

Stage 2 and 3 reports

CN468

December 12, 2022

Abstract

This report explains what I have done for stages 2 and 3 of the coursework.

1 Introduction

Basic Ray tracing produces an image without the complexities we see in the real world like soft shadows, diffuse inter-reflection and caustics. However, modelling these more complex scenes is possible with the marriage of maths, physic and an in-depth understanding of the visual world. In this report, I will be discussing implementations of ray optics for reflection and refraction, quadratic objects, Additional objects and photon mapping with caustics.

2 Stage 2: Feature-Rich Rendering

2.1 Quadratics

What is the theory behind Quadratic objects?

Quadratic objects are geometric shapes that have a degree of two; they are shapes that can be described by quadratic equations such as $x^2 + y^2 + z^2 - c = 0$. Given that these objects are purely mathematical we can deduce the intersection points between the object and a line by solving the quadratic equation to get the points where the value of the quadratic equation is zero. In the case of ray-tracing the line that intersects with the quadratic object is defined as follows.

A Ray:

$$R(t) = P + D(t), \text{ where } D \text{ is the direction vector and } p \text{ is the origin position vertex.}$$

The equation calculate the position given a t displacement along the ray with respect to the direction of the ray.

we can further deconstruct this equation to it dimensional parts, namely the x,y and z dimensions.

$$\begin{aligned} x &= p_x + tD_x \\ y &= p_y + tD_y \\ z &= p_z + tD_z \end{aligned} \tag{1}$$

The Quadratic equation:

$$ax^2 + 2bxy + 2cxz + 2dx + ey^2 + 2fyz + 2gy + hz^2 + 2iz + j = 0$$

These points can be found using the quadratic formula, by substituting the x,y and z ray equation into the quadratic equation, then collect the coefficients for the squared terms the x,y and z terms and the rest as shown by [Ken22b]; we will call these collections A, B and C. Now we can use the quadratic formula to solve for t . The quadratic equation:

$$t = \frac{-b \pm \sqrt{b^2 - 4AC}}{2A}$$

The discriminant:

$$b^2 - 4AC$$

where discriminant > 0 means we have real solutions.

What if the ray(line) never intersects the object? We can guard for this using the discriminant. We acknowledge that the ray and the quadratic object exist in the real space, so we can use the discriminant to determine if the quadratic equation has a real solution. When we have the real t_0 and t_1 values we can calculate the two points of intersection with the ray equation, using the equation (1). After getting the hits points one would need to calculate the normal. using the equation:

$$\begin{aligned} \text{normal.x} &= A * \text{position.x} + B * \text{position.y} + C * \text{position.z} + D \\ \text{normal.y} &= B * \text{position.x} + E * \text{position.y} + F * \text{position.z} + G \\ \text{normal.z} &= C * \text{position.x} + F * \text{position.y} + H * \text{position.z} + I \end{aligned}$$

The final step is to apply a transform, this is to bring the object into view in the world space, this can simply be done by applying a transformation matrix to the following quad matrix representation.

$$T^T * \text{quad} * T^{-1}$$

Below are a set of 3 quadratic coefficients and the images they produce:


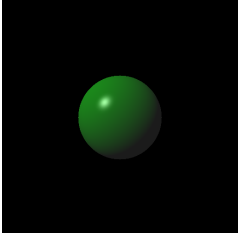

Type	Equation	My Images
Cone	$x^2 + y^2 - z^2 = 0$	
Sphere	$x^2 + y^2 + z^2 - j = 0$	
Cylinder along x	$y^2 + z^2 - j = 0$	

Figure 1: A table of three quadratic object a cone, a sphere and a cone. depicting the type of the object the coefficients and the image of the object rendered using ray-tracing.

