363 – Ray Tracing Assignment

Declaration

I declare that this assignment submission represents my own work (except for allowed material provided in the course), and that ideas or extracts from other sources are properly acknowledged in the report. I have not allowed anyone to copy my work with the intention of passing it off as their own work.

Name: Tom Kirkwood

Student ID: 34767988

Date: 02/06/2023

The implementation of this basic ray tracer includes the following features and extra features: Refraction through a glass sphere, 2 light sources with shadows, 2 reflecting planes facing each other, checkered floor, basic anti-aliasing, fog, depth of field and a textured sphere. The entire scene is surrounded by a box made of 6 different colored planes.

Build instructions:

To build and run the program enter the following commands in linux:

```
g++ Ray.cpp RayTracer.cpp Plane.cpp Sphere.cpp SceneObject.cpp TextureBMP.cpp -I/ -lGL -lGLU -lglut

./a.out
```

Features:

Anti-Aliasing

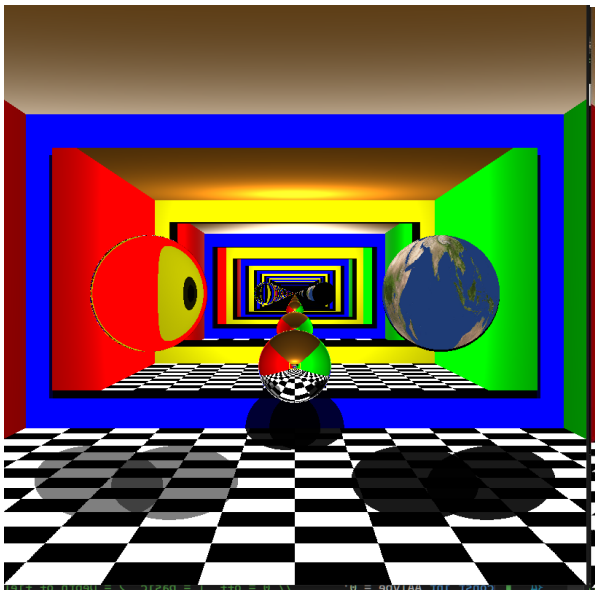


Figure 1: without anti-aliasing

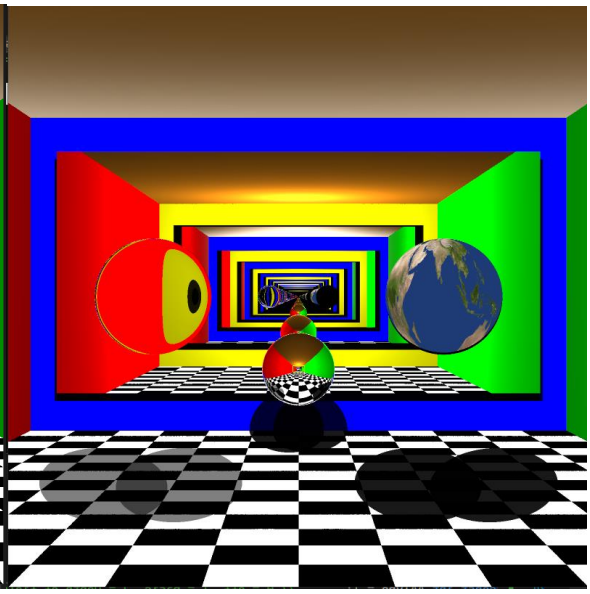


Figure 2 With anti-aliasing

To implement anti-aliasing each pixel is split into 4 sub pixels. A ray is generated for each sub pixel and the rays are traced using the trace function. The color of the 4 sub pixels is added up and the average is taken of the 4 colors as shown below in the code snippet.

```
glm::vec3 col1 = trace(Ray(eye, glm::vec3(x1 + cellXSub, y1 + cellYSub, -EDIST)), 1);
glm::vec3 col2 = trace(Ray(eye, glm::vec3(x2 + cellXSub, y2 + cellYSub, -EDIST)), 1);
glm::vec3 col3 = trace(Ray(eye, glm::vec3(x3 + cellXSub, y3 + cellYSub, -EDIST)), 1);
glm::vec3 col4 = trace(Ray(eye, glm::vec3(x4 + cellXSub, y4 + cellYSub, -EDIST)), 1);


return (col1 + col2 + col3 + col4) / 4.0f;
```

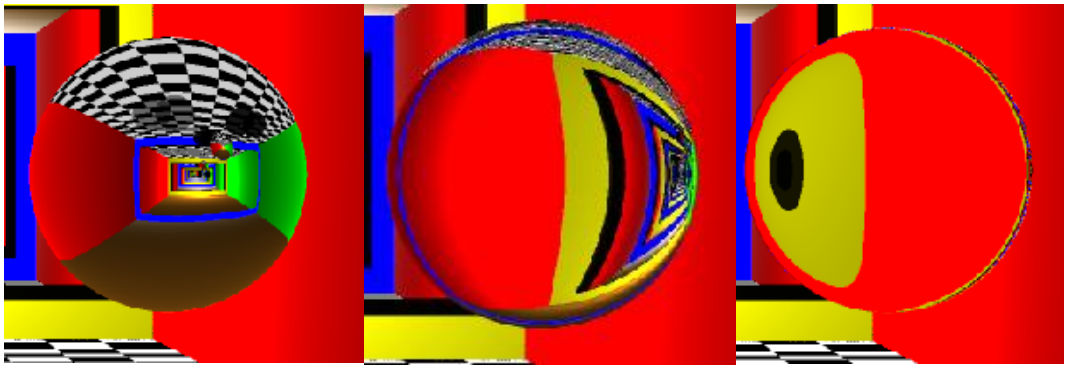
The implemented basic anti-aliasing does have some flaws in that if there are hard lines it will blend the 2 different colors together which can cause some undesired artifacts as shown below. To fix this you could check the rendered pixels next to the new pixel to see if the near by pixel colors are a line or not which was a similar technique I tried using to implement adaptive aliasing.

Anti-aliasing undesired artifacts

Refractive sphere

The refractive sphere with different eta values can be seen below and was modeled using the following equations:

```
float eta = (1 / (obj->getRefractiveIndex()));

glm::vec3 n = obj->normal(ray.hit);
glm::vec3 g = glm::refract(ray.dir, n, eta);

Ray refrRay(ray.hit, g);
refrRay.closestPt(sceneObjects);

glm::vec3 m = obj->normal(refrRay.hit);
glm::vec3 h = glm::refract(g, -m, 1.0f / eta);

Ray refrRay2(refrRay.hit, h);
refrRay2.closestPt(sceneObjects);

if (refrRay2.index == -1) return backgroundCol;
glm::vec3 refColor2 = trace(refrRay2, step + 1);

refrColor += refColor2 * (1 - 0.2f);
return refrColor + (obj->getColor() * (1-obj->getTransparencyCoeff()));
```



