1 A Ray is Cast

For each pixel a ray is cast, from the eye through the respective "pixel" on the imageplane, to determine the color of that pixel in the resulting image. I then get the intersection closest to the eye for that ray (hit with the shortest raydepth). To calculate intersections with spheres I used the quadratic formula and for triangles I used barycentric coordinates as seen in the "Fundamentals of Computer Graphics" textbook. If nothing is hit the background color is returned, otherwise calculations are made to determine the color at the closest intersection.

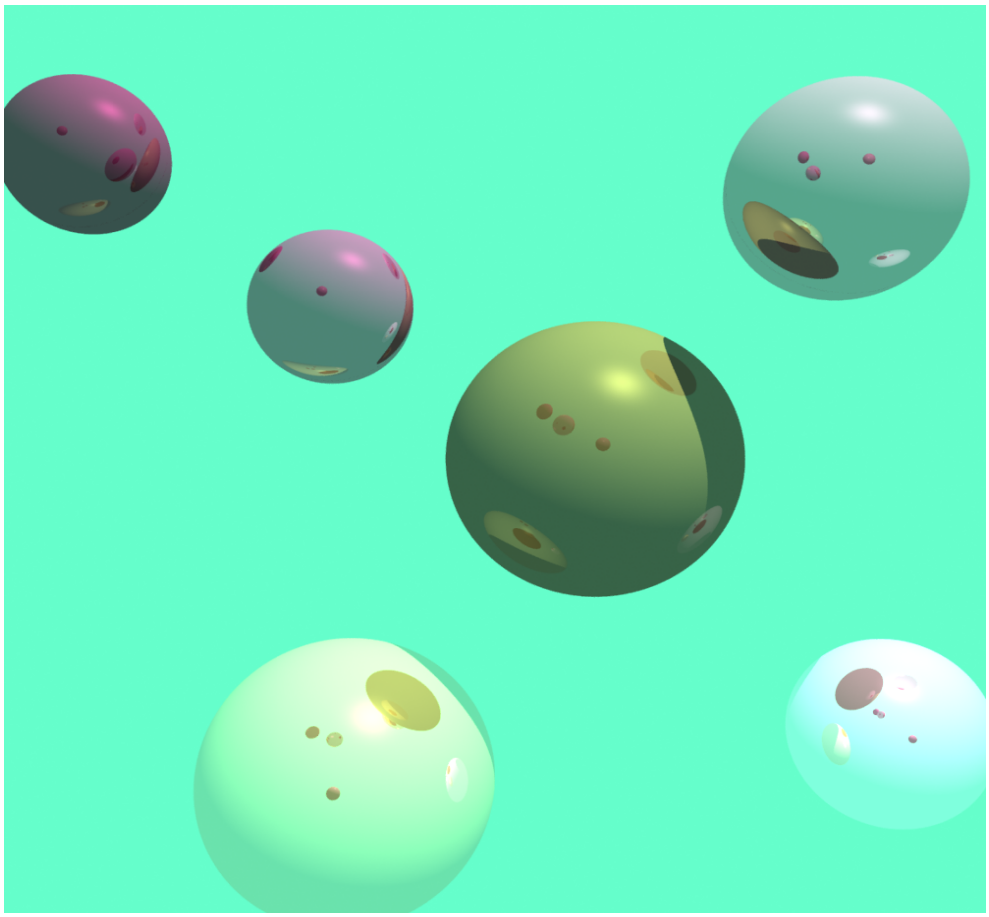

Figure 1: Spheres showcasing recursive reflection and shadows

2 Lighting

Then phong lighting calculations are done at the closest hit. The calculations are done inside the castRay function and are the first calculation I made to determine the colorOut for that ray cast.

3 Shadow Ray

After the lighting calculations I shoot another ray, the shadowray, to calculate if the position of the closest hit should be in shadow or not. The shadow ray moves from that position to the light source to see if any other object is hit. If an object is hit that position is in shadow and the lighting is switched to only ambient lighting. This is implemented inside of the castRay function just after the lighting calculations.

4 Reflections

Next i deal with reflections. This is done by recursively casting a ray that is a reflection of the ray from the closest hit position to the eye. The origin of the ray is set at that position. Then recursively the process starts again. The calculations are done at the closet hit, of the reflection ray, and the color is returned. This is added to the color at the position depending on the reflectively coefficient. I capped the recursive depth at 5. This is implemented after shadow ray in the castRay function.

