**Ray Tracing**

The project implements backwards ray tracing. I.e. the projector rays are casted from the camera and then checked for intersections with the mesh in the scene. If such intersections are obtained the closest one is chosen to be the real intersection that would have occurred between a light ray and the mesh. Based on the surface properties of the mesh new rays are generated, emanating from the point of intersection of the original ray and the mesh. These newly generated rays are then used as reflected, refracted rays. The newly generated rays can also be used to evaluate lighting at the intersection point.

Method Used

A recursive function is used to simulate the branching of rays after collision with surfaces. The pixel color value depends on a weighted sum of color values generated by each ray / mesh intersection. The weights depend on the mesh surface properties. These properties can be defined to generate the desired materials.

Lighting (diffuse)

The project demonstrates 2 types of lights (point and directional lights). Lighting at a point is calculated as follows:

1. A ray is casted from the ray/mesh intersection point towards the light (in the direction opposite to the direction of light in case of directional lights).
2. If the ray does not intersect with any mesh before reaching the light source (infinity in case of directional lights) then the surface point is light by the light.
3. Since the surfaces implemented are perfectly diffuse, color calculations are easy.

Ray Mesh Intersection Calculation

Since the intersection must be calculated with millions of rays. A structure (called collTriangle in the project) is initialized to represent each triangle and accelerate intersection calculations.

This structure contains the mathematical description (a point and a normal) of the plane of the triangle, and 3 planes perpendicular to the triangle containing sides of the triangle.

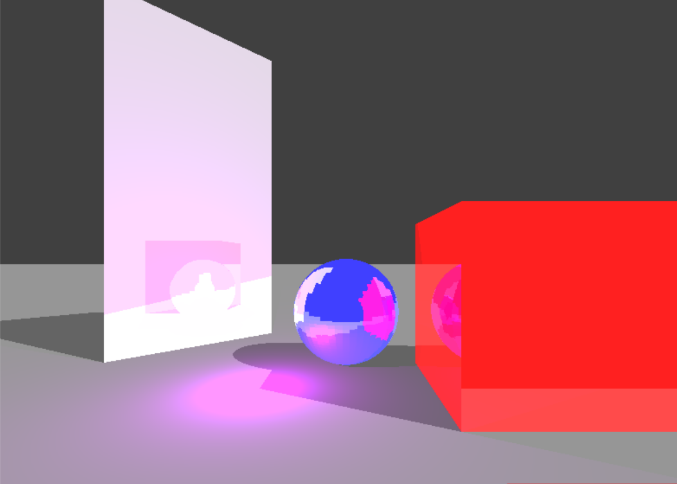
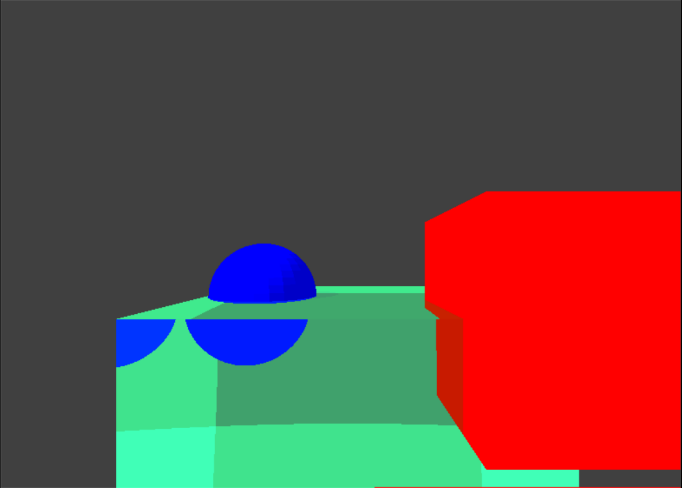
The plane of the triangle is used to calculate the ray/mesh intersection and the other 3 side planes are used to check whether the intersection was inside the triangle or not.

Problems Faced

Some of the initial images obtained were either too saturated or too dark and required precise lighting to result in the full range of colors.

Images before fixing this issue:



Reason for the Issue:

Each screen pixel can display 256 shades per color(r,g,b) ie each pixel requires 3\*8 bits of data i.e. 3 Bytes of data.

Since the color calculations take place on float data types it is easy to over shoot 256 and hence the resulting image is too saturated.

Solution:

The solution to this problem comes from biology. The human eye is adaptive in nature i.e. if too much light enters the eye the iris shrinks to reduce the light inflow preventing the light sensitive cells in the eye from getting overwhelmed.

Such a solution can be implemented in software. The entire scene is rendered with floating point color operations. The brightest pixel on the screen can then be used as an upper limit. The conversion from floating point to 8 bit unsigned integral values then becomes a map from (0 – determined upper limit) to (0 – 255).

Images after applying the fix:

