Computer Graphics – Assignment 2

**Objective**

The objective of this assignment is to help you understand how matrices work in conjunction with 3D mesh data. Before we only worked in a 2 dimensional space, but now we’re moving on into 3D space. In order to see objects as they would be in the real world, we need to involve 3 different matrices that serve different purposes.

First let’s talk about the three matrices and what their purpose are. A *world* matrix is used for objects to dictate where they are in Cartesian space. This matrix can include rotation, scale, and translation information. Each object in your scene will have their own *world* matrix. Second matrix we will discuss here will be the *view* matrix. This matrix acts like a camera in your scene. This *view* matrix has a position and a direction of where to look. Lastly, we have a *projection* matrix. This matrix helps us see objects as if they were distant from us by shrinking or growing the items in our scene. In essence this matrix is what gives us the feel of real 3D by adding perspective to our scene.­ Keep in mind that all of these matrices will be a 4x4 matrix.

1. First we should start by creating the vertex information for our grid and our cube. We need to draw lines so take a few minutes to think about how many total vertices we will need for our objects. The grid will be 10x10. Meaning we will have 11 horizontal lines and 11 vertical lines. I shouldn’t have to go into a lot of detail about the cube, but if you are still confused on the total amount of lines needed to draw a cube then I suggest drawing this out.
2. Keep in mind we would like for the grid to be placed around the origin (0, 0, 0). For the cube, make sure to create it around the origin as well but we will be translating this upwards by half of its height.
3. The grid will have a total width of 1 and a total depth of 1. (Hint: If we are placing this directly in the middle of our screen, where would the left and right side of our grid be located)
4. The cube will have a total width of 0.5 and height of 0.5 and lastly a depth of 0.5.
5. Once we have our geometry, the next step would be to create the *world* matrices for each. The *world* matrix for our grid can be the identity. No translation, rotation, or scaling for this matrix needed. We simply want the grid to be drawn at the origin.
6. The *world* matrix for our cube will be translated upwards by half of its total height. We will also ask of you to rotate the cube over time on its y-axis.
7. Integrating the *view* matrix is next. Create a 4x4 rotation matrix rotated -18 degrees on the X axis. Then translate this matrix backwards along the local Z-axis -1 unit.
8. Once our camera matrix has been fully built in World Space, we will need to inverse this matrix (temporarily) in order for it to be a proper *view* matrix. (Reverse motion)
9. Once we have our *view* matrix involved the next matrix we need to create will be our *projection* matrix. For our *projection* matrix our Near Plane is at 0.1, our Far Plane is at 10.0, the Vertical Field Of View is 90 degrees and the Aspect Ratio is based on your window’s dimension.
10. With all of our matrices created its time to create a vertex shader that will implement all three of these. We need to multiply our vertex by each matrix in this specific order. VertexW = VertexL times World. VertexV = VertexW times View. VertexP = VertexV times Projection.
11. Once all three matrices have been multiplied in, the last step is to do the perspective divide in order to see “depth” in our scene.

**Grade Breakdown**

The grading breakdown for this assignment will be split up into 4 portions. Each portion is based on reaching specific steps in the above instructions.

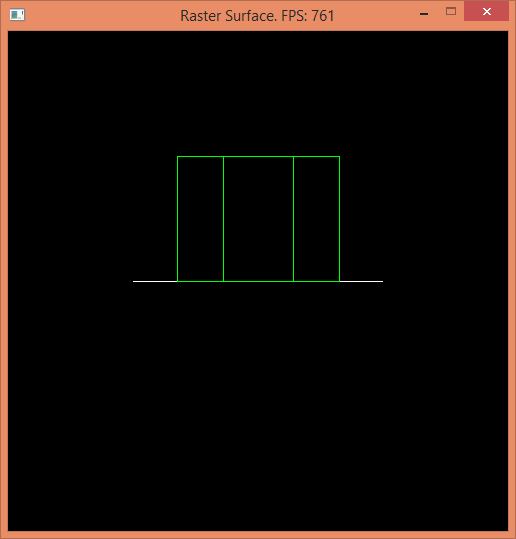
* 25% - Cartesian space geometry. We are no longer working in 2D space
* 50% - Rotate the cube
  + This can be done by step 6 and you will need to rotate the vertices in your shader
* 75% - Working view matrix
  + This can be done by step 7 and you will need to include the multiply in your shader
* 100% - Working projection matrix

**Example Images**

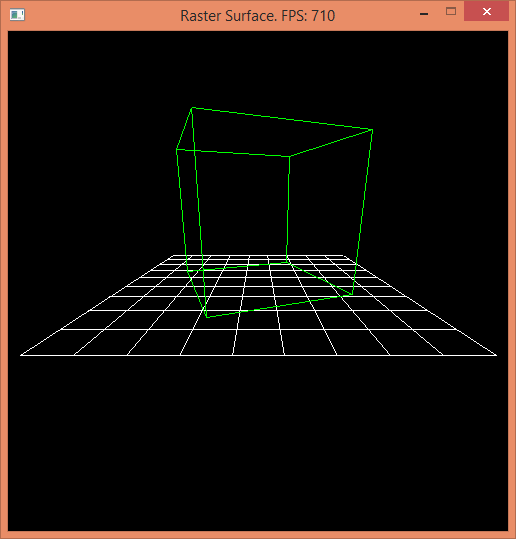
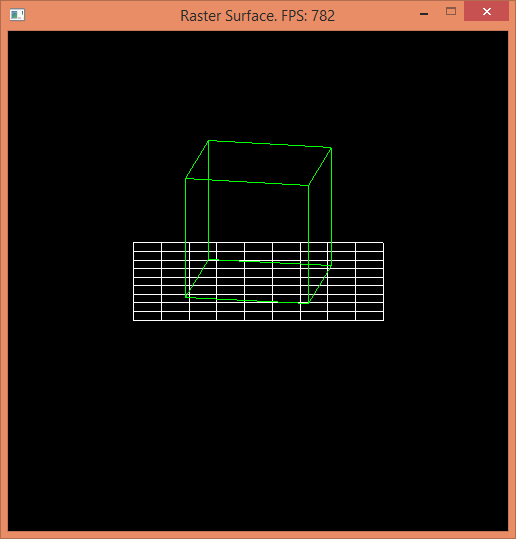
These are example images of what you should be seeing as you progress through today’s lab.

**25% 50%**

A picture containing chart

Description automatically generated****

**75% 100%**

****

**Additional Challenge**

If your code is structured to properly utilize the vertex shader concept shown in the lecture slides, then it should be a straightforward process to draw a second cube somewhere else in the scene. You already have the vertices and drawing code for a cube, so if you draw a second cube, but tell the vertex shader to use a different world matrix, then you can get a second cube in the scene using all of the same code. So, try making a new world matrix, with a different set of rotation and translation operations, you can even include scaling to shrink or expand the cube, then apply that world matrix when drawing the second cube. Make sure the original cube is still being drawn with the other world matrix so that both cubes are placed in different positions. Initially, you will probably want to make small changes so that you can be sure the cube is positioned in front of the camera, so that it shows up, but once you can see your second cube, then you can more easily make adjustments to reposition the cube.

# Submission

Programming assignments will be turned into FSO. Follow the directions listed there carefully, failure to do so can significantly impact your grade in a negative way. If you did not entirely write your own code for this assignment, don’t turn it in. Weekly MOSS results will be posted to the Discord if available.