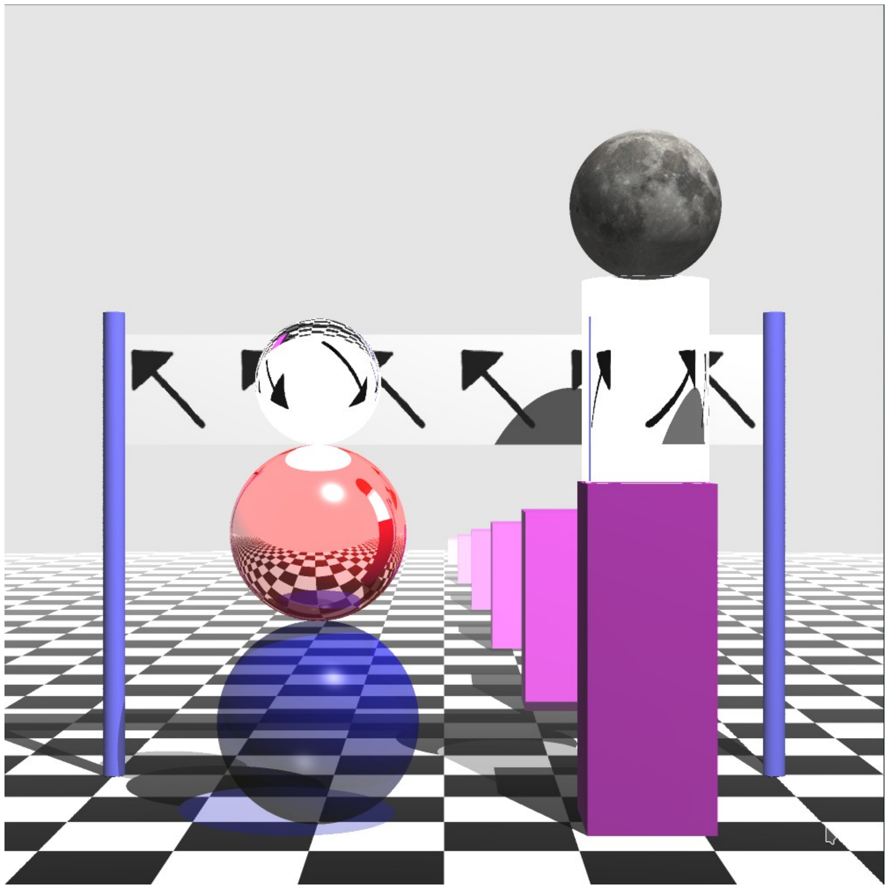
Jamie Barnes - 56505830

COSC363 – Assignment 2



My raytracer scene has 3 shperes on top of each other to the left. The bottom sphere being transparent, the middle being reflective, and the top being refractive. On the right there are 6 cuboid columns the decrease in height as they get further back and into the fog. There is also a refractive cylinder with a sphere textured to be the moon placed on top. Both the refractive objects have a plane placed behind them textured with diagonal arrows to show the effect of refraction on the sphere and cylinder. The virtical plane is supported by two tall cylinders on both sides.

My ray tracer includes:

- Transparency
- Shadows (coloured and tinted)
- A cuboid object made of planes
- A chequered floor
- Textured plane and sphere
- Cylinders
- Refraction
- Anti-aliasing
- Fog

Jamie Barnes - 56505830

Textured Sphere

The sphere is textured using UV mapping to map the points of the sphere to the points on a texture. I referenced the equations provided on the wikipedia page: https://en.wikipedia.org/wiki/UV_mapping

I can confirm that the texture is the correct way up.

$$u = 0.5 + \frac{\arctan2(d_x, d_z)}{2\pi},$$
$$v = 0.5 + \frac{\arcsin(d_y)}{\pi}.$$

Cylinders

A cylinder is created by providing a center point, a radius, and a height.

