Homework 3

Austin Kreulach
CAP5705 - Computer Graphics
University of Florida

October 29, 2021

1 Functionalities and Renders

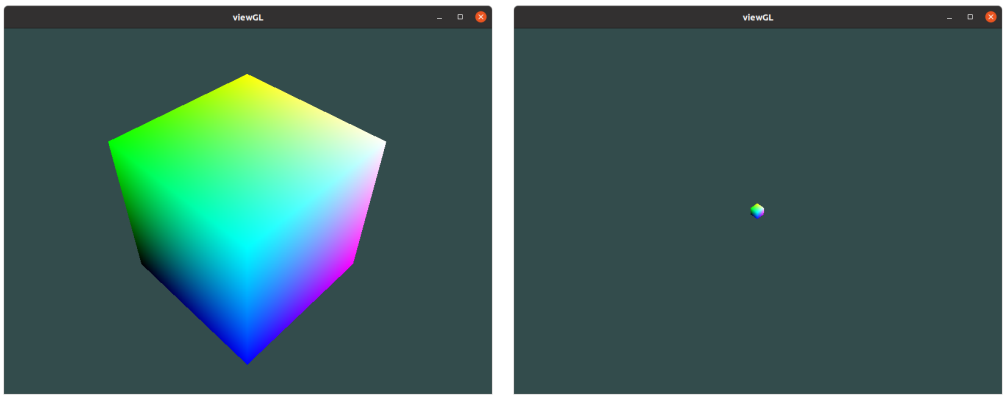

Figure 1: First functionality, scaling

As seen in the above image, the Rainbow Cube can now become larger and, in the second image, smaller. Scale was successfully implemented. It is controlled by the UP and DOWN arrow keys. This was done by including a Model View Projection matrix into the vertex shader and modifying it in the input function called in each loop. The UP and DOWN arrow keys increase or decrease a scale parameter in the Model matrix.

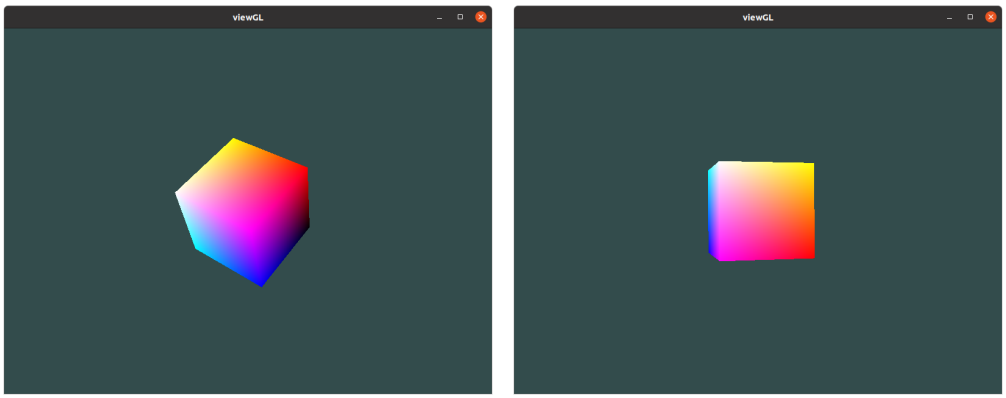

Figure 2: Second functionality, rainbow cube can now rotate.

In these renders you can see that the Rainbow Cube can now also rotate. Two arbitrary rotations were chosen for display here. **Bonus functionality**, there's really no good way to show that this is done via mouse movement, but it is. Feel free to confirm by running the attached code. I included the mouse movement in the same input function as the keyboard input for scaling. I check the mouse position each frame and update two rotation angles based on how far from the center the mouse was moved, then move the mouse back to the center. This does mean that if you tab away the cube rotates wildly as the mouse position is still being read but it is no longer being reset.

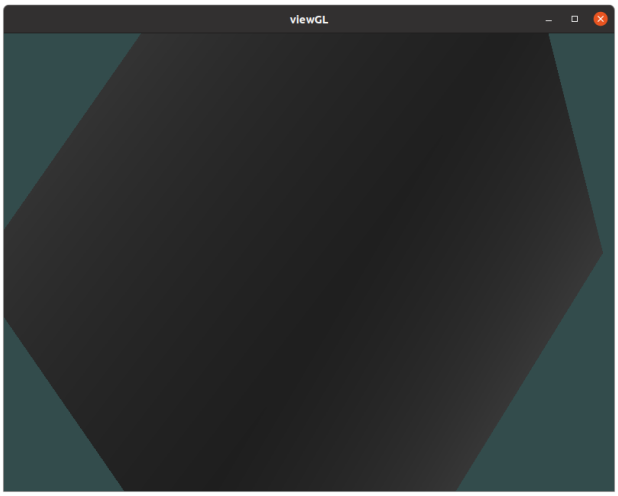

Figure 3: Third functionality, implementing Z buffer by showing Z value as grayscale coloring. This image shows an incorrect implementation.

