CM30075

Advanced Computer Graphics

January 2021

INTRODUCTION

The aim of this report is to analyze and describe the implementation of Photon Mapping, Reflection and Refraction using C++. This would lead to a deeper understanding of the core of Computer Graphics.

REQUIREMENTS

1. Reflection and Refraction

Explanation: The first requirement consists of creating realistic reflections and refractions that can be visible in the final image. For reflection, we compute the direction of the outgoing ray using the reflection law:

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

Where I is the incoming ray and N is the normal at the hit point between the incoming ray and the object's surface. For refraction/transparency we use the equation:

$$T = (eta * I - (eta * cosi - sqrtf(k)) * N$$

Where I si the direction of the incoming light, N is the normal at the hit point, eta is etai/etat (1/index of refraction), and k = 1-eta*eta*(1-cosi*cosi), where cosi is the dot product between N and I. To make the reflection and refraction more realistic and to allow both physical phenomenons appear on the same object we make use of the Fresnel equation that helps us work out kr and kt. By kr and kt we refer to the reflection coefficient and transparency coeficient respectively.

$$kr=(Rs*Rs+Rp*Rp)/2$$

$$kt = 1 - kr.$$

where Rs is the parallel wave of light Rp is the perpendicular/polarised wave of light denoted by

Rs = ((etat * cosi) - (etai * cost))/((etat*cosi)+(etai*cost))

Rp = ((etai*cosi)-(etat*cost))/((etai*cosi)+(etat*cost)).

Etat, etai and cosi are the same terms mentioned above and

cost = sqrtf(1-etai/etat*pow(sqrtf(1-cosi*cosi),2)).

Implementation: `fresnel(Vector &direction, Vector &normal, float &ior,float &kr)`, `transparent(Vector direction, Vector N, float ior)`, `reflection(Vector direction, Vector normal)` methods in the `Scene3` class are used to implement the above functionalities and they were created using bits of code from the lectures and [1]. They return the kr, refracted ray direction and reflected ray direction respectively. These methods are used in the `photon_raytrace` and `photon_trace` methods to compute the new direction of a reflected/refracted photon and the colour of a reflected/refracted ray. To restrict the computational time we use these functions only when the bool reflection/refraction material attributes allow them.

Fig. 1.1, fig. 1.2 and fig. 1.3 show the results after using the above implementation.

Fig. 1.1 The teapot has an index of refraction of 1.5 (similar to glass).

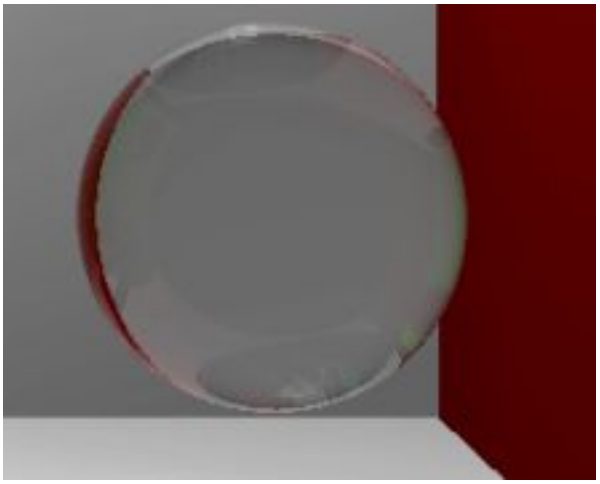


Fig 1.2 depicts a sphere with an index of refraction of 1.33, similar to water and the reflection can also be observed mostly around the edges.

