# Advanced Computer Graphics Report

Zachary Upstone

November 2023

## 1 Introduction

In this report I will briefly summarise how I implemented a ray tracer and photon mapping (Jensen, 1996) in C++ to produce images such as Appendix B(Figure 20). I will firstly look at how I implemented feature rich rendering features including reflection, refraction, quadratic surfaces, depth of field and bounding objects. I will them finish off by talking about my implementation of photon mapping. (Jensen, 1996).

## 2 Feature-Rich Rendering

Feature rich rendering allowed me to add more advanced techniques to my ray-tracer allowing for increased efficiency, but more importantly a dramatic increase in realism. Before feature rich rendering I was producing images such as Appendix A(Figure 1).

### 2.1 Reflection

Before I implemented any of the feature rich rendering step I was shooting rays from the camera and collecting the colour they hit. To add reflection to this existing code required a new material class, this is called Global Material, it takes weights for both reflection and refraction. When the ray collides with a material a method is run in scene where the rays are traced called compute_once, this takes the ray and works out its reflection using the equation below:

$$R = E – 2(N.E)N \tag{1}$$

Where $N$ is the normal of the object where the hit landed and $E$ is the incoming rays direction. With this new ray I can run the trace function again creating a recursive loop when it unwinds I add all the contributions of the reflection to the colour. Eventually the recursion gets down far enough it will make very minor contribution to the colour so I set a limit to break the recursion to help with compute time. This produces images like Appendix A(Figure 2), a lot of noise is present this is due to self shadowing. When the new ray is made its start point uses the hit point but this means it immediately hit the material again causing shadows, to fix this I make the ray start a little along its path away from the hit point. To compute this I use $\epsilon$ which is a small constant and multiply it by the rays path then add the hit points location. I end up after this with an image like this Appendix A(Figure 3) as you can see it has clear reflections of the room and the above Phong sphere.

## 2.2 Refraction

To implement refraction was very similar to reflection, where the ray hit I calculated a new refract ray based on the materials refractive index using these equations:

$$T = \frac{I}{\eta} - (cos\theta t - \frac{cos\theta i}{\eta})N \tag{2}$$

$$cos\theta t = \sqrt{1 - \frac{(1 - cos^2\theta i)}{\eta^2}} \tag{3}$$

$$cos\theta i = N.I \tag{4}$$

Where $N$ is the normal of the object where the hit landed ,$I$ is the incoming rays direction and $\eta$ is the refractive index of the material. Again to avoid self shadows you add a small portion of the direction to the start location, in this case I was also careful to note if you were entering of leaving an object as when leaving you inverse the refraction instead, this is done by using $1/\eta$ instead. This is shown in Appendix A(Figure 4). I also implemented Fresnel's equations here, towards the edges of a refractive material it becomes more reflective shown by this graph Appendix C(Figure 21). to implement this when a refraction occurs you also create a reflection ray. When you add the colour each ray produces you also multiply by the respective Fresnel variable (kt for transmission, kr for reflection) which can be worked out using the below equations and sum to 1:

$$kr = \frac{(\frac{\eta cos\theta_i - cos\theta_t}{\eta cos\theta_i + cos\theta_t})^2 + (\frac{cos\theta_i - \eta cos\theta_t}{cos\theta i + \eta cos\theta_t})^2}{2} \tag{5}$$

$$kt = 1 - kr \tag{6}$$

Where $\eta$ is the refractive index, $\theta_i$ is the angle of incidence and $\theta_t$ is the angle of transmission. A final picture of a refractive object with Fresnel implemented is at Appendix A(Figure 5).

## 2.3 Quadratic Surfaces

To implement quadratic surfaces you start with an input equation like the one below:

$$f(x, y, z) = Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0 \tag{7}$$

Where $A, B, C, D, E, F, G, H, I, J$ are constants inputted to determine the shape of the resulting quadratic. I then work out where and if the rays will hit it, I followed the equations found in Owen (2012), firstly you work out a quadratic equation which when solved will give us the intercept(s), there are between 0-2 intercepts a ray can make.

