Name: Saman Ighani
Student ID: 78966609
User Code: sig16

COSC363 Assignment 2 – Ray Tracer

Instructions for building the ray tracer

Assuming the opengl, GLUT and glm associated libraries are present on the machine. unzip the folder contained in the zip file, import the project into qtcreator or similar ide using the makefile provided build and run.

Through the linux commandline assuming cmake and make are present dependencies on the system:
1. Change directory to point to the location of the unzipped project folder
2. Type in “cmake CMakeLists.txt” without the speech marks and wait until finish
3. Now type in “make” without the speech marks and wait until finish
4. Now type in “./RayTracer.out” without the speech marks and enjoy.

Ray Tracer Description

Making up the base of the ray tracer were features implemented from Labs 7 and 8 with additional functions found from lectures and different mathematical equations used to model procedural textures on spheres and planes. The ray tracer contains refraction, reflection, shadows, transparency and multiple light sources for planes, spheres, a cube and objects alike. The scene implements a transparent sphere, a refracted sphere, two reflective spheres, a sphere with a procedural texture modelled using a mathematical equation, a cube made of a set of six planes and two plains with procedural textures modelled with their own respective mathematical equations for the floor and back wall of the scene.

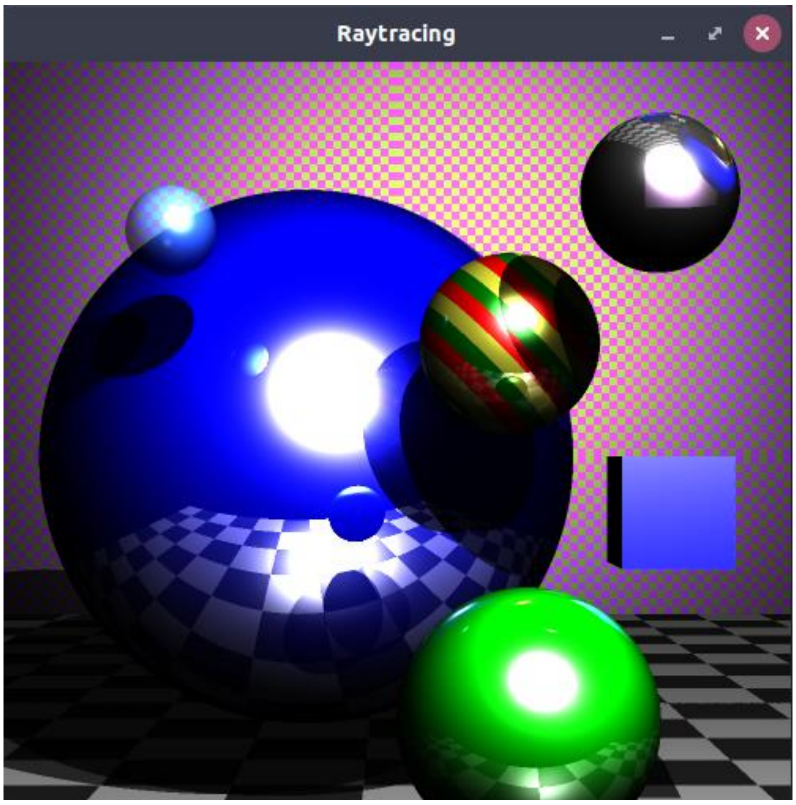


Figure 1: Whole ray tracing scene

Name: Saman Ighani
Student ID: 78966609
User Code: sig16

Features

Anti-Aliasing

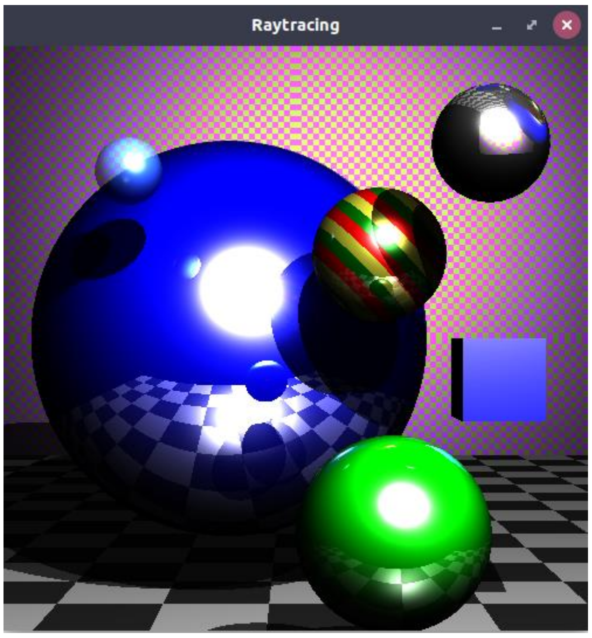


Figure 2: Without anti-aliasing

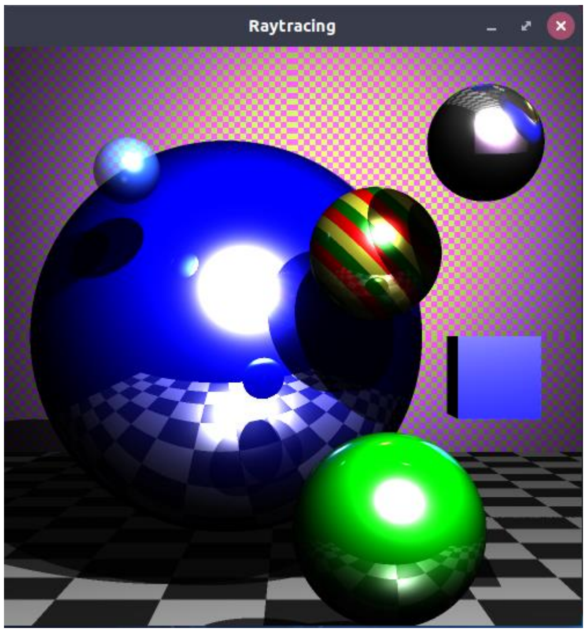
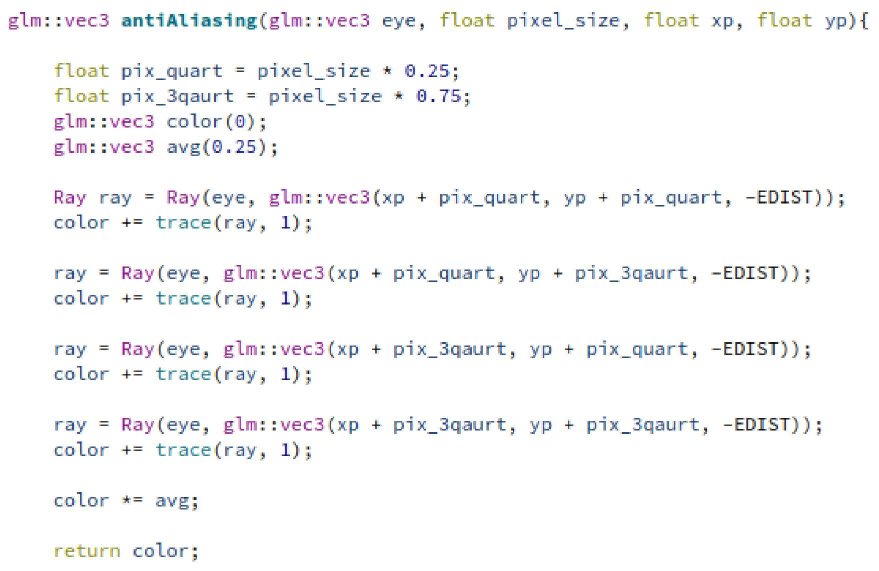