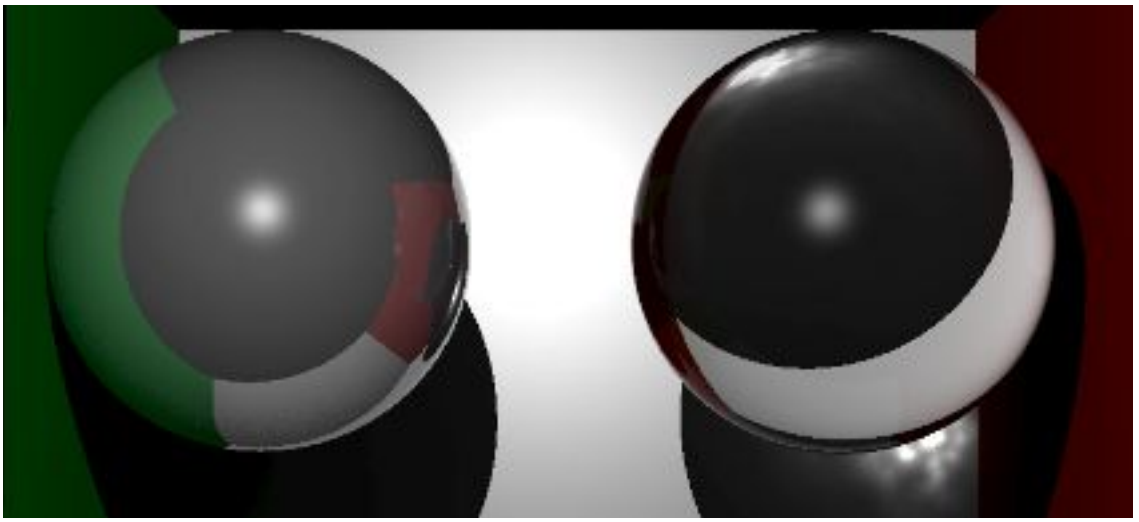


Fig. 1.3 For the left sphere the intention was to mimic steel (ior = 2.5). The reflections of walls (as well as shadows), floor and second sphere are easily noticeable. For the right sphere, the intention was to mimic glass (ior= 1.5) allowing for reflection and refraction at the same time. At a closer look, the reflection of the room and firs sphere can be observed around the edges.

2. Photon Mapping

Explanation: The second requirement consists of implementing Photon Mapping for more accurate lightning in the scene. Photon Mapping is a two-pass algorithm and the implementation is based on [2]. In the first pass, we crate two maps containing photons. One map for the caustic photons and the other for the rest of the photons. To create the maps, we emit photons from the light source in random directions and let them bounce in the scene. We use path tracing to follow the photons through the scene. When a photon hits a surface we store the photon in a kd tree that represents our map. The kd tree used is part of the nanoflan library [3]. To determine the next action of the photon (absorption or specular bounce, refraction) we use the Russian Roulette [4]. A random number is generated each time a new photon hit happens. Using the specular and diffuse component of each material we can compute the probability of the arriving photon to be either absorbed or reflected. The random number mentioned above is used to determine the interaction of the photon with the surface it hits. In this way, we avoid creating more photons for each case every time a photon hit happens, therefore reducing the computation time. The new direction of a reflected photon is calculated using BRDF on the surface. For each photon, we want to remember its position, the direction it is coming from and its intensity, and we use an array for this that is then stored in the kd tree as a node. This data structure optimises the search that will be performed in the second pass and therefore speeds up the rendering process. A kd tree uses binary search and is optimal for searching a specified i number of nearest neighbour search. We base our rendering on this algorithm.

In the second pass we render the image using the photon maps and we begin by performing a traditional ray tracing visibility determination. We start at the eye and fire out rays into the scene to determine what can we see. For specular paths and direct lightning, we use accurate calculations. For indirect illuminated portions of the scene, we use the photon map to approximate the colour that

arrives at those areas to speed up the rendering process. We finding the i nearest photons at each intersection and compute the BRDF for every photon. The diffuse and specular Phong components are added together and divided by the circle area that contains all the photons and has the smallest radius so that the overall intensity contribution of widely spreaded photons is small. BRDFs are functions that use the incoming and outgoing ray angles and have as a result the colour of a material. We use a simplified version of the reflected radiance at a surface point x in direction w: each BRDF contribution is multiplied by the intensity of the photon arriving at a surface divided by the area of the narrowest circle containing all the photons and then sum all the BRDF contributions together to determine the colour of the surface.

