322COM – Assignment 1

Table of Contents

[Goal 1](#_Toc529468142)

[Method 1](#_Toc529468143)

[Reflection 2](#_Toc529468144)

**No table of figures entries found.**

# Goal

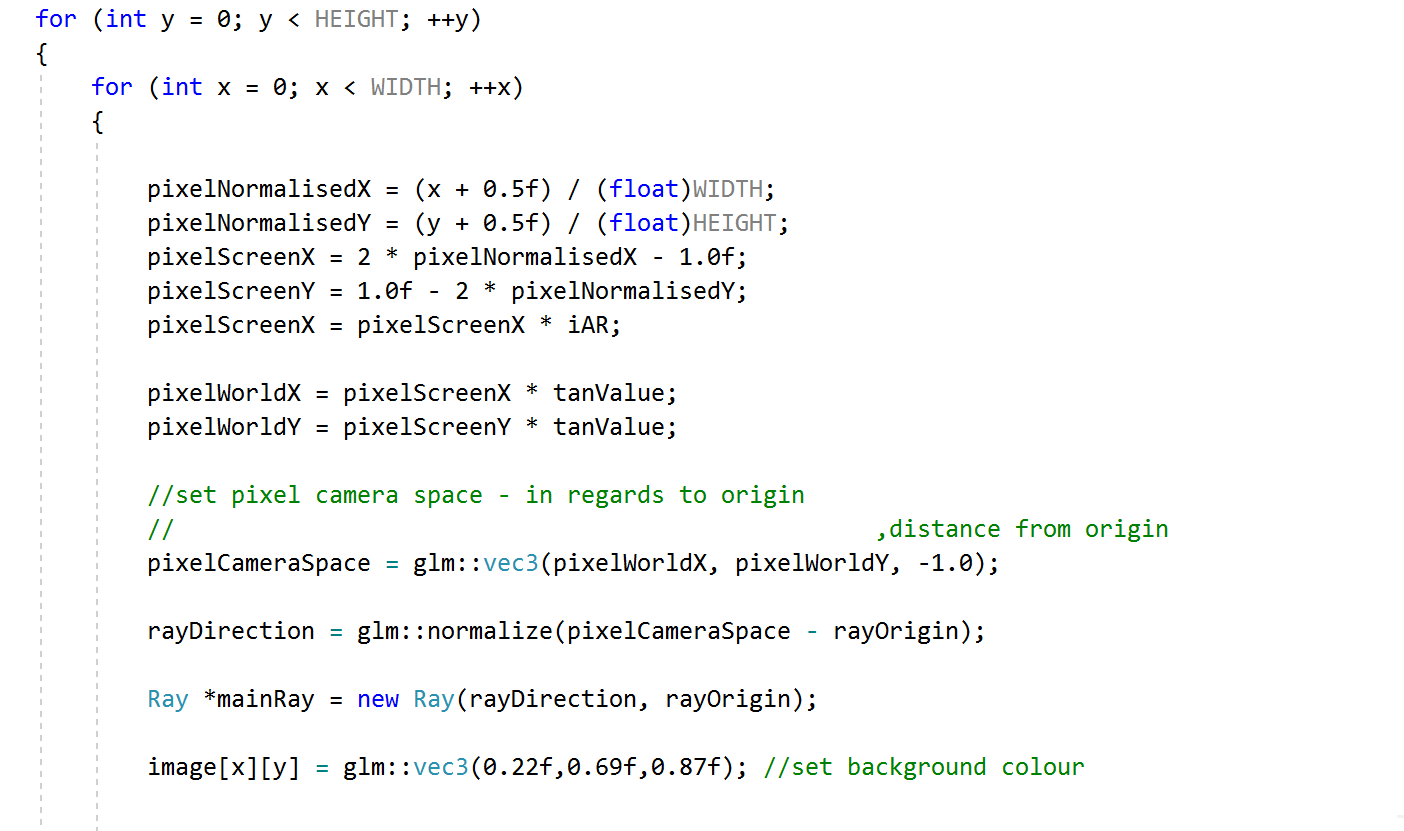
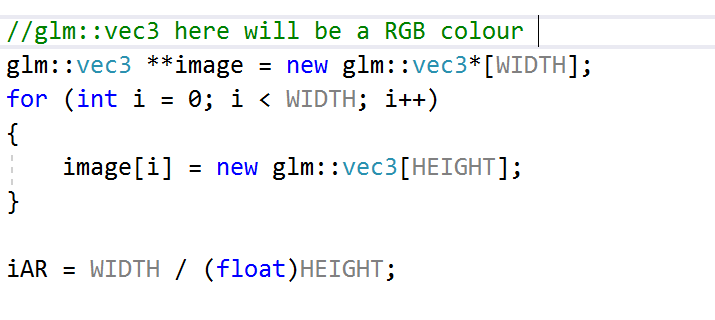
The goal of this module was to produce a real-time Ray tracer using computer graphics algorithms and techniques based on C++.

