Primitive Ray Tracer

Jet Hughes

1 Introduction

This report describes the implementation of a simple ray tracer. We will discuss the implementation of ray-object intersections, lighting, shadows and reflections, constructive solid geometry and super-sampling. We also present our results and identify known issues.

2 Implementation

The ray tracer was developed on Ubuntu using GCC 12. The implementation includes:

- Ray-object intersections for bounded planes, cubes, capped cylinders, and tori.
- The Phong illumination model with shadows and reflections.
- Constructive Solid Geometry (CSG) operations.
- Basic grid super-sampling for anti-aliasing.
- A Spotlight light source

2.1 Primitives

This section details the implementation of various geometric primitives used in the ray tracer. A scene is made up of many primitive objects, one or more light sources, and a camera. To render a scene, we cast rays from the centre of the camera through each pixel in the image. The colour of the pixel is computed by tracing the ray through the scene. If the ray intersects an object, the pixel's colour is computed depending on the ray's properties, the objects' surface, and the light's properties. A ray is defined using the parametric equation:

$$\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d} \tag{1}$$

Where $\boldsymbol{r}(t)$ is the position of the ray at time t, $\boldsymbol{o}$ is the origin of the ray, and $\boldsymbol{d}$ is the direction of the ray. The intersection of a ray with an object is calculated by substituting the ray equation into the object's equation and solving for t. The intersection point is then calculated by substituting the value of t back into the ray equation. We then calculate the normal at the intersection point. Each object is defined simply by an equation that describes its surface.

Plane We define a plane at the world origin with its normal facing the positive z axis. The surface of the plane is defined by the equation:

$$\boldsymbol{n} \cdot (\boldsymbol{p} - \boldsymbol{o}) = 0 \tag{2}$$

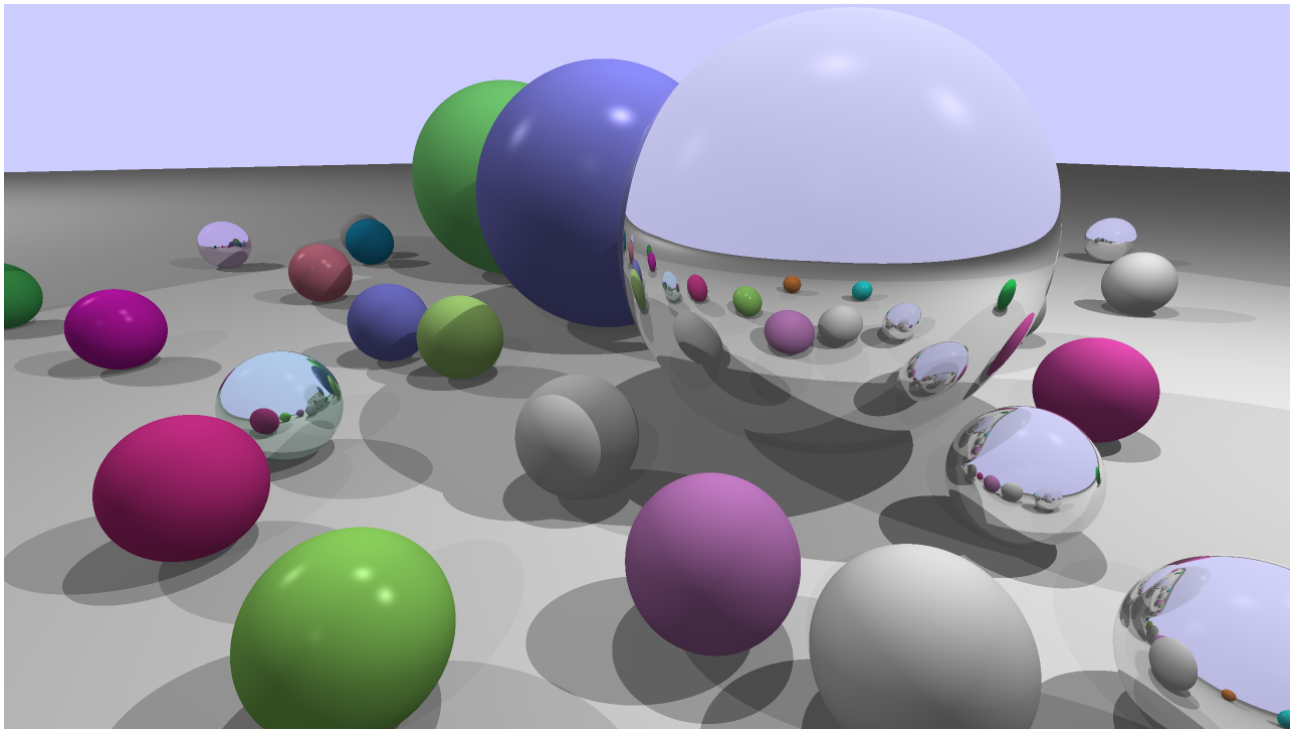


Figure 1: Spheres

Where $\boldsymbol{n}$ is the normal to the plane, $\boldsymbol{p}$ is a point on the plane, and $\boldsymbol{o}$ is the origin of the plane. We can then calculate the intersection of a ray with the plane by substituting the ray equation into the plane equation and solving for t. To define the bounds of the plane we check if the x and y coordinates of the intersection point are smaller than the length of one side of the plane.

Cube The cube primitive is defined as six planes, one for each face of the cube. The cube is centred at the origin with a side length of 2. We calculate the intersection of a ray with the cube by checking if the ray intersects any of the six planes. If the ray intersects a plane, we check if the intersection point is within the bounds of the cube. The normal at the intersection point is the normal of the plane that the ray intersected.

Cylinder The cylinder primitive is centred at the origin with a radius of 1 and a length of 2 along the z-axis. The equation for the body of the cylinder is:

$$x^2 + y^2 = 1 \tag{3}$$

The caps of the cylinder are defined as circular planes at $z = -1$ and $z = 1$. If the ray intersects a cap the normal is the normal of the cap. If the ray intersects the body of the cylinder, the normal is simply:

$$n = \begin{bmatrix} x & y & 0 \end{bmatrix}^T \tag{4}$$

If the ray intersects either cap the normal the normal of the intersected cap.

Torus The torus primitive is defined by its major radius R and minor radius r. The surface of the torus is defined by the equation:

$$(x^2 + y^2 + z^2 + R^2 - r^2)^2 = 4R^2(x^2 + y^2) \tag{5}$$

The intersection of a ray with the torus is calculated by solving the quartic equation that results from substituting the ray equation into the torus equation. The full derivation is not shown here.

2.2 Constructive Solid Geometry

Constructive Solid Geometry (CSG) operations are implemented to create complex shapes by combining simpler primitives using the boolean operations: union, intersection, and difference. The CSG operations are represented as a state machine with four states: In A and B, Not in A or B, In A only, and In B only. Transitions between states occur at intersection points depending on which object is intersected. We first find intersections of the ray with each object and sort them by distance. Each intersection is flagged with the object it belongs to. We then iterate through the intersections and track which object(s) the ray is currently inside.

Union For the union operation we simply return the first intersection.

Intersection For the intersection operation, we keep only the hits that occur when the ray is in A and B.

Difference For the difference operation, we keep only the hits that occur when the ray is inside A and not B.

2.3 Lighting

Phong Illumination The Phong illumination model is used to compute the lighting for the objects. The Phong illumination model defines the illumination of a pixel (I) using the equation:

$$I = k_a I_a + \sum_{i=1}^{n} (k_d I_i(\hat{\boldsymbol{n}} \cdot \hat{\boldsymbol{l}}) + k_s I_i(\hat{\boldsymbol{r}} \cdot \hat{\boldsymbol{v}})^{\alpha}) \tag{6}$$

The model splits light into three components: ambient, diffuse, and specular. The ambient component is used to simulate indirect lighting. The diffuse component shades the object based on the angle between the normal and the light source. The specular component defines highlights on the object. It is based on the angle between the viewing direction and the reflection of the light source. The value α is used to define the shininess of the objects.

Shadows Shadows are computed by casting a ray from the intersection point towards the light source. If the ray hits an object before it hits the light, that pixel is in shadow.