The normal photon map and the caustic photon map are kept and computed separately. For the caustic map, we only look at the specular reflections and do not consider other types of photon bounces to allow for better performance (in terms of speed and accuracy) when the caustics are created. The knn search and BRDF are used to compute the resulting colour of the caustic photons. The result is again multiplied by the cone filter (weighting neighbours based on their distance from the search point, to reduce the blur in the pixels representing caustics).

The brightness of caustics is generally high compared to the rest of the pixels. Leaving this unresolved results in a dark and undetailed image since the framebuffer scales down all the colour contributions setting the brightest caustic to 1. We can solve this by setting the caustic photon intensity to 1/total number of caustic photons fired into the scene.

The caustic photons are fired randomly just like the normal photons but the photon tracing is performed only for specular bounces (when `depthS` is set to true) and when the caustic photon hits a diffuse surface it stops.

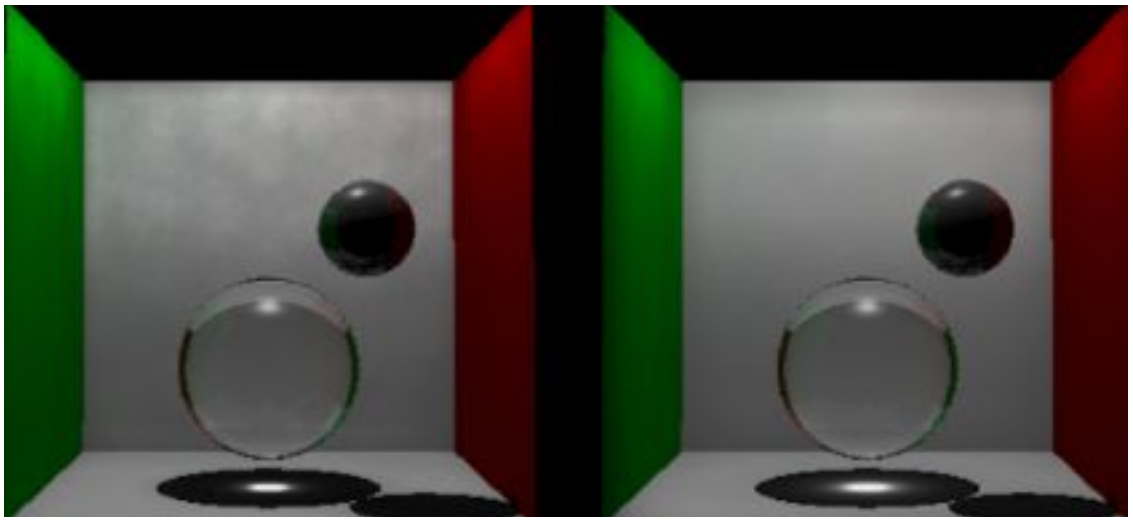


Fig 2.1 Both the above images are rendered at a low resolution but the important aspect that should be noticed is the impact of the number of photons used and how many k nearest neighbours we have. In the left image, we use 100000 photons, 200000 caustic photons, 150 knn and 15 caustic knn, while in the right image we use 300000 photons, 600000 caustic photons, 500 knn and 80 caustic knn. Big differences are seen in the walls and shadow colours.