2.2 CSG

CSG is a way of representing 3D objects using primitive geometry (like spheres) and set operations to create more complex 3D models. Essentially all the work for CSG is done in the intersection method of the class. The theory is quite simple, if you are given a set of object-ray-intersection points for two objects try to combine these hit points in such a way that it maintains the principles outlined by set operation (\cup , \cap , $-$)

The process can be easily explained form the figure below:

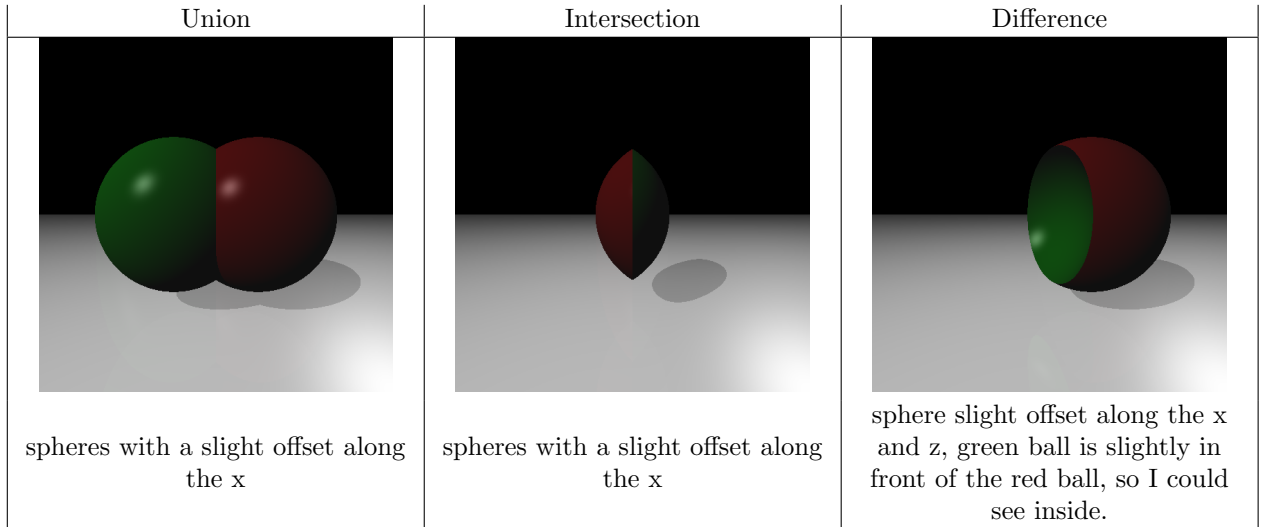


Figure 2: A table of three CSG objects with three set operations performed between 2 spheres. Union on the left, Intersection in the middle and difference on the right.

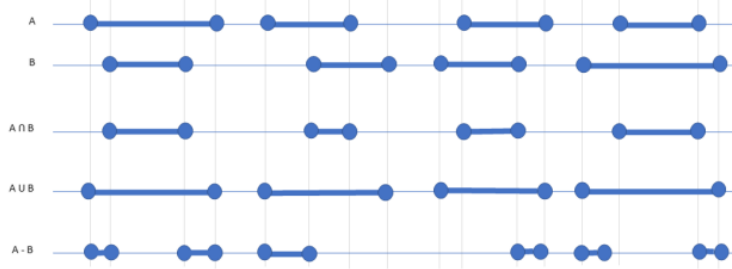


Figure 3: CSG operations as a number line/ray. From CM30075: CSG short notes by [Ken22a]

From figure 3 we can derive a decision table for each set operation and we simplify the decisions table by combining similar outcomes. For example take the decision tree for CSG Difference in figure 5b. we can simplify the decision table: as shown in figure 4b.

A	B	Comparison	Action
outside	outside	$A < B$	Keep A
outside	outside	$A > B$	Discard B
outside	outside	$A = B$	Discard B
inside	outside	$A < B$	Keep A
inside	outside	$A > B$	Keep B
inside	outside	$A = B$	Keep A
inside	inside	$A < B$	Discard A
inside	inside	$A > B$	Keep B
inside	inside	$A = B$	Discard B
outside	inside	$A < B$	Discard A
outside	inside	$A > B$	Discard B
outside	inside	$A = B$	Discard B

(a) full decision table from short notes by [Ken22a]

A	B	Comparison	Action
outside	...	$A \geq B$	Discard B
inside	...	$A > B$	Keep B
...	outside	$A < B$	Keep A
...	inside	$A < B$	Discard A
inside	outside	$A = B$	Keep A
inside	inside	$A = B$	Discard A

(b) Simplified decision table

Figure 4: 2 Figures A and B are decision table (A) from a CSG short notes by Ken Cameron and (B) My simplified version

For defining inside and outside in code, I have used *hit* → *entering* as outside and it negation to

be inside.

