Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2011-2012

Assignment 2 - Object and Camera Transformations in OpenGL

Due date: 5 December 2011, Monday, 23:55

1 Objective

In this assignment, you will implement a demo scene in OpenGL. This demo scene will include translation and rotation transformations on both objects and camera, and scaling transformation on objects.

2 Specifications

You will render an object consisting of triangles using OpenGL. You will use the input file we provide as the only command line argument. Each line in the input file defines a triangle, you can parse them in the same manner as your previous homework, minus the "#Triangle" keyword. The color of the object, position of the lights, background color, ambient light color is up to you. You will implement two camera modes. In the first camera mode, the camera will be turning around the object to be rendered, while keeping the object always right in front of the camera. In this mode, by hitting some keys, you will be able to move, scale and rotate the object, in addition to turning camera movement on/off. In the second camera mode, the camera will move freely in the world. The object will stand still.

To render a scene in OpenGL, you need some basic initializations, i.e. setting up the lights, placing the camera, etc. Example initialization code is given below:

```
// any initialization before the main loop of GLUT goes here
   glEnable(GL_DEPTH_TEST);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_NORMALIZE);
glShadeModel(GL_SMOOTH);
// SET LIGHTING
  glEnable(GL_LIGHTING);
GLfloat position[] = {1000.0, 1000.0 , 1000.0, 0.0};
GLfloat blackColor[] = {0.1 , 0.1 , 0.1 , 1.0};
GLfloat whiteColor[] = {1.0 , 1.0 , 1.0 , 1.0};
glLightfv(GL_LIGHT0, GL_POSITION , position);
glLightfv(GL_LIGHT0 , GL_AMBIENT , blackColor);
glLightfv(GL_LIGHT0 , GL_DIFFUSE , whiteColor);
glLightfv(GL_LIGHT0 , GL_SPECULAR , whiteColor);
  glEnable(GL_LIGHT0);
// SET CAMERA
```

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(90, 1 , 1 , 1000000); // You may change parameters w.r.t. your needs.
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(1.0,0,-1.0,0,0,0,0,1,0); // You may change parameters w.r.t. your needs.
```

- The program will start in the first camera mode, named as "Ring Cam". Pressing '1' any time will also open up this mode.
- You will read an object's file name as command line argument. The object will consist of triangles.
- You will place the camera in a suitable position so that it will face the object, and the object will be visible in a reasonable size. The "gluPerspective" and "gluLookAt" functions' parameters in initialization (and in sub-modes where you will change where you look at) will be defined by you. Please check out definitions of these functions.
- In the first camera mode, the object will be motionless in the beginning. The camera will be making a circular movement around the object, while the object is at the center of the screen at any given moment. Imagine it as the moon being on its circular orbit around the world, while same side is always facing the earth.
- Pressing 'Space' in Mode 1 will put the camera and object in its initial configuration.
- Pressing 'h' in Mode 1 will stop the camera from moving. Pressing 'h' again will cause the camera to continue its movement, from the place it stopped.
- As long as you press the button 'z', the object will do harmonic vertical movement. The object will periodically move up and down in a certain range. The ranges and speed are of your choice. Make sure the object stays visible, and the motion is smooth.
- As long as you press the button 'x', the object will do harmonic horizontal movement. The object will move periodically to the left and to the right. The ranges and speed are of your choice. Make sure the object stays visible, and the motion is smooth.
- As long as you press the button 'c', the object will rotate. You may choose to rotate it around its center of gravitiy, or another point or axis in your 3D space. Make sure the rotation movement is clear, smooth and the object stays visible all the time.
- As long as you press the button 'v', the object will periodically enlarge and shrink in all directions. You will apply scaling transformation on the object. The maximum size and minimum size is up to you. Make sure the transformation proceeds smoothly, and the object stays in visible area.
- The requirement for the object to stay in visible area is valid only if the object is in its initial configuration when the transformation starts. The 'z' - 'x' - 'c' - 'v' sub-modes are mutually exclusive, i.e. object should not be rotating and scaling at the same time. You do not need to implement multi-keyboard-input case for sub-modes. Note that resetting configuration by pressing 'Space' will get the camera and the object to its initial configuration (such as position, size, etc.). However, selecting another sub-mode after one sub-mode is completed will not move the object to its initial configuration. The object will stay where it is, while still keeping its position, scale and rotation. The new transformation is applied on this modified object. In the case of back-to-back sub-modes, you need not keep the object visible. Just apply the transformation as you would if the object was in its initial configuration.

- There is another camera mode you should implement. This mode is free-flying mode. The camera will be able to fly to any direction. The object will stay wherever it is. Pressing '2' any time the program is running will open up this mode.
- Pressing 'Space' will move the camera to its initial configuration (with default being the same with the first camera mode), and object to its default configuration. This configuration of the object is the same initial configuration in first camera mode.
- The camera has a default position, viewing vector, and up vector. You will define all these. The camera should be facing the object (only if the object is in its initial configuration).
- 'w' will move the camera forward in its viewing direction.
- 's' will move the camera backward in its viewing direction.
- 'a' will slide the camera left, while keeping its up vector and viewing vector the same.
- 'd' will slide the camera to the right, while keeping its up vector and viewing vector the same.
- 'Up' will rotate the camera's up vector and viewing direction vector in counter-clockwise direction with respect to right side vector of the camera, while keeping its position. In effect, you look up.
- 'Down' will rotate the camera's up vector and viewing direction vector in clockwise direction with respect to right side vector of the camera, while keeping its position. The effect is that you look down.
- 'Right' will rotate the camera's direction vector and right side vector in clockwise direction with respect to the up vector. You end up looking to the right from the same position, with the same up vector.
- 'Left' will rotate the camera's direction vector and right side vector in counter-clockwise direction with respect to the up vector. You end up looking to the left from the same position, with the same up vector.
- 'r' will rotate the camera's up vector and right side vector in counter-clockwise direction with respect to the direction vector. You will be looking at the same point, with your face leaning right.
- 'e' will rotate the camera's up vector and right side vector in clockwise direction with respect to the direction vector. You will be looking at the same point, with your face leaning left.
- Rotating clockwise with respect to a vector means that, if the camera is at the origin, and you are at a point on that vector(positive side), the rotation is on clockwise direction.
- You may place your camera, direction and distance to object based on some features of the input object, such as center of gravity, and size in three dimensions.
- You will decide how fast the camera moves forward, or turns left, or looks right, etc. The only requirement is that the camera moves at a reasonable speed, and in a smooth manner. As long as you press the movement/rotation button, the behavior continues.
- Your program should quit if the user presses 'q'.
- Your program should be able to handle multiple keypresses at the same time. For example, in the free-fly mode, you should implement a game-like flying cam, where you can go forward and turn right simultaneously.

3 Notes

- You should use C++ and OpenGL library.
- We will test your codes on departmental machines using "g++". Please make sure to run tests on ineks.
- A sample run of your program is " '$ ./hw2 bunnylowscene.txt".

4 Submission

Submission will be done via COW. You should upload a single zipped file called "hw2.zip", which includes your source code and a makefile. The executable output of your program is called 'hw2'.

Warning: Late submissions are allowed for this homework, based on the policy on the course web site.

5 Grading

Grading will be made using the scala on the course's website.

6 Cheating Policy

We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations. See the course website for more information.