CPSC 314, Project 2: Navigation

Out: Mon 11 Feb 2008
Due: Fri 29 Feb 2008 6pm PST
Value: 8% of final grade
Points: 100

In this assignment you will implement four different ways of changing your viewpoint. There are up to 5 extra credit points available. The template code has a starting camera position with 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 any of the '1' through '4' keys should switch between the modes of movement, and also reset to the default view.

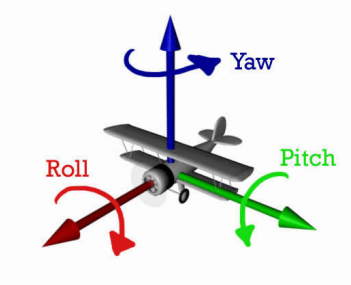
1. Absolute Rotate/Translate Keyboard [10 pts]: Directly control the absolute camera position and orientation with rotation and translation. This mode is easy to program, but hard to use, especially after your orientation changes so that absolute directions do not line up with your current view. You'll increment a value with a lowercase key, and decrement it with an uppercase key, as follows:

- translate forward/backward in World x with 'x'/'X'
- translate forward/backward in World y with 'y'/'Y'
- translate forward/backward in World z with 'z'/'Z'
- rotate forward/backward along the World x axis with 'a'/'A'
- rotate forward/backward along the World y axis with 'b'/'B'
- rotate forward/backward along the World z axis with 'c'/'C'
- increase/decrease speed (the increment moved by a keypress above) with '=/-' keys

2. Absolute Lookat Keyboard [10 pts]: Directly control the eye point, lookat point, and up vector. You'll increment a value with a lowercase key, and decrement it with an uppercase key, as follows:

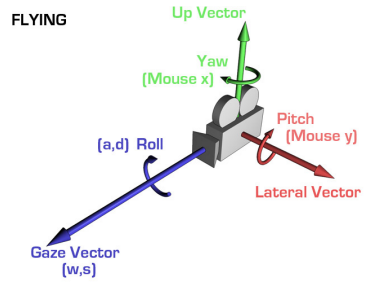
- eye point x with 'x'/'X'
- eye point y with 'y'/'Y'
- eye point z with 'z'/'Z'
- lookat point x with 'a'/'A'
- lookat point y with 'b'/'B'
- lookat point z with 'c'/'C'
- up vector x with 'd'/'D'
- up vector y with 'e'/'E'
- up vector z with 'f'/'F'
- increase/decrease speed (the increment moved by a keypress above) with '=/-' keys

3. Relative Flying [50 pts]: Use mouse drags to control your pitch, and yaw. Use the keyboard to control your roll and forward/backward speed. All of this motion is calculated relative to the current camera coordinate system; that is, with respect to the view you see through the display window. So you can only move forward/backward in the direction that you're looking, like flying a plane.



Yaw is rotating horizontally with respect to your current position, like steering a car or shaking your head for no. Pitch is rotating vertically with respect to your current position, like nodding your head up and down for yes. Roll is tipping left or right by rotating around your current front-to-back axis, like tilting your head so your ear touches your shoulder. See also the animated illustration at http://www.nasm.si.edu/galleries/gal109/NEWHTF/PITCH.HTM

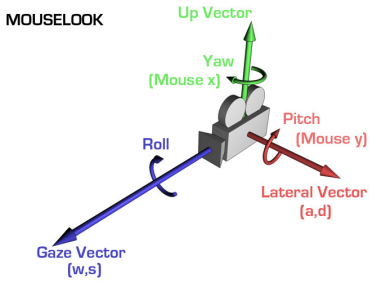
While the mouse button is held down your viewpoint changes, with the speed controlled by the size of the drag vector. In this mode, the size of a mouse drag controls how **fast** you turn.
Keep track of the position at the start of a mouse drag, and consider the size of the vector from that start position to the current mouse position. Dragging further away from the starting point makes you turn faster, and dragging back towards that point slows you down so that you you turn slower. The motion stops when you releasing the mouse button. Use the left button for yaw and pitch. The horizontal component controls your left/right yaw angle, and the vertical component controls your up/down pitch angle. Your roll angle, and forward/backward motion, are controlled with the keyboard. You may need to experiment with weightings when converting drag vectors into motions, in order to make flying feel natural instead of jerky. You hold in a particular position to get a constant turning speed: as long as you hold down the mouse, you'll keep turning.



To summarize:

- change of yaw speed wrt local Camera coordinates with left drag horizontal component
- change of pitch speed wrt local Camera coordinates with left drag vertical component
- roll wrt local Camera up vector with 'a'/'d' keys
- forward/backward motion wrt local Camera z with 'w'/'s' keys
- increase/decrease speed (the increment moved by a keypress above) with '=/-' keys

4. Relative Mouselook [30 pts]
In this mode, all motion is also with respect to the current camera coordinate system. Like in the previous mode, you move forward/back with respect to that coordinate system. There are two differences: first, instead of rolling, you move laterally from side to side with the 'a'/'d' keys and up and down with the 'k'/'l' keys. Mouselook is like a helicopter where you can travel in any direction (up/down, right/left, up/down).
Also, the mouse movements control your orientation differently: they directly control how **far** you turn, as opposed to the flying mode where they control how **fast** you turn. Moreover, you don't hold down a key to drag, you move the mouse freely and check the displacement of the mouse with respect to the center of the screen. After changing the orientation of the camera accordingly, you warp the cursor back to the center. You only turn when you're moving the mouse: when it is still, you are no longer changing your viewpoint.



- change of yaw with left drag horizontal component
- change of pitch with left drag vertical component
- forward/backward motion wrt local Camera z with 'w'/'s' keys
- left/right motion with wrt local Camera coordinates with 'a'/'d' keys
- up/down motion wrt local Camera coordinates with 'k'/'l' keys
- increase/decrease speed (the increment moved by a keypress above) with '=/-' keys

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 flying mode 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 global basis vectors (for example, 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!

- In flying mode you only need to keep checking the mouse position when the button is being held down. While the button is down, continuously rotate. In GLUT, the mouse callback to bind for this behavior is `glutMouseFunc()`. In flying mode, the current size of the drag vector with respect to the starting location matters. You can change that vector size to control the speed of your motion, for a constantly changing vector size.

  In mouselook mode, whenever the mouse moves, rotate the camera (i.e. when the passive mouse call is triggered, a single motion will result whose size depends on the amount of displacement). In GLUT, the mouse callback to bind for this behavior is `glutPassiveMotionFunc()`. After that specific motion is handled, warp the cursor back to the middle of the screen. The `glutWarpPointer()` function will do this warp for you. Note that this call will trigger `glutPassiveMotionFunc()`: beware of excessive recursion! Hint: consider keeping track of whether you've just warped...

Extra Credit [5 pts]

Implement relative flying 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

The template code starts with the camera in the default position, and provides a very simple terrain for you to fly around. Download from `http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007/proj2.tar` and use this command to unpack it: `tar xvf proj2.tar` You will see four files: `p2.cpp`, `Makefile`, `Vec3f.cpp`, and `Vec3f.h`.

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 p2'. 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!