RAY TRACING

COSC363 Assignment 2

Ahmad Alsaleh (Omar Al)

14749959

User ID: aaa113

University of Canterbury

This assignments goal was to implement a ray tracer that can handle different types of geometric objects and global illumination features.

Scene Description

The initial scene opens to a camera (eye projection) pointed at a scene with a ground plane that has different types of objects on it.

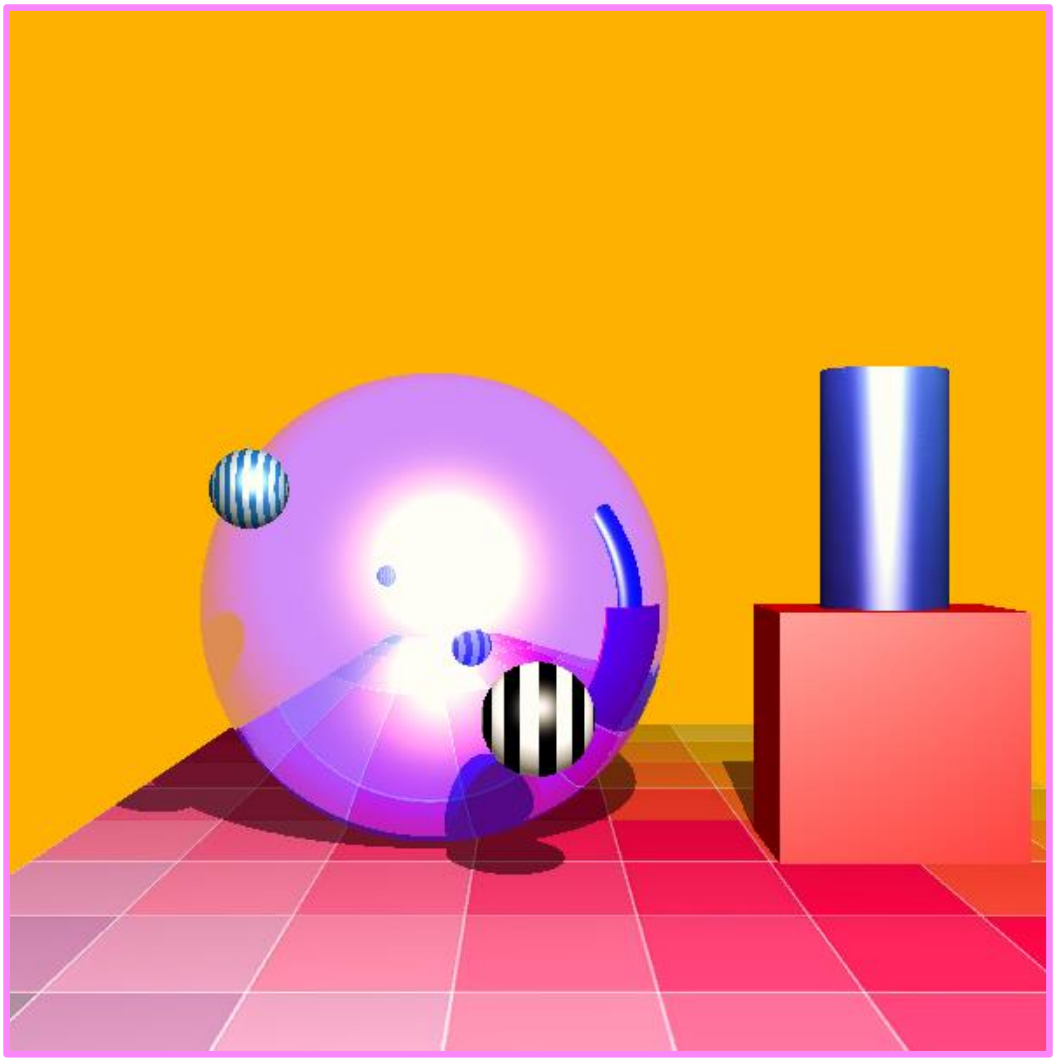

FIGURE 1: SCENE GENERATED FROM RAYTRACING.CPP

SUCCESSES

- Able to implement many extra features
- Rendering quality fairly high and visually pleasing overall
- Excellent implementations of texturing, shadows and reflections
- Relatively low rendering times considering quality