2.3 Ray optics

Ray optics, we extend the Raytracer to implement reflection and refraction.

2.3.1 Reflection

Reflection occurs when light bounces off a surface. When light hits a surface, some of it is absorbed by the material and some of it is reflected.

The law of reflection states that the angle of incidence is equal to the angle of reflection. This is a fundamental principle of geometric optics, the study of how light behaves and interacts with objects. This is based on the fact that light always travels in a straight line. So, when calculating the angle of reflection, we simply need to calculate the angle of incidence $\cos(\theta)$. This is done using the dot product formula, which takes two vectors and returns a scalar value. Using the normal N as our reference vector we can calculate the angle of incidence with $(N.I)$. Now we need to calculate $(N.I)N$, which is a projection of the incidence vector onto normal this tells us in geometry terms the height of the incidence ray along the normal. The projection is negative because the incidence is going into the normal. Next, we need to calculate the vector from the tail of the incidence to the tail of the projection we just calculated. This is done by traversing the incidence and then going up the projection $(I - (N.I) * N)$, this new vector is parallel to the surface. Now we can traverse from the hit point to the head of the reflection direction with $(I - (N.I) * N - (N.I) * N)$, this is our reflection direction.

$$R = I - 2(N.I) * N$$

In the Global material class, I have implemented reflection in the compute once method. as they are part of global illumination. I have a recursive depth of 5 ray bounces. the process is as follows calculating the reflection direction using the method described above, creating a new ray with the hit position as the origin and the reflected ray direction, and then send the ray back into the scene. This produces the image below:

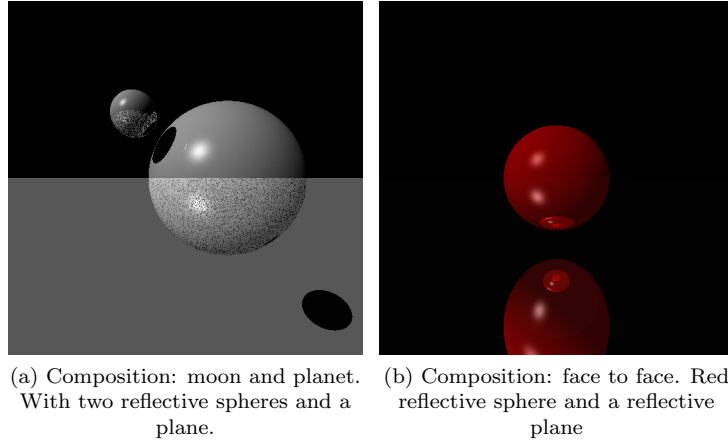


Figure 5: (a) Image of reflection with noise due to floating point rounding. reflection rays start inside the object. (b) Image of reflection without noise as reflection origin offered by some small epsilon ($1 * 10^{-4}$)

2.3.2 Refraction

The theory of refraction states that when light passes from one transparent medium to another, it changes direction. This is due to the fact that light travels slower in a medium than in a vacuum, and the ratio of the speed of light in the two mediums is known as the index of refraction. This is expressed as $\eta = \frac{c}{v}$, where c is the speed of light in a vacuum and v is the speed of light in the medium. The

amount of bending of the light ray depends on the angle of incidence and the refractive indices of the two mediums. This is described by Snell's law, which states that $\frac{\sin\theta_1}{\sin\theta_2} = \frac{\eta_2}{\eta_1}$. This law can be used to compute the direction of the refracted ray, with the equation

$$T = \eta * I + (\eta * c1 - c2) * N$$

, where η is the ratio of the indices of refraction, $c1$ is the cosine of the angle of incidence, and $c2$ is $\sqrt{1 - (\frac{\eta_1}{\eta_2})^2 * \sin^2(\theta_1)}$, I is the incident ray, and N is the normal at the hit point. Additionally, the normal direction will need to be inverted if the incident ray enters the second medium from the outside. I have also inverted the hit normal when the hit is inside the medium. After I have calculated my refraction direction I create a new ray with the hit position as the ray origin with a bias of $1 * 10^{-4}$ along the ray direction. Then I send the ray back into the scene. This will produce the following scene