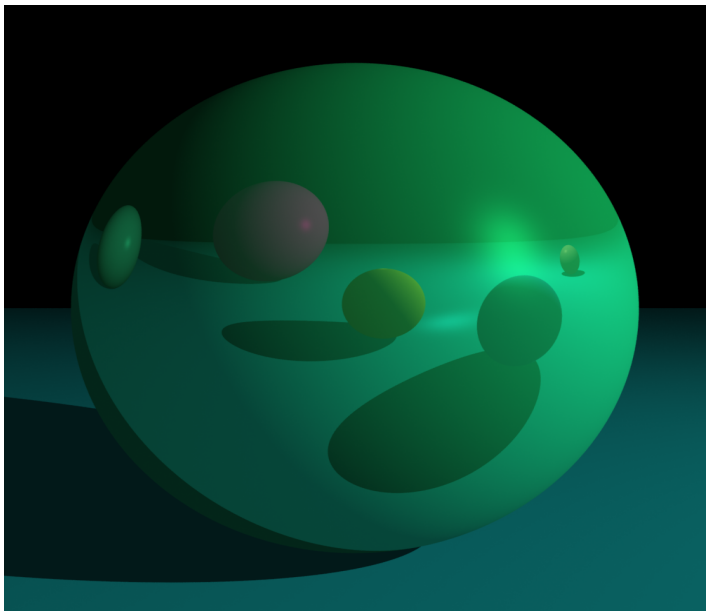

Figure 2: A green sphere with the reflect coefficient set to 0.5

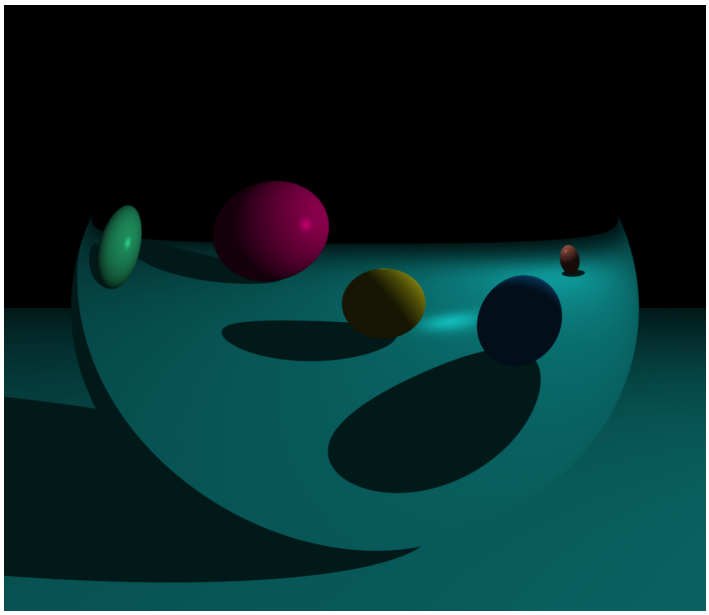

Figure 3: A black sphere with the reflect coefficient set to 1.0

5 Bonus: Reading From a File

My file i/o works by looking at the tags in the files to set different parameters. When setting a vector, spaces should be left between the different floating point numbers (as seen in files). Reflect should be set between 0 and 1. LightArea should be an integer, the larger the integer the greater the light area. To turn off area lighting set LightArea to -1. I have included the files I used to create the images in this document, though the colors may be different. Those files can be found in the main directory of my raytracer and are called Shadow.txt, Spheres.txt, Reflect.txt and Tree.txt. They should be used as reference for how an input file needs to be formatted.

I will now outline how to set different shapes in the file. All parameters must be set in the file for each shape. (The numbers below should be changed as desired and are just for demonstration)
Sphere: Radius: 1.0 Origin: 9.0 3.0 4.0 Color: 0.2 0.3 0.5 Reflect: 0.5
Triangle: A: 1.0 2.0 3.0 B: 3.0 4.0 5.0 C: 4.0 3.0 8.0 Color: 0.1 0.1 0.1 Reflect: 0.0
Plane: Origin: 0.0 0.0 0.0 Normal: 0.0 1.0 0.0 Color: 1.0 0.0 0.0 Reflect: 0.0

Currently the file is set to Spheres.txt in my program. (File pathing may need to be changed to work on a different system)

