CPSC 314, Project 2: Terrain Navigation

Out: Thu 19 May 2005
Due: Wed 25 May 2005 11:59pm PST
Value: 9% of final grade
Points: 100

In this assignment you will implement two different ways of flying around above a colored ground plane. There are up to 5 extra credit points available.

Terrain [10 pts]

You need to make a ground plane of a mesh of many small triangles with a random color for each triangular face. Your terrain should be a grid of $100x100$ vertices in the xz plane, with the corners of the terrain at (0,0,0), (100,0,0), (100,0,-100), and (0,0,-100).

Terrain Hints:

- You will need to store a color for each face, since each face has a random color but that color should persist between frames. Consider what kind of data structure you need to store these colors. You should initialize the data structure with the random colors at startup time, and then traverse it each time you draw a frame.
- The easiest way to specify triangles in OpenGL is with the GL_TRIANGLE mode.

Navigation [90 pts]

You will implement two different ways of flying around your scene: absolute camera movement, and relative camera movement. Absolute camera movement is easier to program, but you'll probably find that relative camera movement is a lot easier to use! Start with a camera that has an eye point of (50, 10, 10), looking at the point (50,0,-75), and the y axis is up. Hitting 'r' should reset the camera to this default view. Hitting 'm' should switch between the two modes of movement and also reset to the default view.

Absolute Camera Movement [20 pts]: You should implement absolute camera movement where you control the x/y/z coordinates of the eye point, lookat point, and up vector with keyboard keys. You'll increment a value with a lowercase key, and decrement it with a uppercase key as follows: eye x: 'q'/'Q', eye y: 'w'/'W', eye z: 'e'/'E', lookat x: 'a'/'A', lookat y: 's'/'S', lookat z: 'd'/'D', up x: 'z'/'Z', up y: 'x'/'X', up z: 'c'/'C'. (Those are the keys on the top, middle, and bottom rows of the keyboard on the left side, so that you can hit them all without moving your hand.) The '+/-' keys should increment and decrement your speed in absolute camera coordinate keyboard motion.

Relative Camera Movement [70 points]: You should implement camera movement where you fly around the scene by incrementally controlling forward/backward speed, roll, pitch, and yaw with mouse drags. All of this motion should be calculated with respect to the current camera coordinate system. Pitch is rotation around the horizontal axis, like nodding your head up and down for yes. Yaw is rotation around the vertical axis, like shaking your head from side to side for no or steering a car. Roll is rotation around the front-to-back axis, like tilting your head so your ear touches your shoulder (like the up vector in the lookat model). There's a good animated illustration at http://www.nasm.si.edu/galleries/gal109/NEWHTF/PITCH.HTM
When a mouse button is held down you move, and you stop moving when the button is lifted up. The vertical component of the vector created by a left mouse drag controls your forward or backward speed (up is forward and down is backward), and the horizontal component controls your yaw angle (left is yaw to the left, and right is a yaw to the right). You should keep track of the position at the start of a mouse drag, and consider the size of the vector from that start position to the current mouse position. Dragging further away from the starting point makes you go faster or turn more, and dragging it back towards that point slows you down and makes you turn less. Releasing the mouse button stops the motion. The right mouse similarly controls roll with horizontal drags and pitch with vertical drags. You'll need to experiment with how to set the weights when converting drag vectors into motions, in order to make flying feel natural instead of jerky.

Navigation Hints:

- Remember that you'll have to flip the y coordinate, since the window system will be sending you coordinates that start at the upper left instead of the lower left.
- Remember that viewing transformations belong in the modelview matrix, not in the projection matrix. See Steve Baker's projection abuse article at http://sjbaker.org/steve/omniv/projection_abuse.html.

- The relative motion specification requires incremental changes of roll/pitch/yaw angles and forward/backward motion with respect to the current camera coordinate system. You could imagine keeping track of cumulative roll/pitch/yaw values with respect to some set of basis vectors kept in world coordinates, but that would require a lot of calculation. And transforming roll/pitch/yaw angles into the eye/lookat/up vector format required by gluLookat would be even more work.

  In contrast, if you assume that you know the current camera coordinate system (let's call it Current), it's easy to calculate the simple new incremental motion, where a drag means a simple motion with respect to the current x, y, or z axis of Current. This new incremental transformation (let's call it Incremental) needs to be applied with respect to the current transformation; that is, p' = Incremental * Current * p. The good news is that Current is exactly the modelview matrix used by OpenGL to draw the previous frame. If you don't wipe that out with glIdentity, that matrix is still intact and contains the information you need. However, OpenGL only allows you to postmultiply a matrix, which would result in the incorrect operation p' = Current * Incremental * p. The trick is to first explicitly store the Current matrix, which you can do with the `glGetDoublev` command that dumps out the contents of the top of the OpenGL matrix stack. Then you can get the desired effect by wiping the stack with glIdentity, first applying the incremental transformation, and finally multiply by the stored Current. This trick saves you a lot of work by using the OpenGL matrix stack as both a calculator and storage device!

Extra Credit [5 pts]

Implement relative motion the hard way, by keeping track of cumulative roll/pitch/yaw values with respect to some set of basis vectors kept in world coordinates.

Template

You can use the same template as for project 1. You should comment out the line `glEnable( GL_LIGHTING );`. Also note that, sadly enough, the project 1 template incorrectly places viewing transformations in the projection stack. You shouldn't do this!

Handin/Grading/Documentation

The grading, required documentation, and handin will be the same as with project 1, except for two changes. First, use the command 'handin cs314 proj2'. Second, there is no need to submit image files since there will not be a Hall of Fame for this project. Stay tuned for the ultimate Hall of Fame competition with Project 4!