Assignment 3: Camera

Written Homework Due: Friday, March 27 (9:30 AM)
Programming Project Due: Friday, March 27 (5:00 PM)

1 Introduction

For this assignment, you will be filling in `Camera`, which is a class that provides all the methods for almost all the adjustments that one could perform on a camera. It will be your job to implement the functionality behind those methods. Once that has been completed, you will possess all the tools needed to handle displaying three-dimensional objects oriented in any way, and viewed from any position. Your camera class will be used for the rest of the class.

2 Demo

When you start the demo and pick the camera lab you will again see the standard control panel and display. The display will show a grid of 9 cubes and a set of axes. The X axis is red, the Y axis is green, and the Z axis is blue. These axes represent the world's coordinate space.

The controls have a set of "canned" camera moves. Below we use "at point" to indicate a point along the look vector of the camera.

- Fixed position: The camera is positioned at $(3, 2, 6)$, looking at the origin, and with up set to $(0, 1, 0)$. This is the default start location. Click Play/pause to force a reset.
- Eye position: The eye does a sinusoid pattern while spinning around the Y axis, always looking at the origin. (The eye point and look vector change, but the at point and *input* up vector do not.)
- Focus point: The eye point is fixed on the positive Y axis, and the at point is spun around, looking down at the plane. The at point and look vector change, the *input* up vector does not.
- Camera zoom: The camera zooms in and out. Just changes the height angle.
- Roll: The eye and at point are fixed, the up vector rotates through the circle.
- Near and Far: The near and far planes are moved towards and away from the camera, causing the scene to be "wiped" in and out.
- Interactive: Allows the user to move the camera around using keyboard controls (see below)
- IBar: Allows the user to change the camera through an interactive widget (see Extra Credit section)

You can print the current world to camera and projection matrices at any time.

When you are in interactive mode, the window will listen for certain key presses and make appropriate calls to the camera class to change its position and orientation. The keys are as follows:

- W/S: Move forward/backward
- A/D: Step left/right
- Left/Right (arrows): Turn left and right (yaw)
- Up/Down (arrows): Move up and down (vertically)
- PageUp/Down: Tilt the view up/down (pitch)

These controls should be fairly intuitive if you have played first person games like Quake. Feel free to change the default controls if you find a different setup more natural. Make sure you document your changes though, or I might think you are missing part of the assignment!

3 Requirements

For this assignment you must implement a fully functional subclass of `Camera`. This involves the following:

- Setting the camera's absolute position or orientation given a new eye point, look vector (or at point), and up vector.
- Setting the camera's height angle.
- Setting the near and far clipping planes.
- Changing the width and height (which changes the aspect ratio).
- Having the ability to, at any point, spit out the current eye point, camera to world, world to camera and projection matrices.
- Moving or rotating by a specified amount (for user interaction).

4 Support Code

The method names are mostly self-explanatory. Again, the GUI is set up for you and all you have to do is fill in the methods in `Camera.h`. Note: because of the way OpenGL computes shading, what's returned from `getWorldToCamera()` and `getCameraToWorld()` will not be exact inverses of each other. Specifically, the `getWorldToCamera()` method returns the world to camera matrix except the scaling part (e.g., just translation and rotation). The `getCameraToWorld()` returns the inverse of the complete world to camera matrix (e.g., rotation, translation, and scaling). In addition, the `getProjection()` routine returns the perspective matrix plus the scaling part in the world to camera transformation.

5 Extra Credit

Add the aspect ratio, center of projection, and skew to your camera. These can then be edited using the IBar. Check out the demo first. The IBar represents the edge of a cube projected onto the screen. By changing the shape of the projection you change the camera. The IBar is labelled for you with what action is mapped to which limb. More details are in the supplementary handout. Note that the mappings of the limbs to the actions is slightly different than described in the handout. Feel free to rearrange these if you want, but make sure you document any changes, and fix the labelling to reflect your changes. Note that you will have to change the `rotateAroundAtPoint` routine also to correctly handle the center of projection change.

For other extra credit, try implementing an orthographic camera (pretty simple), or an oblique camera (a bit harder).