Assignment 2 Report

Raytracer scene- The raytracer shows a scene with a blue reflective sphere, a red sphere, a glass sphere, a cyan coloured transparent sphere, a textured sphere, a checked patterned sphere, a cube, a tetrahedron, a textured floor. The objects are surrounded by four other planes which are magenta coloured, constructing walls around the objects in the scene. There are also two lights, lighting the scene and anti-aliasing is also used in the ray tracer. The program using anti-aliasing should run about 1-2 minutes.

Reflective sphere- This sphere is built on the reflective sphere created in lab 8, using its surroundings it gives its reflection of the scene, using 0.8 as the coefficient of reflection. It uses recursion to accumulate the colours in the reflected direction.

```
if(ray.xindex == 0 && step < MAX_STEPS)
{
        glm:: vec3 reflectedDir = glm:: reflect(ray.dir, normalVector);
        Ray reflectedRay(ray.xpt, reflectedDir);
        glm:: vec3 reflectedCol = trace(reflectedRay, step+1); //recursion
        colorSum = colorSum + (0.8f*reflectedCol);
}
```

Glass sphere – This sphere uses refraction with the eta value of 1.0/1.5 to make it seem that the sphere is made of glass, by showing the objects behind it to appear like the scene is upside down, using 0.8 as the coefficient of transmission. It uses recursion to accumulate the colours in the refracted direction.

```
if(ray.xindex == 1 && step < MAX_STEPS)
{
        glm:: vec3 g = refract(ray.dir, normalVector, glass);
        Ray refractedRay1(ray.xpt, g);
        refractedRay1.closestPt(sceneObjects);
        glm:: vec3 m = sceneObjects[refractedRay1.xindex]->normal(refractedRay1.xpt);
        glm:: vec3 h = glm:: refract(g, -m, 1.0f/glass);
        Ray refractedRay2(refractedRay1.xpt, h);
        glm:: vec3 refractedCol = trace(refractedRay2, step+1); //recursion
        colorSum = colorSum + (0.8f*refractedCol);

}
```

Cyan coloured transparent sphere – This sphere also uses refration with the eta value of 1.0/1.01 to make it seem like it has magnified the scene objects behind it. The sphere was also coloured with the rgb values of (0, 0.5, 0.5) to get it's cyan colour, it also uses 0.8 as the coefficient of transmission and uses recursion to accumulate the colours in the reflected direction.

```
if(ray.xindex == 3 && step < MAX_STEPS)
{
        glm:: vec3 g = refract(ray.dir, normalVector, magnifier);
        Ray refractedRay1(ray.xpt, g);
        refractedRay1.closestPt(sceneObjects);
        glm:: vec3 m = sceneObjects[refractedRay1.xindex]->normal(refractedRay1.xpt);
```

```
            glm:: vec3 h = glm:: refract(g, -m, 1.0f/magnifier);
            Ray refractedRay2(refractedRay1.xpt, h);
            glm:: vec3 refractedCol = trace(refractedRay2, step+1); //recursion
            colorSum = colorSum + (0.8f*refractedCol);

    }
```

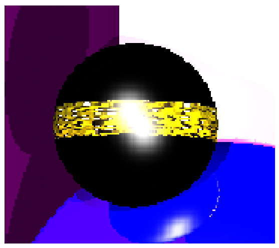
Textured sphere- This sphere is textured using the image specified in the references section and uses

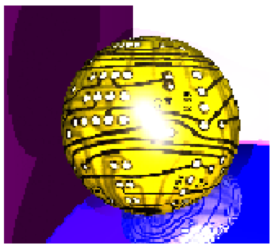
```
            texcoords = 0.5 + (atan2(ray.xpt.z+110, ray.xpt.x+10)/(2*M_PI))
            texcoordt = 0.5 - asin((ray.xpt.y-10)/5)/ (M_PI)
```

to map the image to the sphere. I had trouble mapping the texture using the equation below (from reference 3) at first since the input value of asin needed to be between -1 to 1, and since my sphere had a radius of 5 some of the values put into the asin function were out of the range and therefore gave out NaN values, resulting in the first image of the sphere below. I solved this by also dividing the value by 5 (value of the radius of the sphere). The values added and subtracted (10, -10, 110) above were used to offset the value put in the functions as the sphere is not centred at the origin point, but instead is centred at (-10, 10, -110).

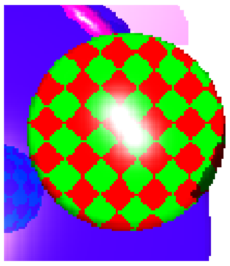
$$u = 0.5 + \frac{\arctan 2(d_z, d_x)}{2\pi}$$

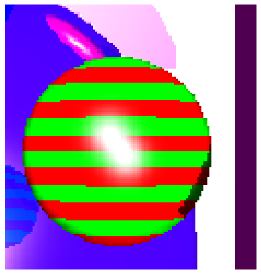
$$v = 0.5 - \frac{\arcsin(d_y)}{\pi}$$

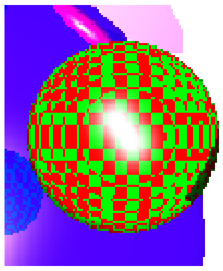




Patterned sphere- The sphere is patterned using sin(M_PI*ray.xpt.x/0.7) + sin(M_PI*ray.xpt.y/0.7), with 0.7 affecting the size of the checks created in the sphere. If the result is greater than 0 colour that part of the sphere red, otherwise colour it green. The output of this is shown in the first image below.







