OpenGL based Real Time Ray Tracer

Aditya Singh Rathore (2018007)
Gaurav Rai (PhD)

Abstract

In this project we aim to implement an OpenGL based Ray tracer. We create a full screen quad and use the fragment shader to compute color of each pixel. Using the pixel coordinates, we compute rays and find intersection with two primitives-Sphere and Square. Further, we implement Blinn-Phong shading, shadows and reflection effects.

1 Literature Review

Ray tracing is a simple and elegant technique. We found all information needed for implementing ray fracer in Chapter 4 and 13 of Peter Shirley's Book Fundamentals of computer Graphics.[1] which describes a recursive ray tracer. We have built our program on the algorithm as descibed in above book.

Further, to test Realtime nature of the scene, we have used a camera for demonstration. General idea in all camera movement is to capture keyboard and mouse displacement in every frame and compare with previous frame position, thus infering it as movement of object. We used idea described in this website : http://www.opengl-tutorial.org/beginners-tutorials/tutorial-6-keyboard-and-mouse/

For intersection of ray with primitives - Square and Sphere, we found material in Peter Shirley's book to be quite sufficient. The general idea is to try and find point of intersection with primitive and ray. This would be solved to find the parameter t. if $t > 0$, we might have an intersection. Stack Overflow discussion https://computergraphics.stackexchange.com/questions/8418/get-intersection-ray-with-square provides a good method for any arbitrary oriented square.

2 Approach

2.1 Ray Generation

- We created a full screen quad.
- We know that Fragment shader will be called on each pixel.
- In the fragment shader, we can get the coordinates of each pixel.
- We pass camera position and direction as uniform to the shader.
- Using camera details and pixel coordinates, we can generate parameterised equation of ray $e + td$. e is origin of ray, d is the direction and parameter t.

2.2 Camera

2.2.1 Change in orientation

- We use change in (u, v) coordinated of mouse with respect to centre point of screen.
- Change in u coordinate is change in orientation along $Horizontal\ Angle$.
- Change in u coordinate is change in orientation along $Vertical\ Angle$.

2.2.2 Change in Position

- We keep tract of arrow keys placed, per frame.
- We define a fixed displacement in the direction camera is looking at every time key is pressed.
- This is the shift in position of camera.

2.3 Object Creation

- Each primitive is represented in 3D space. Square using 4 points and sphere using centre and radius.
- We solve $e + td$ and primitive to find the parameter t.
- If $t > 0$, we have a point of intersection.
- For every intersection, we color the pixel with the color of object.
- If there is no intersection, there is no object. We color this pixel with background color.

2.4 Blinn Phong shading and shadows

- Once we find a valid intersection, we calculate and store its normal.
- Point of intersection can be calculated using the value of parameter t.
- We then use Blinn-Phong shading model for each light source.
- For every light source, we check if a ray originating at point of intersection and going towards a light source intersects any other object.
- If it does, that point will not get light, it will be in shadow of the object it intersects.

2.5 Reflection

- If the ray intersects a reflective surface, we note this fact.
- From the surface, we generate the reflected ray with ray direction and surface normal.
- We find the point where this reflected ray intersects. That point's color is the color of this point (as it will reflect this color towards eye ($-ray\_direction$)).
- However, too many rays can get created in the process. Thus, we limit it to a certain maximum number of reflections.

3 Milestones

3.1 Completed

- Ray Generation
- Object Creation
- Ray-Object Intersection
- Blinn Phong shading and shadows
- Dielectrics : Reflection

3.2 Could not be achieved

-
- Dielectrics : Refraction

4 Implementation Details

4.1 Camera

5 Outcomes

5.1 Object Creation

- We are able to achieve object creation of primitive and complex structures from combining primitives.
- Our intersection algorithm work in all orientations of objects.

5.2 Camera

- Camera movement is smooth and accurate.
- However, it keeps mouse occupied, thus, complicating any UI gadgets usage.

5.3 Blinn-Phong Shading

- We get good results with no distortions, even with multiple light sources.
- Shadows are accurate.
- Well modelled parametrs can produce results indistinguishible from reality.

