CSC418 Winter 2008

Assignment 3
OpenGL Practicum

Due March 10th at 11:59pm.

In this assignment, you will implement a basic 3D object viewer using OpenGL, GLUT, and the GLUI user interface toolkit. You will be able to view basic shapes and more complex hierarchies of these. Your geometric objects will have lighting and material properties to increase the realism of your scenes, and you will provide camera controls in order to view your scene from different angles.

1. Overview:

Your program will have a viewport window from which your scene will be visible, and any number of GLUI user interface windows to support property editing. The UI windows will allow you to edit the geometric and material properties of your objects and move them around in the scene. There will be similar controls for editing light properties, as well as turning them on and off.

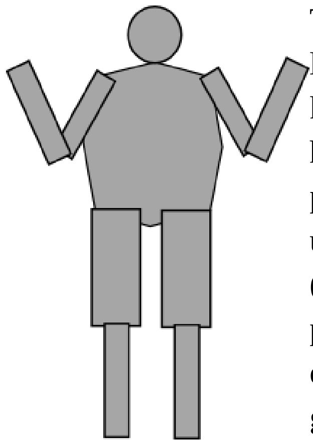
2. Scene structure:

In class and tutorial we have studied what is required to generate a 3D image on the computer: lights and materials, geometry, and a camera. We've also studied how complex objects can be modeled as a hierarchy of simpler objects, combined with affine transformations. Thus, the top level objects in your program should be a list of lights, the displayed object or hierarchy, and a camera.

3. Geometry:

Your modeler will display a single object at a time. An object may be a 3D geometric primitive, such as a box, a sphere, or a cone, or a hierarchical object as described in the next section. The command to display one of these geometric objects should be accessible from the GLUT menu or from a GLUI window. When the command is invoked, your modeler should replace the current object with the one requested by the command. The new primitive should appear in the scene at a default location with reasonable default values for all its properties. For example, the SPHERE command, invoked from a button or menu, would create a sphere of radius 1 centered at the world origin with a grayish material. The camera should also be reset to its default location.
As a minimum, you should have support for the following primitives: box, tetrahedron (pyramid with 4 triangular sides), and sphere, NOTE: You are required to generate the geometry and normals for these primitives yourself; that is, you need to tessellate the primitives yourself. You will lose marks if you use calls to glutSolidSphere() and the like (only glutSolidTeapot() is allowed!)

4. Object hierarchies:



The hierarchy object will be a hard-coded collection of boxes and spheres that look like a person (head, body, two arms with elbow joints and two legs with knee joints). When the creation command of the hierarchy is invoked, the hierarchy (i.e. the person) should appear, replacing the previous object. The 3D primitives making the person should be arranged in a linear list (or array). The user interface should provide means for selecting the next or previous primitive (body part) in the list. Once a primitive is selected, the user interface should provide means to rotate one of its joints in 3D (there should be a way to rotate every joint in the hierarchy). You can base your code on the example code given in tutorial.

5. Transforming objects:

You will provide methods to move the objects in your scene interactively. A single primitive object, or the top level body part in your hierarchy, will have 6 degrees of freedom: 3 for translations (x,y,z) and 3 for rotations (x,y,z). All sub-objects in your hierarchy will only have the 3 rotational degrees of freedom.
To invoke a translation of your object, use the mouse buttons along with the SHIFT key. Rotations should be accessed with the CTRL key. Use one mouse button per axis: for example, with the CTRL key down, let the left mouse button handle X rotations, the middle button Y rotations, and the right button Z rotations. See the example from tutorial 1 for handling mouse motions.
Note: do NOT use the GLUI translation and rotation controls or you will lose marks.
The top level object should also have a GUI-controled scale parameter which uniformly scales the entire object or hierarchy.

6. Materials:

Each object should have editable material properties. When the hierarchical object is displayed, each 3D primitive should have its own material properties. These should be edittable through the user interface. Edittable properties include ambient, diffuse, and specular (RGB) colors and shininess. The controls can be numeric; do not spend time trying to add a sophisticated color picker.
You should also add a normal-as-color scheme, which is a useful tool to check the geometry of your objects before you implement lighting. It works as follows: consider that the (x,y,z) components of a unit normal vector lie between -1 and 1, while the floating point RGB values of a color lie between 0 and 1. To produce a color display that is indicative of the normal direction, we simply use the (x,y,z) values of the normal as the RGB components of the color using the formula RGB = ((x,y,z) + 1)/2 to map the values to the range [0,1]. This gives you better idea of the geometry of your object than a flat shaded polygon.
This mode should be activated when lighting is switched off (see below).