The other two patterns above were also patterns that I also tried, The horizontal striped pattern created with checking sin(M_PI*ray.xpt.y/0.7) is greater than 0, if it is that part of the sphere is red, otherwise it is green, again 0.7 affects the size of the stripes created. The 3D Checkerboard pattern is created with checking that (((int)(ray.xpt.x/0.5) + (int)(ray.xpt.y/0.5) + (int)(ray.xpt.z/0.5)) % 2 == 0), if it is that part of the sphere is red, otherwise it is green, with 0.5 affecting the size of the checkers created. The patterns created above were inspired by patterns from reference 2.

Sphere – The spheres uses the simplified phong's illumination to colour them using ambient, diffuse, specular reflections, noted in the lecture 9 notes.

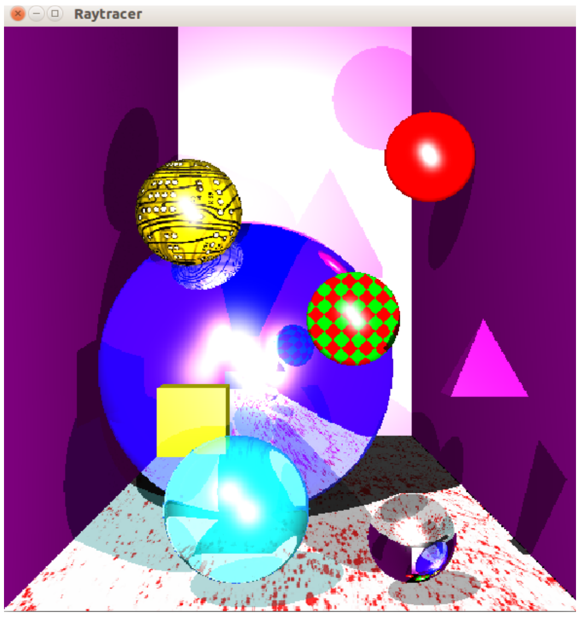
Cube – The cube is made with 6 planes created, with the x,y,z values for top, bottom, left, right, near and far sides instantiated and all coloured with the colour passed in the function drawCube, Note that this function can draw any parallel-pipped object.

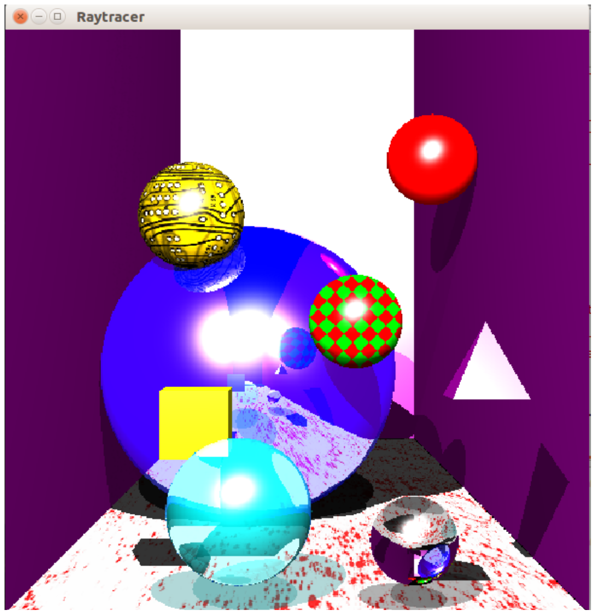
```
void drawCube(float top, float bottom, float left, float right, float near, float far, glm::vec3 col)
```

Tetrahedron – The tetrahedron is made with 5 planes created with the x,y,z values for top, bottom, left most, right most, near most and far most sides instantiated and all coloured with the colour passed in the function drawTetrahedron. The points of the planes meet at the top by using the point in the middle of the specified left and right and the point in the middle of specified the near and far.

```
void drawTetrahedron(float bottom, float top, float left, float right, float near, float far, glm::vec3 col)

        float mid_x = (left + right)/ 2;
        float mid_z = (near + far)/ 2;
```

Lights – The scene has two light source created in the points of light1(-18, 40, -3) and light2(15, -10, -30), shadows are drawn accordingly for the lights. The first image below shows light sources in the points of (-18, 40, -3) and (15, -10, -30). The second image below shows light sources in the points (-18, 40, -3) and (15, 80, 10). Showing how the colours calculated change accordingly when light sources move.