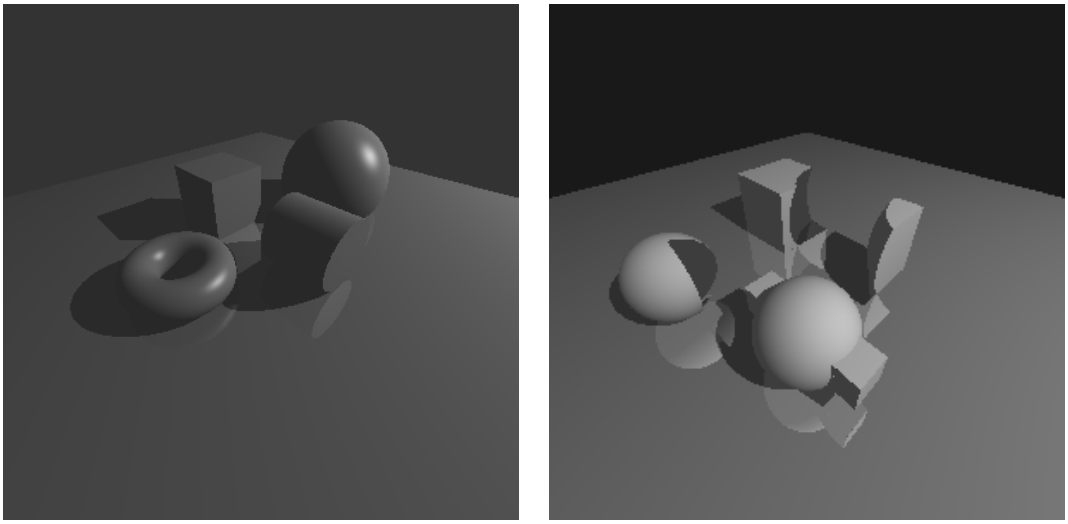
Reflections Reflections are calculated by bouncing rays off the hit points. The reflected ray is found by reflecting the view direction around the surface normal. The direction of the reflected ray is given by:

$$\widehat{\boldsymbol{m}} = 2(\hat{\boldsymbol{n}} \cdot \hat{\boldsymbol{v}})\hat{\boldsymbol{n}} - \hat{\boldsymbol{v}} \tag{7}$$

The reflected ray is then:

$$\boldsymbol{r}(t) = \boldsymbol{h} + t\widehat{\boldsymbol{m}} \tag{8}$$

Reflections are computed recursively and limited by some number of maximum bounces. The resulting colour is computed as a weighted sum of the colour at each bounce.



(a) Scene containing the five primitives (b) Scene with CSG operations

Figure 2: Examples of the four shapes, and the CSG operations

2.4 Super-sampling

Super-sampling is implemented to reduce aliasing and enhance image quality. We cast four evenly distributed rays for each pixel and take the average of the four colours.

2.5 Spotlight

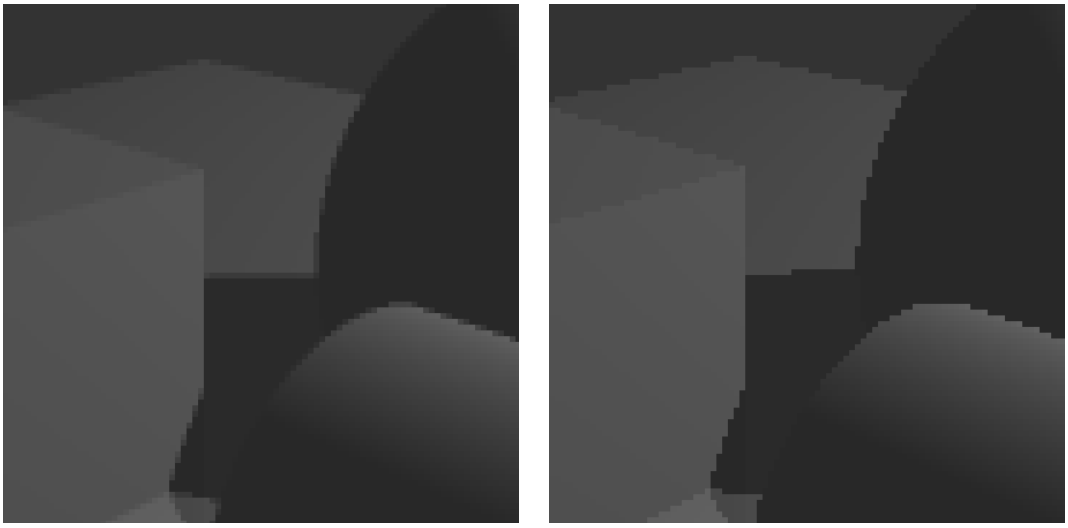
The spotlight casts light from a point in a directed cone. The illumination of a point is calculated by comparing the angle between the direction of the light, and the direction from the point to the light. If the point lies within the cone the illumination is computed according to the inverse square law. Otherwise, the illumination is zero. This is a simplified model of a spotlight with a hard shadow. In reality, the shadow may be softer.

