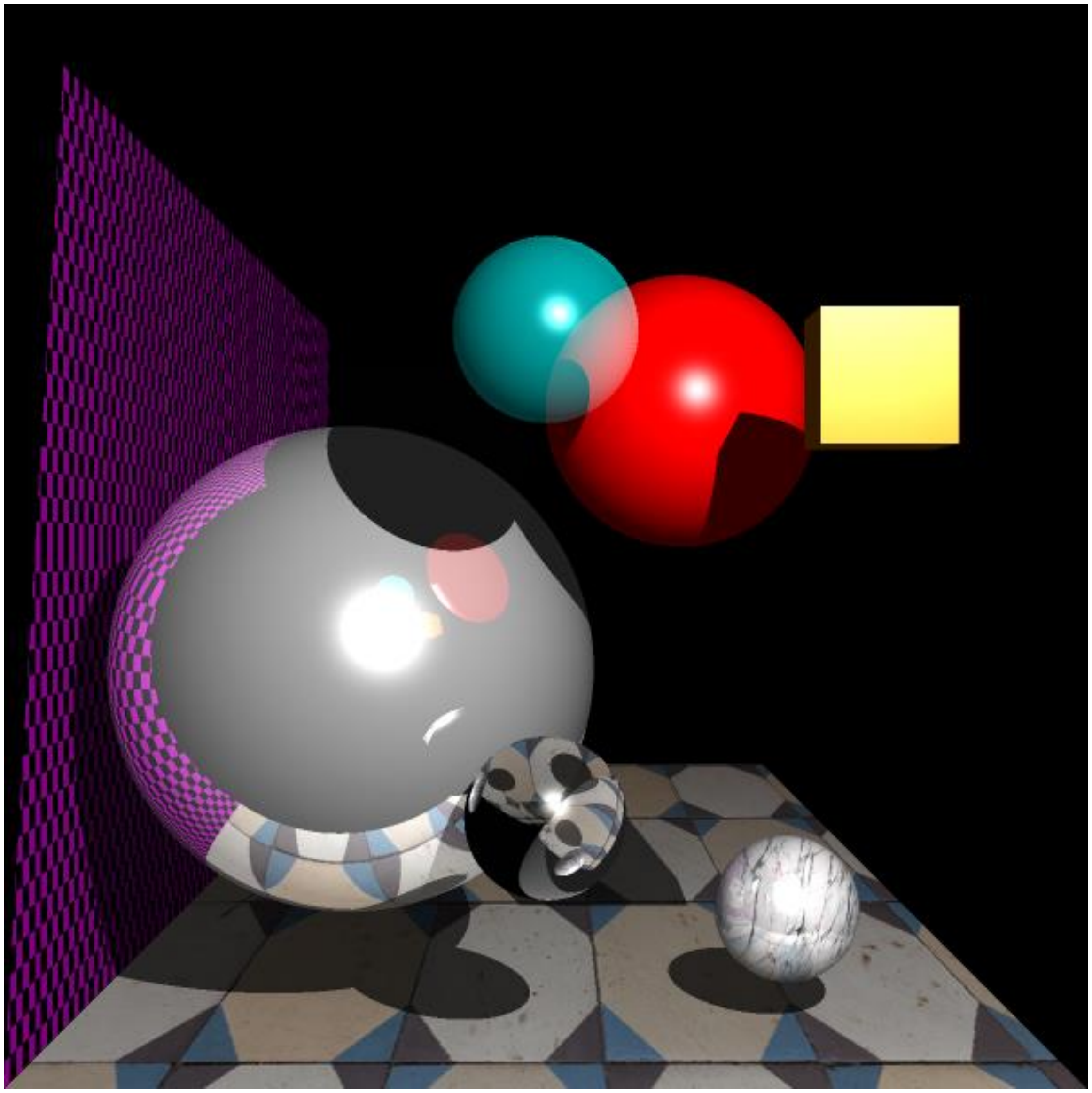
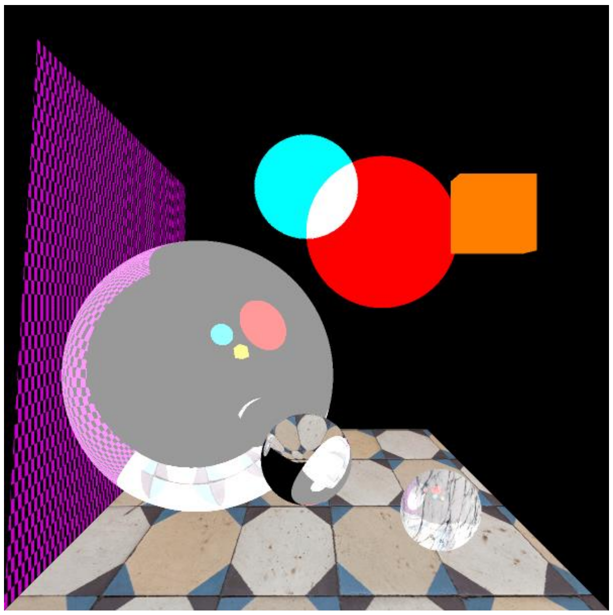
COSC363 Computer Graphics Assignment 2 – Ray Tracing

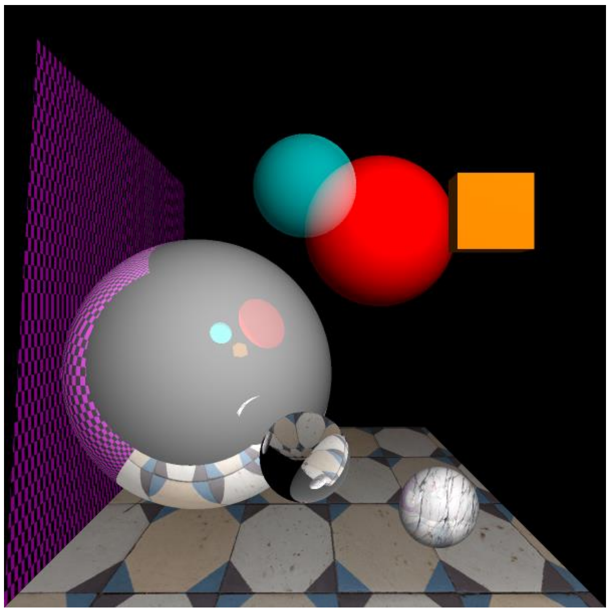
Alex Bull 11213398

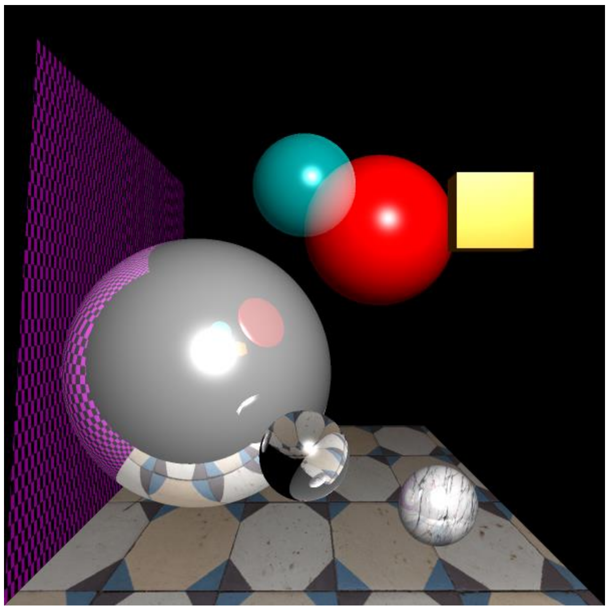


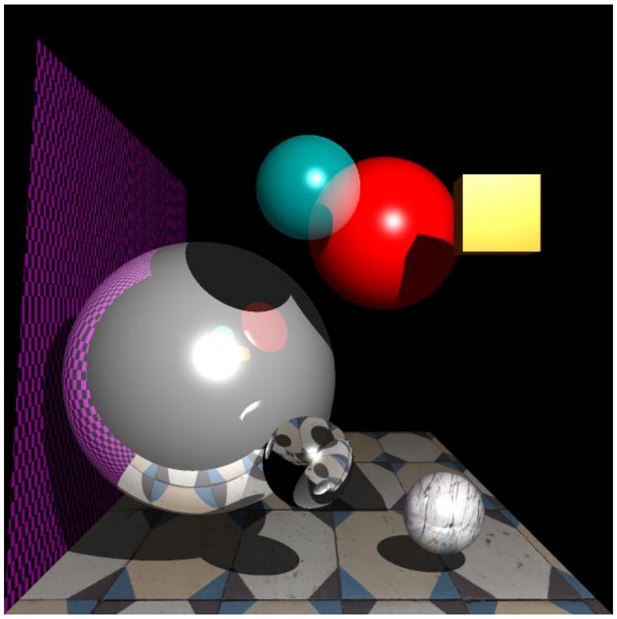
My ray tracer was successful in rendering the core components of the brief. These components being: one light source with diffuse and specular reflections generated, shadows, one reflective sphere, one cube (made of multiple planes), and a textured planar surface. It then went on to have five of the extension components implemented. These were: a transparent object (that wasn't just a special case refractive object), a refractive object, anti-aliasing/super sampling, an image textured non-planar object, and an object textured with a procedural pattern. OpenGL functions were only used for rendering the ray traced image.

