Ray Tracing

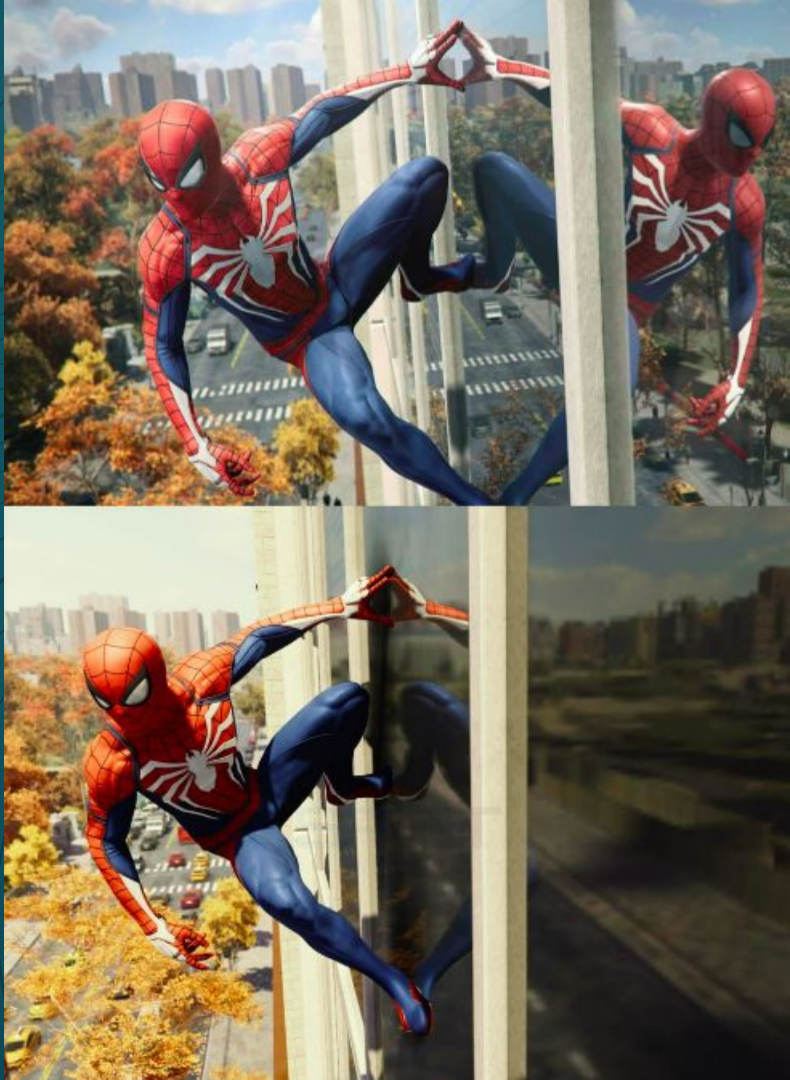
B.VIGNESH
CED18I007

About the concept…

Being one of the most famous gaming buzzwords of the current time, ray tracing is a rendering technique which simulates the presence of “light” in any scene, thereby producing objects close to real life scenarios. The main idea behind ray tracing is to implement the concept of light falling on an object and hitting our eyes, but in the reverse way. It tries to capture all the possible light rays that might land on the observer’s eye, by retracing the path.It’s applications include many, such as gaming, architecture modelling, animation, etc.

Effects of ray tracing on a scene

As shown to the right, the above image represents a case where ray tracing has been implemented, while the below image is a case without ray tracing. It can be clearly seen that ray tracing simulates the scene closer to a real life scenario.



Project breakdown

This project was broken into the following steps :

