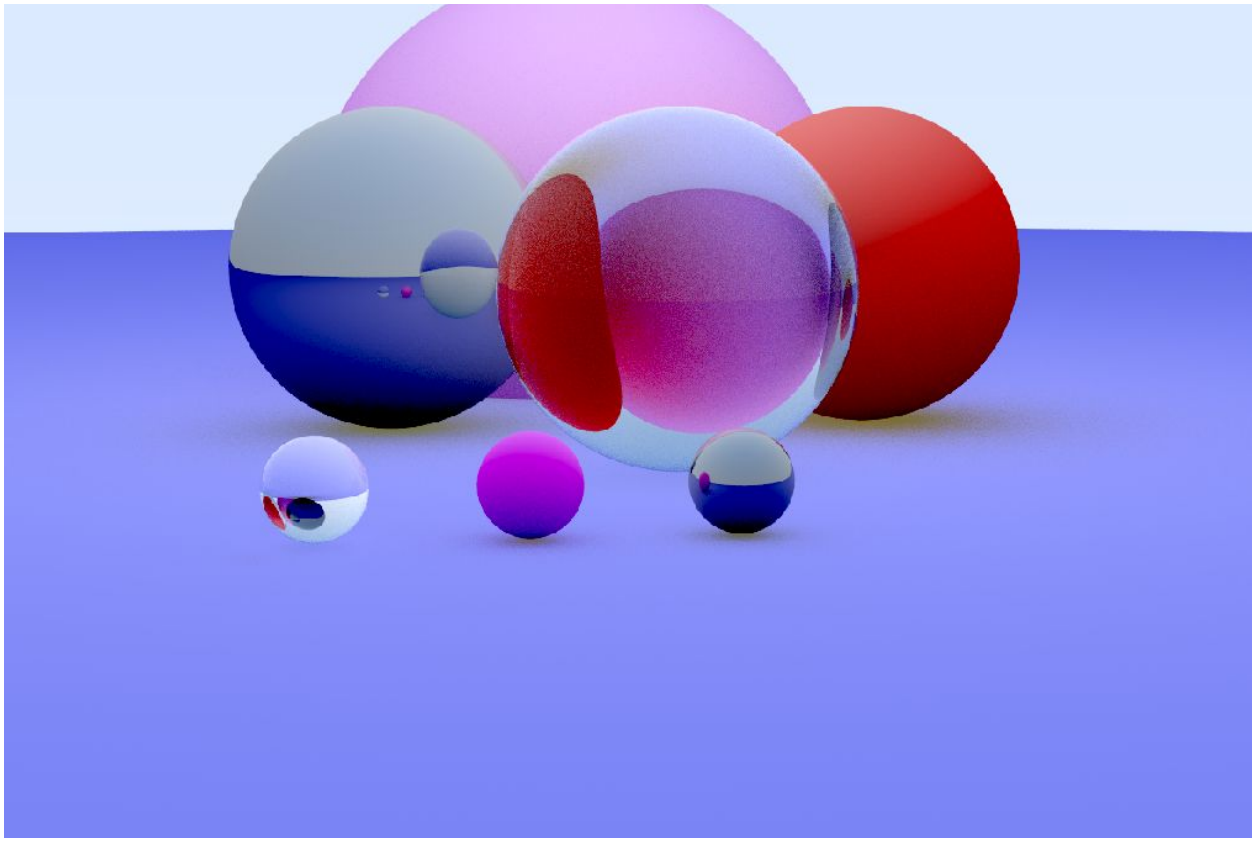
Simple Raytracer

Ben Jollymore, A00400128

Introduction

This is a simple ray tracing program implemented in C++, compiled using G++ in the MinGW environment for Windows. Executable must be run from the command line and output piped into a file. It does not rely on any external libraries. This program was originally built using Eigen and OpenGP libraries, but due to dependency loading and linker issues plaguing development, all externals were dropped late stage and the program was refactored to be entirely self contained. For every pixel, a ray is cast from the camera point and is tracked as it moves through the scene, interacting with the different spheres. Each ray moves by recursively either reflecting, scattering or refracting through the spheres in the scene, with the albedo from each sphere being combined to yield a final color. This colour is combined with the light from a single point, intensity determined by the angle in which the light intersects the sphere. There are a total of up to 50 bounces per ray through the scene. Sample output of the program is below. Note grainy-ness decreases with number of anti aliasing passes. Between 10 and 50 passes were taken for renders in this document, in the interest of time.

