CAP4730 Ray Tracer Report

Jonathan Bryan

February 2024

1 Introduction

In this project I created a ray tracer which renders simple 3D objects. To start I used GLFW and GLEW to render a rectangular texture to the screen. By editing each pixel of this texture we can create objects and give the illusion of depth. For linear algebra and general math I used the popular C++ library GLM. To create a simple user-interface for editing the scene and saving images or animations I used Dear ImGui. For exporting images I used stb_image_write to create jpeg images and compile those images into videos using ffmpeg. The rest of this report will dive into the specific implementations of ray tracing principles.

2 Scene

For this report I will use a standardized scene to show the different camera, shading, and material techniques implemented in this ray tracer.

2.1 Objects

To create our digital scene we will first need some basic shapes. In this program I have created spheres and triangles. Spheres are defined by a radius and a center point. Triangles are defined by three points.

In the scene we have a plane defined at the origin with the color (200, 200, 200) and material 0. In the code this is simply defined as a triangle which does not check its bounds.

In this scene we have two spheres with the following attributes:

	center	radius	color	material
sphere 1	(-3.0, 2.0, 0.0)	1.5	(75, 255, 0)	material 1
sphere 2	(0.0, 2.0, -2.0)	0.5	(0, 210, 255)	material 2

We also have a tetrahedron which is the color (255, 0, 0), has material 3, and has the following vertices:

vertex	position
v1	(0.0, 3.0, 0.0)
v2	(1.0, 1.0, -1.0)
v3	(-1.0, 1.0, -1.0)
v4	(0.0, 1.0, 1.0)

Another important part of our scene is the lighting. In the demo scene we have two lights which have the following properties:

	position	color
light 1	(-2.0, 4.0, -3.0)	(255, 255, 255)
light 2	(-0.5, 3.5, 0.5)	(255, 255, 255)

2.2 Material

For our scene we can generate different materials that the objects can inherit. Keeping the materials separate from the surface allows us to edit materials and see the results on multiple different geometries. In our demo scene we have 4 defined materials:

	objects	ambient	diffuse	specular	specular power	glazed
material 0	tetrahedron	0.5	0.4	0.8	32	yes
material 1	sphere 1	0.25	0.4	0.6	100	yes
material 2	sphere 2	0.4	0.4	0.25	16	no
material 3	plane	0.5	0.2	0.2	8	no

3 Camera

To generate our image we begin by defining a camera and shooting rays out of it. Casting these rays into our scene and looking for intersections will allow us to differentiate objects. Generic rays can be defined with the following equation:

$$r = \mathbf{p} + t\mathbf{d}$$

Where $\mathbf{p}$ is the origin of the ray and $\mathbf{d}$ is the direction.
Our camera object contains the following:

type	variable	description
vec3	e	position
vec3	u,v,w	basis vectors
int	nx,ny	plane resolutions
float	l,r,b,t	plane bounds
float	d	camera depth

Our camera has two different methods of casting rays into the scene: perspective and orthographic.

3.1 Orthographic Rays

Orthographic rays are cast from the viewing plane directly and are orthogonal to the plane. The origin of the ray $\mathbf{p}$ is defined as a position on our viewing plane which depends on the resolution. It can be calculated using the following equation:

$$\mathbf{e} = \mathbf{e} + u\mathbf{u} + v\mathbf{v}$$

The direction of the ray is simply into the direction the camera is pointed, **-w**.

3.2 Perspective Rays

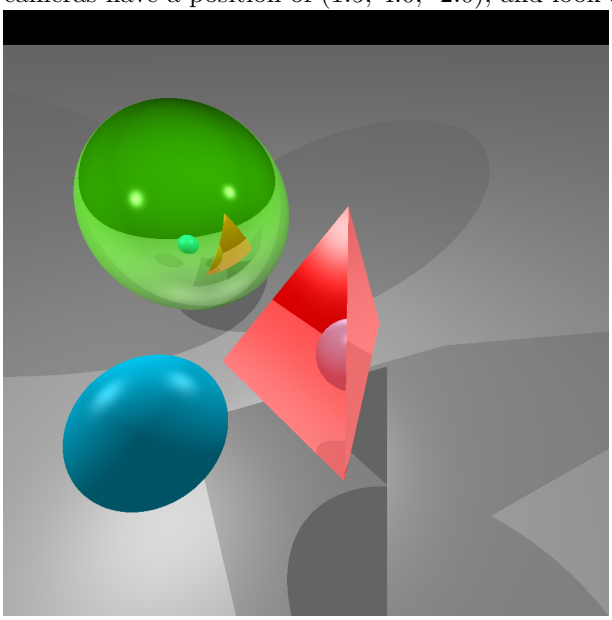
Perspective rays are cast from a single point "behind" the viewing plane. The origin of each ray is **e**, the position of the camera. The direction of the ray is calculated using the following:

$$\mathbf{d} = -d\mathbf{w} + u\mathbf{u} + v\mathbf{v}$$

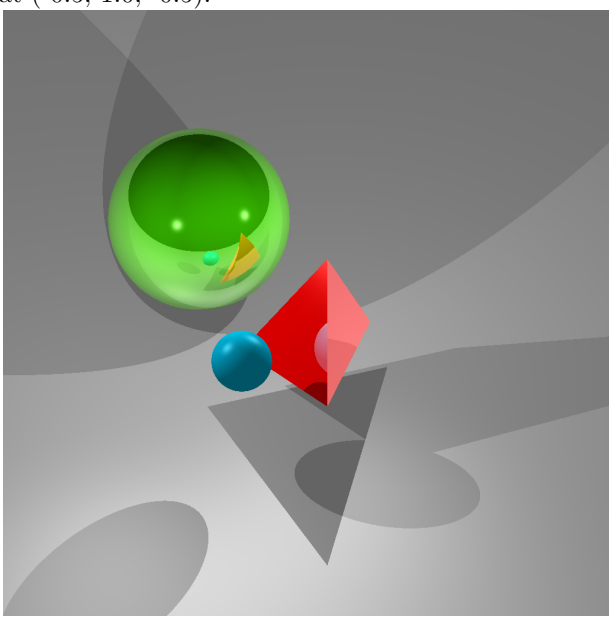
where d is the depth.

3.3 Demo Scene

In our demo scene we can swap between orthographic and perspective by pressing "p" or using the radio buttons in "Camera Settings." The following shows perspective rays on the left and orthographic rays on the right. The scenes are the exact same and have the same camera position/angle. l, r, b, t and d are all set to 5.0 so that the pictures are similar sizes. In the following pictures both cameras have a position of (1.5, 4.0, -2.0), and look at (-0.5, 1.0, -0.5):



Perspective camera, depth 5.0.



Orthographic camera.

4 Shading

Shading is a very important part of adding depth to our objects. The previous images all include multiple lighting techniques so that the objects are visible and have depth. There are three main types of lighting which are added to generate the final color of a surface:

$$L = L_a + L_d + L_s$$

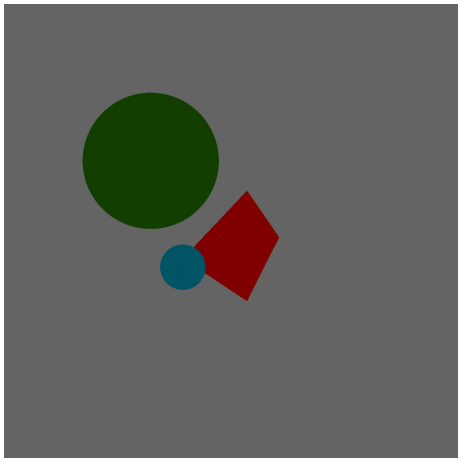
4.1 Ambient

The most basic lighting is ambient lighting, which is a rough estimation of the light that spreads and bounces all over surfaces. This lighting is the reason shadows are not fully dark. The equation

for ambient light is as follows:

$$L_a = k_a \cdot \mathbf{color}_{obj} \cdot \mathbf{color}_{light}$$

Here is the same scene with orthographic projection and only ambient light:



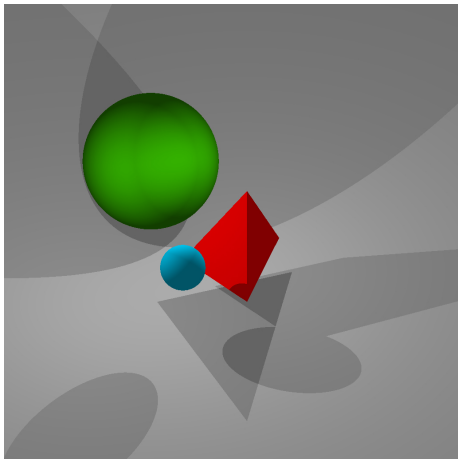
The image looks extremely flat and we can only tell objects apart by color and position.