DIFFICULTIES

- Slow movement with camera due to render times (discussed in detail below)
- Unable to implement cone after many attempts due to wrong equations
- Unsuccessful implementation of a spotlight resulting in removing it completely

Please note basic requirements are in black and extra features are highlighted with orange text.

The following models/objects are in the scene:

- Ground plane as a floor textured using an image
- A reflective sphere with a purple color
- A sphere textured using an image (blue and white stripes)
- A sphere textured using a procedural pattern (black and white stripes)
- A red cube constructed using six planes
- A cylinder with a default color of blue

The following features are also implemented:

- One light source, diffuse and specular reflections are also generated
- Shadows: All the objects in the scene have shadows calculated on both the plane and other objects
- Reflections: The scene contains a blue reflective sphere which reflects all the objects in the scene off it
- Camera motion: Camera moves in six directions when keys are pressed

Control Functions

Special Keys:

- Arrow key up: moves camera forward
- Arrow key down: moves camera down
- Arrow key right: move camera right
- Arrow key left: moves camera left
- F1 key: moves camera higher
- F2 key: moves camera downwards

Ray Tracer Description

The project initially started with creating a simple scene consisting of a set of spheres. A ray tracing algorithm uses a simple camera model where the origin of the rays are specified at the eye position (camera position) and the view axis is along the –z direction. The scene being viewed draws each cell as a quad, and a ray is generated from the origin through the center of each cell.

Lighting, Shadows, and Colors

In the trace function, the function computes the color using Phong's illumination model. The light vector is then calculated then normalized. The dot product of the vector and normal vector is calculated, if the result is positive diffuse colors are added to the ambient color.

For computing specular reflections of the light, we first calculate the reflection vector r, using the reflect function in the GLM library. Then the specular term is calculated and added to the ambient and diffuse components calculated above. This will give the sphere specular reflections coming off the light source which is defined near the eye position in the scene.

For the shadows, we create a shadow ray from the closest point of intersection from the eye view towards the light source. If the shadow ray intersects an object, the the distance from the closest point

of intersection along the shadow ray is smaller than the distance to the light source, this indicated that the point is within the shadow.

Spheres Equations

The equation of a sphere centered with radius r is:

$$(p - C) \bullet (p - C) = r2$$

We consider a ray given by the equation

$$p = p0 + t\,d$$

At the point of intersection, both equations are true.

Therefore, $(p0 + t\,d - C) \bullet (p0 + t\,d - C) = r2$

$(s + t\,d) \bullet (s + t\,d) = r2$, where $s = p0 - C$

$$(d \bullet d)\,t2 + 2\,(s \bullet d)\,t + (s \bullet s) - r2 = 0.$$

Since d is a unit vector, we get

$$t = -(s \bullet d) \pm \sqrt{(s \bullet d) - (s \bullet s) + r}$$

Using the above equation we are able to draw the sphere and calculate the ray-sphere intersections.

Reflective Sphere

Using a recursive algorithm that accumulates color values along secondary rays, we are able to generate reflections of rays from a surface. Assume I_B is the color from the object B being reflected to the other object A being reflected on. Where p_r is the coefficient of reflection, which represents how much of color I_B is reflected onto the surface of object A. Then:

If this secondary ray meets a surface at a point with intensity I_B, then $p_r\,I_B$ is added to the pixel color

The color of the pixel is now

$$I = I_A + p_r\,I_B$$

Plane creation

The plane has a constructor that takes five parameters, four vertices and a color value. Since it is a subclass of SceneObject, the plane class has implementations for the functions intersect() and normal(). The surface normal is calculated as:

$$(B - A) \times (D - A)$$

The value of the ray parameter t at the point of intersection is calculated as:

$$t = \frac{(A - p0).n}{\mathrm{d.\,n}}$$

We then use the isInside() function in Plane.cpp to check if the point of intersection is within the quadrilateral specified by the four points.