5.4 Dielectrics

- Multiple reflections are handled well. However, at scenarios when number of reflections is beyond the Max_depth, we get dark spot.
- Performance is hindered significantly when number of reflective objects are increased.
- With 3 adjacent mirrors and 2 spherical mirrors placed infront of each other, we get good frame rates of around $125\ fps$.

5.5 Screenshots

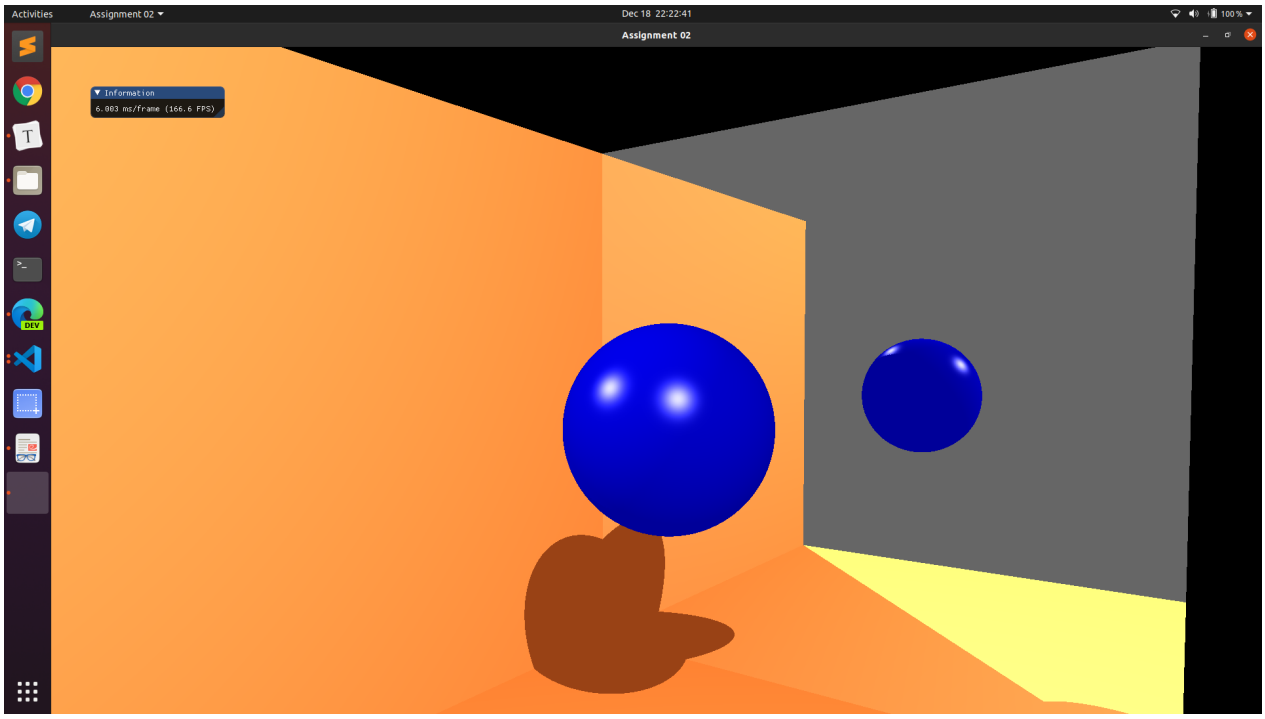

Figure 1: A simple reflection with square mirror.