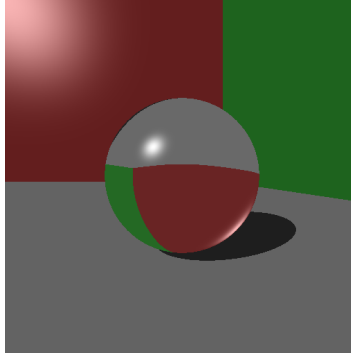
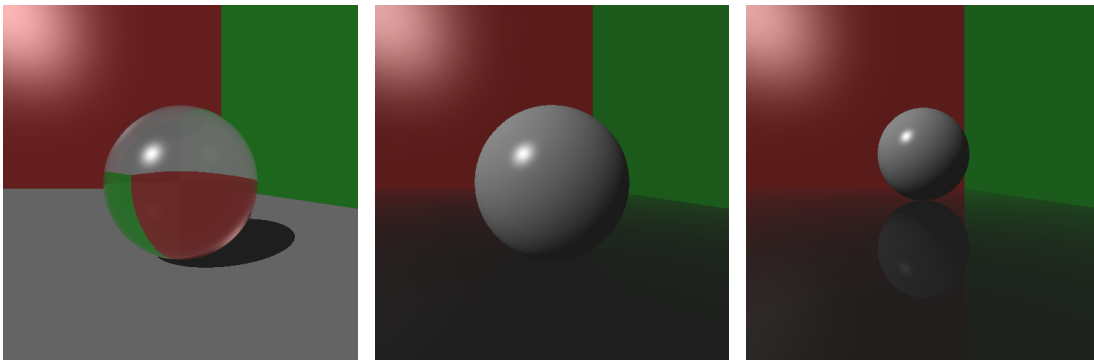


Figure 6: Image of a sphere on a grey plane bottom, a green plane left and a red plane back.

2.3.3 Fennel term

In the real world transparent objects do not only refract light, they also reflect light, consider water or glass. The angle of incidence determines how much light is reflected and refracted, so most transparent objects are view-dependent. The Fresnel equations, named after French physicist Augustin-Jean Fresnel aims to determine the amount of light that is reflected and refracted when a ray of light hits a surface depending on the angle of incidence and the refractive indexes of the mediums involved. In the code, I have given the Fresnel function, the incident ray, the hit position and the two *ior* for both mediums involved. Then the reflected ratio KR is calculated using the equation (2)



(a) Fresnel term on the sphere, sphere showing reflection and refraction on a glass sphere.

(b) Fresnel term on the plane with sphere close, in the image we see little to no reflection on the plane.

(c) Fresnel term on the plane with a sphere on top and far away, in this image a-lot of reflection on the plane.

Figure 7: Fresnel term applied on a glass sphere on the left, on the plane in the middle with sphere close and on the right with the sphere far.

$$\begin{aligned}
FR_{parallel} &= \left(\frac{\eta_2 \cos \theta_1 - \eta_1 \cos \theta_2}{\eta_2 \cos \theta_1 + \eta_1 \cos \theta_2} \right)^2 \\
FR_{perpendicular} &= \left(\frac{\eta_1 \cos \theta_2 - \eta_2 \cos \theta_1}{\eta_1 \cos \theta_2 + \eta_2 \cos \theta_1} \right)^2 \\
KR &= \frac{1}{2} * (FR_{parallel} + FR_{perpendicular})
\end{aligned} \tag{2}$$

We use the equations to find the reflective coefficients parallel $FR_{parallel}$ and the perpendicular $FR_{perpendicular}$ to the plane and then calculate the average between them. Now that we have the reflective coefficient, we can calculate the refractive coefficient by subtracting the reflective coefficient from 1 ($KT = 1 - KR$). This produces the images in 8

2.4 Additional object (cube)

For My additional feature, I have added a new cube object in the cube_object class. We create the Cube in the constructor using the centre position and the size $Cube(vertexcentre, floatsiz)$. In the constructor, we use the centre and the size of the cube to calculate the min and the max point of the cube. With the equations:

$$\begin{aligned}
Vector \ max &= center.x + size, \ center.y + size, \ center.z + size \\
Vector \ min &= center.x - size, \ center.y - size, \ center.z - size
\end{aligned} \tag{3}$$

