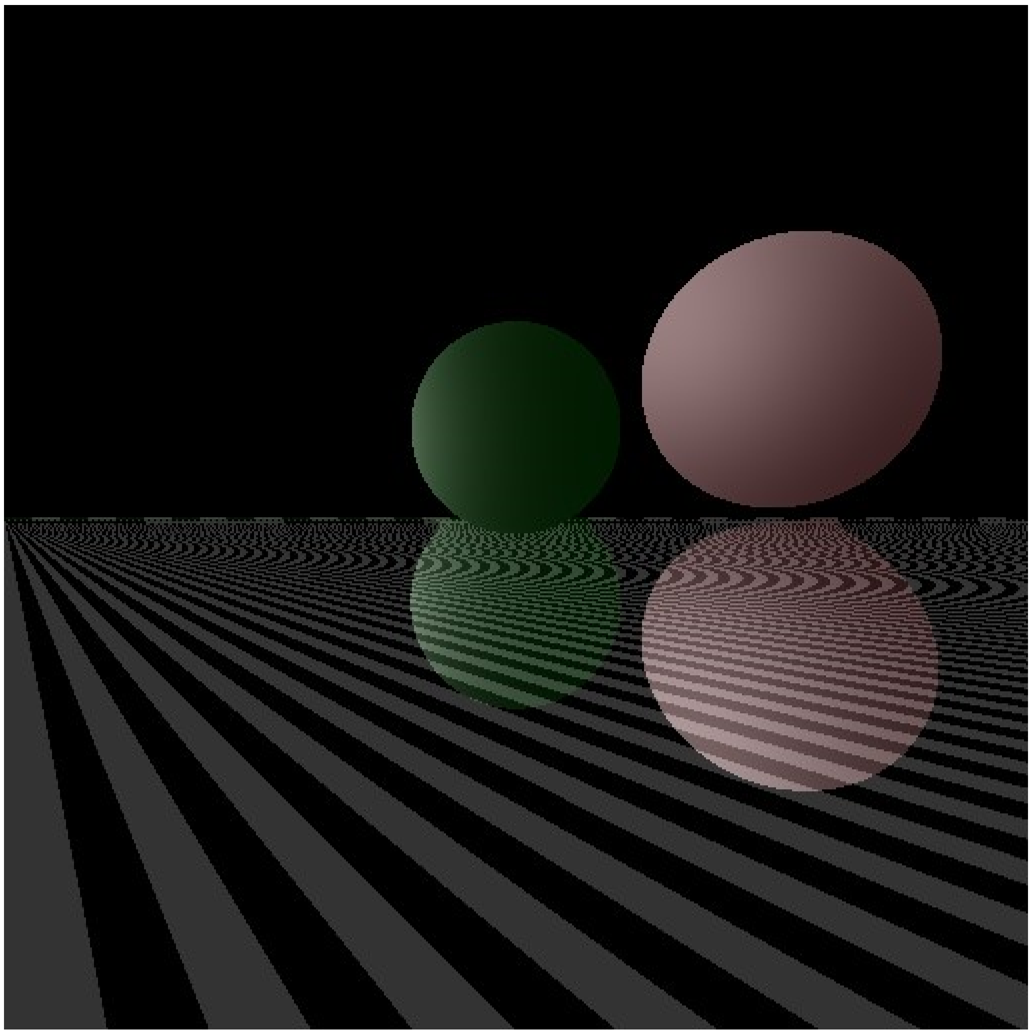
Computer Graphics CAP 5705- Assignment 1

First-Hit Ray Tracer

Authors - Ganti Balasai Srikanth, Nikhil Sai Siddam Shetty



Introduction

The rendered world consists of two spheres and an infinite plane which is also a glazed surface where the spheres are reflected upon. This raytracing solves for ray-object intersection problem, adds ambient, diffuse and specular shading, introduces a glazed surface and also gives users the option to switch between orthographic and perspective geometry. At the end we also create a movie with all the rendered images together.

Ray Object Intersection

We send rays into the scene and find out if these rays hit the objects in the scene, the implementation of this changes with the change in object type but at its core we put the mathematical equation for the ray into object equation and find out if there is a hit or miss. In this implementation we have two spheres in the scene with the below dimensions

Objects	Center	Radius	Colour
Sphere 1	(-0.5, 1, 2)	0.5	red
Sphere 2	(-0.8, 0, 5)	1	green

Ray Equation : $P(t) = A + tb$

If the ray hits the sphere it should satisfy the below equation

$$(P(t) - C) . (P(t) - C) = r^2$$

Upon further simplification we see that this is a quadratic equation and we need to solve for t , the scenarios we encounter are $t >= 0 \, and \, t < 0$, if $t >= 0$ we get a hit and if it is less than 0, it is a miss.

This is implemented in code using the function hit_sphere() where we solve for t and based on the result we give properties to that intersection point.

Shading

The shading model used is based on the Phong reflection model, which is a popular method for approximating the way surfaces reflect light.

The light source is defined at the position (-0.6, 2.5, 4).

The shading for each intersected point is determined by three components:

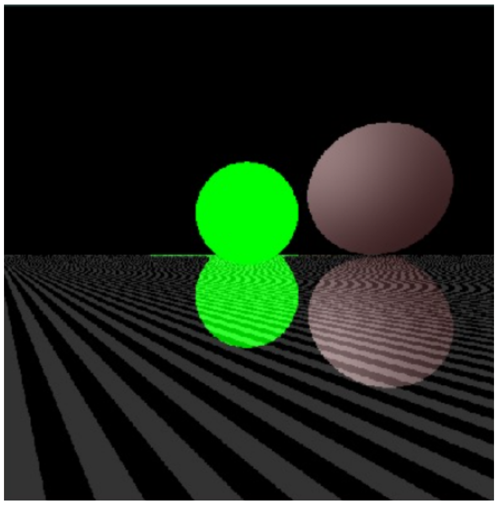
1. Ambient lighting: A constant color value representing light that is present everywhere in the scene.

2. Diffuse lighting: Dependent on the angle between the surface's normal vector and the light direction.

3. Specular lighting: Represents shiny highlights on the surface, contingent on the angle between the viewer direction and the reflection of the light source.

We have implemented a ray_color() function that takes a ray and a depth value as input. It checks if the ray intersects with any of the predefined objects (in this case, two spheres).
If there's an intersection, it calculates the shading based on lighting conditions and returns the resulting color. If the maximum recursive depth is reached or if there's no intersection with any objects, it returns a black color.

The function uses the Phong reflection model to calculate the color at the point of intersection. This model combines ambient, diffuse, and specular reflections to generate a realistic color


(Green sphere is not shaded and red is shaded)

Glazed Surface

Within the digital rendering framework, we've engineered a surface with "glazed" characteristics, adept at both showcasing its intrinsic design and mirroring the surrounding entities. When a ray intersects this surface, determined by the equation

$$IntersectPoint = r.origin() + r.direction() \times t_{plane}$$

Two core visual elements are assessed: the plane's inherent design and the reflective depiction of its environment. The reflection is computed using the principle

$$r_d = i_d - (2 \times dot(i_d, n) \times n)$$

Where,
r_d = reflected ray
i_d = incident ray
n = normal

To ensure accuracy and prevent any self-interference, a minimal bias is incorporated. The value of depth thus determines how many "bounces" a ray can make in the scene and, indirectly, the quality and accuracy of reflections and refractions in the final rendered image.

The final visual output on this surface results from a balanced blend of its innate design and the mirrored scene. This methodology facilitates the accurate portrayal of a glazed surface, capturing both its design and reflective essence.

Movie

Producing a sequence of images that, when viewed in order, will show the effect of a camera orbiting around the point (-1, 0, 2) in the xz-plane, maintaining a constant y-coordinate. This orbiting motion is achieved by updating the x and z coordinates of the camera for each frame, based on a calculated angle, and then rendering the scene from the updated camera position.