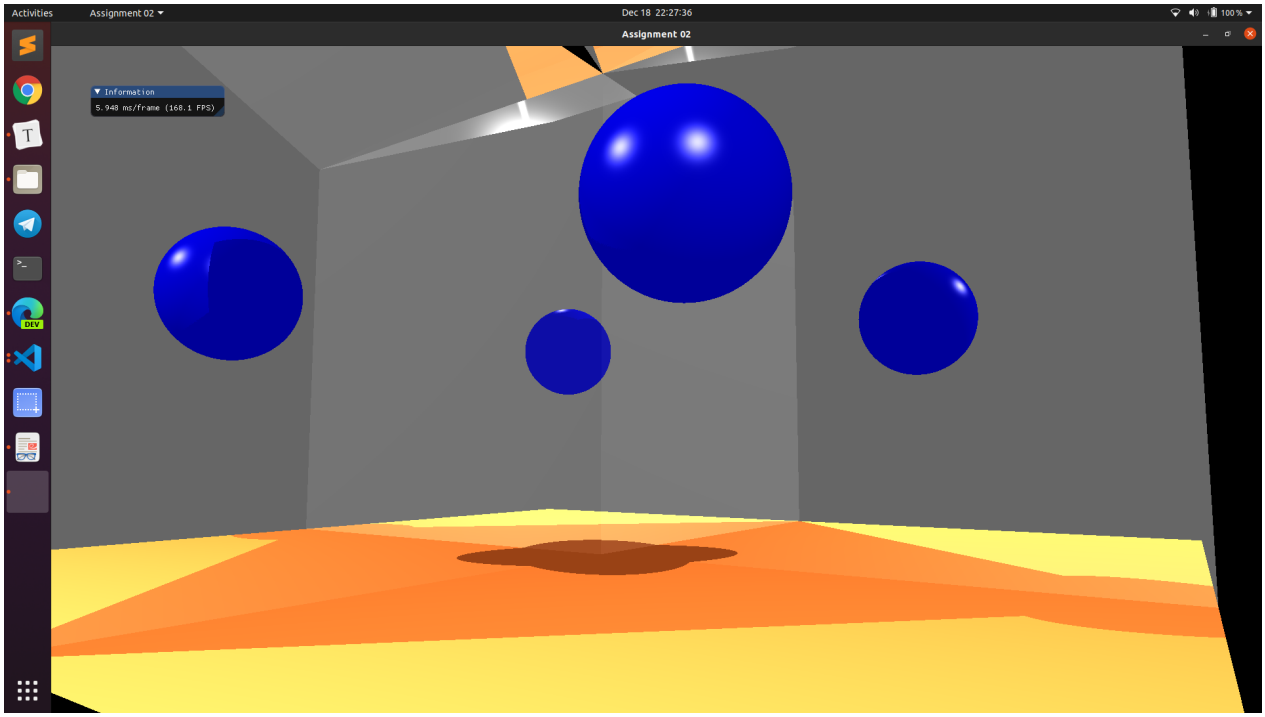

Figure 2: Single Object With multiple reflective surfaces.