For calculating the intersection points, we can use the min and max. We will call them the bounds. We want to make an axis-aligned box, so the bounds of the box define a set of lines parallel to each axis of the coordinate system. $y = min.x$, we want to find the intersection between the line $y = min.x$ and $y = ray.pos.x + t.x * ray.dir.x$ we substitute y to make $min.x = ray.pos.x + t * ray.dir.x$, we rearrange to get the equation:

$$t.x = \frac{(min.x - ray.pos.x)}{ray.dir.x}$$

We repeat the same for the max.x bound and again for all the axis. This will get us the points where the ray intersects the planes defined by each face of the box. Intersecting the plane does not mean we intersect the cube. To find the point that intersects the cube, we simply compare the t values. This will produce a Cube in the scene. Because the box is an AABB we have no orientation, so applying transform only moves the box along each of the axes with respect to the translation performed.

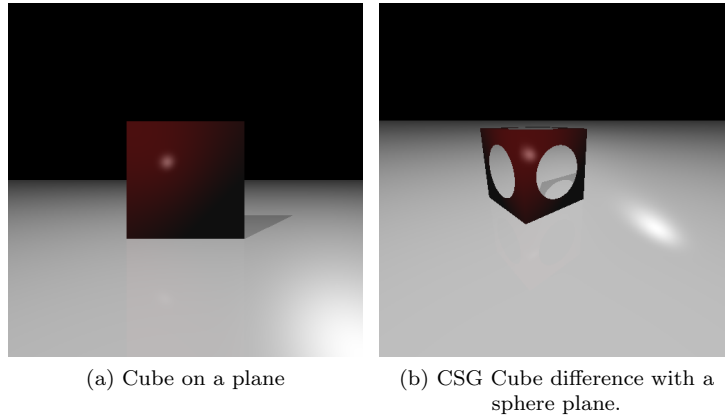


Figure 8: Additional cube object on the right and CSG cube object on the left.

3 Stage 3: Photon Mapping

Photon mapping introduced by Jensen [Jen96] is a way of simulating global illumination, including inter-reflections, caustics, colour bleeding and more. Photon mapping simulates the effect of packets of energy (photons) bouncing around the scene. It can produce images that are hard to achieve using traditional methods. Photon mapping consists of two parts: constructing the photon maps and rendering.

3.1 Phase 1: constructing the photon map

We construct a photon map by shooting a large number of photons in random directions from the light source in the scene. When a photon hits a surface, it can be stored, reflected or transmitted depending on the object that is hit. The diffuse object can store and reflect the photon whilst specular objects (in our case, objects with global material) can reflect and transmit. Each diffuse hit is stored in a data structure, which is, for us, the kd-tree. I have used a Russian-roulette to determine what the photon should do when it hits an object, Russian roulette is a Monte Carlo technique used to determine whether a given photon should be absorbed, reflected or transmitted and because the photons are terminated randomly, this method can help reduce bias in the photon map, which can improve the overall accuracy of the rendered image. With regards to generating the photons, I initially tried to uniformly distribute the photons around the light source using Solid angle dependency - Inverse CDF, which got some weird effects so I proceeded by using sampling with random numbers. The photon maps can be seen in the appendix [figure:11]

We perform the same operation with Caustics; the method I have used is to shoot the photons randomly into the scene if the photons hit a specular object (in this case, if the photon hits a Global Material), it is traced until it hits a diffuse material where it is absorbed. I have used a kd-tree [kd-] (<https://github.com/gishi523/kd-tree/blob/master/kdtree.h>). I found on GitHub; this tree is simple to use and doesn't require much alteration to get it up and running. This implementation uses `vector<photons>`, which can be considered a list of photons as input to build the entire kd-tree in one go. as opposed to dynamically building the tree for every new photon object interaction.