Figure 3: With anti-aliasing

Anti-aliasing is implemented in the scene by dividing each pixel into four parts. For each part a ray is traced returning a colour value, the colour value is averaged from the value given by the original pixel. This removed the jaggedness of objects and shadows of the scene and makes the image look a lot cleaner and smoother. Without anti-aliasing the application will execute in approximately five seconds whereas with anti-aliasing on the application takes four times longer at approximately 20 seconds. Code implementation shown below.



```
glm::vec3 antiAliasing(glm::vec3 eye, float pixel_size, float xp, float yp){

    float pix_quart = pixel_size * 0.25;
    float pix_3qaurt = pixel_size * 0.75;
    glm::vec3 color(0);
    glm::vec3 avg(0.25);

    Ray ray = Ray(eye, glm::vec3(xp + pix_quart, yp + pix_quart, -EDIST));
    color += trace(ray, 1);

    ray = Ray(eye, glm::vec3(xp + pix_quart, yp + pix_3qaurt, -EDIST));
    color += trace(ray, 1);

    ray = Ray(eye, glm::vec3(xp + pix_3qaurt, yp + pix_quart, -EDIST));
    color += trace(ray, 1);

    ray = Ray(eye, glm::vec3(xp + pix_3qaurt, yp + pix_3qaurt, -EDIST));
    color += trace(ray, 1);

    color *= avg;

    return color;
```

Figure 4: Code implementation of anti-aliasing

Name: Saman Ighani
Student ID: 78966609
User Code: sig16

Multiple Light Sources

There are two light sources in the scene. Shadows for figures are cast using both light sources. Both light source calculations are computed identically with different parameters used respectively.

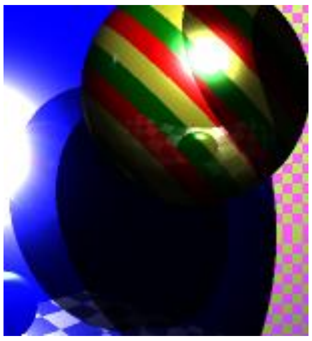
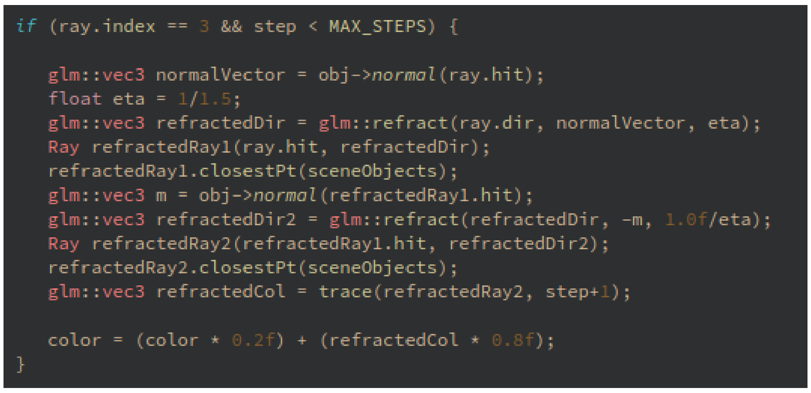


Figure 5: Shadows from two different light sources

Refraction

The sphere in the top right of the scene is refracting light through itself using an eta of 1/1.5 which is of the same refractive index of a glass sphere. It is implemented in code recursively where two refractions rays and normal vectors are calculated for a ray going into the object and a ray that leaves the object. As show in the snippet below.



```
if (ray.index == 3 && step < MAX_STEPS) {

    glm::vec3 normalVector = obj->normal(ray.hit);
    float eta = 1/1.5;
    glm::vec3 refractedDir = glm::refract(ray.dir, normalVector, eta);
    Ray refractedRay1(ray.hit, refractedDir);
    refractedRay1.closestPt(sceneObjects);
    glm::vec3 m = obj->normal(refractedRay1.hit);
    glm::vec3 refractedDir2 = glm::refract(refractedDir, -m, 1.0f/eta);
    Ray refractedRay2(refractedRay1.hit, refractedDir2);
    refractedRay2.closestPt(sceneObjects);
    glm::vec3 refractedCol = trace(refractedRay2, step+1);

    color = (color * 0.2f) + (refractedCol * 0.8f);
}
```

