Justin Dietrich, Ryan Scott
10/9/2022
CST - 310
Prof. Citro

Project 4 + 5
Render Your Scene With Primitives

Project 4: Render Your Scene With Primitives

In this project we make a basic recreation of the picture from the previous project. We explain what we used and how we did it. Later projects will have us improving the picture.

Mathematical characteristics:

When making shapes that are consistent in shape and angle relative to others, you need to use mathematics to keep that consistency. For example, the ds stack has 3 different rectangular prisms stacked upon each other, each at the same angle but with two being smaller than the bottom one. When creating where these angled lines end, they all have to line up with each other. To do this, we calculated the ratio of x distance to z distance moved from the bottom ds, and kept the ratio of the top ds's the same when determining their vertices (the code looks messy there, because we kept it formatted with how we were calculating it instead of inputting the resulting float by itself).

We also created a drawCircle(x, y, z…) function that draws a circle at the desired location, with the desired color and radius, using sin and cos functions that create 100 lines to form a circle. The function also has options to make it an oval, to have the circle standing or lying flat, or to have a hollow/filled circle.

The primitives used to render:

All of the objects are made with glut. We use GL_TRIANGLES, GL_POLYGON, GL_LINES, GL_QUADS, and our circle function. Each object has its own mesh function, which combines the above drawing functions to make them. We skipped the wires for now, and will add them to a later project, since they will take some time. They will be implemented using GL_LINES.

All the transformations used and explanations on their use:

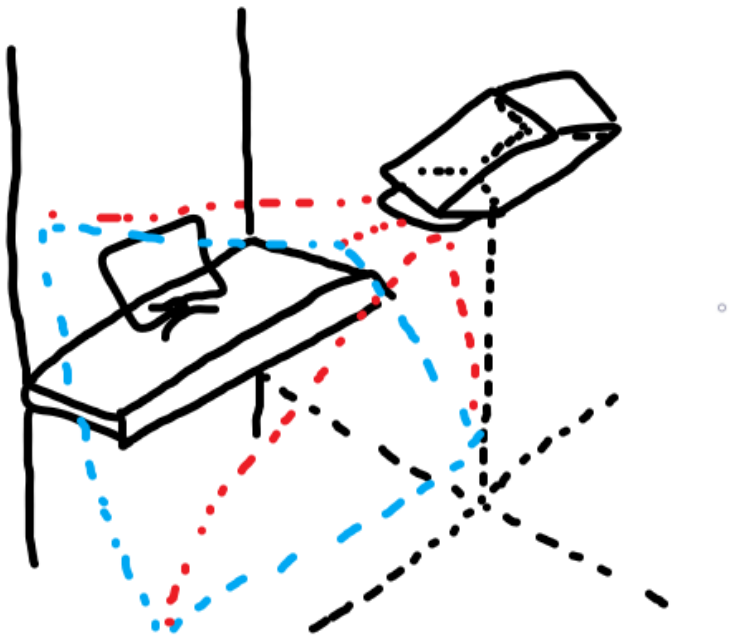
Things that would look like rectangles when viewed straight on turn into parallelograms when viewed at an angle. For things at an angle, we use parallelograms to simulate rotation. The circle function also uses a stretching method if you want to make an oval, and can interchange the y and z axis when drawing a circle to simulate a 90-degree rotation.

The shader(s) used and explanations of the reasoning and what it/they accomplished:

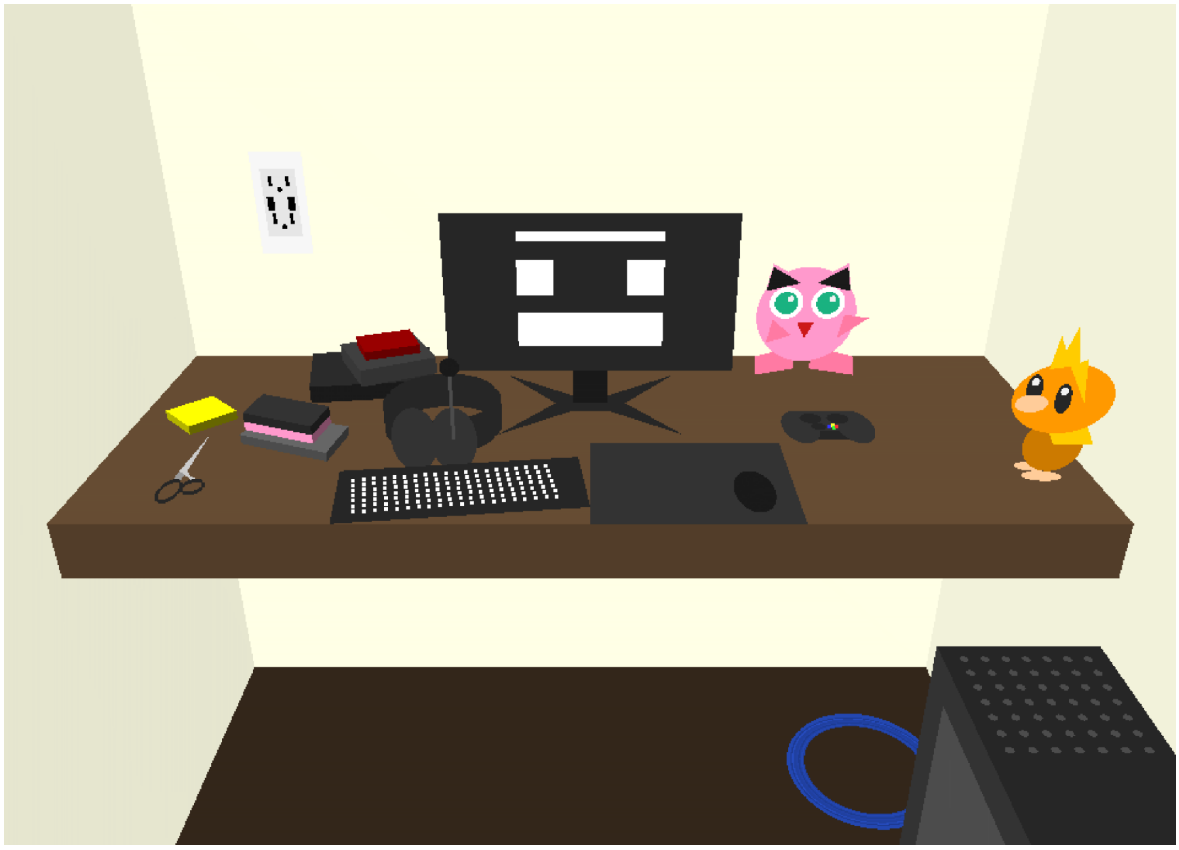
There's no Shader.h file. We do all of the shading manually, meaning we choose colors that are slightly brighter and darker when making different faces of an object. By doing this, we are able to make a more convincing 3D image.