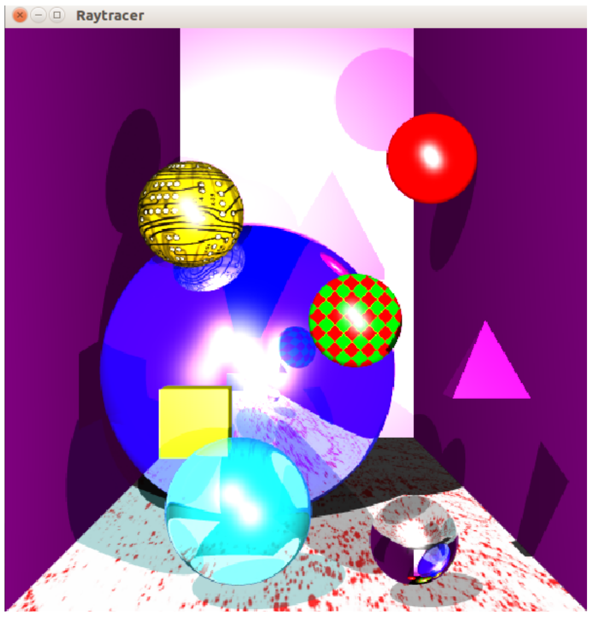
Floor – The floor is textured using an image specified in the reference section. With a1 = -20, a2 = 20, b1 = 0, b2 = -200, using idea from the lecture notes the plane maps the image with

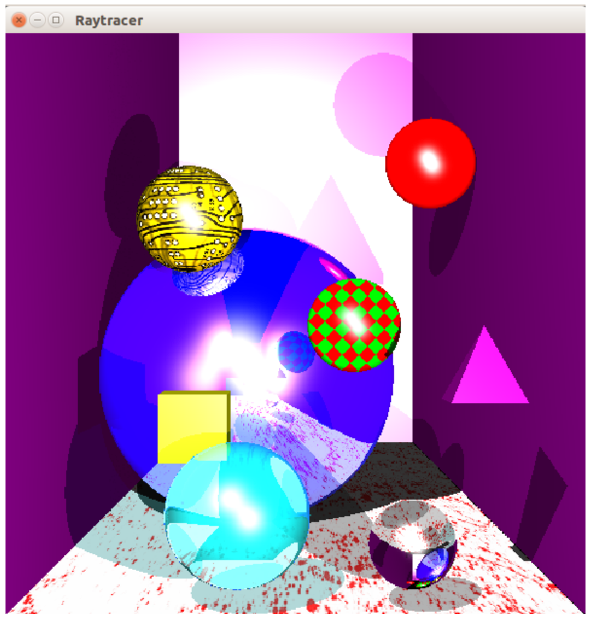
```
texcoords = (ray.xpt.x + 20)/ 40;
texcoordt = (ray.xpt.z)/ -200;
```

The shadows of transparent spheres are adjusted to be lighter than solid objects, and having the shade of the colour of the transparent object itself by getting the current colorSum * 0.7 and adding 0.3 * of the object colour to now be the colorSum. This was done for both lights to generate their shadows accordingly. The sample for light 1 is shown below, this was done for the 2nd light source as well, where xindex 1 is the glass sphere and xindex 2 is the cyan coloured transparent sphere.

```
colorSum = floor_texture.getColorAt(texcoords, texcoordt);

//light1
if((shadow1.xindex == 1 && shadow1.xdist < lightDist1))
        colorSum = glm:: vec3(0.7f * colorSum) + glm:: vec3(0.3f * sceneObjects[1]->getColor());

if((shadow1.xindex == 3 && shadow1.xdist < lightDist1))
        colorSum = glm:: vec3(0.7f * colorSum) + glm:: vec3(0.3f * sceneObjects[3]->getColor());

if ((lDotn1 <= 0 || (shadow1.xindex>-1 && shadow1.xdist < lightDist1)) && (shadow1.xindex != 1
&& shadow1.xindex != 3))
        colorSum = glm:: vec3(ambientTerm * colorSum);
```

Anti-aliasing – anti-aliasing is used on the ray tracer to make the edges of objects to be smoother by sub-dividing pixels into 4 parts and calculating the colour values in those 4 parts and getting the average value of those colour to represent the colour of the pixel itself. Below I have inserted an image showing the difference between the ray tracer having anti-aliasing and not having it. (I have also inserted a file called Raytracer0.cpp which is the version of the ray tracer without anti-aliasing). As a result of anti-aliasing, some of the pixels in the edges of the between red and green in the patterned sphere smoothed the differences in the 4 parts by making the edges yellow, compared to the one without anti-aliasing where the edges were either red or green. The pseudo code I used to do anti-aliasing was from reference 1.





References –

Sphere Texture - https://www.textures.com/download/electronics0039/32165
Floor Texture - https://www.textures.com/download/splatterhomogeneous0003/10708
Other references –

Lecture notes 9, 9a

1. http://pellacini.di.uniroma1.it/teaching/graphics13a/slides/03_raytracing.pdf
2. http://web.cse.ohio-state.edu/~shen.94/681/Site/Slides_files/texture.pdf
3. https://en.wikipedia.org/wiki/UV_mapping