While solving it is important to note:

- I check the divisor of the quadratic formula (in my code a variable named Aq) to make sure I am not dividing by 0.
- I also check to see if the discriminant is negative as if it there will be no real solutions.
- I also only compute the second hit if the first hits t value is not NULL as if it isn't it won't be visible in the scene.

Finally to compute the normal's I take the partial derivatives of $F$ with respect to $x, y, z$ individually and then combine the results. All this allows us to create shapes like a cylinder Appendix A(Figure 6), a cone Appendix A(Figure 7) and any other shape that can be constructed from quadratic equations. The CSG class can be seen in Appendix A(Figure 8), the spheres of which is its constructed can be seen in Appendix A(Figure 9).

To transform the quadratic surfaces I follow the equations below:

$$Q' = T^T Q T \tag{8}$$

Where $T$ is the transform matrix and $Q$ is the initial quadratic. This equation can be found in Cameron and Yang (2023).

## 2.4 Additional features

I chose to do two additional features, to start with I implemented bounding objects for polymesh objects but this was quick so I also wanted to implement another, I chose depth of field as it can later be applied to photon mapping and allows a large increase in realism, where bounding objects only improve the codes efficiency.

### 2.4.1 Depth of Field

To implement Depth of Field I used the thin lens model of which a diagram can seen at Appendix C(Figure 22) (Yang and Cameron, 2023). This works by instead of firing a single ray I pretend to have a lens where the preexisting ray would be the centre ray. I then fire an amount of rays around this that slightly deviate from the path when they meet the lens the rays that are deviated correct there course so as to collide with the centre ray at the focal length. In my code I simply treat the camera as the lens moving the start location of the ray allowing me to avoid complicated calculations correcting the direction. All of this means that if an object is at the focal point the rays won't spread out from the centre ray and if it isn't they will spread out, by averaging the colours returned by the rays this will cause out of focus objects to blur as they pick up surrounding colours from each sub-ray creating a focus effect seen in Appendix A(Figure 12). This can also be extended to photon mapping as seen in Appendix B(Figure 13).

#### 2.4.2 Bounding Objects

I also implemented bounding objects to help speed up ray tracing especially when a polymesh object is far away. They work by surrounding a polymesh with a simpler object (I used a sphere) then before checking if rays hit the teapot checking they hit the sphere. Since checking the sphere is a lot faster than checking every triangle in the polymesh it speeds up ray tracing significantly. In Appendix A(Figure 10) you can see my teapot and in Appendix A(Figure 11) you can see the bounding sphere I used, it is very large as it takes the Max x,y,z and Min x,y,z of the polymesh and finds the difference for its diameter, this is to ensure no rays that could hit are missed. Without bounding 553650 rays are checked to see if they hit the teapot which contains 6320 triangles whereas with the bounding turned on only 250775 rays are checked against the teapot polymesh (these numbers were both collected from the referred scene), this obviously varies with how far away the teapot is but either way you can see a significant difference.

## 3 Photon Mapping

Finally I move onto photon mapping(Jensen, 1996), this involves completely overhauling how the raytracer works it can be broken broadly into two steps (1)photon mapping and (2) rendering.

### 3.1 Photon Mapping

In my implementation I have created a photon class which is an extension of a normal ray so it can also include a type, a colour and a intensity. I also implemented a point light class so that I can fire the photons from it. They are fired randomly from this point light.

#### 3.1.1 KD tree

All photons are stored within a balanced KD tree structure, the implementation I used can be found here Holt (2017). This is compact and efficient and allows us to find M photons in a tree with N photons in $O(M \cdot \log 2(N))$ time (Jensen, 1996). This is very important when dealing with as many photons as I am. I created 2 separate KD trees, one for the global photon map and one for the caustic map.

### 3.1.2 Global Photon Map

I start by first firing a set amount of photons to fill my global map, these photons start with the light colour and "normal" type. For most of my images I used 100,00 - 1,000,000 photons, the directions are decided by random uniform distributions. The photons act similar to Monte-Carlo rays, they reflect and refract, when they do so they pick up the colour of the material multiplied by the lights colour, when they hit they are stored in the KD tree along with their locations and the hit materials colour.(It should also be noted that I made all materials have a small chance of reflecting photons to replicated realistic colour bleeding effects as seen in Appendix B(Figure 14). How the photon map is visualised can be seen by making the search area really small as seen in Appendix B(Figure 15).

