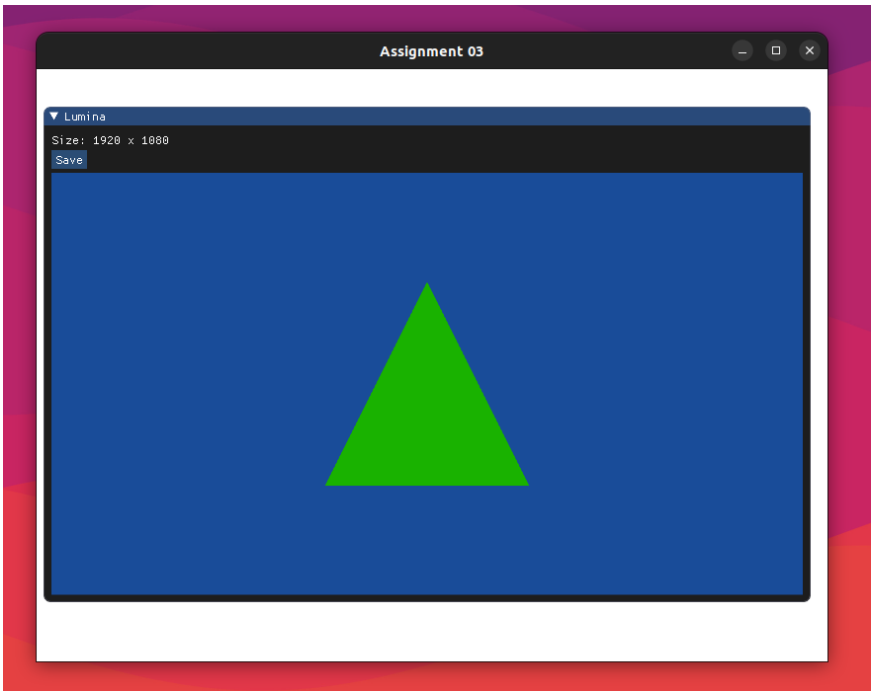
Assignment 04: Raytracing

Documentation

Tanishq Jain, 2021294

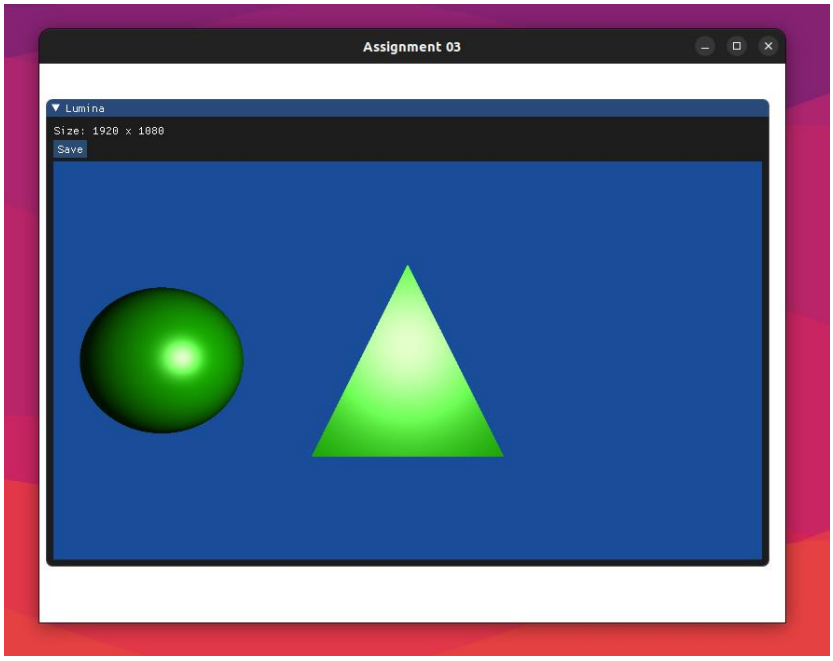
Question 1 - Extend the Object class to implement a Triangle that can be rendered in the raytracer.

We create a new class, Triangle which extends Object class. It is defined using the co-ordinates of its 3 vertices – vertexA, vertexB and vertexC. To check if a given ray ray intersects with our triangle, we implement an intersect() function. It finds the barycentric co-ordinates of the point of intersection of ray with plane containing the triangle. Then we use a condition to check whether this point lies inside / outside the triangle. If it lies inside the triangle, we set the ray parameters and update the value of t if it is smaller than current value.