eta = 1/1.5 *eta = 1/1.05* *eta = 1/1.01*

Fog

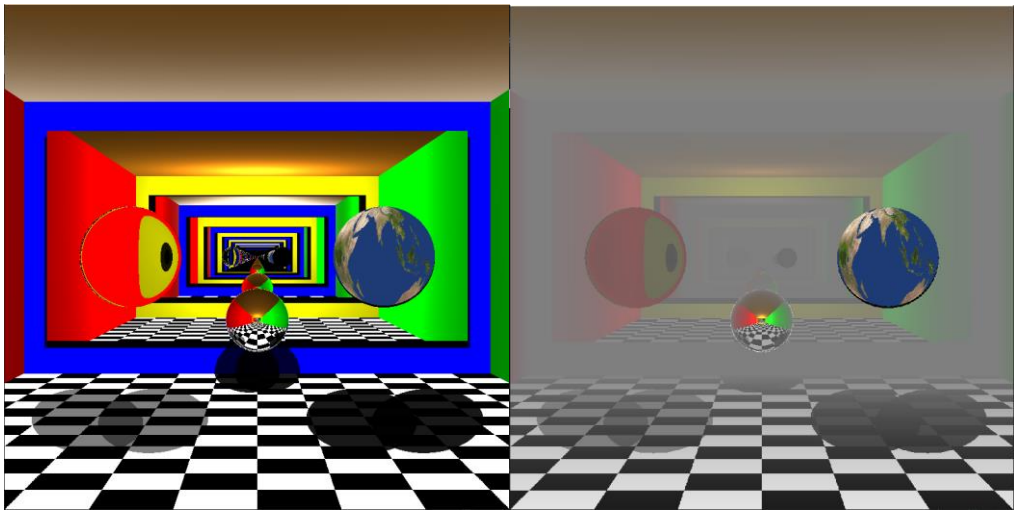


Figure 3: without fog *Figure 4 With fog*

The fog affects in figure 4 was created using the following equations:

$$\lambda = \frac{(ray.hit.z) - z_1}{z_2 - z_1}$$
$$color = (1 - \lambda)color + \lambda\, white$$

The lambda value is clamped to 1 when ray.hit.z is less than z2 and is clamped to 0 when ray.hit.z is greater than z1. In figure 4 z1 is -40 and z2 is -120.

Depth of field

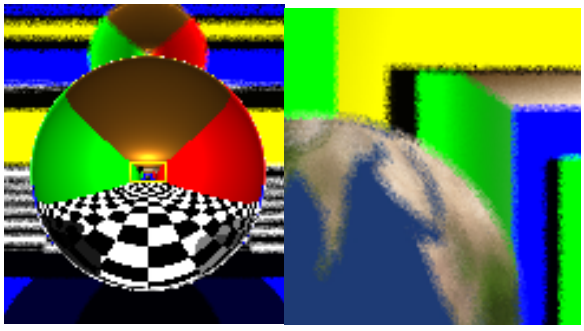


Figure 5: In focus object *Figure 6 Out of focus object*

Depth of field was achieved by generating a random offset of the origin of 10 randomly generated rays using the code below. In figure 5 an object in the focus distance appears to be in focus compared to the objects in figure 6.

```
glm::vec3 colorSum(0);
float rayCount = 10.0f; // Number of rays to be sent out per pixel
const float convergeDistance = 0.1f;

for (int i = 0; i < rayCount; i++) {
    // generate random ray origin offset
    float theta = (rand() / float(RAND_MAX)) * 2.0f * 3.1415f;
    float x = convergeDistance * cos(theta);
    float y = convergeDistance * sin(theta);

    glm::vec3 offset(x, y, 0);
    glm::vec3 target = eye + (ray.dir * DDIST);
    glm::vec3 eyeOffset = eye + offset;

    Ray eyeRay = Ray(eyeOffset, glm::normalize(target-eyeOffset));
    // Add the color of the new ray
    colorSum += trace(eyeRay, 1);
}

return (colorSum / rayCount);
```

Successes:

I think some of the features such as the depth of field and anti-aliasing made the scene look quite a bit nicer and I'm happy with how I implemented them in such a way that makes modifying the features easy. Refraction was the first major feature I implemented which was a success as it was satisfying getting a feature like refraction working as it should

Failures:

Some of the extra features don't render how I would like to such as lighting which doesn't render the back of objects nicely which is why the textured sphere looks black in the reflection as shown in the figure below.

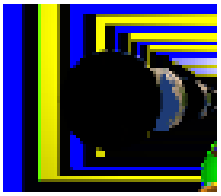


Figure 7: backside of objects appears black in mirrors.

I believe this was due to the ambient lighting or normal vector of the object not being calculated properly. Although I see the basic anti-aliasing as a success, I found trying to implement adaptive aliasing a failure due to not being able to successfully implement it without major artifacts.

Render times:

All renders were done with the max steps set to 50, an EDIST value of 1000 and a resolution of 800 x 800.
When doing a render with no extra features turned on: approximately 1.5 seconds.
With anti-aliasing: approximately 5 seconds.
With depth of field (10 rays per pixel): approximately 13 seconds.

Sources:

Earth texture: https://www.solarsystemscope.com/textures/

The following source was used to help implement different features of ray tracing:

https://www.youtube.com/watch?v=Qz0KTGYJtUk&ab_channel=SebastianLague