Ray Tracing Project B

Ray Tracing Spheres, Cubes, Cones, and Cylinders

Jason Rhee

JJR975

I have designed two different ray tracing scenes that include Phong lighting and materials, checkerboard procedural materials, spheres, cubes, cones, cylinders, ground plane grids, and a WebGL half. In order to navigate the WebGL half use the arrow keys to move on a plane. The up arrow key changes the height in accordance with the tilt. The W key tilts up, the S key tilts lower, the D key rotates right, and the A key rotates left. In order to modify and interact with the ray tracing half, press T to trace the first scene, which contains three cubes and two cubes all in different positions with Phong lighting and materials and shadows. Lights may also be modified in the first scene. Press E to trace the second scene, which contains a flattened sphere, a cone, and a cylinder with checkerboard procedural material. Press p to toggle 4x4 Jittered Antialiasing. Input in different values in the input boxes and press Toggle Position in order to change light positions. Press Toggle Lamp On/Off in order to turn on and off the lamps. The light changes applies to the first scene. The default is the normal ray tracing.

To start the project, I built off of the Rescue code which traces a ground plane grid. I decided to add transformations to the grid in order to move around for different views. I did this by writing ray transformation functions within the CGeom class. I made sure to invert all of the WebGL transformations in order to create a matrix that transformed the rays. The transformations will be useful to draw the many other objects on screen.

I then implemented a 4x4 jittered antialiasing. I did this by splitting each little square that the ray goes through into even smaller squares and then randomizing where in the tiny square that the ray goes through. I couldn't go much farther with antialiasing, because I couldn't figure out how to achieve a perspective view so that the ground plane grid actually looked like a floor. I knew antialiasing was achieved because when I set the grid z value far from the eye point and rotated the grid, the grid was still crystal clear. Below is a picture of antialiasing.

Figure 1: The ground plane grid with 4x4 jittered antialiasing.

I then decided to work on tracing my sphere. I followed the ray going through the sphere twice strategy guided by Professor Tumblin's lecture slides. I traced the sphere and transformed it in order to add lighting, materials, and shadows. This inspired me to work on trying to ray trace more different shapes. I traced a cube, a cone, and a cylinder as guided by the FS Hill readings. I now have a collection of many ray traced shapes.

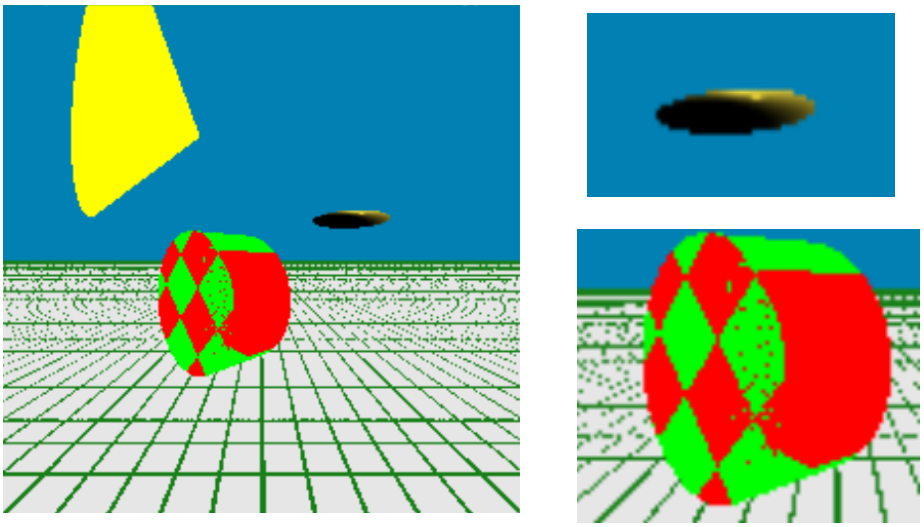

Figure 2: The flattened sphere, the checkerboard cylinder, and the cone.

In the image above, I have implemented a cone, cylinder, and a sphere that has been flattened by ray transformations. I have implemented a checkerboard procedural material on the cylinder by adding up the floors of the xhit, yhit, and zhit values and then found the absolute value of the total modulo 2. If the value was a 1, I colored it red, and if it was 0, I colored it green. I applied a ray scale, higher on the x and z values, to the sphere in order to flatten it.