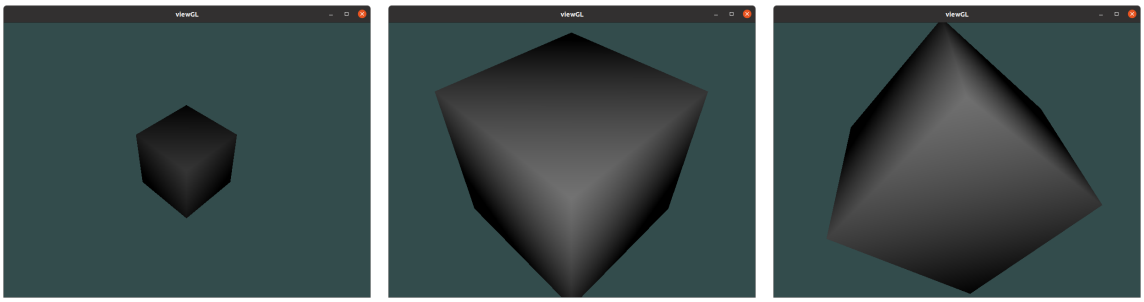

Figure 4: Third functionality, correctly implemented Z buffer grayscale.

Next I implemented the grayscale depth-buffer. I achieved this by defining a far plane at $z = 5.0$ and dividing the Z coordinate of a pixel by this 'plane' in the fragment shader and assigning the value to all elements of the color vector. This originally left distant pixels light and close ones dark, but I chose to invert this by subtracting the result from $1.0f$. You can see the values are correct across rotations by comparing the second and third images. You can see the values are correct across scaling by comparing the first and second images.

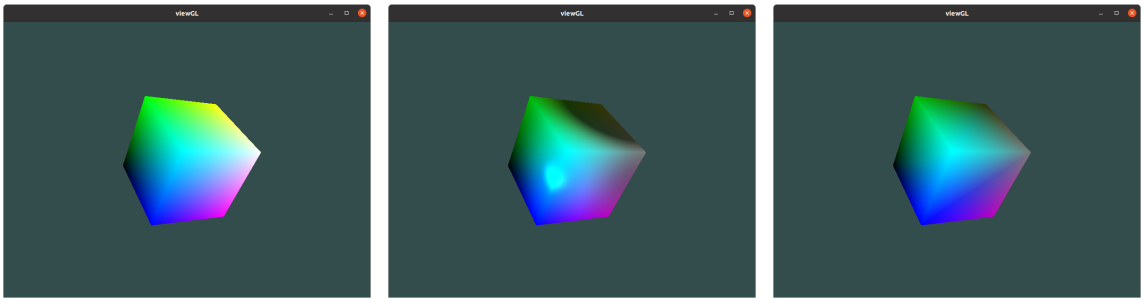

Figure 5: Fourth functionality, correctly implemented Phong and Gouraud shading.

Shown here is the Rainbow Cube under true color, Phong shading, and Gouraud shading from left to right. Note the very strong specular highlight in the Phong shading inside a triangle, and note the somewhat visible border between the two triangles forming the bottom right face under Gouraud shading. I take these as evidence that I implemented both correctly. Phong shading followed the schema from class, with a light being placed back and to the left of the camera.

When I implemented the two new shading styles I decided to create four separate shader files, load them all, and switch between them at the will of the user. Pressing Z loads true color. X loads the grayscale Z-buffer. C loads Phong shading. V loads Gouraud shading. All other functionalities continue to work when switching between shaders.

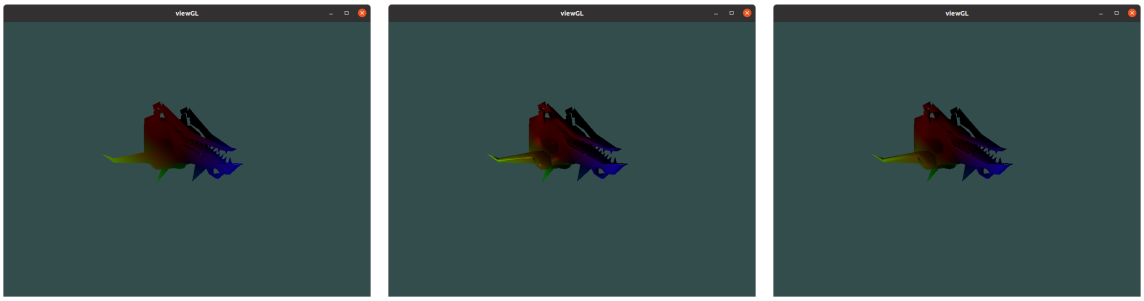

Figure 6: Fourth functionality, correctly implemented Phong and Gouraud shading.

Shown here is the Rainbow Dragon Skull under true color, Phong shading, and Gouraud shading from left to right. The colors are generally darker here so I rotated the skull to show off the yellow-greenish horn. If you zoom/focus on the horn you can see that in the true color it is noticeably duller than under Phong where the specular highlights are very strong and Gouraud has much softer specular highlights and more noticeable borders between triangles. Once again, these I take as evidence of successful implementation. Also note that I did, in fact, load a second mesh.