Lab 4 - Transformation Matrices

You may work in pairs on this assignment. To receive credit, demonstrate your completed program to me by Feb 27th.

In this lab, you'll learn how to combine transformation matrices to move objects around in 3D space. In the previous lab you implemented an orthographic projection matrix. In this lab, the orthographic projection matrix is provided and you'll be changing it to a perspective projection matrix. You'll also learn about model and view transformation matrices that are often used to move objects and cameras around to compose a scene. A unit cube, centered at the origin, is provided with its faces set to red, green and blue using color attributes like you've used in previous labs. By default the red faces are on the x-axis, the green faces are on the y-axis and the blue faces are on the z-axis. You'll be transforming the cube using translation, rotation and scale matrices that you'll construct from the values set on corresponding sliders. Lastly, you'll be constructing a view matrix using values from two sliders. All three matrices will be combined in your vertex shader to transform the points in your scene.

Part 1 - Perspective Projection

Construct a perspective projection matrix. Given the aspect ratio of your window, the desired field of view (FOV), and a near and far plane, create a 4x4 matrix that will be passed to your vertex shader as the projection matrix. Pick values for the FOV and near and far plane that allow you to see most of the grid (if not all of it). You're not allowed to use the glm::perspective function.

Part 2 - View Matrix

There are two sliders that will control the position of your camera. The camera should be positioned around a circle of radius 20 using the angle slider. The camera Y slider should position the camera along the Y axis between -20 and 20. Construct a view matrix such that the camera is focused on the origin. Update the vertex shader to accept a uniform variable for your view matrix and use it correctly to transform your vertex data.

Part 3 - Model Matrix

Use the translation, rotation and scale values from the sliders to construct a model matrix that will be passed to your vertex shader to adjust the cube's transform. You're not allowed to use the glm functions, translate, rotate or scale. Update your vertex shader to accept a uniform variable for the model matrix and use it correctly to transform your vertex data. Make sure that the scale and rotation are performed relative to the center of the cube.

Things to notice

Notice how when we switched from an orthographic projection to a perspective projection, lines that were once parallel may not be any more. Objects also get smaller as they get further away. This approximates how we see the real world and is great for 3D games or viewing a scene in a 3D editor. Orthographic projections, on the other hand, are great for engineering or modeling environments when we want to focus on a front, top or side view. After this lab is completed, feel free to use glm::ortho and glm::perspective to construct your projection matrices.

Our 3 rotation values are called Euler angles and are common in 3D modeling and animation software, but they have their problems. Drag the Y rotation to 90 degrees. Now try dragging the X and Z rotation sliders. Notice how they both now rotate the cube on the same axis. By rotating around Y 90 degrees, we've made the X and Z gimbals parallel, so we lose a degree of freedom. This is called gimbal lock. Gimbal lock artifacts can happen when animating Euler angles. Quaternions can be useful in overcoming gimbal lock.

Notice how the provided code includes glEnable(GL_DEPTH_TEST) and how the glClearColor also clears the GL_DEPTH_BUFFER_BIT. The depth buffer is used so objects that are closer to the camera cover objects behind them. When rendered, the depth value of the object is written to a separate buffer. When the depth test is enabled, each fragment tests whether its current depth is less than the one currently in the depth buffer. If it is, then it renders the fragment, otherwise it discards it. This behavior can be changed using the glDepthFunc.

Recommended Reading

Perspective projection
Another perspective matrix derivation
Model, View and Projection Matrices
Transformation Matrix
Vector and Matrix Math - Interactive Demos
Translation Matrix
Scale Matrix
Rotation Matrix