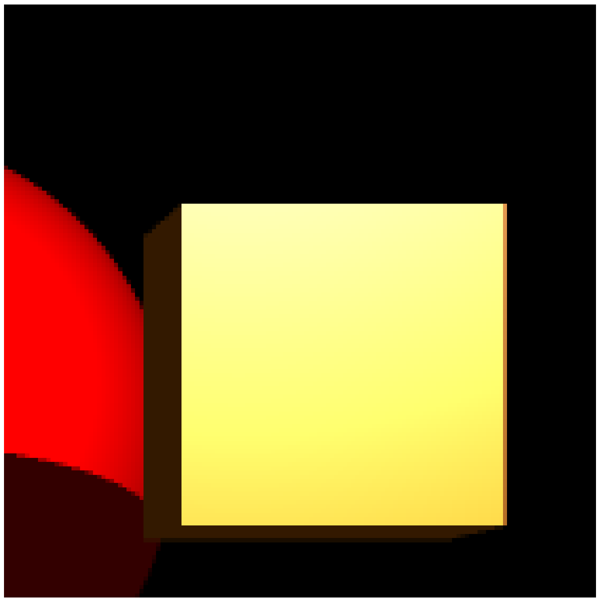
One aspect that had room for improvement is the shadows cast by transparent and semi-transparent and colored objects. The shadow is rendered by realistically we would expect the shadow to be dimmer if it weren't solid and if it were also colored, a tinted shadow.


The screenshot above shows the scene before lighting is implemented.

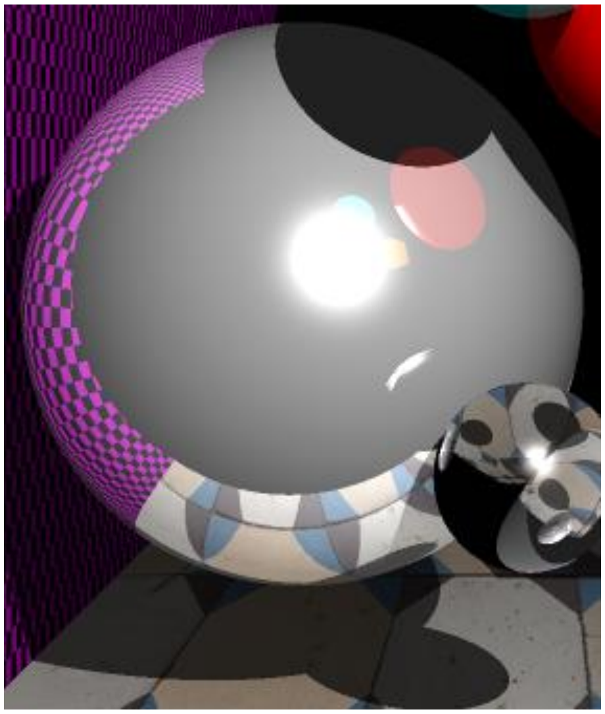

Lighting rendered taking into account the light source at [15, 20, -20].


The scene after specular relections were added.


The scene after shadows were added.


An orange cube made of six planes.

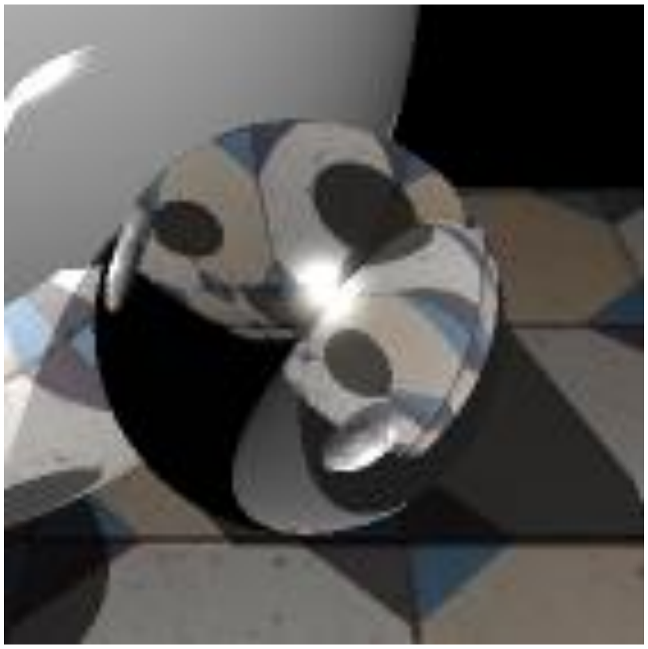
The planar floor's texture.


