1 Discussion

1.1 Building

```
user@domain:~/path/to/project/COSC363-Assignment-2$ ./build.sh
```

If `build.sh` isn't executable, then apply the following permissions:

```
user@domain:~/path/to/project/COSC363-Assignment-2$ chmod +x ./build.sh
```

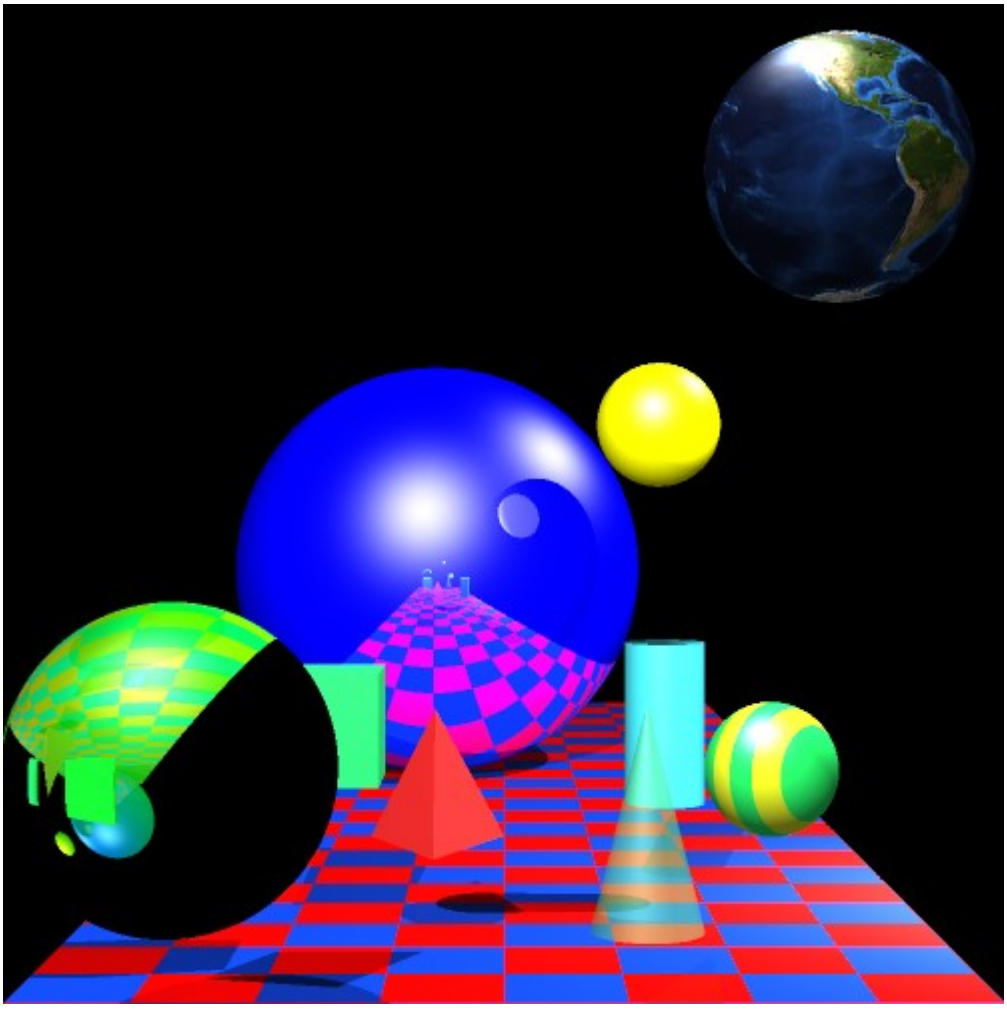
1.2 Successes

I am particularly happy that the I completed all the extra features I intended to attempt. Additionally, the final product doesn't look terrible, which is always a bonus.

1.3 Failures

Initially, the triangles which composed the tetrahedron did not have lighting properly applied. Initially, I discovered that removing one of the conditions for shadows fixed the problem. A day later, I realised that the "fix" had disabled all shadows. After traversing my Git commit history, I found that the root cause was that I had defined the triangles of the tetrahedron in a *clockwise* manner. After switching to an *anticlockwise* direction and adding back the deleted code, the tetrahedron gained realistic lighting and the shadows reappeared.

Whilst looking for the code which had disabled shadows, I realised that I had not properly distinguished between `materialCol` and `colorSum`. In a rather lackadaisical fashion, expressions which should have been assigned to `materialCol` were often assigned to `colorSum`. However, this seemed to have minimal impact.

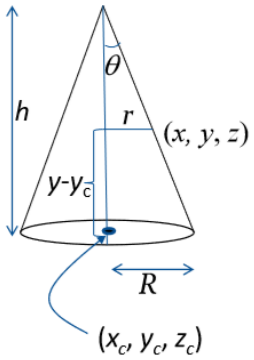


2 Extra Features

2.1 Cone

A cone is a subclass of the abstract class `SceneObject`.