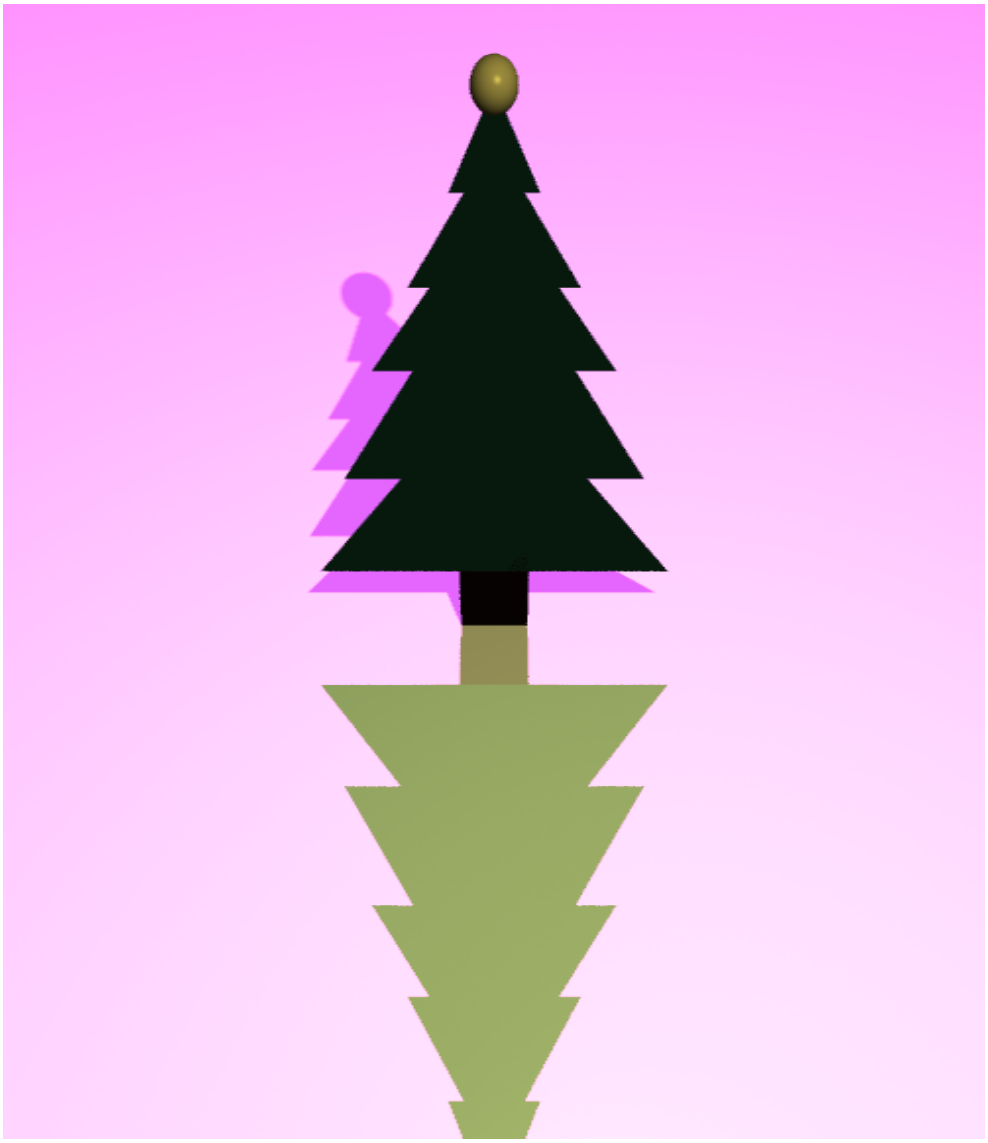

Figure 4: Tree created from the Tree.txt file

6 Super-Sampling

I implemented anti-aliasing through super-sampling. For each section of the image plane I created a n x n grid, where n is defined in the pixelTo3D function as int anti_aliasing. Once I had the grid I randomly choose a point in each gird square. Then in the render function I cast a ray through each point and averaged the results of all the raycast to get the resulting color for that pixel. This is done in the render function.

