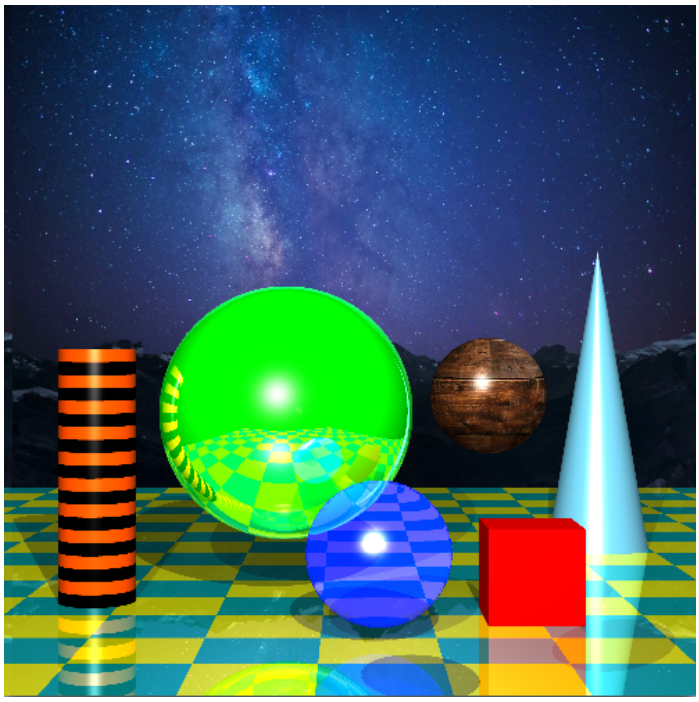
COSC363: Computer Graphics Assignment 2 Ray Tracer

1. The Basic Ray Tracer

The basic ray tracer implements a simple scene containing the floor plane, spheres and a cube constructed using six planer surfaces. The classes involved were given in the lab 7 / 8 files



BUILD COMMAND:

```
g++ -Wall -o "%e" RayTracer.cpp Ray.cpp SceneObject.cpp Sphere.cpp Plane.cpp Cylinder.cpp
Cone.cpp TextureBMP.cpp -lm -lGL -lGLU -lglut
```

A. The light vector is the vector from the point of intersection towards the light source which is defined as:

```
glm::vec3 lightVector = light - ray.xpt;
```

B. To add shadows to the scene, we create a shadow ray from the point of intersection towards the light source and the compute of the closest point of intersection of objects with the ray.

```
ray.closestPt(sceneObjects);
```

The following code is the condition for deciding whether point P is in the shadow or not.

```
if ((lDotn <= 0) || ((shadow.xindex > -1) && (shadow.xdist < lightDist))) colorSum +=
ambientCol * materialCol; // If the value of l.n. is negative, return only the ambient
component
else colorSum += ambientCol * materialCol + (lDotn * materialCol + specular) *
lightIntensity; // Otherwise it should return the sum of ambient and diffuse colours
```

C. The reflection on the first sphere is done by the recursive algorithm introduced in the labs which accumulates colour values along secondary rays. The variable colourSum stores the accumulated color values.

```
    // -------- Reflection on first sphere ----------------------------
    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); }
```

D. The box / cube is built with my drawCube function which creates six planes (front, left, back right..) so it creates six pointers of plane.cpp to construct the cube.

E. The plane is constructed with the plane.cpp class given in the labs. The pointer for the floor plane is created by defining four parameters which are the vertices of the floor plane which is shown flat across the bottom of the scene. The texture for the floor is given by the following formula which will set a different material color on neighbouring quads generating a checkered pattern.

```
int texcoords = (int)((ray.xpt.x + 50) / 8) % 2;
int texcoordt = (int)((ray.xpt.z + 200) / 8) % 2;
if((texcoords && texcoordt) || (!texcoords && !texcoordt)){
   materialCol = glm::vec3(0,0.5,0.5);
   } else {
   materialCol = glm::vec3(0.75,0.75,0); }
```