The intersect function uses the equations provided in the lectures:

$$t^2(d_x^2+d_z^2)+2t\{d_x(x_0-x_c)+d_z(z_0-z_c)\}+\{(x_0-x_c)^2+(z_0-z_c)^2-R^2\}=0$$

The top and bottom caps required a lot of edge cases that needed to make sure that traversing rays in a transparent or refractive sphere weren’t caught, or that rays that originated from the caps also weren’t caught. This took a lot of time and trial and error but I believe that it now all works correctly.

The normal function checks if the point lies on either of the caps and returns a corresponding vector. Otherwise, the function returns a normalised vector that is formed from subtracting the given point from the center point and zeroing the y component.

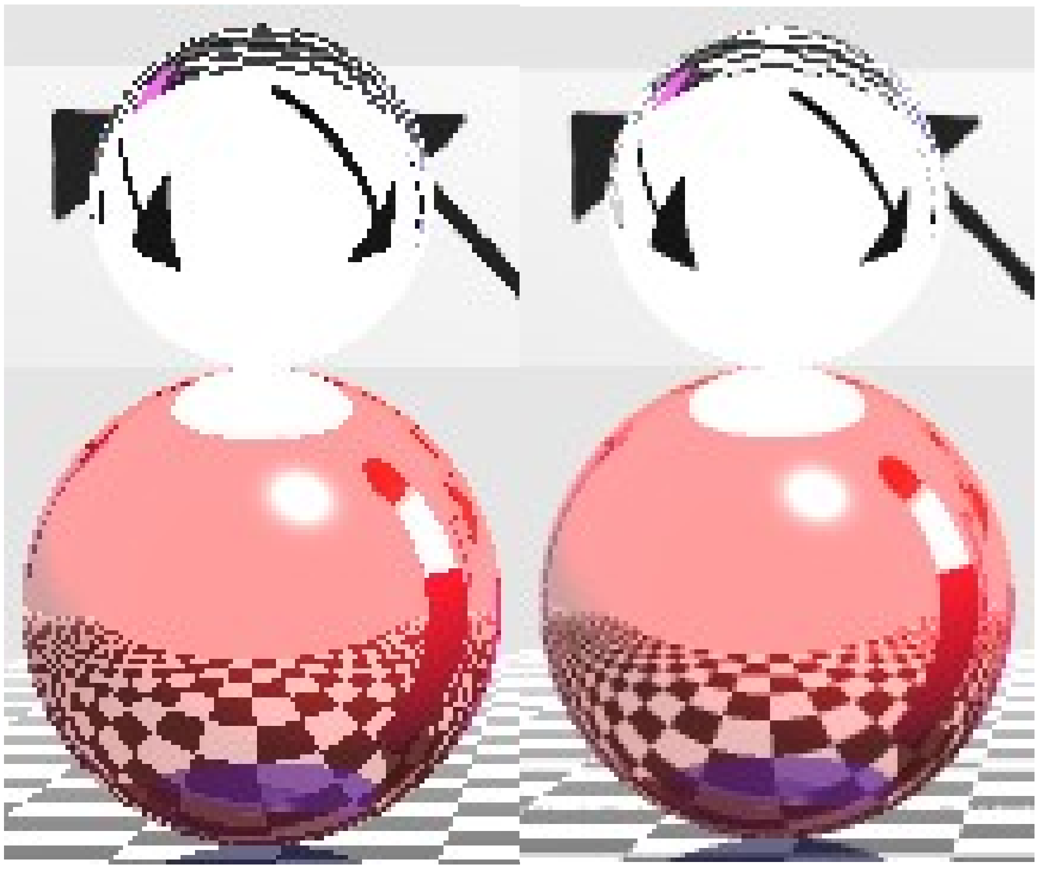
Refraction

Refraction is performed using the methods provided in the lectures. When a ray hits a refractive object the trace function creates a new refracted ray using the glm::refract function and the given normal values and uses the difference in direction of the incident ray and the normal vector to determine whether the ray is entering or exiting the object.

Jamie Barnes - 56505830

Anti-aliasing

Anti-aliasing is implemented through super-sampling. It calculates 4 rays in a 2 by 2 grid and averages their colours to provide the colour for one pixel. It does this for all pixels and none of the 2 by 2 grids overlap. This is done to reduce the jaggered edges on objects and textures.