A cone is given by the formula $(x - x_c)^2 + (z - z_C)^2 = R^2$ ①.



The height of a cone in relation to its radius, is $\tan(\theta) = \frac{R}{h}$[1], where θ is half the cone angle.

The radius of the cone at a given height y is given by:

$$r = \left(\frac{R}{h}\right)(h - y + y_c) \text{ ②}$$

To obtain the equation for the points of intersection, equation ② is substituted into equation ①, to form:

$$(x - x_c)^2 + (z - z_c)^2 = \left(\frac{R}{h}\right)^2 (h - y + y_c)^2 \text{ ③}$$

The ray equations of $x = x_0 + d_x t; y = y_0 + d_y t; z = z_0 + d_z t$ is then substituted into equation ③, and solved for t. The resulting equation is

$$at^2 + bt + c = 0$$

where

$$a = d_x^2 + d_z^2 - \frac{R^2}{h^2} d_y^2$$

$$b = 2\left(d_x(x_0 - x_c) + d_z(z_0 - z_c) - \frac{R^2}{h^2} d_y(y_c + h - y_0)\right)$$

$$c = d_x^2 + d_z^2 - \frac{R^2}{h^2}(y_c + h - y_0)^2$$

The surface normal vector is given by:

$$\vec{n} = \frac{\vec{v}}{\|\vec{v}\|}$$

where

$$\vec{v} = (x_0 - x_c, r \times \frac{R}{h}, z_0 - z_c)$$

$$r = \sqrt{(x_0 - x_c)^2 + (z_0 - z_c)^2}$$

[1] R is the radius of the cone; h is the height of the cone. The bottom of the centre of the cone is given by (x_c, y_c, z_c).

2.2 Cylinder

The intersection equation is:

$$t^2(d_x^2 + d_z^2) + 2t\big(d_x(x_0 - x_c) + d_z(z_0 - z_c)\big) + (x_0 - x_c)^2 + (z_0 - z_c)^2 - R^2 = 0$$

Rays are checked on both the first and second intersection of the cylinder. This ensures that both the front and the back of the cylinder would be displayed. Otherwise, only the front half of the cylinder would be rendered.

The surface normal vector is given by

$$\vec{n} = \left(\frac{x_0 - x_c}{R}, 0, \frac{z_0 - z_c}{R}\right)$$

In order to prevent artifacts from appearing on the front of the cylinder, checks for the ranges of t_1 and t_2 were performed. If either t_1 or t_2 were below a threshold 0.01, then -1.0 is returned, indicating that there is no intersection.

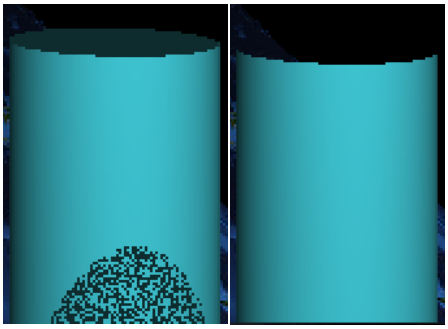


Figure 1: Before and after checks for t were performed

2.3 Tetrahedron

The tetrahedron is constructed out of four triangles. The `Triangle` class is a subclass of the `SceneObject` abstract class.

The surface normal vector is given by

$$\vec{n} = \frac{(\vec{b} - \vec{a}) \times (\vec{d} - \vec{a})}{\|(\vec{b} - \vec{a}) \times (\vec{d} - \vec{a})\|}$$

The intersect equation and method are the same as for a plane.

2.4 Multiple light sources

The scene includes two light sources. For each light source, ray-tracing features like shadows and lighting are performed separately. This can be shown with the dual shadows of the cylinder, and the two specular highlights on the large, reflective, blue sphere. To achieve this in the source code, all the lighting functionality is duplicated.

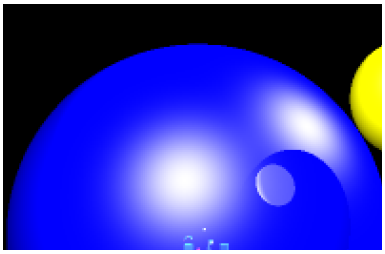


Figure 3: Multiple specular highlights

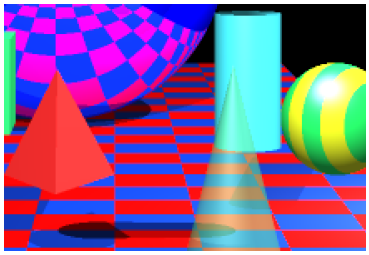


Figure 2: Cylinder and its two shadows

2.5 Transparency

Transparency is achieved by creating a new ray. The source point of the new ray is the closest point of intersection of the incoming ray with the object. The direction of the new ray is the direction of the incoming ray. The colour is then retrieved by tracing the new ray. The current colour is scaled by the transparency constant, and the obtained refractive colour is scaled by $1 - constant$. This results in the transparent object

The transparent object also has dimmer shadows. This is done by scaling the shadows generated from the primary and secondary lights.