I then attempted to add a diffuse shading light so that I could light my objects and create a shadow. I simply calculated the surface normals of the sphere and the cube, because they were the two objects in the scene that I was lighting. The surface normals for the sphere were simply the xhit, yhit, zhit values subtracted by the center which is the origin. The cubes surface normals were 1/-1 on each of the faces in accordance with the axis. I calculated the diffuse color by adding the ambient color to the calculated diffuse color, which is the light vector dotted with the normal vector multiplied by the material color values. I then proceeded to find the specular lights by multiplying that with the product of the specular material color and the dot product of the reflection and view vectors to the shininess power. I turned on and off the lights by scaling the specular and diffuse vectors with 0 for no light or 1 for light. I calculated a light vector by subtracting the lamp position by the x,y, and z values of the ray intersections. Users may change the position of the two lamps.

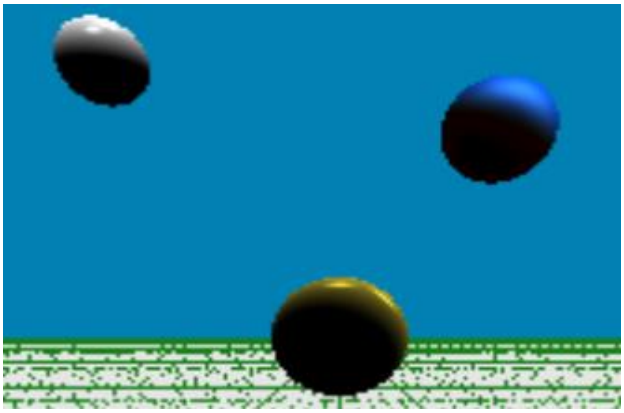

Figure 3: Three spheres in different positions lit by two lights. There are three different Phong materials.

I then implemented shadows. I did this by reversing the light vector so that it goes from the light to the x,y,and z position where the eyeRay meets the object. If the shadow ray hit another object, it would continue on its path and color the pixel on the other objects in the shade black. I added these rays to both lamps in order to create combined shadows.

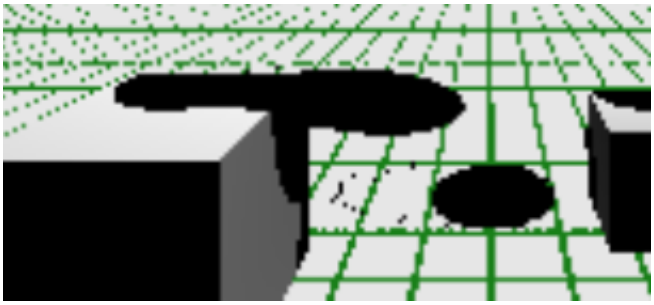

Figure 4: Two cubes and a ground plane grid with cast and combined shadows from the two lights and three spheres above.

I then added a ground plane grid coded as I would in WebGL and added full functioning 5-DOF controls. I also added a sphere from Basic Shapes starter code. I attempted to match the preview with the ray tracing, however I couldn't figure out how to position and rotate my camera in the ray tracing without messing up the ray tracing image I had created so I felt kind of stuck.

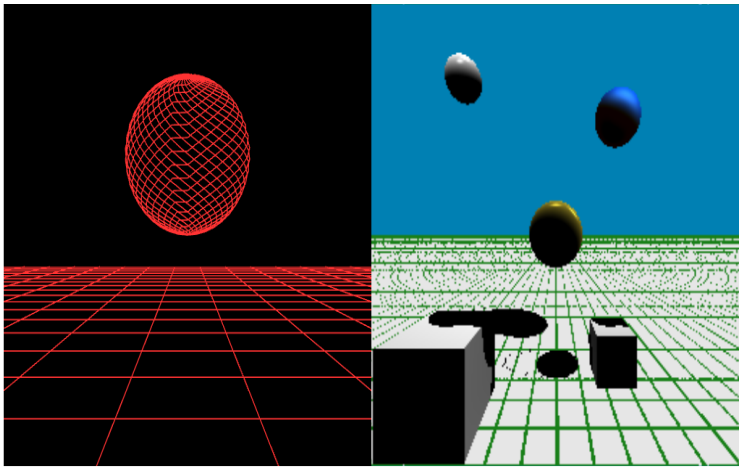

Figure 5: The light adjustable ray tracing on the right and the navigable webGL on the left.

I also tried very hard to implement reflection, however I couldn't extract colors because of the way my code was organized. I plan to work more on organizing my code before it becomes too large where it becomes difficult to organize or modify.