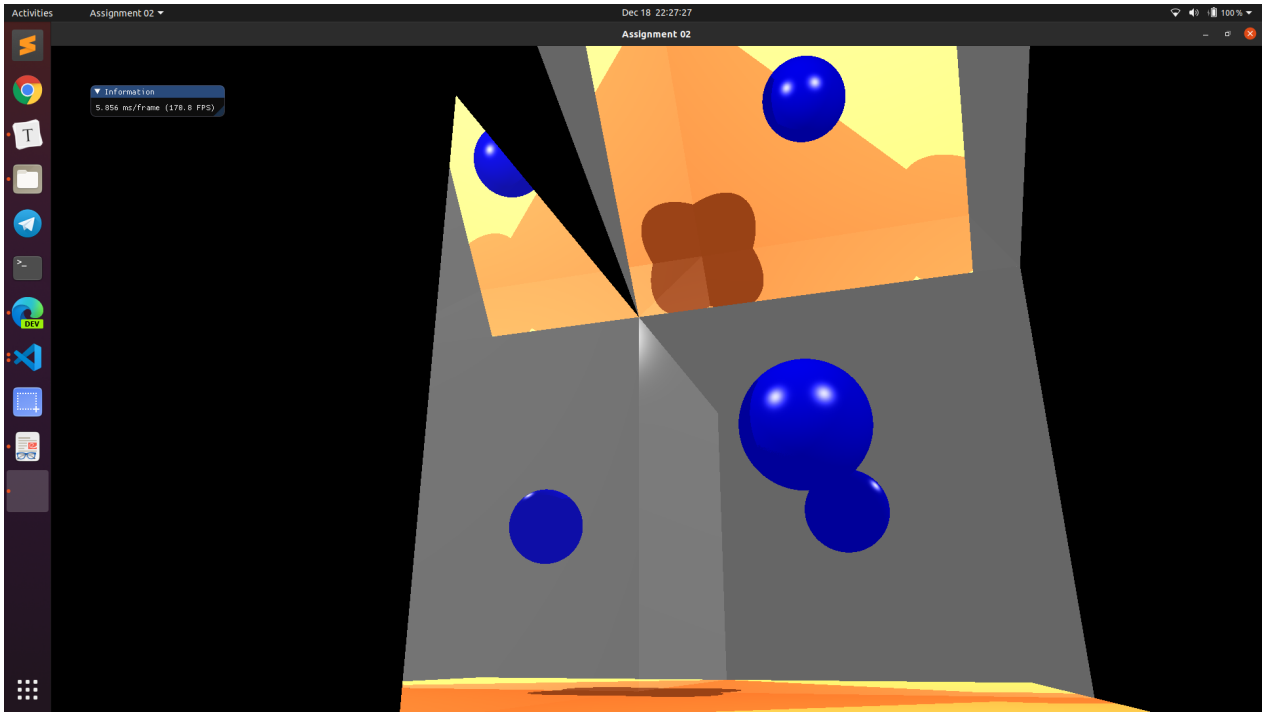

Figure 3: Non Axis Alligned Square mirror.

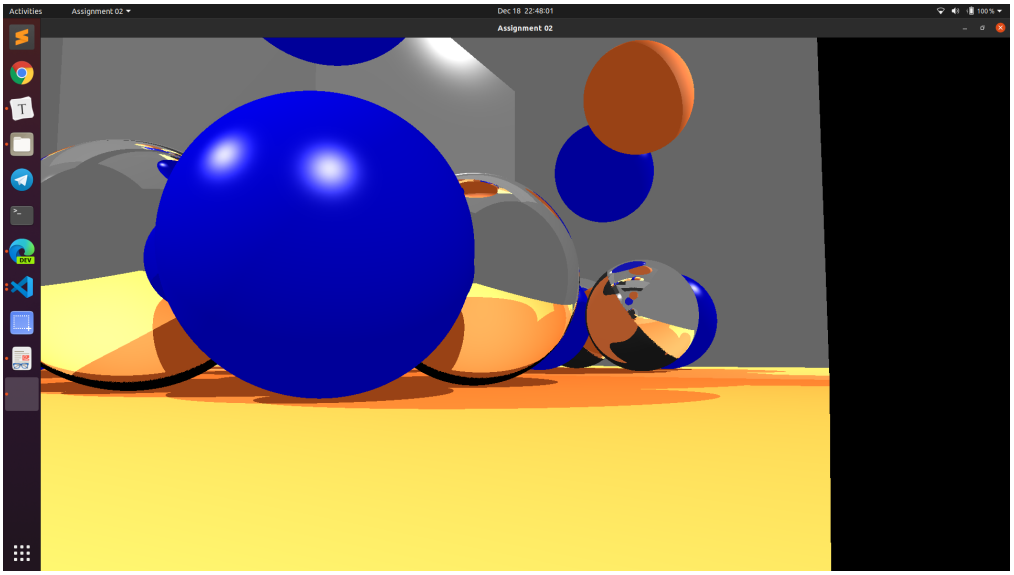

Figure 4: Spherical Mirrors.