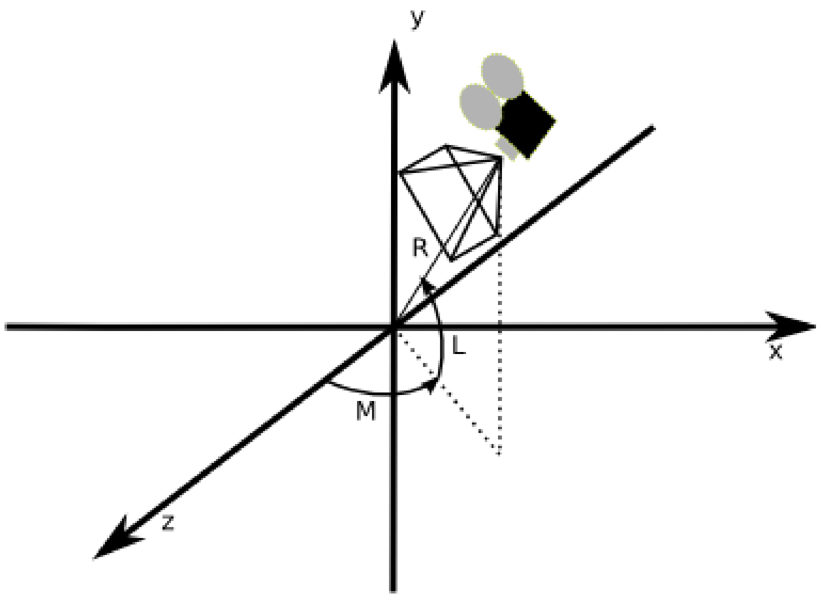
To aid you in debugging, also provide a GUI checkbox to draw normal vectors as short line segments in the middle of your triangles for face-normal shading or at the vertices for vertex-normal shading.

7. Lights:

Your scene should have three lights (although you can add more if you like). Your UI should provide a way to select the lights (independent of which object or part of the hierarchy is displayed) and change the light properties. Properties that should be changeable are positional v. directional style and ambient, diffuse, and specular colors. You should also be able to turn individual lights on and off. There should also be an edittable global ambient term that allows you to see the current object when all lights are turned off. The position of your lights should be fixed in the scene, independent of camera motion. Additionally, add a global switch to turn lighting off, and render your object using the normal-as-color scheme described in the Materials section.

8. Camera motion:

For this assignment, you will implement a two-angle camera. This camera is described by a lookAt point, an up vector, and a point in spherical coordinates (R,M,L) defining the camera center (see the lecture notes, pg. 29, for a refresher on spherical coordinates). R describes the radius from the lookAt point to the camera center, M is the longitudinal (meridian) angle of the camera (0 to 2π), and L is the latitudinal angle (-π to π).



To enable you to see the objects on the screen from many viewpoints, you will implement interactive camera controls for panning, tumbling, and dollying the camera. Panning involves moving the camera parallel to the image plane, tumbling rotates the camera about the lookAt point, and dollying moves the camera closer or further away from the lookAt point.
The camera motion in your viewer should be triggered by a mouse button when the ALT key is pressed. Use tumbling with the left mouse button, panning with the middle button, and dollying with the right button.
Also provide GLUI controls to change the focal length of the camera and the value of the far plane.

9. Submission instructions:

Your program must be submitted on the CDF system. You should submit your code files and a Makefile that compiles your program correctly by simply calling 'make.' Name your program `a3`. Additionally, include a text or pdf file with a brief summary of your design decisions and code structure. This summary should be succint, and it should help the grader understand your code (in other words, if you were grading the assignment, you'd like this document to tell you what you need to know).

To submit, use the command on the CDF system appropriate for your course:

```
submit -c csc418h -a a3 <filenames>
submit -c csc2504 -a a3 <filenames>
```

The **submit** command timestamps and uploads the files in the command line to a special folder where we can access them to grade. More information about **submit** can be obtained from the man page on CDF.

10. Resources:

OpenGL Red Book: http://fly.cc.fer.hr/~unreal/theredbook/
GLUT API: http://www.opengl.org/documentation/specs/glut/spec3/spec3.html
GLUI API: http://glui.sourceforge.net/
NeHe OpenGL Tutorials: http://nehe.gamedev.net/

11. Checklist:

1. Geometry:
    1. Box:
        1. Correct geometry
        2. Correct normals
        3. User interface command to make a box the current object.
    2. Tetrahedron:
        1. Correct geometry
        2. Correct normals
        3. User interface command to make a tetrahedron the current object.
    3. Sphere:
        1. Correct geometry
        2. Correct normals

3. User interface command to make a sphere the current object.

2. Hierarchical transforms:
   1. Complex hierarchy:
      1. Objects in correct locations
      2. Moves, rotates, scales as one figure.
      3. Correct rotations of lower branches.
      4. User interface command to make the hierarchy the current object.
   2. Method for hierarchy traversal
   3. UI update on hierarchy traversal
3. Lighting:
   1. Correct use of glLight() and glMaterial()
   2. Three lights defined.
   3. Global ambient term
   4. Normal-as-color mode
   5. UI
      1. Light selection from GUI control.
      2. Change light type (positional/directional)
      3. Change RGB ambient, diffuse, specular of lights
      4. Change RGB ambient, diffuse, specular, and shininess of material
      5. Turn lights on/off and turn lighting off.
4. Camera:
   1. Correct use of glMatrixMode(), gluLookAt(), and camera shape functions.
   2. Reset camera button.
   3. Change near and far planes from UI.
5. Mouse interaction:
   1. Correct routing of callbacks based on mouse button and modifier keys.
   2. Camera movement:
      1. Pan
      2. Tumble
      3. Dolly
   3. Translations
   4. Rotations
6. Short Report