A reflective grey sphere.

The planar floor as seen in the scene. The equation used in mapping the texture is:

texcoords = (ray.xpt.x - a1)/(a2-a1)

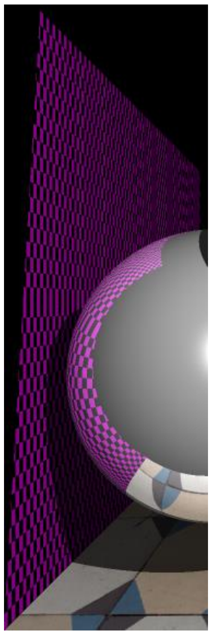
texcoordt = (ray.xpt.z - b1)/(b2-b1)



A sphere which refracts the light entering it. Ray tracing refractions involve the passing of a ray through an object and calculating the secondary ray's direction, then adding what it hits to the color sum. I made this sphere black (0, 0, 0) and allowed for the secondary rays to influence the color of it a large amount in order to give off the effect that the sphere was made from glass. I further this impression I made the refractive index the same as glass in the refraction calculations/code.

```
n = sceneObjects[ray.xindex]->normal(ray.xpt);
g = glm::refract(ray.dir, n, eta); //eta is refractive
index of glass
Ray refrRay1(ray.xpt, g);
refrRay1.closestPt(sceneObjects);

if (refrRay1.xindex == INDEX_OF_RELEVANT OBJECT){
m = sceneObjects[refrRay1.xindex]
->normal(refrRay1.xpt);
h = glm::refract(g, -m, 1.0f/eta);
refrRay2 = Ray(refrRay1.xpt, h);
refractCol = trace(refrRay2, step + 1);
} else {
refractCol = trace(refrRay1, step + 1);
 }
colorSum += (refractCol * 1.0f);
```

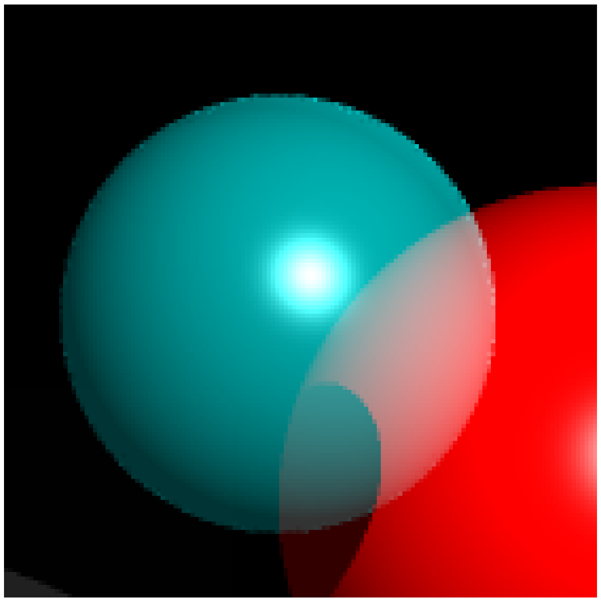


A simple procedural generated texture was used on the side plane. A procedural texture is a texture that isn't stored e.g. an image, and instead uses functions to calculate the RGB value at each point of the object.

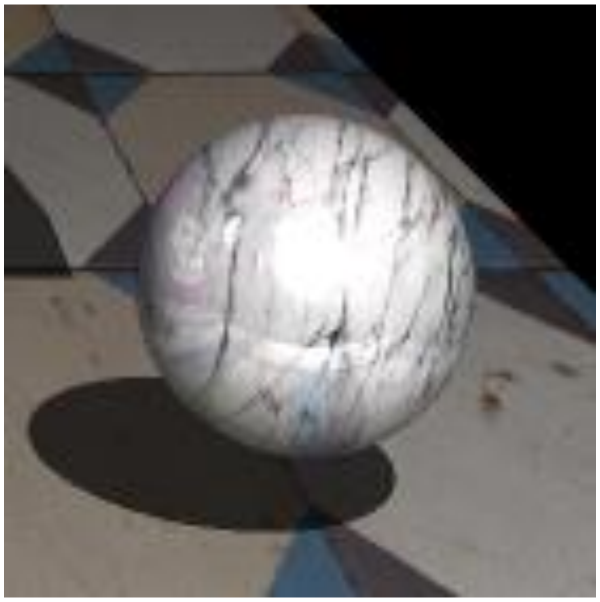
In this scene the equation used is:

```
 if ((int(ray.xpt.y) + int(ray.xpt.z)) % 2 == 0){
 materialCol = glm::vec3(0, 0, 0);
} else { materialCol = glm::vec3(0.8, 0, 0.8); }
```

Which simply means if point y + point z is an even value the color at the point is black, otherwise it is pink.