Figure 6: Implementation of refraction

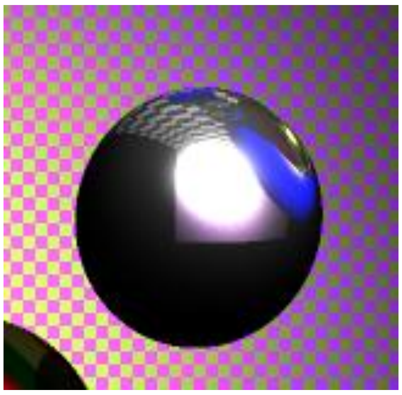


Figure 7: Refractive sphere

Procedural pattern generated onto sphere

A procedural pattern was generated onto the sphere as shown below. Using three different colours for its stripes. A value is found by finding the floating-point remainder of the x and y values of the ray by the number 3. A different coloured stripe is shown depending on the remainder given.

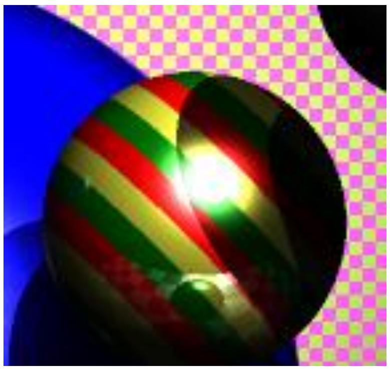
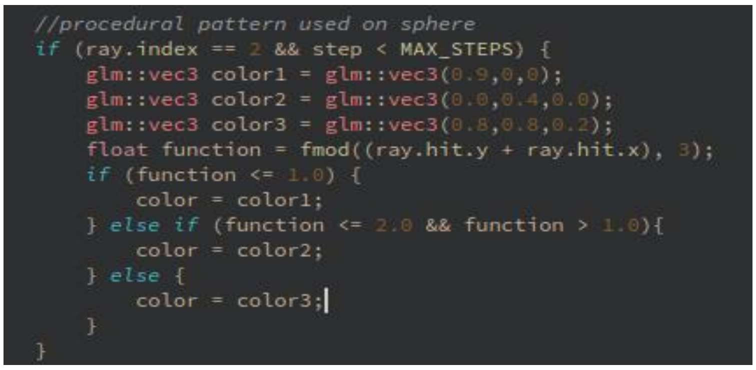


Figure 8: Procedural patter on sphere



```
//procedural pattern used on sphere
if (ray.index == 2 && step < MAX_STEPS) {
    glm::vec3 color1 = glm::vec3(0.9,0,0);
    glm::vec3 color2 = glm::vec3(0.0,0.4,0.0);
    glm::vec3 color3 = glm::vec3(0.8,0.8,0.2);
    float function = fmod((ray.hit.y + ray.hit.x), 3);
    if (function <= 1.0) {
        color = color1;
    } else if (function <= 2.0 && function > 1.0){
        color = color2;
    } else {
        color = color3;
    }
}
```

Figure 9: Implementation of procedural pattern

Name: Saman Ighani
Student ID: 78966609
User Code: sig16

Successes and Failures

Starting with the successes I had was with implementation of the anti-aliasing. It was very straight forward to implement and due to its ability to make the application look smoother it was very satisfying when completed. Other successes I had was with refraction, it did take some trial and error and a long time to eventually figure out the correct implementation but once complete it was very satisfying. The multiple light source implementation was a nice success as originally I had its implementation before all of the refraction and transparency implementations and this caused the shadows to show improperly, eventually I realised I had to do all the light source calculations after the refraction and transparency implementations to show correctly.

A failure I had was trying to map a graphical bmp texture to a sphere, the program would force quit because the bmp image could not map correctly to the sphere. I realised that the centre point was incorrect and once fixed the application would build and execute but the sphere would show out of window behind the camera view. After trial and error with moving the spheres location I still could not get the texture to map correctly to the sphere and so I left this portion out of the source code.

References

- Ray tracing algorithm and information from course lecture notes and labs.