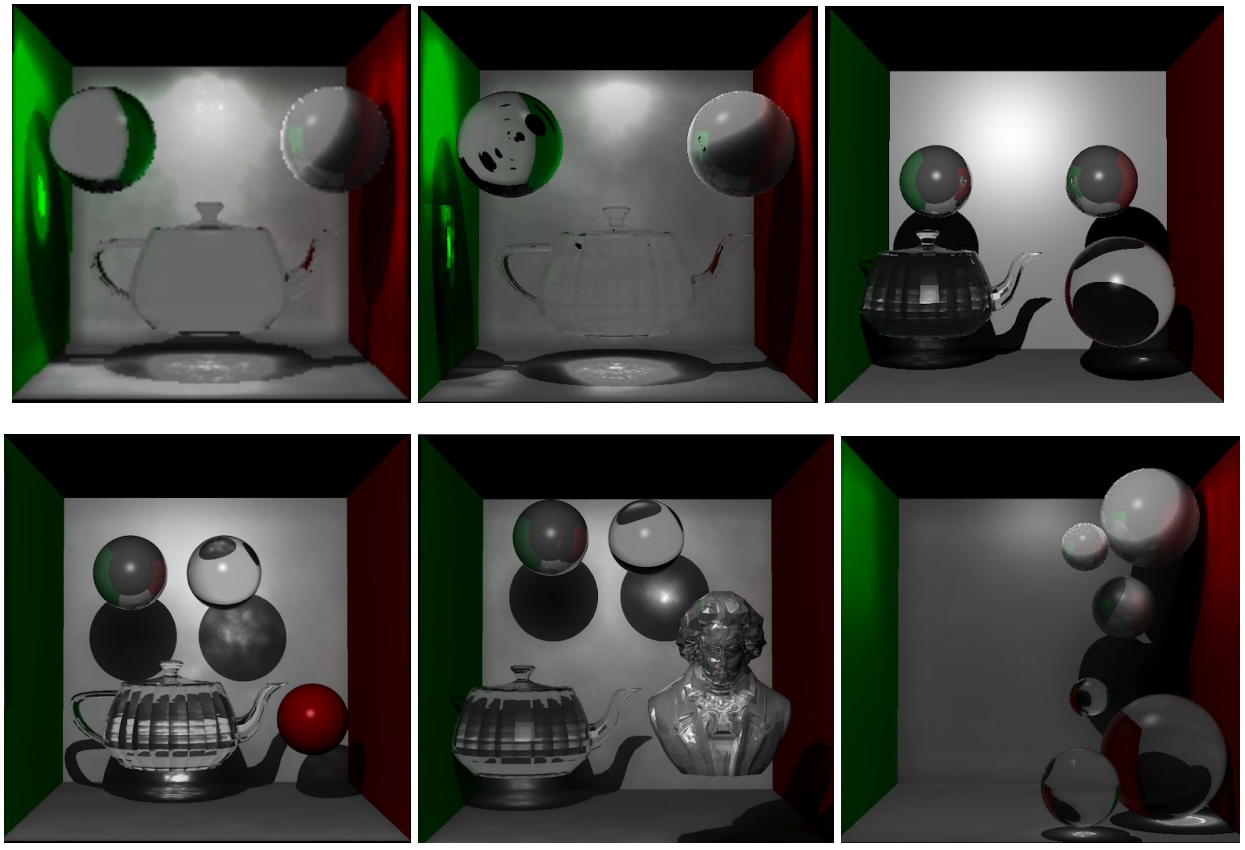
