CAP 5705 - Computer Graphic
Khanh Dong
September 22nd, 2025

First Hit Ray Tracer Report

1 Overview

The program is built upon the previous assignment, now with perspective camera, shading, and reflection. The scene contains two spheres from the previous assignment, a tetrahedron, and a plane. The tetrahedron is made up of 4 triangle surfaces, stored in a vector of Triangle structs, with the ray intersection determined by a function using the Möller–Trumbore algorithm. The implementation is based on this tutorial. The plane is defined by a Plane struct, and the intersection is calculated by the dot product of the ray's direction and the plane's normal vector. The spheres are made to move up and down by updating their y-axis locations each loop.

1.1 Camera

The program has already implemented orthographic perspective as well as camera movement in the first assignment. The perspective camera in the code is implemented by positioning the camera at a fixed viewpoint in 3D space and generating rays that originate from this point, but in different directions depending on the pixel location on the image plane. This is achieved by computing normalized direction vectors which correspond to each pixel's position on the camera's virtual view plane using the camera's orthonormal basis vectors (right, up, and forward) and a field of view angle, simulating how objects appear smaller as they get farther from the camera, thus creating the natural depth perception of perspective projection. The implementation was based on this tutorial.

The different perspective switch uses a boolean flag that is toggled by the user's input (pressing "p"). This conditional logic was inspired by the sample report in the assignment instruction.

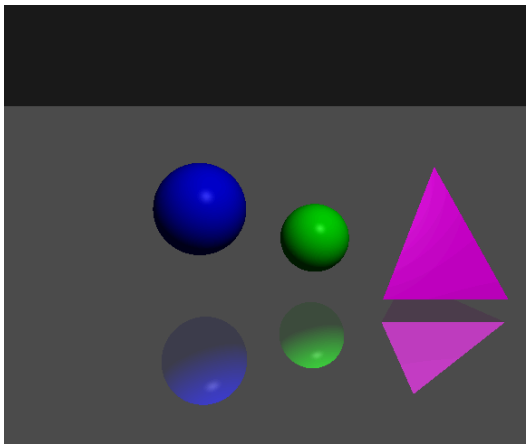

Figure 1. Perspective Camera

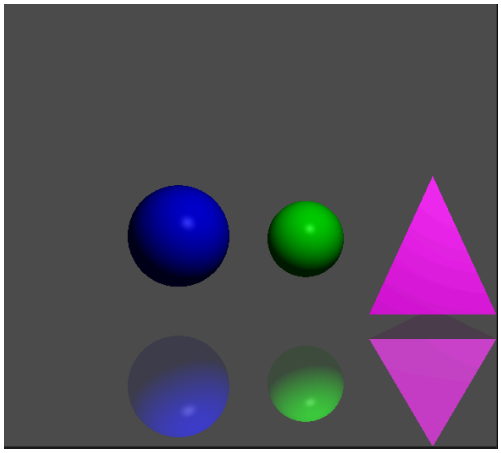

Figure 2. Orthographic Camera

2 Shading

The shading in the code is implemented using a Phong lighting model, composed of ambient, diffuse, and specular components. The shade function first calculates ambient light as a weak constant color scaled by the object's base color. It then determines if the point is in shadow by casting a shadow ray to the light source, checking for intersections with other scene objects; if blocked, only ambient light is returned. If not in shadow, diffuse lighting is calculated as the object color multiplied by the light color and the dot product of the surface normal and light direction, representing how much light hits the surface directly. Specular highlights are computed by reflecting the light about the surface normal and measuring the angle to the viewer direction, raised to a shininess exponent to model glossy reflections. The final color is the sum of ambient, diffuse, and specular components, clamped to a maximum intensity 1 per channel to prevent oversaturation. The light in the ray tracer is implemented as a simple point light source represented by a struct containing its position in 3D space and its color/intensity vector (a white point light located at coordinates (2,5,5)). The implementation was based on this tutorial as well as this guide and this video.

