COMP 410/510, Computer Graphics, Spring 2018

Programming Assignment #1

Model Animation with Shader-based OpenGL

Due Date: Friday, March 9, 2018

Problem Description and Learning Objectives:

In this assignment, you will design and implement a basic graphics application that animates a 3D model on the screen (as shown above). This will be an interactive program that allows the user to select an object to draw (from a choice of three objects), to choose various drawing attributes for the object (such as color and polygon mode), to adjust speed, etc. Each time the user selects one of the drawing options, the image on the screen will be redrawn with the new choice. To develop this program you will need to learn how to use both **shader-based OpenGL** and **GLUT** (or FreeGLUT) 3D graphics libraries. The GLUT library includes functions for implementing event-driven input and display handling operations. This assignment will be an introduction to developing event-driven graphics programs. Your program must be developed using **shader-based** OpenGL and C/C++.

Problem Specification

Your program will display, animate and switch between three different models, cube, sphere (sphere.off) and a car model (car.off). The latter two models will be loaded from off files provided. When loaded, the models are expected to do a heart-beat animation (scaling up slowly to touch the bounds of screen and then scale down continuously) with a certain speed. Note that the figure shown above is only a demonstration; your program is not expected to generate exactly the same output.

You can preview the given off files using MeshLab (http://www.meshlab.net). Each of these files includes a vertex list first and then a triangle list as shown below. Off files are actually ASCII files and you can see their content using any text editor. But for your application you need to write your own loader:

```
###
OFF
1231 1078 4166
28.750000 8.500000 -28.250000
-28.750000 8.500000 -28.250000
29.500000 8.500000 -30.000000
...
...
3 9 1 3
3 12 10 2
3 3 11 13
3 14 16 6
3 7 17 15
```

Note that the first two numbers in the header indicate the number of vertices and the number of triangles in the list, respectively (the third number is the number of edges, which you can simply ignore in this assignment). Then follows a list of x y z coordinates, and then a list of indices. On each indices row, first the number of vertices specifying that polygon is given, which is always 3 in our case, followed by the three indices themselves (hence each row represents a triangle of the model).

Animation will continue as such until the user terminates execution. During execution, the user may change the object type, color, drawing mode and animation speed. Your program must handle user input from keyboard and mouse, and set the drawing modes as specified below:

Object type, drawing mode (wireframe or solid) and color must be set through pop-up menus created by GLUT library functions. You must create a hierarchical pop-up menu with three top-menus (object type, drawing mode and color) each popping up to related submenu entries.

- **Object type** -- set the current object to be drawn, one of the following two choices:
  - Sphere (to be loaded from sphere.off)
  - Car (to be loaded from car.off)
  - Cube (to be created in the code)
- **Drawing mode** -- wireframe (i.e., as lines) or solid mode
- **Color**-- set the current color in which to draw lines or triangles (the 8 colors at the corners of the color cube plus black will be sufficient)

Animation speed, termination and help functionalities should be controlled through keyboard handler GLUT library functions as specified below:

- **Arrow key** -- set the animation speed (up and down arrow)
- **i** -- initialize the pose (top middle of the window)
- **h** -- help, print explanation of your input commands (simply to command line)
- **q** -- quit (exit) the program

Your program must also properly handle the **reshape event**; so you must define your own reshape callback function.

Transformations (i.e., scaling up and down in our case) and projection have to be implemented in vertex shader. Since we haven't yet seen how to illuminate surfaces, when you display your objects in solid (filled polygon) mode, you'll be able to observe them only as silhouettes.

You can use the default orthographic projection (the viewing volume being the cube centered at the origin with sides of length 2). Note that since reshape callback function is invoked when the window is first opened at the beginning of program execution, it is a good place to put all projection-related settings (that then means you won't need to set projection in init() function).

Program Requirements

1. You are expected to design the program as an event-driven main application that responds to keyboard, mouse and reshape events. Thus, you should follow the main program and function module model given in your textbook and lecture notes.
2. Your program files must have documentation comments.
3. Your program will be graded for:
    - good programming **design**,
    - good programming **style**,
    - good program **documentation,** and
    - **correctness**.
4. You may be required to run your program and demonstrate that it works.