4.2 Diffuse

Diffuse lighting is responsible for gradients and shadows (4.5) in our scene. For each ray intersection diffuse lighting is calculated across each light in the scene. The equation for diffuse lighting is as follows:

$$L_d = k_d \cdot \mathbf{color}_{light} \cdot \max(0, \mathbf{n} \cdot \mathbf{v}_L)$$

Where $\mathbf{n}$ is the normal of the surface at the point of intersection and $\mathbf{v}_L$ is the direction towards the light source. The following is our demo scene with ambient and diffuse lighting enabled:



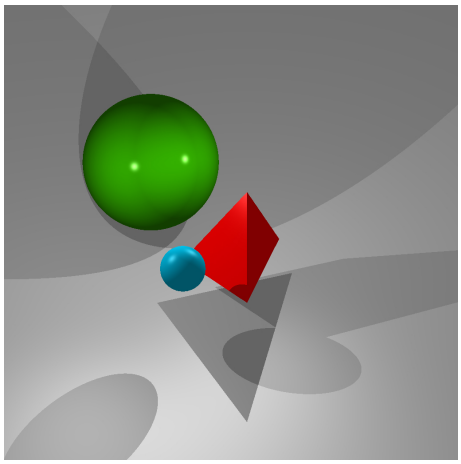
The scene already has a lot more depth due to the diffuse lighting and shadows. Shadows are included in this section because they come along with diffuse lighting in the code.

4.3 Specular

Specular highlights are bright spots that come from reflection of light off a surface. Specular reflections are calculated with the following equation:

$$L_s = k_s \cdot \mathbf{color}_{light} \cdot \max(0, \mathbf{v}_E \cdot \mathbf{v}_R)^n$$

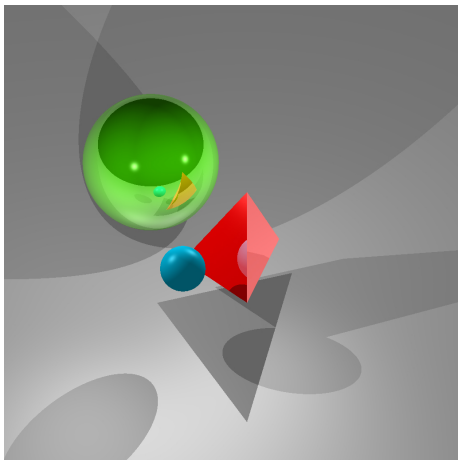
Where k_s is the specular coefficient, $\mathbf{v}_E$ is the direction of the camera, $\mathbf{v}_R$ is the direction of reflection from the light source, and n is the power of the reflection. A higher n leads to a smaller and more concentrated highlight as the light is less scattered.



Our objects are beginning to look glossy and shiny. We can tell which direction the light is coming from due to the specular reflections. We can also tell materials apart based on how concentrated their reflection is.

4.4 Glazed

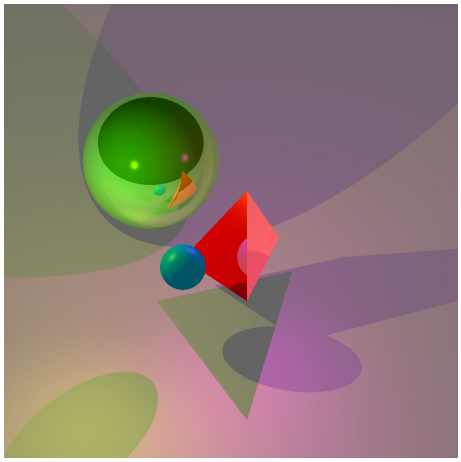
Glazed surfaces work like mirrors in the ray tracer. When we intersect a surface we reflect a new ray off the surface and intersect it with the scene. Once we retrieve the color of that object we add it to our object. This method of creating a mirrored surface is recursive and can allow for an infinite number of bounces. To keep things simple my ray tracer allows for two bounces. In the demo scene the green sphere and the red tetrahedron are both glazed:



The scene is looking quite realistic. We can see objects bounce off one another and really tell the materials apart.

4.5 Shadows

Shadows in this ray tracer are quite simple. Each time we calculate diffuse or specular lighting with a point light we check if $\mathbf{v}_L$ (the direction of the light) intersects with any other objects in the scene. If it were to hit an object, we return black instead of the light color. Since this is calculated per point light the shadows are additive and overlap. We can see this happend when we change the light colors to yellow and pink:



We can tell which light is casting shadows because of the color they leave behind. On the left we have green shadows, left by the absence of pink light. On the right we have blue shadows left by the absence of yellow light. Where they meet we get a teal mix of the two.

5 Videos

To generate demo videos with camera animations we need to store keyframes. Keyframes are a simple data structure that store the time, position, camera settings of a specific frame. We can interpolate between these keyframes to create a smooth transition between positions. The movies can be found within the report directory in the project files.

5.1 Movie 1

In movie 1 we simply move around the scene. I added a few keyframes circling the objects to show how the movement works. It uses a basic orthographic camera.

5.2 Movie 2

In movie 2 we move around the scene and change the depth of the camera. The smooth zooming in and out allows for interesting perspected and we can get a closer look at reflections.

5.3 Movie 3

In this final movie I made the small blue sphere move and the lights flash different colors. This is done by changing the position of the sphere and the color of the lights depending on time.