using different transformations. Start with the simplest objects and progress to the composite objects and finally to scenes composed of multiple different objects. It might save you time during testing to also provide a *Previous* scene button, so that you don't always have to cycle through the old (working) examples to get to your new ones.

Part of the scene functionality should also include demonstrations that you have implemented the viewing and scene transform functionality. For viewing, this can be done with a (small) series of scenes that have the same objects in the same locations, but with different viewing parameters. Use parameters that make it clear what has changed. For example, use the same *lookat* parameters, but simply change from perspective to orthographic projection in a way that the user can see what is happening.

Scene transformations are best shown with your interaction. However, if you do not get the interaction part working, you can still implement the scene transformation functionality and show it off with multiple scenes.

Your control panel should have a label field in it that describes the goal of the current scene: that is, every scene should have its own phrase or short sentence that lets you tell the user/grader what feature(s) that this scene is showing (that was not demonstrated by earlier scenes). A basic version of this feature is already part of the Java demo program.

Point allocation

15 Two additional **single** object children of *Object3D*

30 Multi-scene implementation with clear demonstration of program's non-interactive functionality. These points are mostly a *testing* requirement. Implement it early, so you can facilitate your own testing!

30 Implementation of hierarchically defined *Object3D* classes

25 Interactive modification of scene transformation parameters using sliders/scrollbars.

Notes:

1. **Please do not use packages.** Our submission and grading tools can't handle them (yet).
2. The *threeD* demo program provides the framework for setting up the basic environment for doing simple 3D lighting and rendering.
3. It is not necessary for you to use all (or even any) of my skeleton implementation framework reflected in the *threeD* demo. Also, feel free to modify any parts of it you feel could be done in a better way in the context of your overall program design.