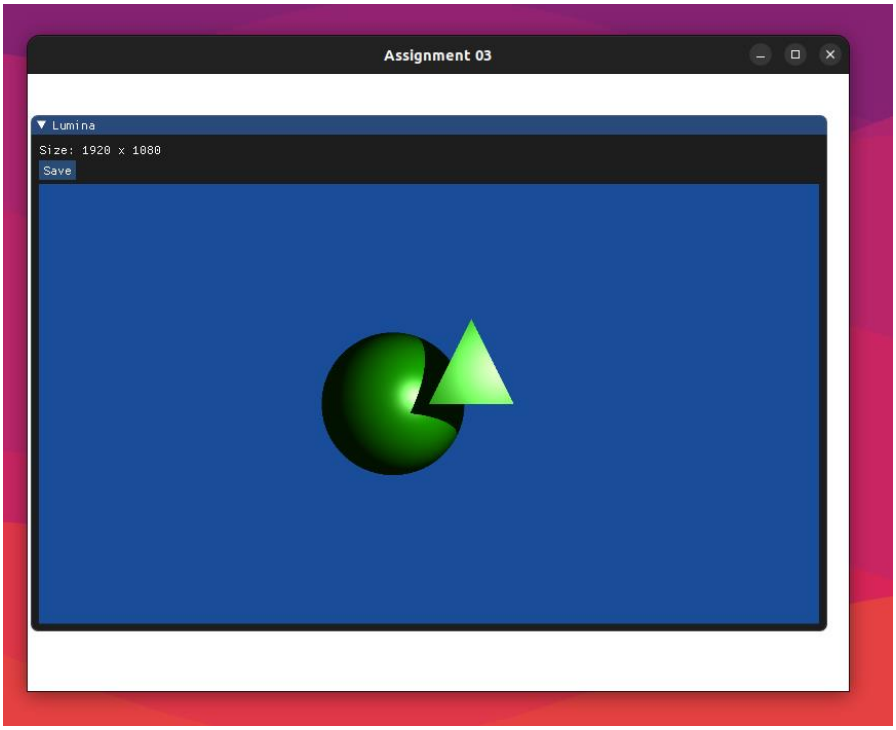
Question 2 - Implement Blinn-Phong shading for the shapes.

In order to implement blinn-phong shading, we first implement a subroutine getNormal() in triangle.cpp and sphere.cpp. It returns normal to the surface at point of intersection with the ray and is required for shading calculations. Then, in material.cpp, we update the shade() method to implement shading rather than a flat color. Initially an ambient component is added to the color. Then, we iterate over each light source and add the corresponding diffuse, specular components. Finally, we return the overall color.



Question 3 - Implement shadows in the raytracer.

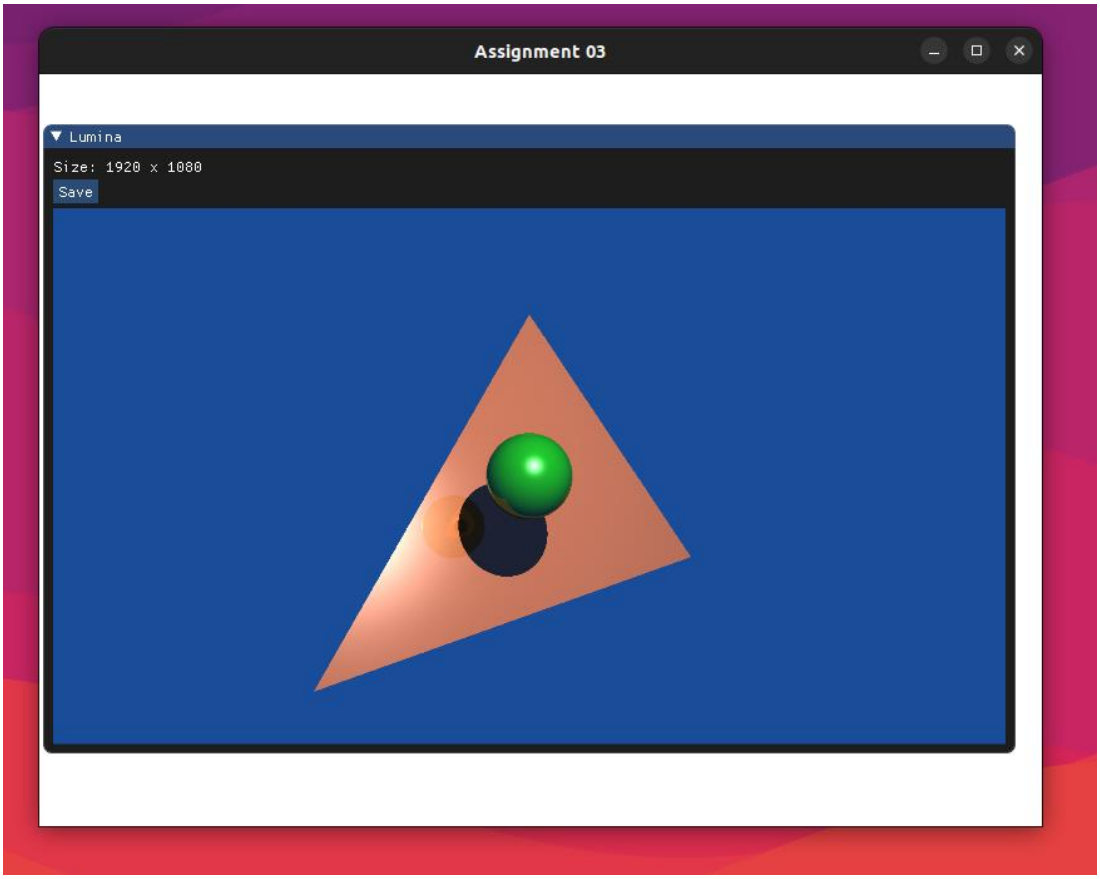
To further implement shadows in our raytracer, we add code to the shading model computations in material.cpp. While iterating over each light source, we create a shadow ray. Its point of origin is the point of intersection of the previous (incident) ray, and direction is from origin to the given light source. Then, we iterate over objects in the world and use the intersect() method to check if the object intersects with this shadow ray. If any object intersects with the shadow ray, it means that the object acts as an obstruction between the surface and the given light source. Hence, we do not add the diffuse and specular components from this light source.