2. Extensions

2.1 Primitives other than a plane, sphere or box (CYLINDER)

A cylinder object class is made to generate a cylinder shaped primitive in the scene. The class contains a point of intersection and a surface normal. Referring to the slide [09]-38:

- The ray equation is given by: $x = x_o + d_x t \; z = z_o + d_z t$ and where c = center, R = radius, d = direction of the ray and o is the ray origin
- The intersection equation is given by:
  $t^2(d_x^2+d_z^2) + 2t\{d_x(x_o-x_c) + dz(z_o-z_c)\}+\{(x_o-x_c)^2+(z_o-z_c)^2-R^2\} = 0$

```
float a = pow(dir.x, 2) + pow(dir.z, 2);
float b = 2 * (dir.x * (posn.x - center.x) + dir.z * (posn.z - center.z));
float c = pow((posn.x - center.x), 2) + pow((posn.z - center.z), 2)- pow(radius, 2);
```

- The surface normal vector:
  $n = (x - x^c, 0, z - z_c)$

```
glm::vec3 n = glm::vec3 ((p.x - center.x) ,0 ,(p.z - center.z));
```

2.2 Primitives other than a plane, sphere or box (CONE)

A cone object class is made to generate a cone shaped primitive in the scene. The class contains a point of intersection and a surface normal. Referring to the slide [09]-45.
Equations used:

- The ray equation is given by: $x = x_o + d_x t; \; y = y_o + d_y t; \; z = z_o + d_z t;$
- $\tan(0) = R/h$ (0 = half cone angle)
- Equation of the cone with base at (x_c , y_c , z_c): $(x-x_c)^2 + (z-z_c)^2=(R/h)^2 (h-y+y_c)^2$

```
glm::vec3 d = pos - center;
   float yd = height - pos.y + center.y;
   float stan = pow((radius / height), 2);
   float a = (dir.x * dir.x) + (dir.z * dir.z) - (stan*(dir.y * dir.y));
   float b = 2 * (d.x * dir.x + d.z * dir.z + stan*yd*dir.y);
   float c = pow(d.x, 2) + pow(d.z, 2) - (stan * pow(yd, 2));
   float delta = b * b-4*(a*c);
```

- The surface normal vector

```
    float r = sqrt(pow(p.x - center.x, 2) + pow((p.z - center.z), 2));
   glm::vec3 n= glm::vec3 (p.x - center.x, r * (radius/height), p.z -
center.z);
```

2.3 Multiple light sources including multiple shadows generated by them

By adding another light source the same way as the first light source was made, an second light vector is added to the scene. By having a different light intensity for each light vector each light vector will have two shadows but with different darkness due to having different intensities which is visible by the difference of figure 1.1 and 1.2.

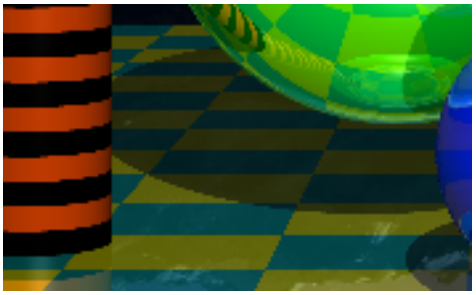


Figure 1.1

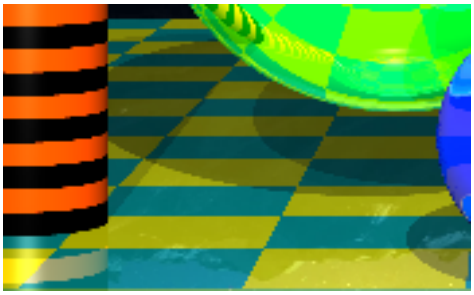


Figure 1.2 multiple

2.4 Refraction of light through an object

Refraction is added to the blue sphere. The algorithm recursively traces the rays, where for each step the ray is traced twice. Two normal vectors are computed so the refractive rays are transmitting in and out of the sphere. Referring to figure 1.3, 1.4, 1.5 we can see the differences with the ETA values as demonstrated in lecture slides [09]-23,24