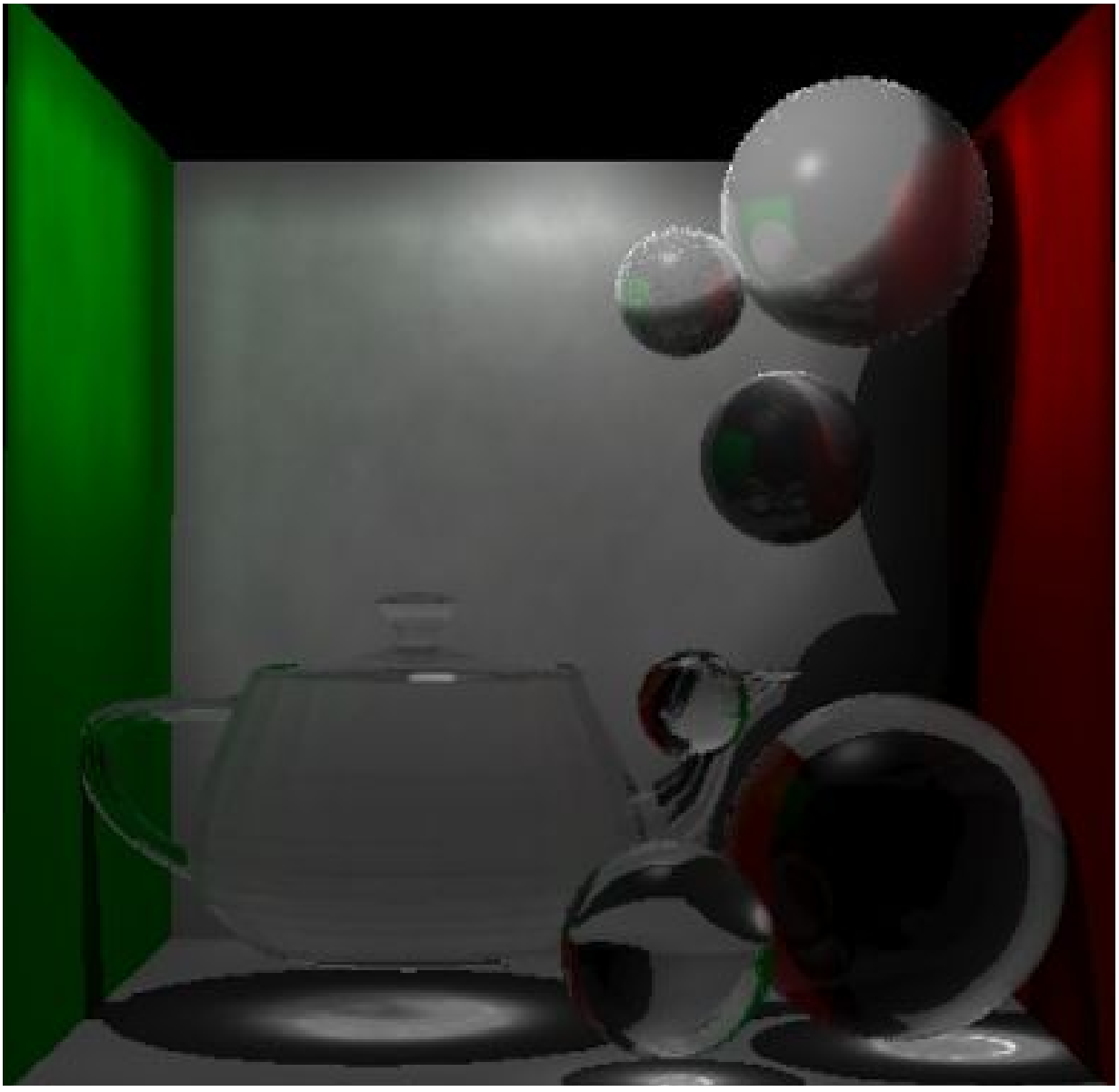