Aliased : Anti-aliased

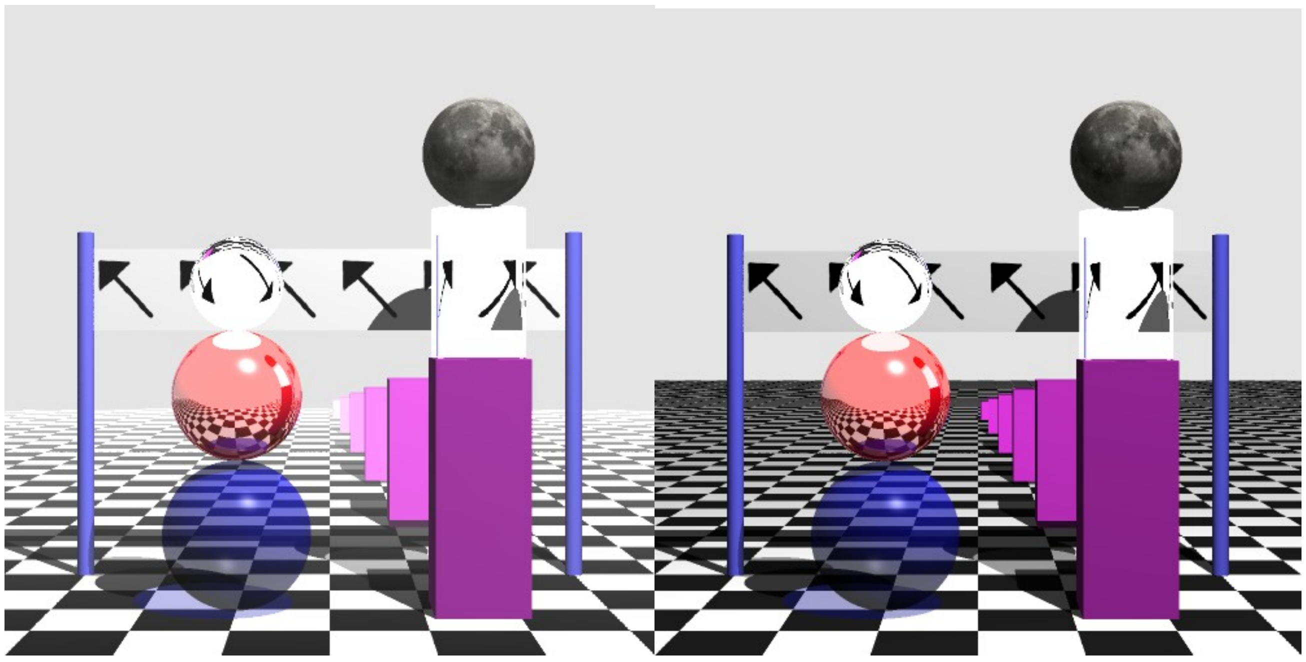
Jamie Barnes - 56505830

Fog

Fog is implemented in the trace function using the ray distance from it’s initial point to it’s hit point. It takes this distance and mulitiplies it with a gray scale colour vector after a certain distance which it then adds to the colour returned by the function. To prevent ghost objects in the off-white fog background colour, a limit is placed to set every colour to the same off-white of the fog after a certain distance.

$$\text{Colour} = \text{Colour} + \text{glm::vec3}(0.0045) \times (\text{rayDistance} > 50\,?\,\text{rayDistance} - 50 : 0);$$

$$\text{if}\,(\text{rayDistance} \geq 250)\,\text{colour} = \text{backgroundColour};$$



Compiling and Running

To compile the files run the command:	g++ -o out *.cpp -lGL -lGLU -lglut
To run the output run the command:	./out

The scene has 15 objects in the scene and is rendered at 500x500px (with AA effective 1000x1000). Time estimate: 40s

References:

Moon Texture: https://svs.gsfc.nasa.gov/cgi-bin/details.cgi?aid=4720

UV Mapping: https://en.wikipedia.org/wiki/UV_mapping