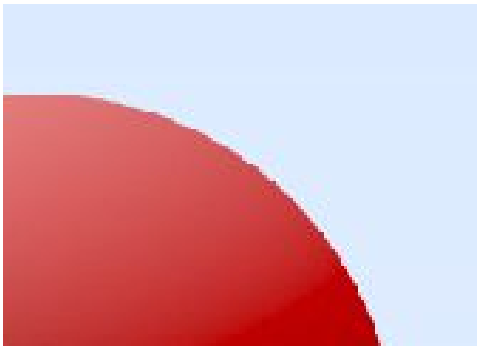


Extra Features

Some additional features aside from the base functionality were added to this program, as discussed below:

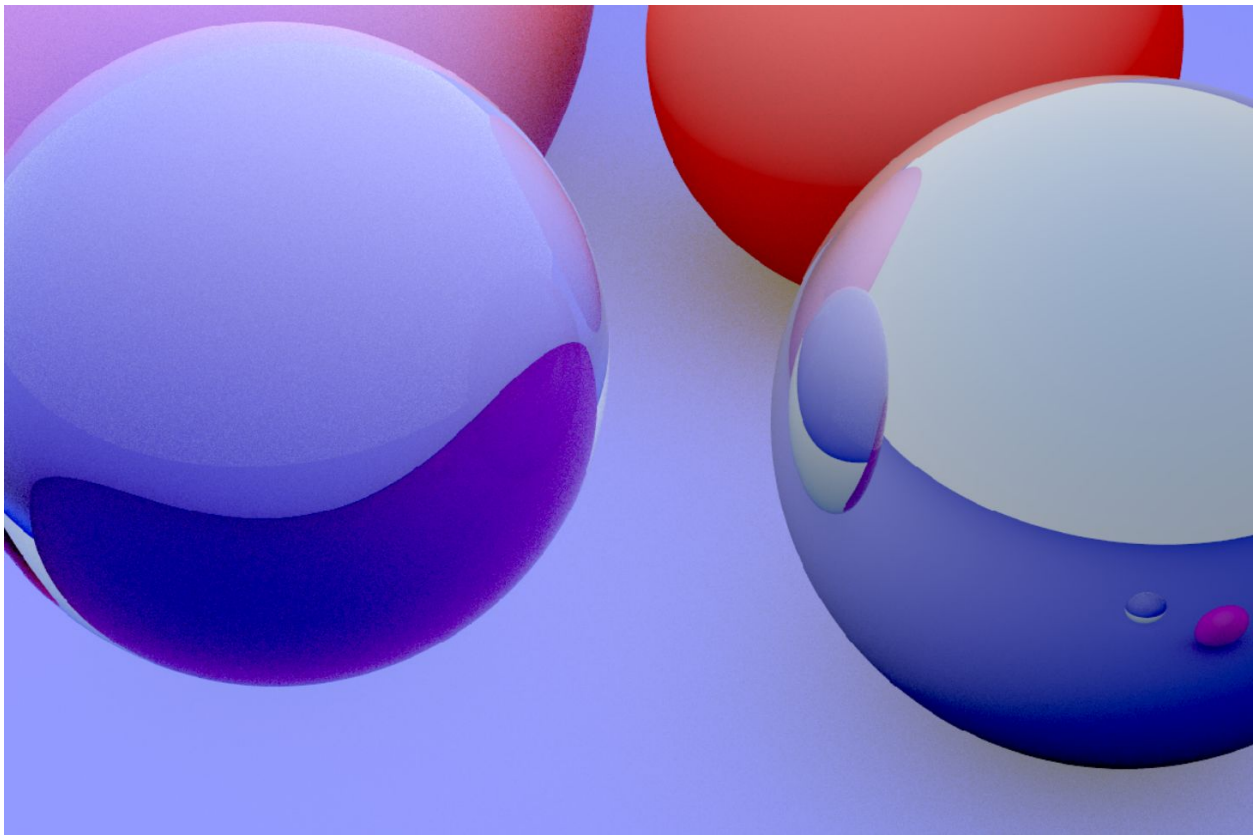
Anti Aliasing

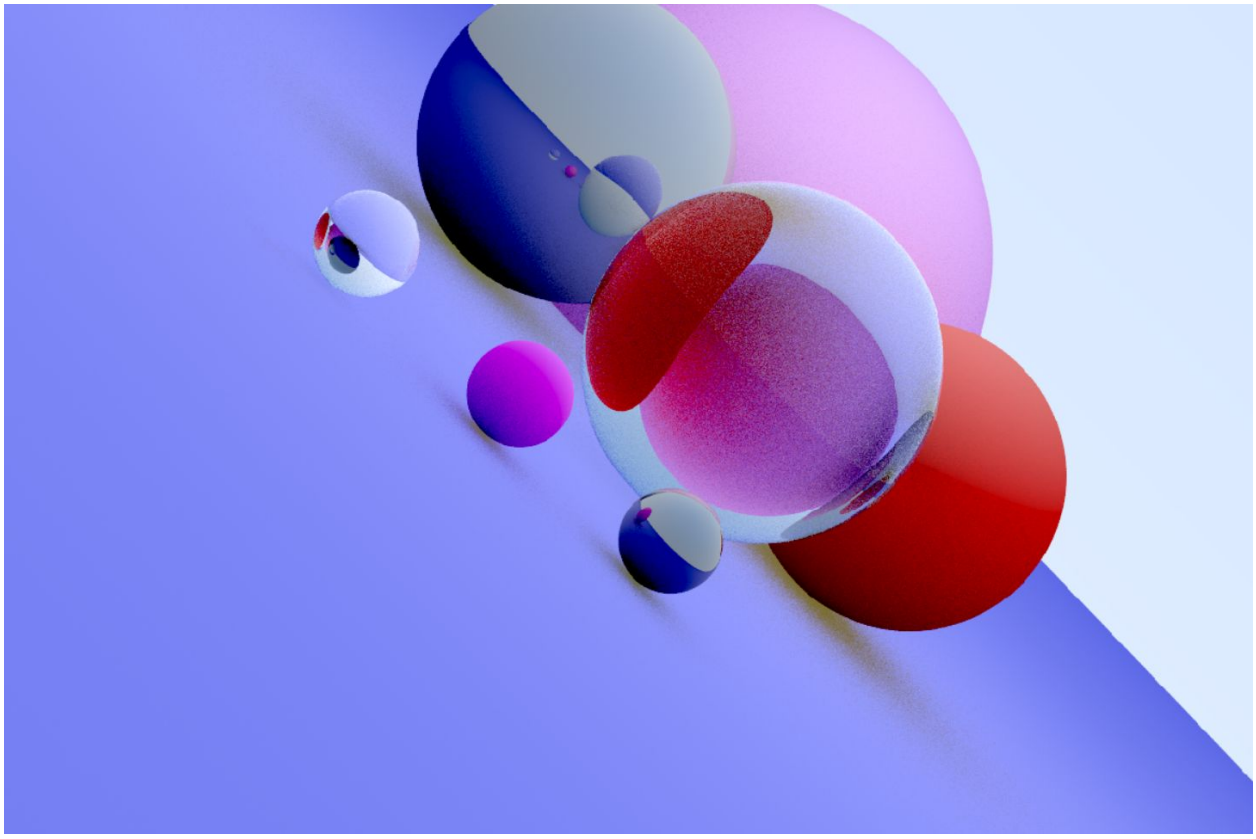
Simple anti-aliasing was implemented by shooting not one ray to each pixel, but instead are sampled by 10 rays (this being an arbitrary value), each offset from pixel center by some random value between 0 and 1. The total colour is then averaged by the number of samples taken. This feature is not perfect, but it is better than without any anti aliasing.