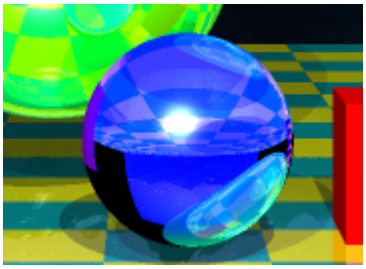


Figure 1.3 ETA: 1.5

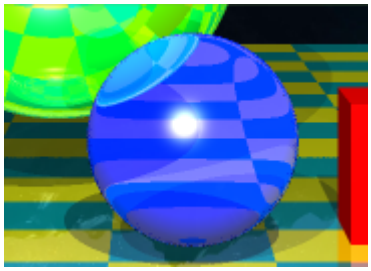


Figure 1.4 ETA: 1.01

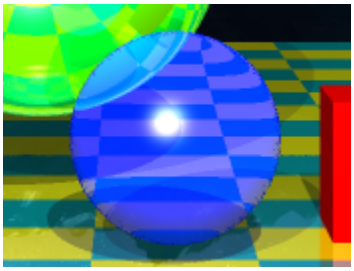


Figure 1.4 ETA: 1.003

2.5 Transparent object

By multiplying the same sphere's current colour by the transparency float value which is set to 0.80 corresponding to 80% transparency and adding the colour that is refracted through the sphere. We can see on figure 1.6, 1.7 and 1.8 the differences this the transparency values.

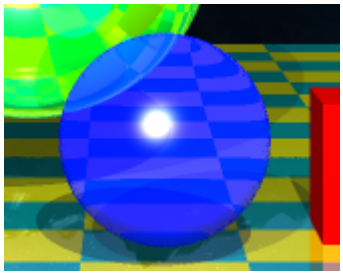


Figure 1.6 50%

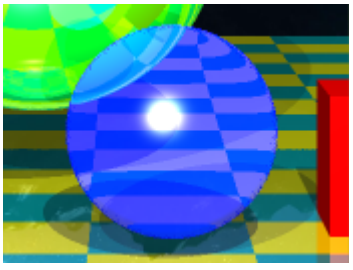


Figure 1.7 80%

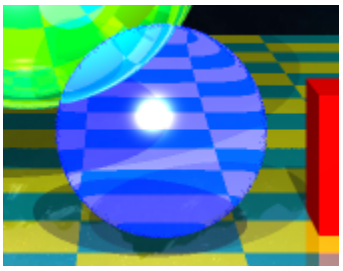


Figure 1.8 99%

```
  if(ray.xindex == 1 && step < MAX_STEPS){
        glm::vec3 refractedDir = glm::refract(ray.dir, normalVector, 1.0f/ETA);
        Ray refractedRay(ray.xpt, refractedDir);
        refractedRay.closestPt(sceneObjects);
        if (refractedRay.xindex == -1) return backgroundCol;

        glm::vec3 normalVector2 =
sceneObjects[refractedRay.xindex]->normal(refractedRay.xpt);
        glm::vec3 refractedDir2 = glm::refract(refractedDir, -normalVector2, ETA);
        Ray refractedRay2(refractedRay.xpt, refractedDir2);
        refractedRay2.closestPt(sceneObjects);
        if (refractedRay2.xindex == -1) return backgroundCol;

        glm::vec3 refractedCol = trace(refractedRay2, step + 1); //Recursion!
        colorSum += colorSum * transparency + refractedCol * (transparency); }
```

2.6 A non-planar object textured using a procedural pattern

The cylinder object is textured with a procedural pattern of strips across the horizontal of the cylinder. The pattern will interchange colours on even and odd y point values. So that it creates a stripe pattern between black and orange. Which it stored in the material colours variable..

```
    if(ray.xindex == 5) {
        if (int(ray.xpt.y - 5) % 2 == 0){
            materialCol += glm::vec3(0, 0, 0); // Black
        } else{
            materialCol += glm::vec3(1, 0.3, 0); // Orange}}
```