This raytracer will need to be built from scratch and include the ability to be able to handle interaction and implement global illumination.

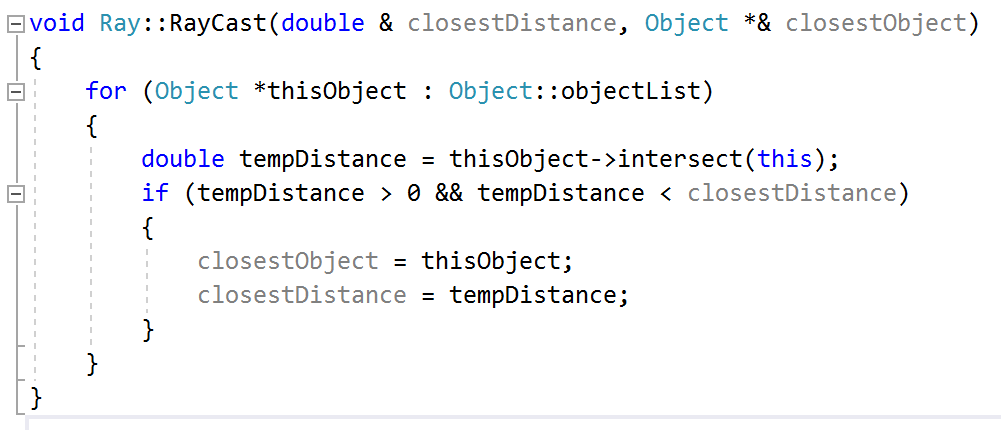
# Method

The core output for this program was to produce an image of a scene containing objects with lighting and shading. To produce this image, I had to create rays that pass from the camera through every pixel and return a colour for that pixel on the image.

To do this I created a multidimensional pointer array, (as opposed to a regular dimension array that would stack overflow down the line,) that used vector 3s to contain colour.



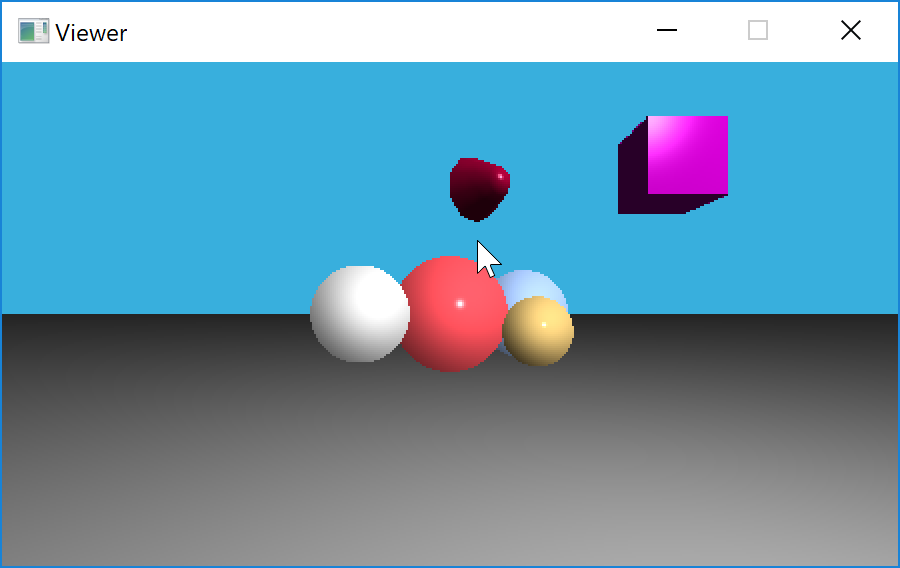
There are three types of primitives within my scene: there are spheres, planes, and triangles. Each of which are required to create a class that would inherit the method of getting the intersection from a base object class and overwrite it per primitive, as they contain different calculations.