Figure 5: Without Anti-Aliasing

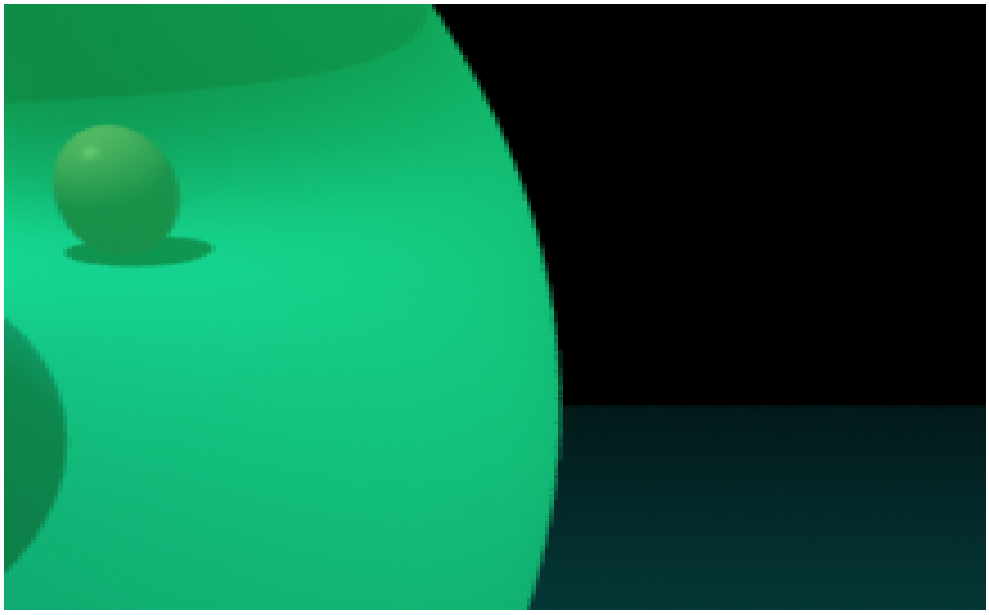

Figure 6: With Anti-Aliasing, Super-Sampling from a 5x5 grid

7 Area-LightSource

I created an area light by creating a grid of point light sources and then casting a ray for each different light value and averaging their results. I implemented this in the render function.

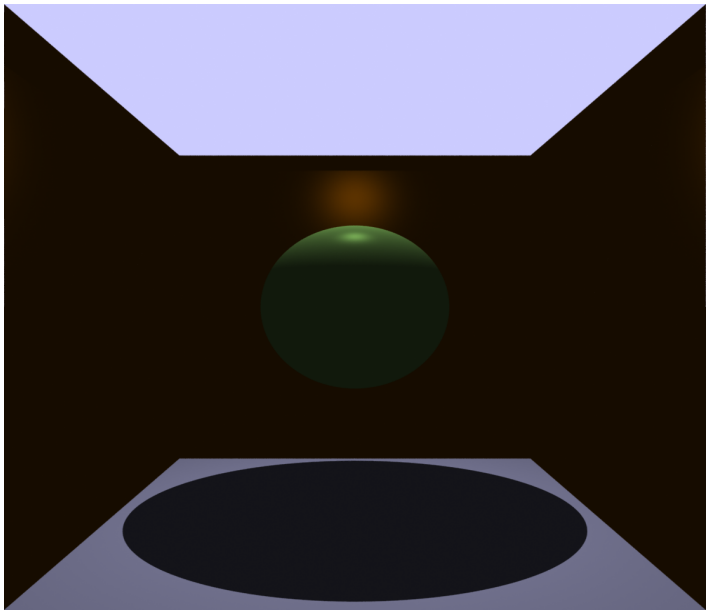

Figure 7: Without Area-Lighting

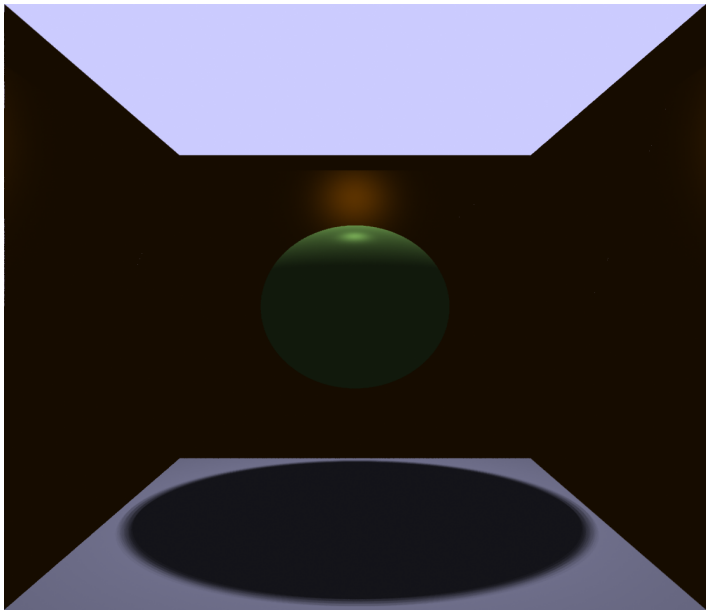

Figure 8: With an Area-Light of size 3 adding an additional 36 light points