Known Bug

When the movie is generated we get a clear picture of the camera rotating around the objects but the reflections on the plane appear to be stationary.

Perspective Switch

Visualization in computer graphics is heavily dictated by the type of perspective chosen. In the Raytracing project, there are two distinct camera geometries being utilized: orthographic and perspective. The user's input dictates which of these two modes is active.

In the case of an orthographic camera, rays that project onto the scene are parallel. This results in objects being depicted at a consistent scale regardless of their depth in the scene, leading to a depiction that lacks depth perception. Mathematically, for a 2D representation, the direction of the ray is represented as in terms of the ray equation

$r(t) = o + td$

Where *o* is the origin and *d* is the direction of the ray.

On the other hand, in a perspective camera, rays emanate from a single point (the camera or the eye) and diverge as they travel forward. This results in objects appearing smaller as they get farther away, creating an illusion of depth. The direction of the ray in the perspective view is

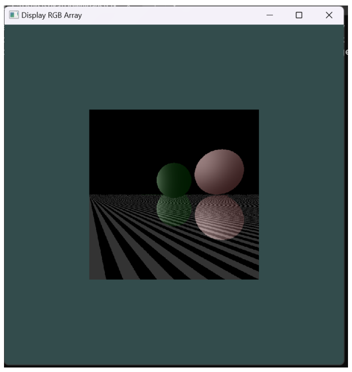
determined as the difference between the position on the image plane (position_on_image_plane) and the camera center.
image_plane_center, is ascertained using the equation

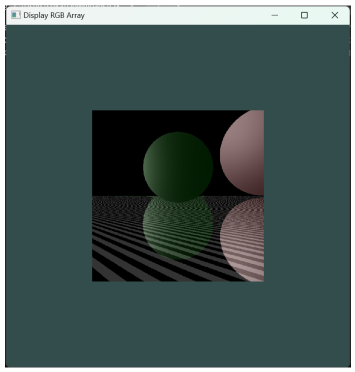
image_plane_center=c+distToImage×w,
where
c denotes the camera's position, distToImage is the distance between the camera and the image plane, and w represents the normalized direction in which the camera is aimed.

Demo Scene:

In the ray tracing framework, the user can seamlessly toggle between orthographic and perspective views. By default, the system employs an orthographic view, but upon user input, specifically pressing the 'p' key, it switches to a perspective view. This choice alters the ray direction calculation within the rendering loop: in orthographic mode, rays extend directly from the camera through pixel centers, while in perspective mode, rays pass from the camera center through distinct positions on the image plane, introducing depth and a more realistic portrayal of the scene. This design facilitates flexibility in visualizing the rendered scene through different projection techniques.

For perspective camera, **distToImage** = *2*.





Orthographic camera

Perspective camera

Bonus Feature

Pattern on a plane

A pattern is achieved on the plane through a simple checkerboard logic based on the plane's intersection point coordinates. When the ray intersects the groundPlane, the intersection point, denoted as intersectPoint, is calculated using the formula

$$IntersectPoint = r.origin() + r.direction() \times t_{plane}$$

This point's y and z coordinates determine the color of the pattern. Specifically, the sum of the y and z coordinates of intersectPoint is used to decide the pattern's color: if the sum is even (determined by the modulus operation % 2 == 0), the color is set to white (color(1, 1, 1)); otherwise, it's set to black (color(0, 0, 0)).

This approach creates a pattern with alternating black and white squares based on the spatial position of the intersection point on the plane. The final output color for the plane is a mix of this inherent pattern color and the color obtained from the reflection, controlled by the reflectivity parameter.

Sources

1. https://raytracing.github.io/
2. https://subscription.packtpub.com/book/game-development/9781787123663/1/ch01lvl1sec18/reflection
3. https://learnopengl.com/Lighting/Basic-Lighting
4. Raytracing Basics - https://www.youtube.com/watch?v=gfW1Fhd9u9Q&list=PLIrATfBNZ98edc5GshdBtREv5asFW3yXl
5. Fundamentals Of Computer Graphics - Peter Shirley, Steve Marschner