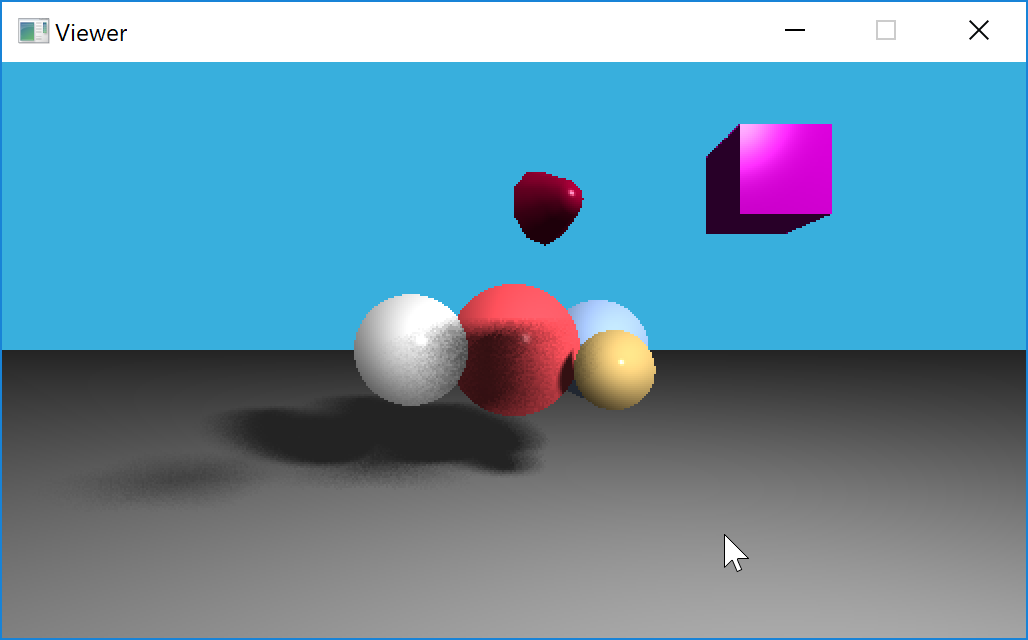
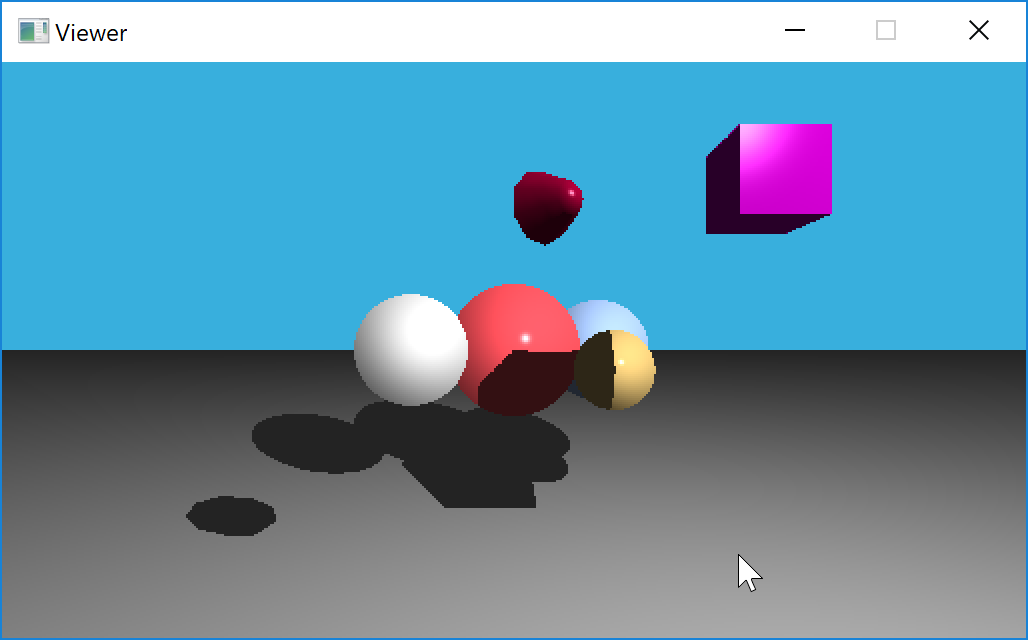
The triangle primitive would be further used in the loading of a model from an object importer to allow for more complex shapes within the scene.



