Kayleigh Kinsey

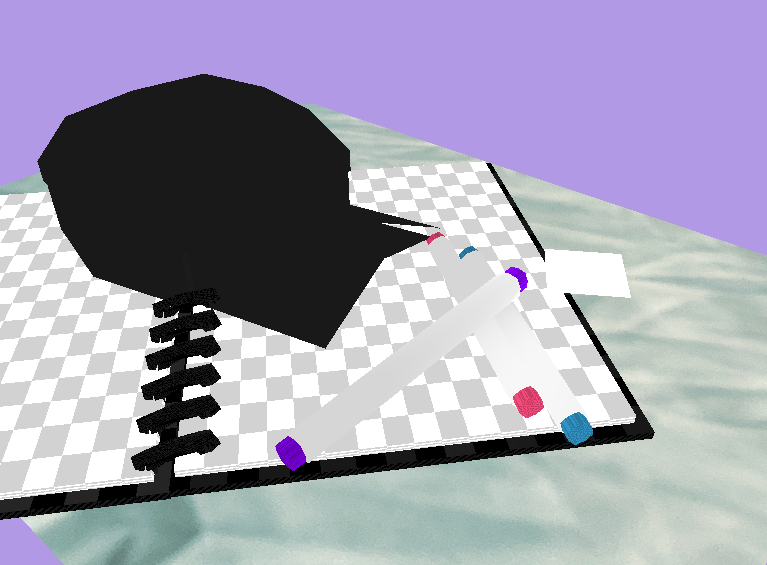
12/10/23

Comp Graphics and Visualization

Here is the image I used as a reference for my project:



And here is how the final scene turned out:



**Development Choices**

I chose the objects I did because they were easy to represent with simple shapes and they were solid in color. Building my scene was much more challenging than I expected, so I’m glad I chose such simple objects.

Every shape in the scene has its own mesh object. These meshes are created in the main function and contain each shape’s vertices, texture data, normals, and indices. All of the textures for my scene are stored in an array that is referenced during rendering.

Since my objects are all solid-colored, I made some simple tiled textures to add interest to certain aspects of my scene and to help define their shapes.

The markers are each made with two cylinders. A loop renders each cylinder and textures and colors it as needed. The colors for my marker caps are stored in an array and referenced during rendering.

The hat is built from four shapes: a cylinder, a half-sphere, and the “hat bill” mesh, which is essentially two planes. I originally tried to use a half-sphere for the hat body without the cylinder, but it looked to flat, so I used the half-sphere as a top and a cylinder for most of the hat’s body.The reason I made the hat bill its own mesh instead of using my plane mesh is that I planned on moving the vertices around to create a smoother shape with a more rounded arch. The bill mesh in use right now was a base that I intended to enhance. I ran out of time to do that before turning this assignment in.

The sketchbook is made out of four cubes and 12 toruses. Since the rings on the real sketchbook are very small, I decided it would be simpler to use a single, thicker torus to represent each pair of rings.

The ground is a simple plane that uses a photo of my sheet at the texture. It’s the only texture I didn’t make from scratch.

**3D Navigation**

The scene can be navigated with the mouse and keyboard. I used the camera class from learnopengl.com for this functionality.

W, A, S, and D, will move the scene on the x and z axes to give the illusion of moving around in the 3D space.

E and Q will move the scene on the y axis to give the illusion of moving up and down.

The mouse can be used to change the camera’s view direction so the user can look around the scene.

The mouse’s scroll wheel modifies the speed at which the camera moves. Scrolling up will slow the camera movement and vice versa.

The P key can be used to toggle the camera view between perspective and orthographic.

**Coding Practices**

One way I organized my code was by keeping most of my lighting, texture, and color details in my namespace instead of directly referencing them during the rendering loop. This made it easier to apply different textures and colors in a single loop. For example, I was able to create all of the markers in one loop without writing the same code over and over. As the program got bigger, my rendering loop became more redundant, but the pieces are still in place for me to improve it with minimal pain.

Another way I organized my code was by creating only two fragment shaders for my scene objects: one for shiny objects and one for matte objects. I was worried at first I’d have to make a fragment shader for every single object (and if I wanted more realistic lighting, that may be a good thing to do) but I quickly realized the objects in my scene could be sorted into two basic categories of texture.