EXTRA FEATURES

Texturing

Texturing is done by mapping the coordinates of the point of intersection (x, y, z) to image coordinates, and assigning the color of the pixel to that point. Using the equations below we are able to calculate the intersection of the ray with the point we want to texture and the color of the pixel of that point.

1. TEXTURING A PLANE USING AN IMAGE

```
if(ray.xindex == 3)
{
 texcoords = (ray.xpt.x - a1)/(a2-a1);
 texcoordt = (ray.xpt.y - b1)/(b2-b1);
 col = texture.getColorAt(texcoords, texcoordt);
}
```

2. TEXTURING A SPHERE USING AN IMAGE

A sphere is textured with blue and white stripes by loading a BMP image onto the sphere using the method below.

```
//Sphere texture
if (ray.xindex == 1)
{
float texcoords = asin(normalVector.x) / M_PI + 0.5;
float texcoordt = asin(normalVector.y) / M_PI + 0.5;
materialCol = texture1.getColorAt(texcoords, texcoordt);

}
```

3. TEXTURING A SPHERE USING A PATTERN

A sphere is textured with black and white stripes. This is done by checking on which coordinates the ray lands on and splitting the sphere vertically into stripes. The color is determined by which stripe the ray lands on and is given a unique color.

```
//Sphere texture
if (ray.xindex == 2)
{
    float texcoords = asin(normalVector.x) / M_PI + 0.5;
    //float texcoordt = asin(normalVector.y) / M_PI + 0.5;
    int odd_even = texcoords/0.1;
    if (odd_even % 2 == 0)
    {
        materialCol = glm::vec3(1,1,1);
    }
    else
    {
        materialCol = glm::vec3(0,0,0);
    }
}
```

Camera Motion

Using the arrow keys to move (left = Left direction, right = Right direction, Up = Forward, down = backwards), and F1 to move the camera higher, and F2 to move the camera lower. Using these controls we are able to modify the eye position, thus also modifying the position where the rays originate and correctly rendering the scene. This is done by using a special function to take keys as an input and modify the x, y, and z coordinates of the eye.

Issues faced using camera motion:

- The scene has to re-render every time a key is pressed using glutPostRedisplay. This causes the user having to wait every time a key is pressed.
- If a key is pressed multiple times, during the time it renders, it keeps taking inputs and moves accordingly. A delay was introduced by using the usleep function after a key was pressed to prevent multiple key inputs for Linux in c++ however it did not solve the problem.

Cylinder Equations

A cylinder at the origin with axis along the y-axis, radius R and height h is given by equation 1:

$$x^2 + z^2 = R^2$$

A cylinder at (x_c, y_c, z_c) with axis parallel to the y-axis is given by:

$$(x - xc)^2 + (z - zc)^2 = R^2$$

With a normal vector at (x,y,z) normalized to:

$$(\frac{(x - xc)}{R} \quad , \quad 0 \, , \quad \frac{(z - zc)}{R}$$

The intersection equation is given by:

$$t^2\left(d_x^2 + d_z^2\right) + 2t\left\{d_x\left(x_0 - x_c\right) + d_z\left(z_0 - z_c\right)\right\} + \left\{\left(x_0 - x_c\right)^2 + \left(z_0 - z_c\right)^2 - R^2\right\} = 0.$$

Using the intersection equation above, we substitute the values in said equation into the quadratic formula below for the values, a, b, and c. Where x is the equivalent to t. From the result we are able to compute and draw the cylinder.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

References

Lectures 9 & 9a: http://learn.canterbury.ac.nz/course/view.php?id=1439&section=1

Lab 7 & Lab 8: http://learn.canterbury.ac.nz/course/view.php?id=1439&section=2