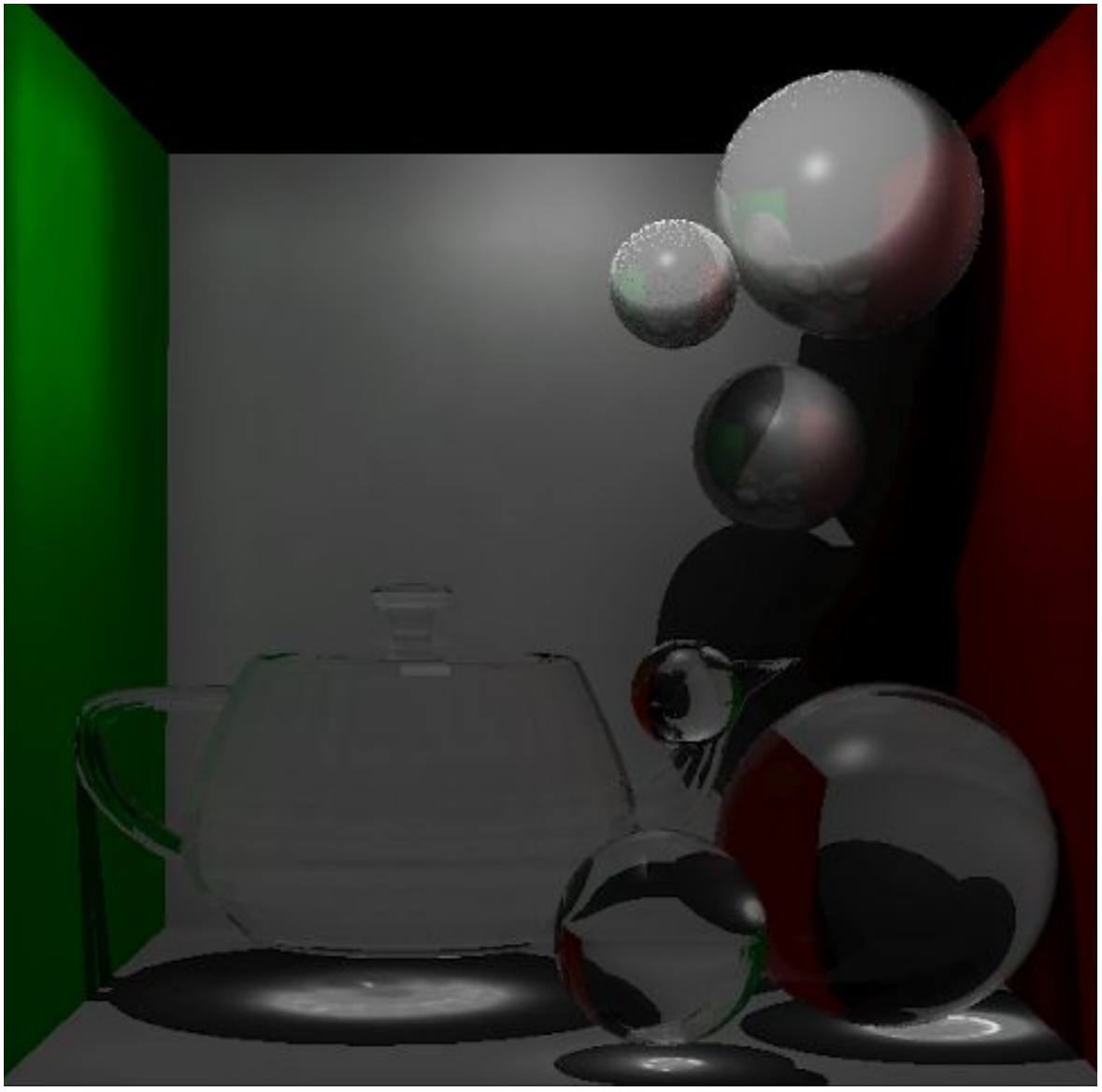
Fig 2.2 Shows my experimentation with different caustic intensities and the overall evolution of caustics.