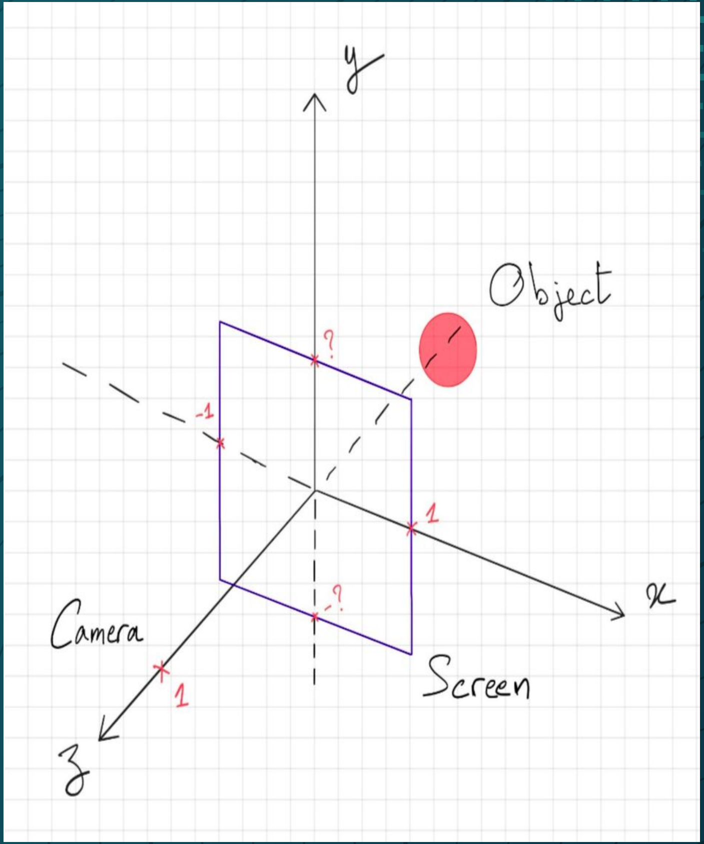
- Defining the scenery.
- Tracing all the rays from the eye to the light source.
- Calculating the colour that our eye captures for each light ray that hit us.

The big picture here is to attempt to find all the possible intersections of the light rays that our eye could capture, and evaluate the colour that each of the light ray provides due to reflection.

Defining the scenery

The eyes' capability to capture light rays is based on its field of view. We define the field of view here with a screen that stands in between the camera(us) and the scenery. This provides a compact and bounded field of view to the observer.

Furthermore, we define the scenery with our primary objects being spheres. The light source here is a point object.



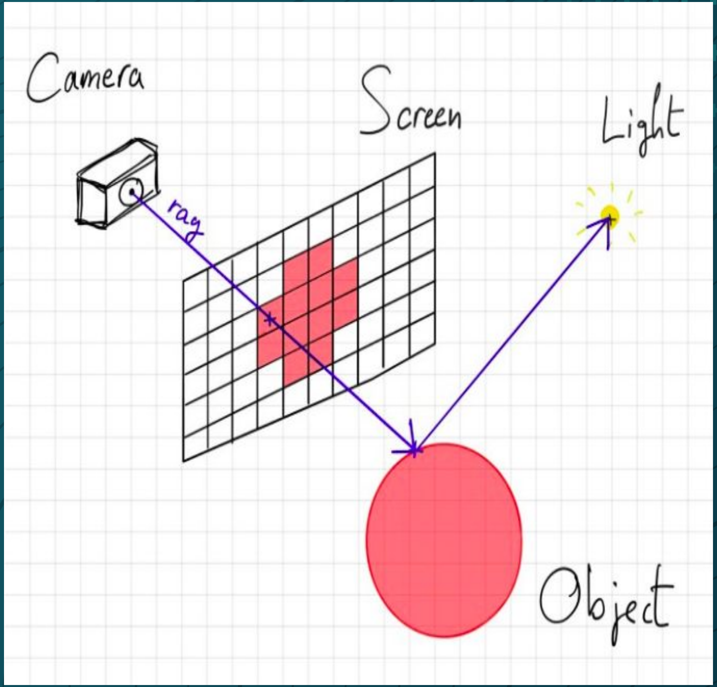
Tracing light rays and finding intersections

This step involves finding the light rays that emanate from the camera to the screen, and finding their intersection with its nearest intersection with the objects in the scenery, if any. We then retrace the path of this ray to the light source from the intersection point we found out. In this way, we find the ebay that would have hit this intersection point and reached us.

NOTE : It is important that we check for obstacles when retracing the path from the intersection point to the light source, which has been done properly. This helps us in directly assigning the colour of the intersection point as black, in case any obstacle exists.

Using geometry

The geometry(sphere) of the object has been exploited for finding the intersection point of the ray with the object. A sphere is represented by its centre and its radius. We have used these 2 parameters, along with the ray vector, to find the intersection point. We have similarly retraced the path from the intersection point to the light source.



Colouring the screen

The geometry(sphere) of the object has been exploited for finding the intersection point of the ray with the object. A sphere is represented by its centre and its radius. We have used these 2 parameters, along with the ray vector, to find the intersection point. We have similarly retraced the path from the intersection point to the light source.