Each of the intersection method calls would be compared to each other per pixel to return the closest object. Which would then be used to get the colour of that pixel by sending in information about the intersection point and the object to calculate Phong lighting.

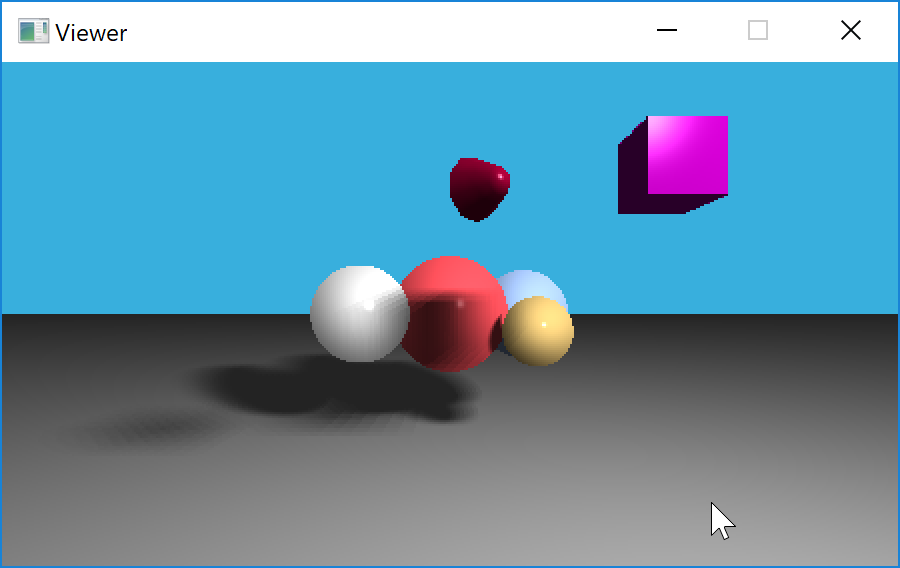


Each of my objects has material has a diffuse colour a specular colour and a shininess value. These values are used to calculate ambient diffuse and specular lighting using Phong shading. The specular and diffuse lighting required a light source with a position and intensity to be contained within the scene.



Further using this light source, I created hard and soft shadows by ray-casting from the pixel to the light source to determine whether it has intersected with another object along the way. The soft shadows required the use of an area light as they would need to have multiple locations in the light that the pixel could cast to based on the number of samples the shadow would use.

Any soft Shadows that would generated by the use of a perfect grid aligned set of rays contained banding. To fix this I added a jitter variable to add a small amount of randomness to the samples. At low samples this would look noisy but at high samples it would have a smoother look than the banding.



Having a multidimensional array of colours would be useless without a way to view it so using a simple method of converting the array to a PPM format and outputting a file I could render a viewable image file. This method is not convenient when variables within the scene can be changed so I strived to render the scene to your screen via the use of a window. To create a window, I required to use of the SDL2 library which allowed me to convert the colour array into a surface which could be rendered on the window.

So, to make things easier I added user interaction to my program via the use of the SDL2 library which would catch the key down event and re-render the scene at depending on the key you press.

# Reflection

So originally, I had set up my Phong lighting in a way that all of my primitive object types would calculate the intersections and colour separately and then compared to each other without any sort of inheritance. Most of my code was based on static functions. After taking some advice from a colleague I decided to refactor most of my object code so that it had a better structure using inheritance and overwriting. So that it could be more readable and have a better structural flow.

I did encounter a bug that still to this day I cannot figure out the solution to where do planes normal is required to be inverted. The end result is still what I want but I do not understand why it requires inversion.

Once I figured out the trick to it I did enjoy implementing shadows into the application and seeing the difference in processing time between hard and soft. It amazes me how much in increase of samples will increase the render time.

If given more time I would quite like to implement multithreading and a ray tracing optimisation algorithm into the application so as to reduce the render time.

I was very happy with the end result of only increasing the memory by under 10 megabytes every time I re-rendered. Earlier in development I did encounter a situation where the memory would increase by hundreds of megabytes every re-render, I solved this by clearing a list of objects every re-render.

I've added a good range of re-renders based on user inputs that I hope will display all of the core features of my application. I enjoyed the fact it was very simple to implement such a wide variety of different instances of the render.

In the future I would not only like to improve the rate in which the scene renders I would also like to include a user-input based way for the camera to move in the scene using matrix transformations. After that I'd like to implement the ability to add multiple lights with in my scene and have them correctly render.