Question 4 - Implement reflective and dielectric materials (make your raytracer recursive).

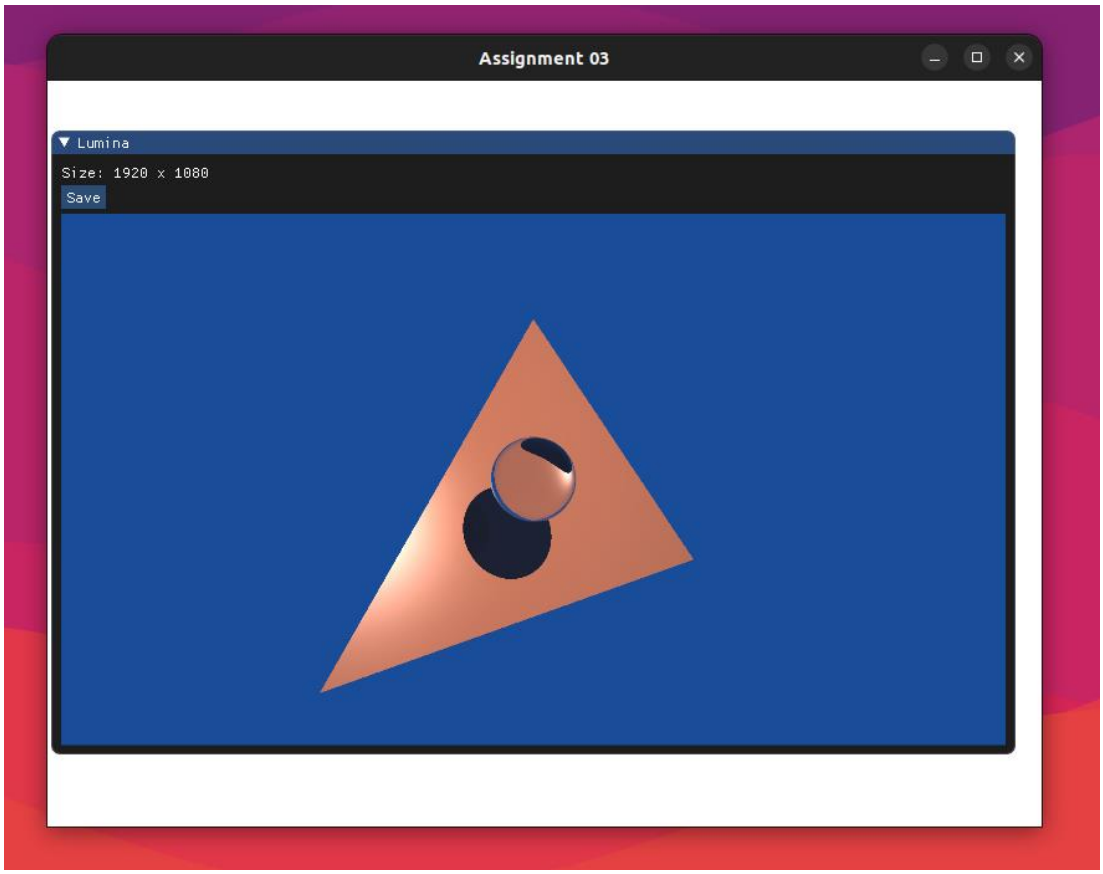
In order to implement reflective and dielectric materials, we modify the shade_ray() method present in world.cpp. Direction of reflected ray is determined using direction of incident ray and normal to the surface. Its origin is same as the intersection point of incident ray. If object is solid, its color is computed as combination of shading model and recursive contribution of reflected ray. Else, we use a condition to check whether light is entering / exiting the dielectric. If it is entering the dielectric, we simply compute refracted ray and apply schlick's approximation to combine values from recursive contributions of reflected and refracted rays. If light is exiting the dielectric, we first apply beer's law to compute intensity loss factor for its travel inside the dielectric. Then, we use a condition to check whether it escapes out / total internal reflection takes place. In case of total internal reflection, we just return the value from reflected ray. Else, we apply schlick's approximation for both reflected, refracted rays. We also set a DEPTH_LIMIT = 5 as a base case for recursion.

1) Setting isSolid = true for sphere



Reflection

2) Setting isSolid = false for sphere



Refraction