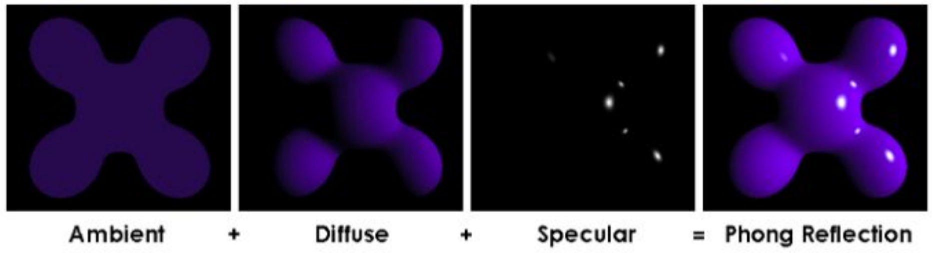
We have used the Blinn Phong model to find the illumination of the intersection point due to the light source.

$$I_p = k_a * i_a + k_d * i_d * L \cdot N + k_s * i_s * (N \cdot \frac{L + V}{\|L + V\|})^{\frac{\alpha}{4}}$$

eq. 7 — Blinn-Phong model

The Blinn Phong model

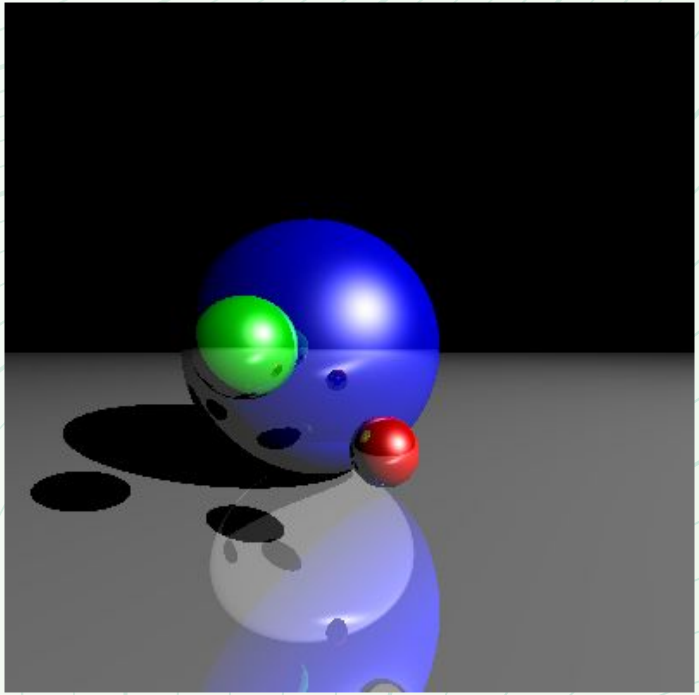
In simple words, Blinn Phong model describes that the illumination of an object is based on its ambient, diffused and specular colour components. The ambient colour is the component in the absence of light, while the diffused is in the presence of light. Specular colour covers the white reflection of light from the object, which makes it seem shiny.



Phong reflection model — Wikipedia

EXECUTING THE CODE :

The code has been written in a jupyter notebook, with its name as '***Spheres_Ray_Tracing.ipynb***', and if executed in the cell order, will save the image as "***image.png***" in the same directory of the notebook. The code can be experimented with different values for sphere radius, sphere centres, screen size, number of light sources, etc.



SCALABILITY OF THE PROJECT:

The project was further scaled into '***Scaled_Up_Ray_Tracing.ipynb***', which produces the images for different camera and screen positions, using rotation matrix in 3D space(Linear Transformation), which were then compiled together as frames to produce a video output.

MULTIPLE CAMERA AND SCREEN POSITIONS:

The camera was rotated 360 degrees along the XZ plane, with its initial position as [-3,0,0]. The circular trajectory of the camera was divided into 50 segments, thereby giving us 50 snapshots, each equidistant from its neighbours. These 50 snapshots were then combined together sequentially to produce a video output.

NOTE: This scaled up version of the project would take about an hour or even more to produce all the 50 snapshots!! Thus, certain amount of parallelism was introduced in order to reduce the running time to around 15 to 20 minutes.The entire scaled up version is stored in the directory '**Scaled_Up_Project**'