Camera Location and View Direction

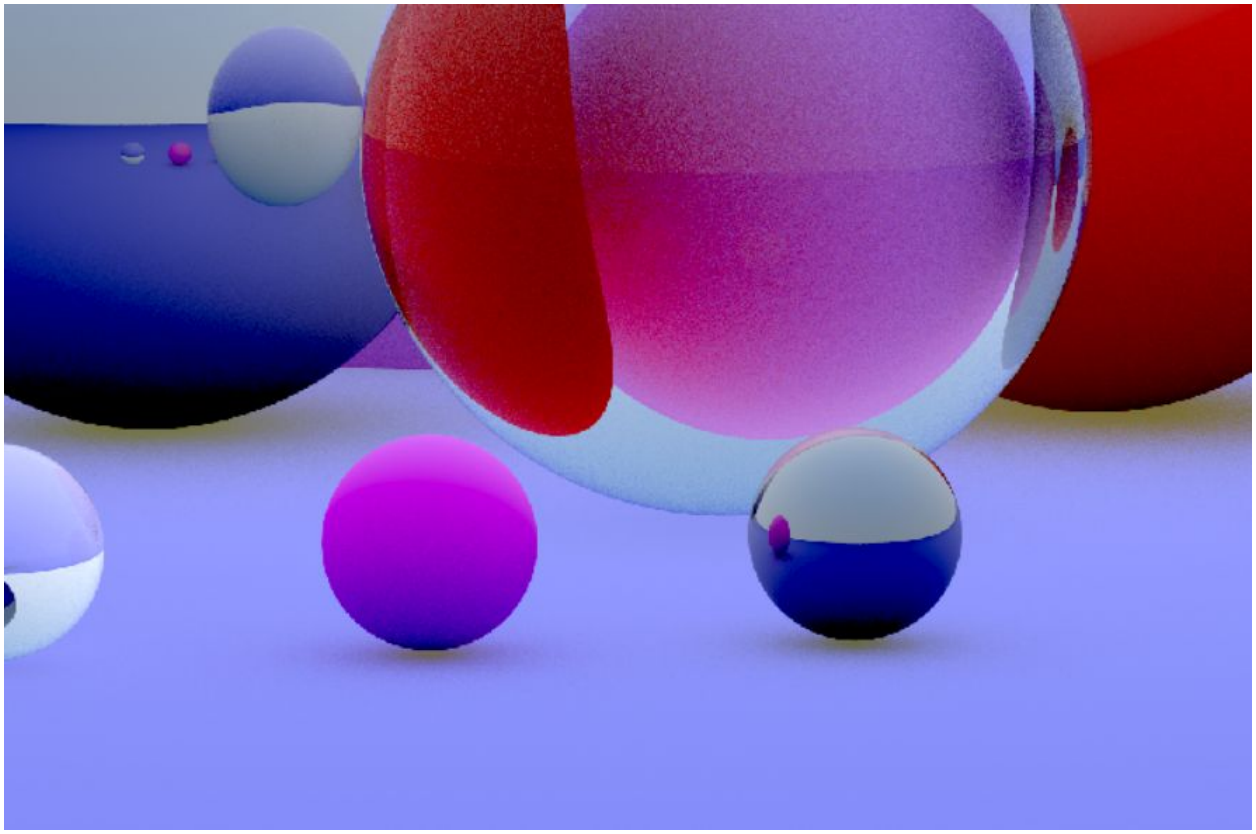
Both the location of the camera and the direction in which the camera is pointing are entirely arbitrary and can easily be changed. It is implemented by re-defining UV space based on the direction of the vector between Camera origin and view target, as well as a vector that defines the vertical axis.





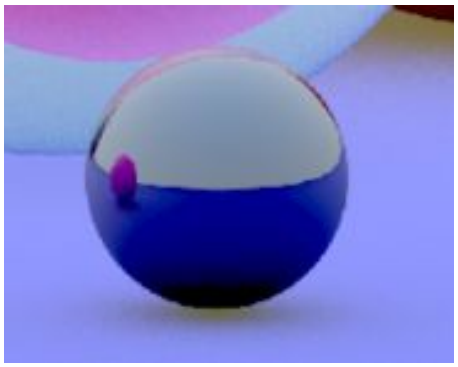
Camera Field of View

The camera field of view is also user configurable. The field of view is defined in the vertical axis, as the angle in degrees between the top of the image plane and the Camera, as defined by an imaginary plane inline with the center of the camera. The angle is converted to radians and is then used to calculate the new height and width of the image plane.



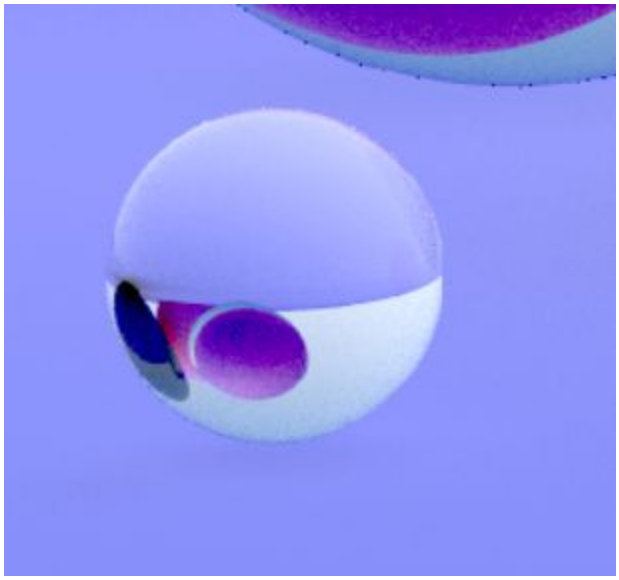
Reflective “Metal” Spheres

Some spheres have the property of being entirely reflective, in a mirror like manner. A ray that intersects will rebound at the same angle it intersected, mirrored across the surface normal of the sphere at that point. If the angle at it rebounds is less than 90 degrees, no light will be absorbed and a slight grey will be added to the ray’s return colour, to simulate a “metal” material.



Refractive “Glass” Spheres

Other spheres have translucent properties, and rays have the possibility of passing through the sphere and exiting the other side, at some angle dictated by the difference in refractive indices between the sphere and the empty space in the scene. All spheres in this scene have a refractive index of 1.5*the index of empty space, to simulate “glass” like properties. Each ray that intersects the sphere has a chance of either reflecting off of or refracting through the sphere, determined by the angle in which the ray intersects the sphere (this is calculated using Schlick Approximation). Any ray that is presently in the sphere will always reflect inwards about the negation of the norm of the sphere, a phenomenon called perfect internal reflection.



Reference Material

https://en.wikipedia.org/wiki/Schlick%27s_approximation
http://www.realtimerendering.com/raytracing/Ray%20Tracing%20in%20a%20Weekend.pdf
http://www.realtimerendering.com/raytracing/Ray%20Tracing_%20The%20Next%20Week.pdf
https://en.wikipedia.org/wiki/Lambertian_reflectance
https://www.youtube.com/watch?v=ARn_yhgk7aE
https://www.youtube.com/watch?v=Ahp6LDQnK4Y
https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/ray-tracing-practical-example
https://medium.com/farouk-ounanes-home-on-the-internet/ray-tracer-in-c-from-scratch-e013269884b6