3 Testing and Results

Figure 1 shows a scene with many spheres with different material properties. It showcases the Phong illumination model, reflections, and shadows. Figure 2a showcases each of the five primitives. Figure 2b shows the CSG operations, union, difference, and intersection. There are some issues with the CSG operations which will be discussed later. Figures 3a and 3b show close-ups of the same scene with super-sampling on and off respectively. Figure 4 shows a scene lit by a spotlight.

4 Issues

CSG operations Figure 2b shows a simple scene with the three CSG operations. Two objects created using the Difference operation are shown at the back of the scene and in the far-left corner. Both objects are rendered incorrectly. The surface created by the subtraction is in shadow despite direct illumination. Figure 5(a) shows the same scene with shadows turned off. In this case, the



(a) Scene with super-sampling (b) Scene without super-sampling

Figure 3: Super-sampling vs no super-sampling

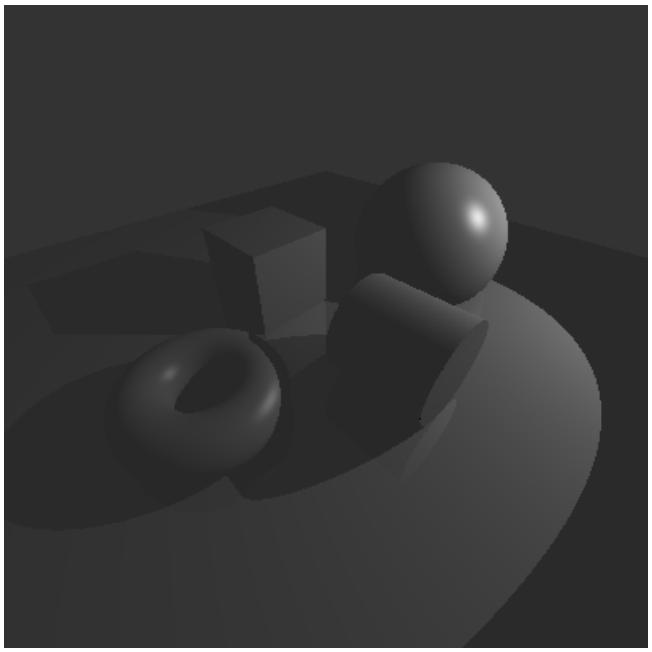
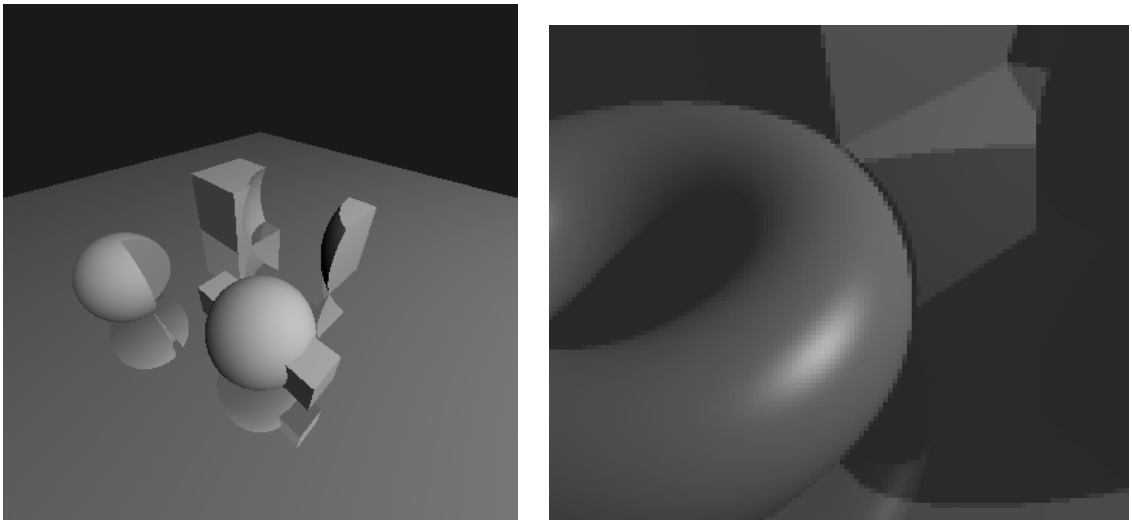


Figure 4: Scene lit by a spotlight



(a) CSG scene with no shadows.

(b) Torus edge artifacts

Figure 5: Images demonstrating known issues.

surface created by subtraction is shaded correctly. Furthermore, the reflections of all CSG objects, bar the union object, are incorrect.

Edge Artifacts Some artefacts around the edges of objects are visible, as shown in 5(b).