CS6250 Topics in Adv. Comp. Graphics

Programming Assignment #1 — Computer Science Department
3D Transformations/Viewing — Bowling Green State University
Fall 2009

30 points
Due date: Monday, September 21, 2009
Time: Electronic submission of program files due by 2:00pm on the due date
Printout (i.e., hardcopy) is due at the beginning of the class on the due date (must be stapled)

This program assignment is designed to help you gain or regain experience with 2D and 3D geometric transformations, perspective viewing, and windowing in an OpenGL environment. The program must be developed in C/C++ using the OpenGL graphics library with GLUT. No other libraries can be used, unless you received approval from your instructor.

Assignment: For this assignment, a program will be designed and implemented that allows rendering and manipulation of a 3D polyhedral object with 'n' vertices (e.g., the object could be a pyramid, a hexagonal prism, etc.). The program should generate wireframe drawings of the object. The program allows limited movement of the object. You are free to dress up the program with appropriate and aesthetically-appealing use of color. The program should be well-structured and easy to use (to allow for future changes).

3D Object Specification:

- The object description should be read from an ASCII text file (i.e., **cs6250_obj.dat**). The file is on the Blackboard. (You should try to test program on this and a couple of your own datasets.)
- The object should consist of vertices and connecting lines, with the vertex coordinates expressed in world coordinates.
- The format of the file is as follows.
  - The first line of this file will contain two integers. The first integer represents the number of object vertices (***k***), and the second represents the number of connecting lines (***l***).
  - The next line of the file should be blank.
  - The next ***k*** lines of the file contain vertex definitions. Each vertex is to be stored as three integer values representing the x, y, and z (world) coordinates of the vertex.
  - The last ***l*** lines of the file contain the connecting line definitions. Each connecting line is represented by two integer values. These integers represent the indices of the vertices that are connected by the connecting line. The first vertex is considered to have index 0 while the second vertex has index 1, etc.

Specifications:

- The following commands should be supported by your program. These commands should be processed using GLUT's keyboard input functions: do not use standard C/C++ input functions (e.g., scanf, cin). A "key" of these commands should be printed to standard out when the program starts.
  - **x**: move the object +10 units in the x direction
  - **X**: move the object -10 units in the x direction
  - **y**: move the object +10 units in the y direction
  - **Y**: move the object -10 units in the y direction
  - **r**: rotate the object by +30 degrees about its VERTICAL axis

        - For full credit for the rotation feature, the program should rotate the object about vertical axis and centroid. One way to estimate the position of the centroid is to use the mean of all of the vertex coordinates. Partial credit is available if the object is rotated about a vertex rather than its centroid. Be sure to document what your program rotates about! For simplicity, we'll just assume that the vertical axis of the object is in the y direction.
    - **p**: perspective/parallel projection toggle
        - The default viewing mode is parallel projection. The direction of the projection for parallel projection is perpendicular to the projection plane.
        - For all projections, you can assume that the CENTER of the screen is also the origin of the coordinate system that the object(s) are represented in.
    - **I**: input file
        - For option I, the user should be prompted for the name of the (3D object) file to input on the screen. The old object should be replaced by the new one.
    - **q**: quit
- For screen size, use a viewing canvas whose size is 450 units in the horizontal direction and 450 units in the vertical direction.
- Assume, for both projections, that the x-y coordinate system in the screen viewport is parallel with the x-y world coordinate system.
- The viewpoint is fixed in the system. You can assume that the center of projection is at (0, 0, 100) in WORLD coordinates.
- Assume that the view reference point is the center of the screen (which is coincident with (0, 0, 0) in WORLD coordinates), VUP is the world coordinate system's positive y axis, and VPN is the world coordinate system's positive z axis.
- We will assume an infinite view volume under both perspective and parallel projections.
- After the user enters any transformation, the program should compute the transformation, efficiently apply the transformations, and then display the transformed object on the screen.
- As soon as the user hits one of the command keys, the screen should be immediately updated.
- The screen update should be completed smoothly.

Extra point: If you want to add extra features to the program, a small number of extra credit points may be awarded (e.g., creative, appealing use of color or adding other options- s: object begins smoothly and continuously rotating about its vertical axis).

Structure and Documentation Note: The program should have a modular design and a reasonable amount of documentation. This means that major/important features, functions/methods, and data structures should be very clearly documented. Remember, you can reduce some documentation work that might otherwise be needed by careful choice of variable and function/method names. Object-oriented approaches must be very clearly described; careful choice of object names is not sufficient for other readers of your program to understand its structure. Programs are also expected to be reasonably efficient.

The initial comment section of the program should include a concise description of the architecture of your program. Your goal in the initial comment section should include providing a concise description of the program's methodology (including any critical functions and data structures) to the grader. Anyone reading the initial comment section should be able to very quickly understand the structure of the program.

Building and submitting the OpenGL program: Your program must include and use the *cs_456_250_setup.h* header file that is on the BlackBoard. Unless you are doing animation, you should use that file exactly as it is. If you want to use an object-based approach, you are free to minimally modify the *cs_456_250_setup.h* to work with your object-based framework. Please, no use of STL for C++ program.

You need to hand-deliver a print-out of your program's complete source code. *(Please have a separate cover page.)* In addition, you need to put your complete source code in the directory specified below on the CS server. In particular, create a Visual Studio Project under a work directory named **cs6250_prog1_yourlastname** (e.g., if you name is Tiger Woods, the directory should be named as **cs6250_prog1_woods**.) and then submit the entire folder. You are restricted to NO MORE THAN **TWO** header files (including *cs_456_250_setup.h*) and **ONE** C/C++ file.

Please check that your program works on the departmental Windows machine.

- Turn-in your softcopy of program for grading:
  1. Go to **Start → CS Classes** and login as **bgsulabs** for both username and password.
  2. Click on **lee\cs6250_turn_in**
  3. Drag your **cs6250_prog1_yourlastname** folder and drop it into the **"cs6250_turn_in"** folder**.**

  (For more details to access the CS Classes, see http://www.bgsu.edu/departments/compsci/docs/cs-classes.html)

```
 …

#include <GL/glut.h>
#include "cs_456_250_setup.h"

…

void display_func( )
{
	/*
		This callback function is automatically called whenever a window needs to be displayed or
redisplayed.
		Thus, usually this function contains all of the drawing commands; the drawing/animation
statements should go in this function.
	*/
}

#define canvas_width  123        // canvas width
#define canvas_height  456       // canvas height
#define canvas_name "NAME OF SCREEN CANVAS"          // name at top of the screen region

int main(int argc, char ** argv)
{
	glutInit(&argc, argv);

	init_setup(canvas_width, canvas_height, canvas_name);

	// window creation
	glutDisplayFunc(display_func);     // registers display callback function

	glutMainLoop( );                   // execute until killed

	return 0;
}
```