### 3.1.3 Caustic Photon Map

The second part of photon mapping is including the caustic photons, these are created in a similar way to global photon with one key difference, they are only stored if they hit a caustic. To make sure I collect enough of these I keep firing these until a quota is met in the tree, this would obviously vary depending on the amount of refractive objects in the scene. caustic photons are given a special type and only stored when they make contact with a diffuse material.

## 3.2 Rendering

Next I render the scene using the photon maps I have made, this works similar to previous raytracing. Rays are traced until they hit an object, if they object is refractive or reflective another ray is produced. Once it reaches the final hit these rays unwind picking up colours from where they hit. To compute colour I search nearby photons to the hit in a radius $r$ and using the equation below can work out the radiance of a hit to display as a pixel:

$$L_s(x, \varPsi_r) = L_e(x, \Psi_r) + \int_{\Omega} f_r(x, \Psi_i, \Psi_r) L_i(x, \Psi_i) cos\theta_i d\omega_i \tag{9}$$

Equation 9 is found in Jensen (1996). The first term $L_e$ goes to 0 if it is not a light source, the second term is the main part it represents the integral of all incoming directions of the product of the BRDF for each direction, the incoming radiance of each direction and the cosine of the incoming direction all with respect to direction. It also should be noted in my implementation I have placed a flag under scene.caustic so that the code doesn't get stuck trying to fill it quota forever if there isn't a caustic in the scene.

### 3.2.1 Ambient Light

Photon mapping is a way of replacing the ambient term in our ray tracing module, To render this ambient term I follow the process above of searching a sphere near the hit in the global photon tree, I also scale down this contribution after to make sure it creates the right final effect and doesn't out weigh the other colour contributions. Our scene with just this lighting looks like Appendix B(Figure 16).

### 3.2.2 Specular and Diffuse Lighting

To implant specular and Diffuse lighting in our model I rely on the standard Monte-Carlo ray tracing method that I already implemented and use compute per light. This is still effective and once added can be seen in Appendix B(Figure 17).

### 3.2.3 Caustic Light

Finally, I have the opportunity now to include caustic lighting, I follow Jensen (1996) approach as mentioned above. To render the caustic map follows a similar method to the ambient light, but for each hit I instead search the caustic map, the other key difference is the search radius is a lot smaller in my case I used 1/5 of the size as caustics tend to come in dense pockets, after implementations I get lighting effects near refractive objects as shown in a close up Appendix B(Figure 18), overall our photon image now looks like Appendix B(Figure 19).

### 3.2.4 Cone Filter

Colour boundaries suffer from a lot of colour incorrect colour bleeding, to fix this Jensen (1996) proposes to use a cone filter, I multiple our radiance prediction to take into account distance using the equation below:

$$w_{pc} = 1 - \frac{d_p}{kr} \tag{10}$$

Equation 10 is found in Jensen (2000), where, $d_p$ is the distance between the ray tracer hit and the photon hit. $r$ is the max radius of the search and $k$ is some constant(in my case 1.1). Then at the end the calculated luminosity is also divided by this factor leading the calculation for radiance to be that below:

$$Lr(x, \overrightarrow{\omega_p}) \approx \frac{\sum_{p=1}^{N} f_r(x, \overrightarrow{\omega_p}, \overrightarrow{\omega})\Delta\phi_p(x, \overrightarrow{\omega_p}) w_{pc}}{(1 - \frac{2}{3k})\pi r^2} \tag{11}$$

Equation 11 is found in Jensen (2000).

## 4 Conclusion

To conclude I have discussed how I implemented both feature rich rendering and photon mapping to create more realistic rendered images, I decided to implement bounding spheres and depth of field as my additional features.

## References

Cameron, K. and Yang, Y., 2023. Week 5 - cm30075: Advanced graphics short note raytracing quadratic surfaces. [Online].
Available from: CM30075: Advanced Graphics Short Note Raytracing Quadratic Surfaces.

Holt, X., 2017. Kd::tree. [Online].
Available from: https://github.com/xavierholt/kd.