3.2 Phase 2: Rendering

In this stage, we perform normal ray tracing, where the ray is shot from the camera into the scene. In the compute once function found in the Phong materials, I have created a function called `get_radiance()` this function computes a radiance estimate at the hit point by performing a radius search in the kd-tree; this will return all the photons within the radius at the hit position. I then perform cone filtering on the photon to weigh the photons; this will give more importance to the photons closer to the hit point. This produces the images below:

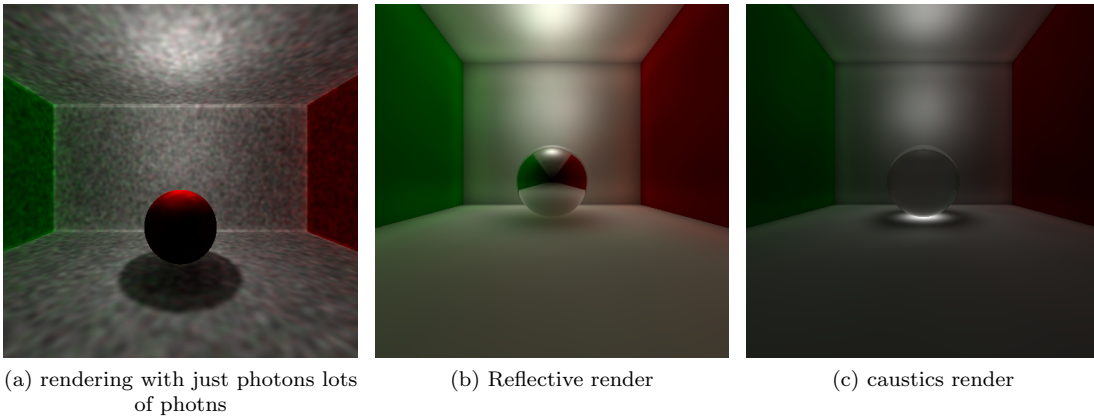
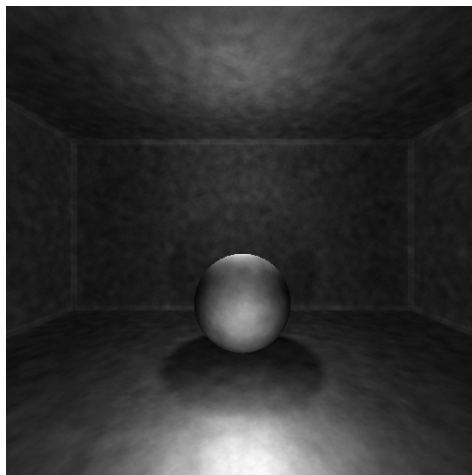
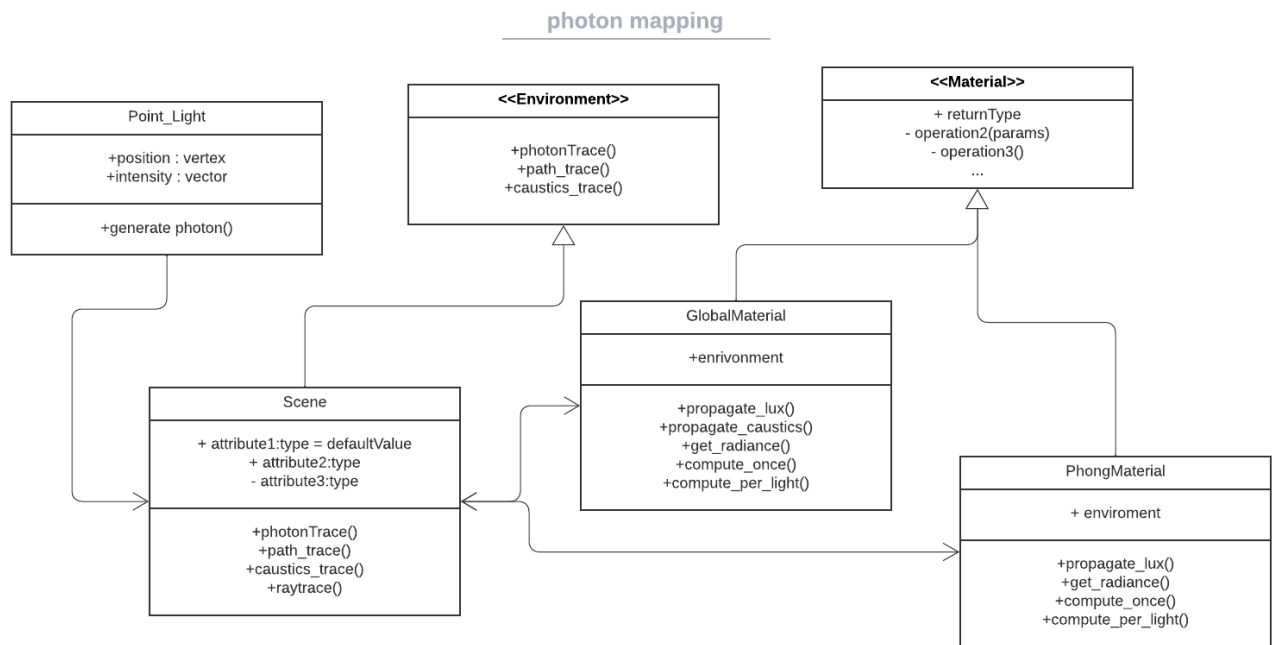