You are required to submit only the source files, i.e., the project files, ready to be compiled and run. Please try to comply with the announced due date. You can send the files via e-mail to your TA (and to me as a carbon copy). Ask for an acknowledgement of receipt when you have sent your files.

Implementation Hints

1. Detailed documentation for **OpenGL functions** are described in your textbook and OpenGL 4 Reference Pages. I strongly recommend you to read Chapters 1 and 2 from your textbook (E. Angel).
2. Use the **GLUT (GL Utility Toolkit) library** functions as described in your textbook. There is also a user manual at http://freeglut.sourceforge.net/docs/api.php.
3. Your program can be based on the example codes from your textbook. You can directly use the given InitShader.cpp code to load your shaders as well as the provided header files. You may use the spinCube example from lectures as a skeleton code to modify.
4. While the coordinates in the sphere.off file are already given within the range [-1,1], that is not the case in the car.off file, hence you will need to normalize them.
5. Note that the viewing volume (that you set by Ortho() function defined in mat.h) is eventually fit into the display window. So the top of the window matches the top of the

view volume, and similarly the bottom of the window matches the bottom of the view volume.

6. To switch between three object types (cube, sphere and car), you can use vertex arrays. A cube is easier to implement so you can start with that.
7. You can use glPolygonMode() function to switch between wireframe and solid type of object rendering.
8. You can use glutIdle() to get animation. With this function, it is not possible to have direct control on the frame rate of the animation. However you should be able to increase or decrease the speed of the animation by adjusting the scale factor. Another possibility is to use glutTimer() function to have more control on the speed of the animation, but that is not a must for this assignment.
9. Adjust the size, speed and positioning of the model properly (by playing with the state variables) so that animation looks nice.
10. You will need **global variables** that hold state values for your program, e.g., a code number for the current object type to draw, a code value for the current draw mode (wireframe or solid), etc. The callback functions (glutIdle() for instance) will simply update these global variables and post a redisplay. The **display()** function will read the current state to determine what object to draw, how to draw it, etc. Note, we are forced to use global data even though that is not good software design because of the way the GLUT library software designers specified the input callback functions.
11. The various drawing mode control options listed above (type of object, draw mode, color, etc.) should be handled by using these GLUT lib functions:
    - glutKeyboardFunc()
    - glutSpecialFunc()
    - glutReshapeFunc()
    - glutMouseFunc()
    - glutCreateMenu()
    - gluAddMenuEntry()
    - glutAddSubMenu()
    - glutAttachMenu()
12. Each of these functions must be called one time at program initialization with a parameter that is the name of your callback function. Your actual callback functions must have the appropriate prototype.
13. You are welcome to design your own user input technique for setting these state variable values with some other combination of key inputs or with mouse inputs (if you define a mouse callback function and call **glutMouseFunction()** or **glutMotionFunction()**).
14. To create pop-up menus, you will need to use glutCreateMenu(), gluAddMenuEntry(), glutAddSubMenu() and glutAttachMenu() functions. Although we will see how to use these functions in the next lectures, they are easy to use and the idea is very simple, so you can already start experimenting on them. I recommend you to read the relevant lecture slides and also Chapter 2.12 from your textbook.
15. Projection will be implemented in terms of matrix multiplication operations in vertex shader, and so is the scaling transformation. You can create the corresponding 4x4

matrices by using the functions given in mat.h file. These matrices should be created in the application and then sent to shader as uniform variables. For example, you can make use of Ortho() function that returns a 4x4 orthographic projection matrix (the viewing volume is specified through the function arguments).

16. **Avoid using OpenGL functionalities deprecated and removed by OpenGL 3.1** (you can actually use any function that is used in the textbook examples, and possibly many others). You can use GLUT although you may get deprecation warning depending on your platform.
17. Be creative and inventive!!! **Extra credit** will be given for use of different OpenGL functions in creative ways and for a clever design. Additionally you may want to add other control options or design and implement a better user interface.