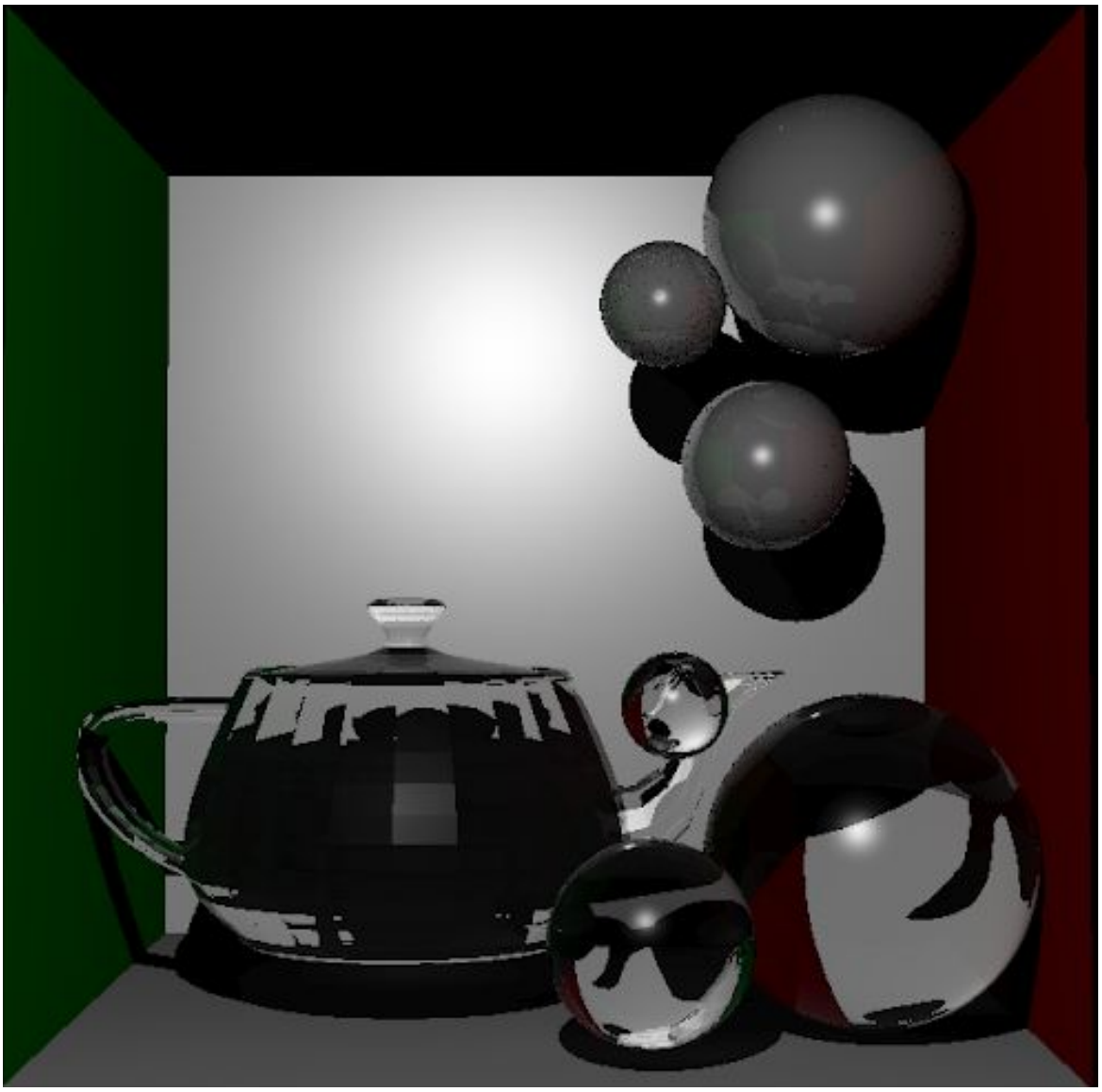
DISCUSSION

The next 3 images represent the same scene with different lighting. The caustics are far from being realistic and the failure of creating a real Cornell box with a light inside the room makes taking advantage of the complex geometry of the teapot very challenging. As further improvements, I would consider fixing the lightning model, implement anti-aliasing for the reflective silver spheres, and adding textures.

I would consider the first images being the best one since it emphasises the reflection, transparency and caustics a bit better than the other two while also having a better balance between the colour of the normal pixels and the ones representing caustics (it is less darker and faded). The major downside are the lower resolution (512x512 compared to 712x712 - my computer runs extremely slow even for these resolutions, probably pointing out the fact that the code lacks efficiency) and the unevenness of the walls, therefore the knn search did not use a large enough i value to compute a more consistent colour.







REFERENCES

[1]
https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

[2] Henrik Wann Jensen: "Global Illumination using Photon Maps". In *"Rendering Techniques '96"*. Eds. X. Pueyo and P. Schröder. Springer-Verlag, pp. 21-30, 1996

[3] https://github.com/jlblancoc/nanoflann

[4] Arvo, James and David Kirk: "Particle Transport and Image Synthesis".ComputerGraphics24(4), pp. 53-66, 1990.