Jensen, H.W., 1996. Global illumination using photon maps. [Online].
Available from: http://graphics.ucsd.edu/ henrik/papers/photon$_map/$.

Jensen, H.W., 2000. A practical guide to global illumination using photon maps. [Online].
Available from: https://graphics.stanford.edu/courses/cs348b-00/course8.pdf.

Owen, G.S., 2012. Ray - quadric intersection. [Online].
Available from: http://www.bmsc.washington.edu/people/merritt/graphics/quadrics.html.

Yang, Y. and Cameron, K., 2023. Week 5 - sampling. [Online].
Available from: Slides for week 5 sampling.

# Appendices

## A Raytracing Photo Appendix

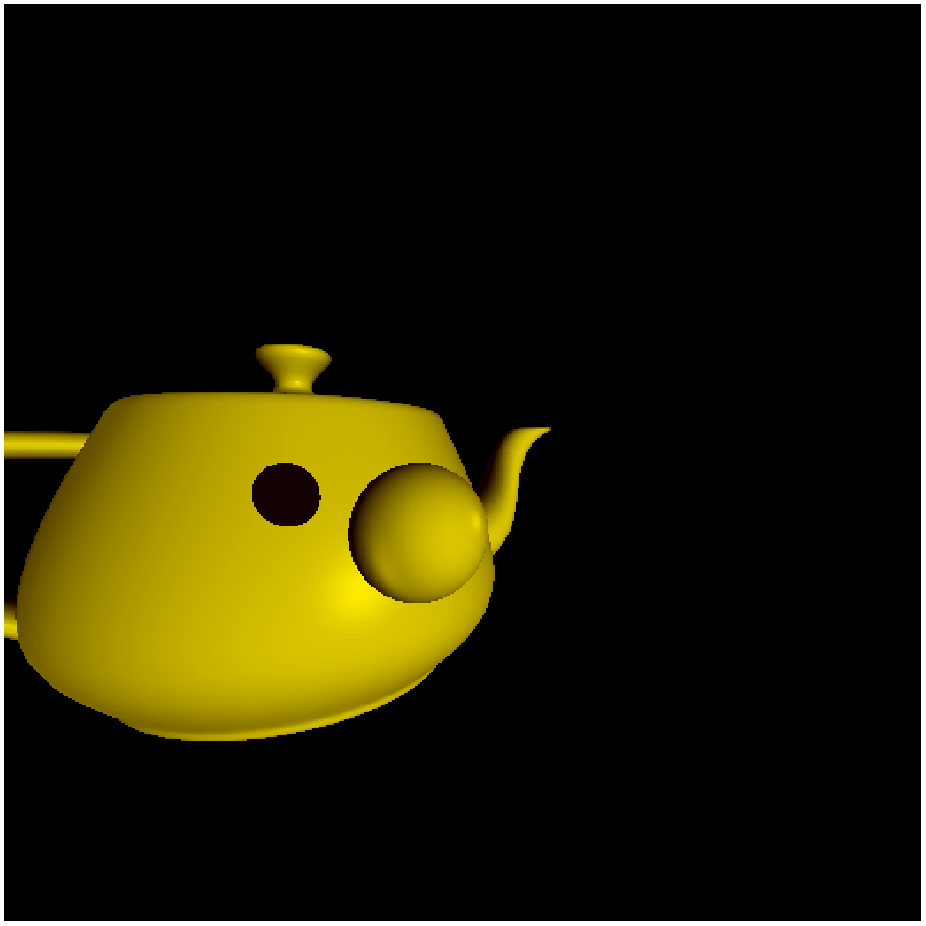