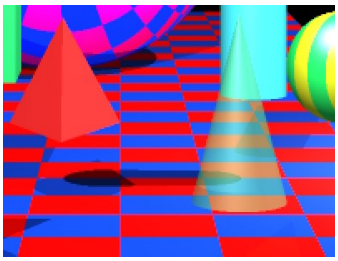


Figure 4: Transparent cone. One of its shadows can be faintly seen at the bottom right of the image.

2.6 Refraction

Refraction is achieved by calculating a ray g which travels inside a sphere, and a ray h which travels outside the sphere. The angles between the input ray d (referred to as `ray` in the source code), the interior ray g, and the outgoing ray h are determined by the η constants and the surface normal n. The interior of the sphere has an η value of 1.5, with the area outside the sphere having an implied η of 1.

The refractive sphere is also transparent and had a green colouring, hence the green tinge of the floor plane in the refraction.

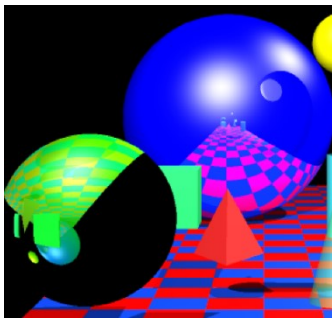


Figure 5: Refractive sphere, and surrounding scene

2.7 Anti-aliasing

Anti-aliasing is a method of preventing jagged edges for polygons and shadows. The type of anti-aliasing implemented is super-sampling, which generates four rays through each quarter of a square pixel and computes the mean of the colour values. As shown in Figure 6, the cylinder with super-sampling has smoother edges.

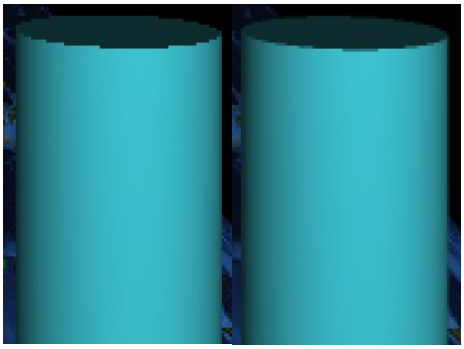


Figure 6: Cylinder on the left, without super-sampling. Cylinder on the right, with super-sampling

2.8 Texturing of a non-planar object

A sphere is textured using an image of the Earth [1]. This involved mapping three-dimensional coordinates to latitude and longitude, which were then mapped to the appropriate two-dimensional coordinates on the rectangular image. [2]

The two-dimensional coordinates for the image (u, v) are derived from:

$$u = 0.5 + \frac{\tan^{-1}(d_z, d_x)}{2\pi}$$

$$v = 0.5 - \sin^{-1}\frac{d_y}{\pi}$$

Figure 7: Sphere mapped with a texture of the Earth

However, in order to make the northern hemisphere appear on the top of the sphere, ±operators were switched, resulting in the following source code:

```
if (ray.xindex == 16) {
  glm::vec3 d = glm::normalize(ray.xpt - earthCenter);
  float u = 0.5 - atan2(d.z, d.x) / (2 * M_PI);
  float v = 0.5 + asinf(d.y) / M_PI;
  materialCol = earthTexture.getColorAt(u, v);
}
```

2.9 Procedural textures

A sphere is procedurally textured using two colours to form stripes. The x and z coordinates of the ray are taken and depending on whether the sum is even or odd, one of two colours is displayed.

Source code:

```
if (ray.xindex == 17) {
  int value = (int)(ray.xpt.x + ray.xpt.z) % 2;
  if (value == 0) {
    materialCol = glm::vec3(0.901, 0.941, 0.156);
  } else {
    materialCol = glm::vec3(0.156, 0.941, 0.403);
  }
}
```

3 References

[1] NASA, “DECEMBER, BLUE MARBLE NEXT GENERATION W/ TOPOGRAPHY AND BATHYMETRY,” [Online]. Available: https://visibleearth.nasa.gov/view.php?id=73909. [Accessed 30 May 2016].

[2] Wikimedia Foundation, “UV mapping,” Wikimedia Foundation, 30 November 2018. [Online]. Available: https://en.wikipedia.org/wiki/UV_mapping. [Accessed 30 May 2019].