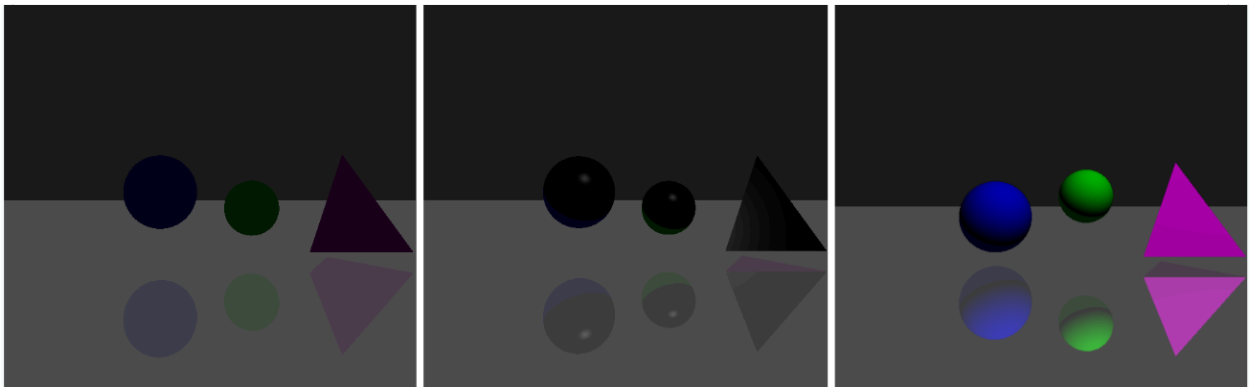

Figure 3. Ambience, Specular, and Diffuse only

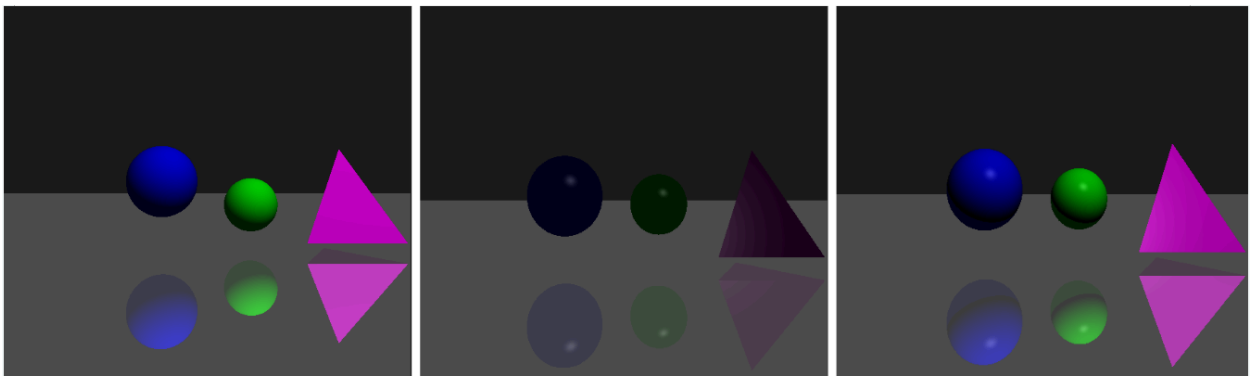

Figure 4. Ambience + Diffuse, Ambience + Specular, Specular + Diffuse

2.1 Glazed Surface

The trace function implements recursive ray tracing with a depth limit to avoid infinite recursion. It first checks if the recursion depth exceeds 2, returning a background color if so. It then finds the closest intersection of the input ray with surfaces in the scene: spheres, triangles, and a plane. For each shape, the intersection test calculates the parameter t along the ray; if it is smaller than current closest, it updates the closest hit record with intersection details including position, normal, and color. For the plane, if intersected closest, the function recursively traces a reflection ray computed by reflecting the incident ray about the surface normal. The final color for the plane combines 30% of its base color and 70% of the reflected color. If the ray hits any other object, the function calls the shade function to compute illumination at the hit point. If no intersection occurs, a constant background color is returned. This function captures reflection effects through recursion and manages complexity and performance with depth limiting, implementing a core ray tracing rendering loop. The camera position has been raised due to the orthographic perspective showing no glazed surface, which is the result of the original position being parallel to the surface. The implementation was based on this tutorial.

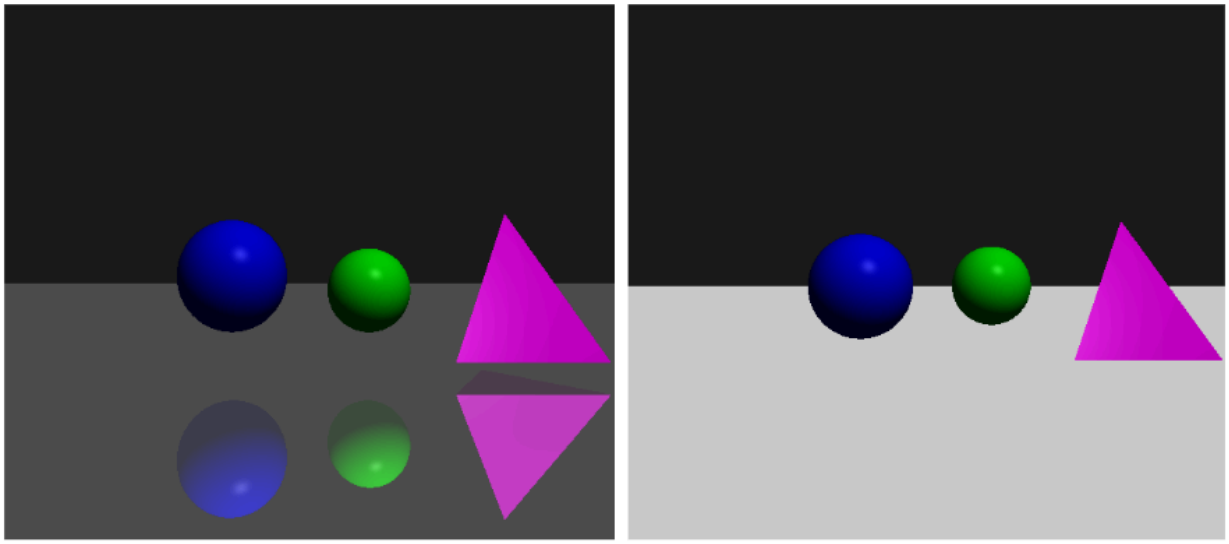

Figure 5. Glazed and No Glazed Surfaces

3 Movie and Image Renders

The render image file saving and movie assembly in the code is implemented by capturing each rendered frame as an image and writing it to disk as a PNG file using "stb_image_write". During each frame of the rendering loop, after the color buffer is fully computed for all pixels, the pixel data is passed to the image writing function which outputs a uniquely named file (e.g., frame_0001.png, frame_0002.png, etc.) to a frames directory. These frames serve as individual snapshots of the rendered scene over time. After rendering completes, FFmpeg is used to combine these sequences of image files into a video file by specifying input pattern, frame rate, and output video format. The implementation was based on the sample report in the instruction.