The transparent sphere was made by passing a ray through the points of the sphere but not affecting the direction of the ray. This meant the rays could pick up scene/colors it came into contact with after passing through the sphere and use those to add to/impact the color sum. This gave the impression of a transparent material.



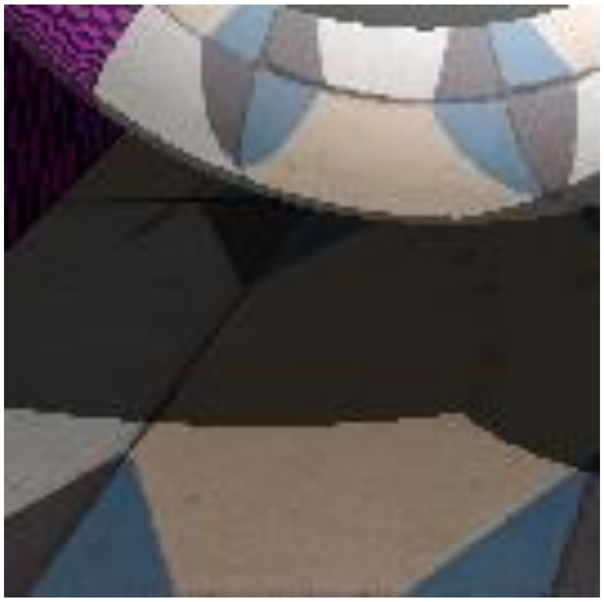
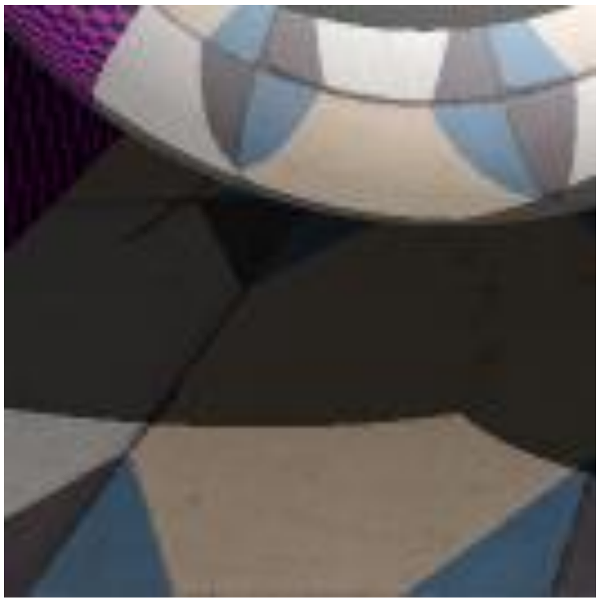
This sphere has an image texture mapped to it with an equation that takes into account the curved surface of a sphere, rather than mapping the texture onto it as if it were flat/planar.

The equation used was:

texcoords = asin(normalVector.x)/M_PI + 0.5;

texcoordt = asin(normalVector.y)/M_PI + 0.5

The texture used on the sphere. The sphere also uses the reflection code for some added reflectivity, however not to the same degree as the larger sphere.

Non anti-aliased render *Anti-aliasing enabled (super sampled) render*

A method of anti-aliasing is called super sampling, and is what was used in this project. Its main function is to make the edges of shadows and objects in the scene look smoother, rather than jagged. This was implemented by dividing the cells of the view pane into quarters and sending a ray through each. The function used was a wrapper to the trace function, which calls the trace function on rays that intersect the center of the quarters/divisons of the cell. So in total, 4 rays are sent through a cell (one ray per subdivision). The colors obtained by the four rays are averaged to get the color value of the cell. The psuedo-code for the function used would look like:

```
Left quarter location on x-axis of cell (centerLeftX)= cell position + (cell width / 2)
Bottom quarter location on y-axis of cell (centerLowerY) = cell position + (cell width / 2)

Call trace() four times, one time for a new ray where:
Ray direction calculation uses (centerLeftX, centerLowerY)
Ray direction calculation uses (centerLeftX + (cellWidth / 2), centerLowerY)
Ray direction calculation uses (centerLeftX + (cellWidth / 2), centerLowerY + (cellHeight / 2))
Ray direction calculation uses (centerLeftX, centerLowerY + (cellHeight / 2))

Get the average of the color results
```