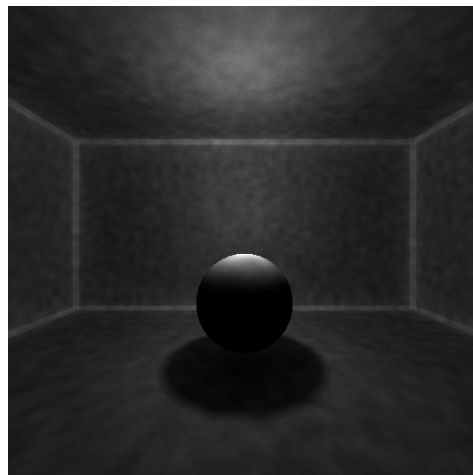
Figure 9

4 Appendix

5 UML Diagram for photon mapping

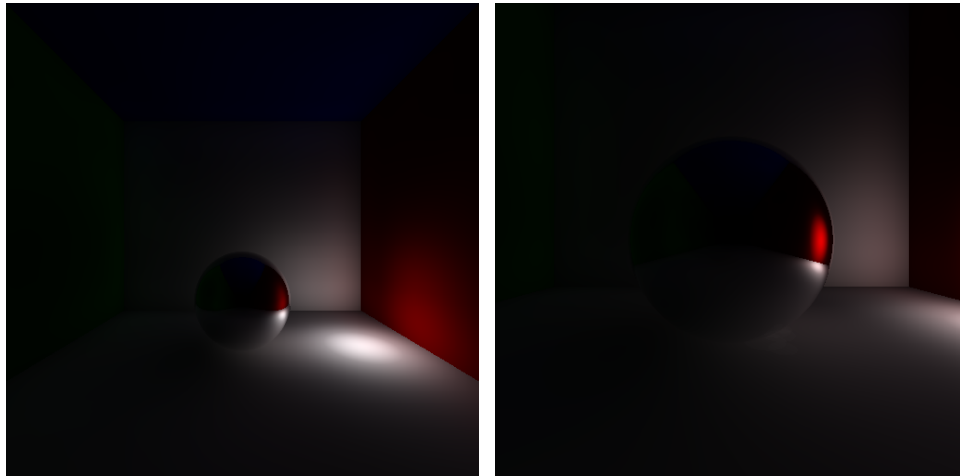


(a) Photon map using inverse CDF, in this image, the light source is directly above the sphere, but it appears to be in front of it.



(b) Photon map using a three uniform random generated numbers

Figure 10: Photon maps both with 10000 photons



(a) A cool image using Photon map.

(b) Another cool image.

Figure 11: Photon maps both with 100000 photons

References

- [Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Eurographics workshop on Rendering techniques*, pages 21–30. Springer, 1996.
- [kd-]
- [Ken22a] Cameron Ken. Cm30075: Advanced graphics short note constructive solid geometry. pages 1–3. University of Bath, 2022.
- [Ken22b] Cameron Ken. Cm30075: Advanced graphics short note raytracing quadratic surfaces. pages 1–3. University of Bath, 2022.