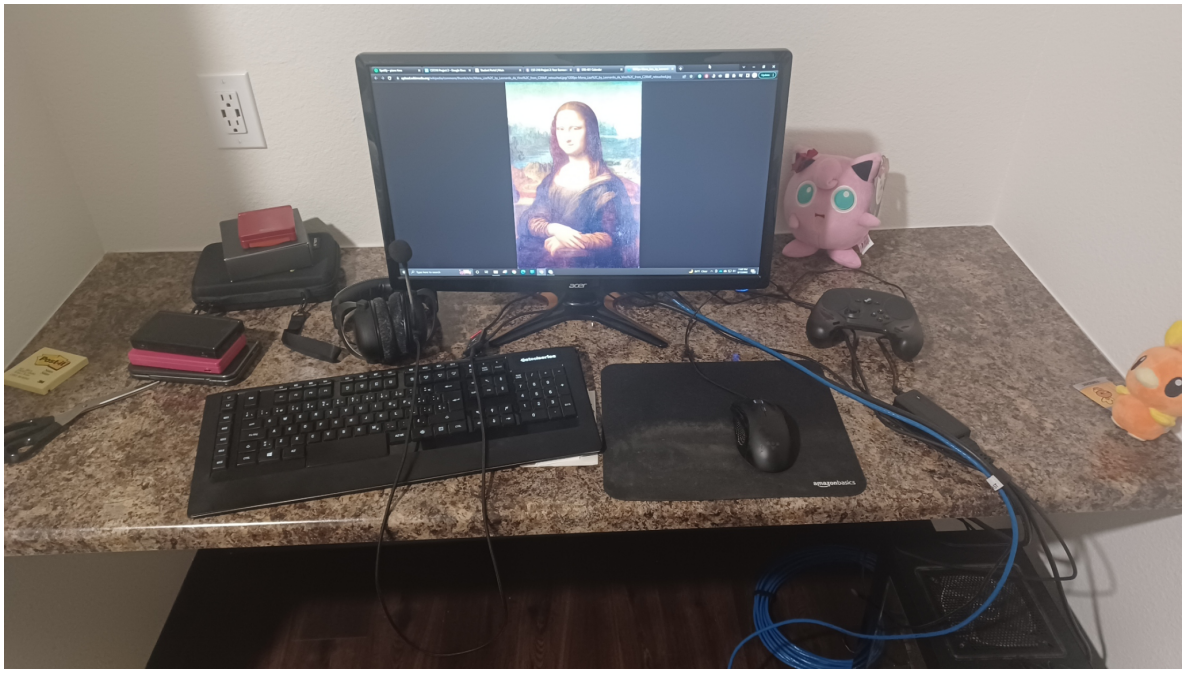
- Little pink guy: GL_TRIANGLES, GL_POLYGON, drawCircle()
- Monitor: GL_QUADS, GL_TRIANGLES, GL_POLYGON
- Mouse + Pad: GL_QUADS, drawCircle()
- Ds's: GL_POLYGON (didn't know about GL_QUADS at this point…)
- Scissors: drawCircle(), GL_QUADS, GL_TRANGLES
- Desk: GL_POLYGON (same case as Ds's)
- Walls/Floor: GL_POLYGON (^^^)
- Little Orange Guy: GL_TRIANGLES, drawCircle(), GL_QUADS
- Desktop: GL_QUADS
- Wire Coil: drawCircle()
- Keyboard: GL_QUADS
- Headphones: drawCircle(), GL_LINES

The camera is at z = 0, while the scene is around z = 30. The camera is above the scene and centered with it. Here is a diagram of the camera relative to the scene.



Comparison:





Project 5: Render Your Scene with Primitives

In this project, we were to improve on the previously submitted image by adding more detail.

Several details have been added to some objects in this submission, such as an outlet on the wall, more items on the desk, a circulation vent on the computer, plushes with more detailed eyes, and a backlit keyboard.

The way we simulated 3D on a 2D screen was by calculating the angles of shapes on objects that have multiple faces, and using the order of rendering objects and parts to appear in front of the screen. It is possible to shade and add more dimension to the same objects by using different colors in order to add a sense of dimension to them.

It is necessary to calculate the vertices of objects that have multiple faces by taking the distance from one vertex to the next, calculating the ratio, inverting the result, and applying the result to the next vertex to obtain perpendicular lines. In addition to the backlit keys, a for loop places a point for each key at a slight diagonal angle, similar to the keyboard's position.

Many object meshes can be considered complex because they consist of many small shapes strung together. Every circle here has 100 vertices. The wall outlet is composed of quads, and the plushes contain shapes such as circles, ovals, triangles, and quads.