2.7 A non-planar object textured using an image

The sphere is textured by the bitmap woodTexture.bmp which maps the coordinates of the point of intersections to image coordinates, and assigns the colour values to that point using the getColorAt function which is defined in the TextureBMP.cpp class given in the labs.

```
    if(ray.xindex == 2){
        glm::vec3 center(6.0, -2.0, -60.0);
        glm::vec3 d = glm::normalize(ray.xpt - center);
        float u = (0.5 - atan2(d.z, d.x) + M_PI) / (2 * M_PI);
        float v = 0.5 + asin(d.y) / M_PI;
        materialCol = texture.getColorAt(u, v);}
```

2.8 Anti-aliasing

The result of the ray tracing algorithm is it distorts the artefacts such as jagged edges on polygons and shadows in the scene. Supersampling is used to generate rays through each pixel, it divides the pixel into four segments and computes the average of the colour values which is then returned.

```
    Ray ray = Ray(eye, glm::vec3(xp + quarter, yp + quarter, -EDIST));
    ray.normalize();
    colorSum += trace(ray, 1);

    ray = Ray(eye, glm::vec3(xp + quarter, yp + three_quarter, -EDIST));
    ray.normalize();
    colorSum += trace(ray, 1);
```

```
    ray = Ray(eye, glm::vec3(xp + three_quarter, yp + quarter, -EDIST));
    ray.normalize();
    colorSum += trace(ray, 1);

    ray = Ray(eye, glm::vec3(xp + three_quarter, yp + three_quarter, -EDIST));
    ray.normalize();
    colorSum += trace(ray, 1);
```

To compute the average: `colorSum*= average;` which is then returned.
The following figure and 1.9, 2.0, 2.1 and 2.2 shows the differences with anti-aliasing.

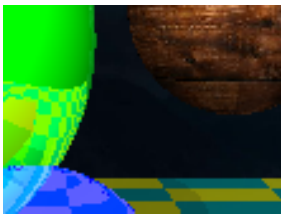


Figure 1.9 no anti-alias

Figure 2.0 anti-alias

Figure 2.1 no anti-alias

Figure 2.2 anti-alias

3. Successes and its failure

There was a problem when trying to implement multiple light sources which was overcome by using a lightIntensity variable. Initially the light sources would look identical in intensity and the shadows were too dark or the light was too strong. By adding the light intensity variable and making the second light source use a (1-light intensity) it its calculation, I was able to make it so the shadows have different intensities and therefore easier to visualize two light sources in the scene render.

Another problem was texturing the sphere, adding the imported texture to the sphere. At first the image would not map to the sphere the way I wanted. The first problem was because of how the texture was being centered to the object, by matching the values of the center to the position of the sphere, this problem was overcome. The second problem was on the texture image itself, I did not see that converting a png image to a bmp still includes the chequered background of the png which would not be there in a png format but otherwise there in a bmp format. So by getting an image with no transparent background, this problem was fixed.

There is a failure on the implementation attempt on adding shear to a cylinder.
Following the equations:

- $[x - (x_c + ky)]^2 + (z - z_c)^2 = R^2$
- $n = (x - x_c * ky , 0, z - z_c)^2$

And the code:

```
    float k = 20.0
    float a = pow(dir.x, 2) + pow(dir.z, 2);
    float b = 2 * (dir.x * (posn.x - center.x) + dir.z * (posn.z - center.z));
    float c = pow((posn.x - (center.x + k * posn.y)), 2) + pow((posn.z -
center.z), 2)- pow(radius, 2);
```

Surface Normal vector

```
glm::vec3 n = glm::vec3 (((p.x - center.x) - k * p.y) ,0 ,(p.z - center.z));
```

But the shear was not added and the cylinder remained the same, the problem was that I wasn't very sure how the k value work. Which would have affected my implementation approach.

References: COSC363 Lecture and lab notes my R. Mukundan
Photos: https://unsplash.com/search/photos/wood-texture