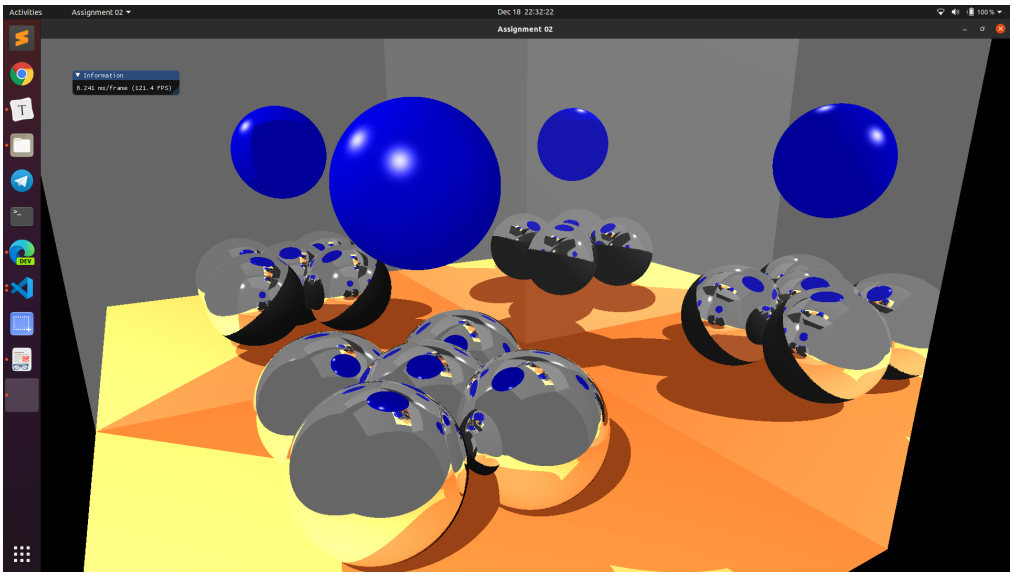

Figure 5: Multiple Reflections from multiple surfaces. Note sphereical surfaces have dark spots when reflections exceed maximum depth.

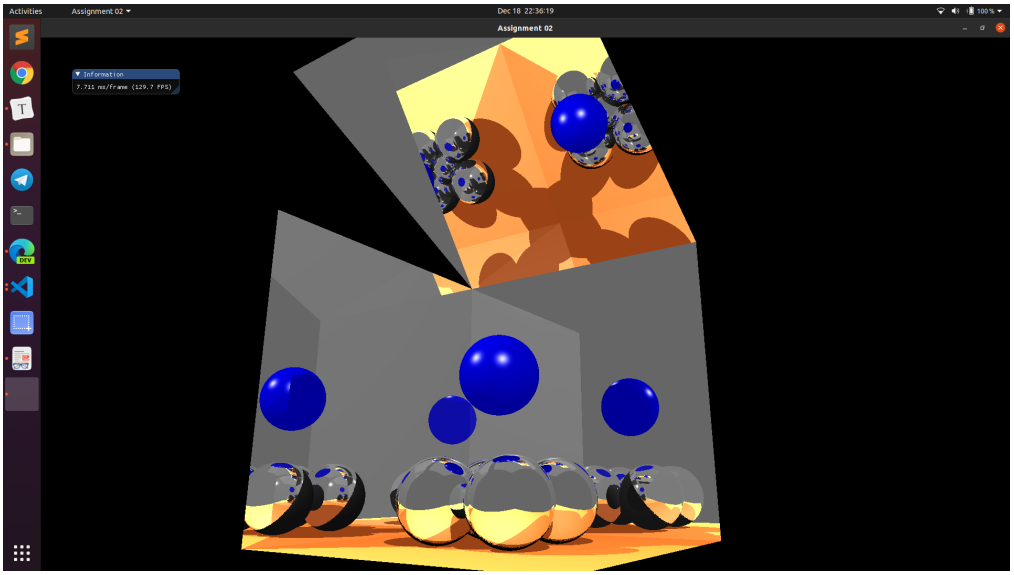

Figure 6: Panoramic View of the scene.

6 Contributions

- Project Setup : Aditya and Gaurav
- Camera : Aditya
- Ray-Object Intersections : Aditya and Gaurav
- Blinn-Phong Shading : Aditya
- Reflection : Aditya
- Presentation and Report: Gaurav

Research :

- Aditya 50%
- Gaurav 50%

Implementation :

- Aditya 70%
- Gaurav 30%

Presentation and Report :

- Aditya 30%
- Gaurav 70%

References

[1] Peter Shirley and Steve Marschner. *Fundamentals of Computer Graphics*. 3rd. USA: A. K. Peters, Ltd., 2009. ISBN: 1568814690.