Figure 1: Basic Ray tracer Phong image

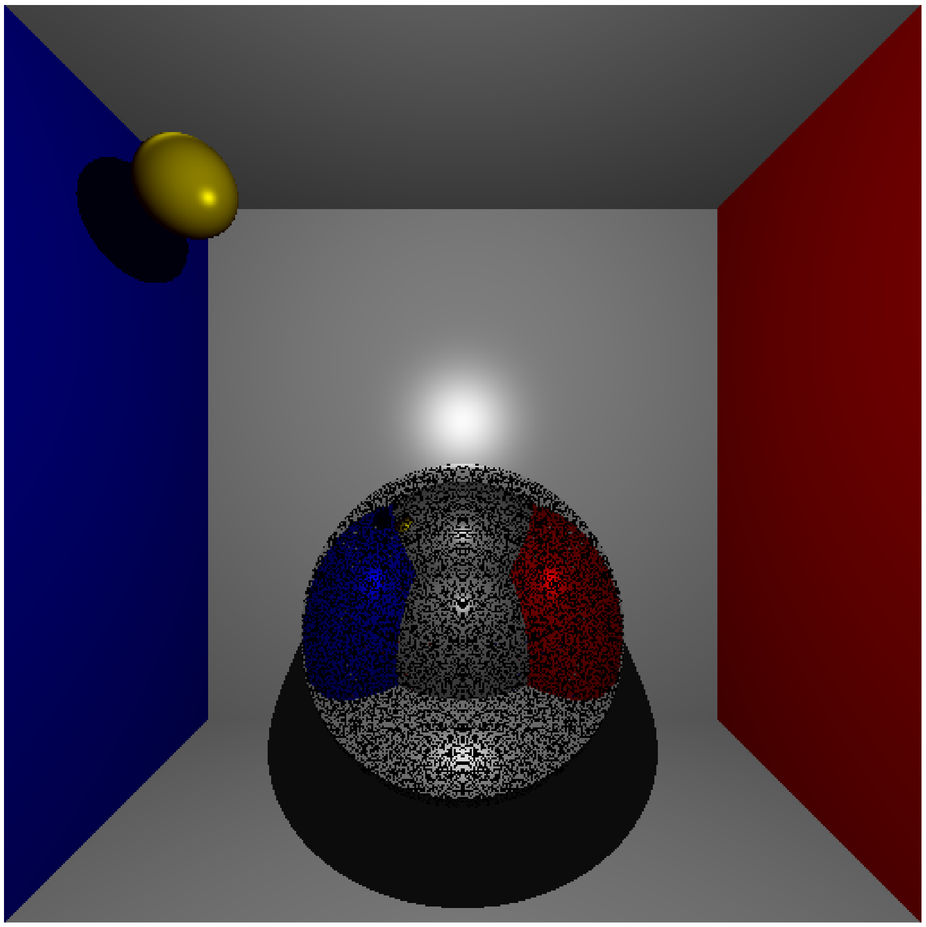

Figure 2: Raytracer reflection with self shadowing

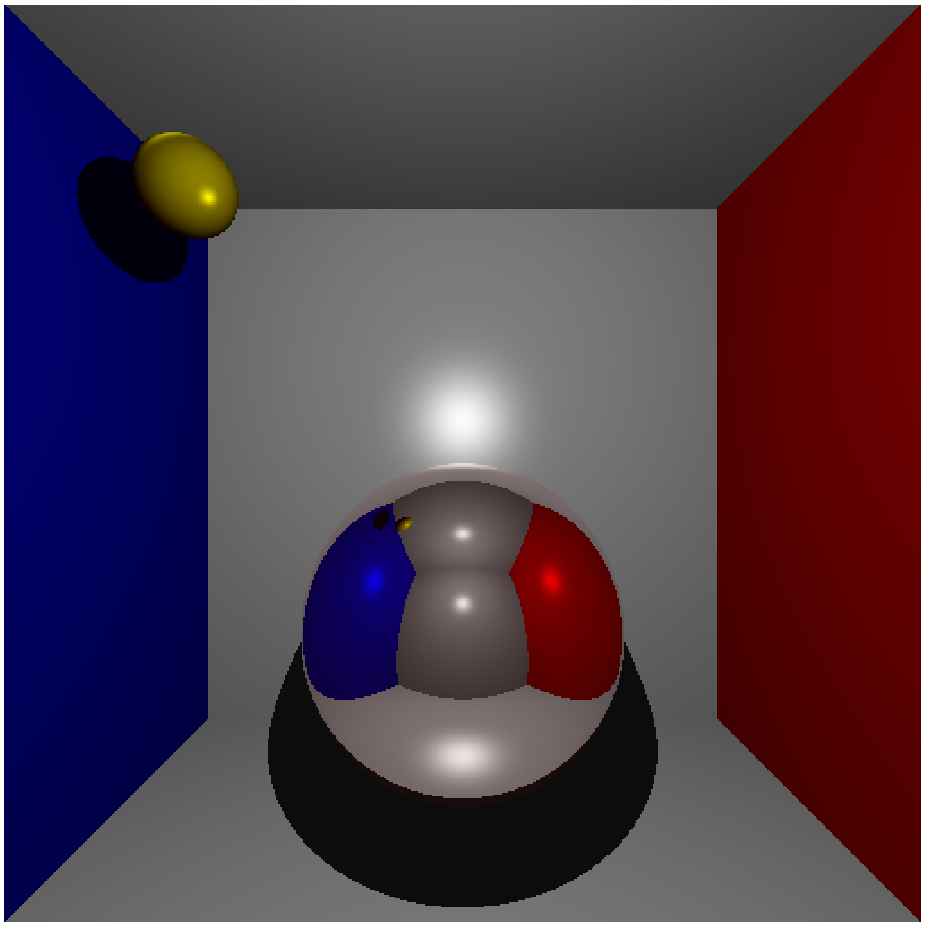

Figure 3: Raytracer reflection without self shadowing

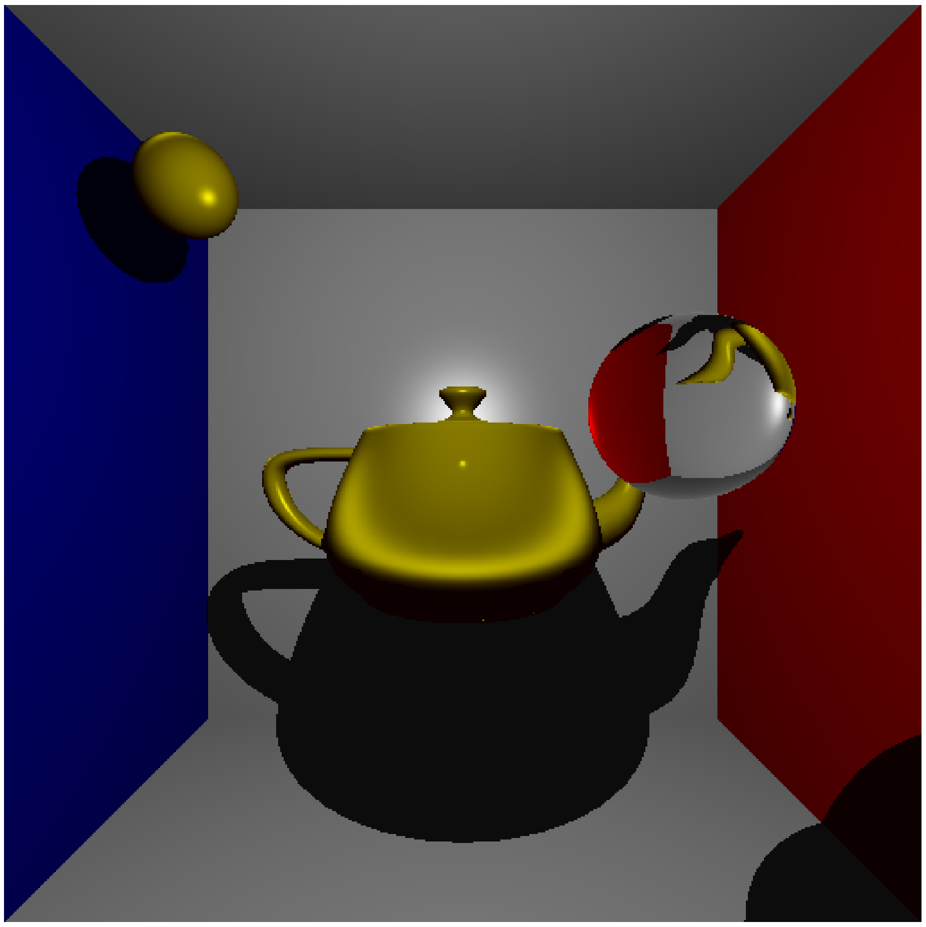

Figure 4: Scene with refractive sphere and no Fresnel

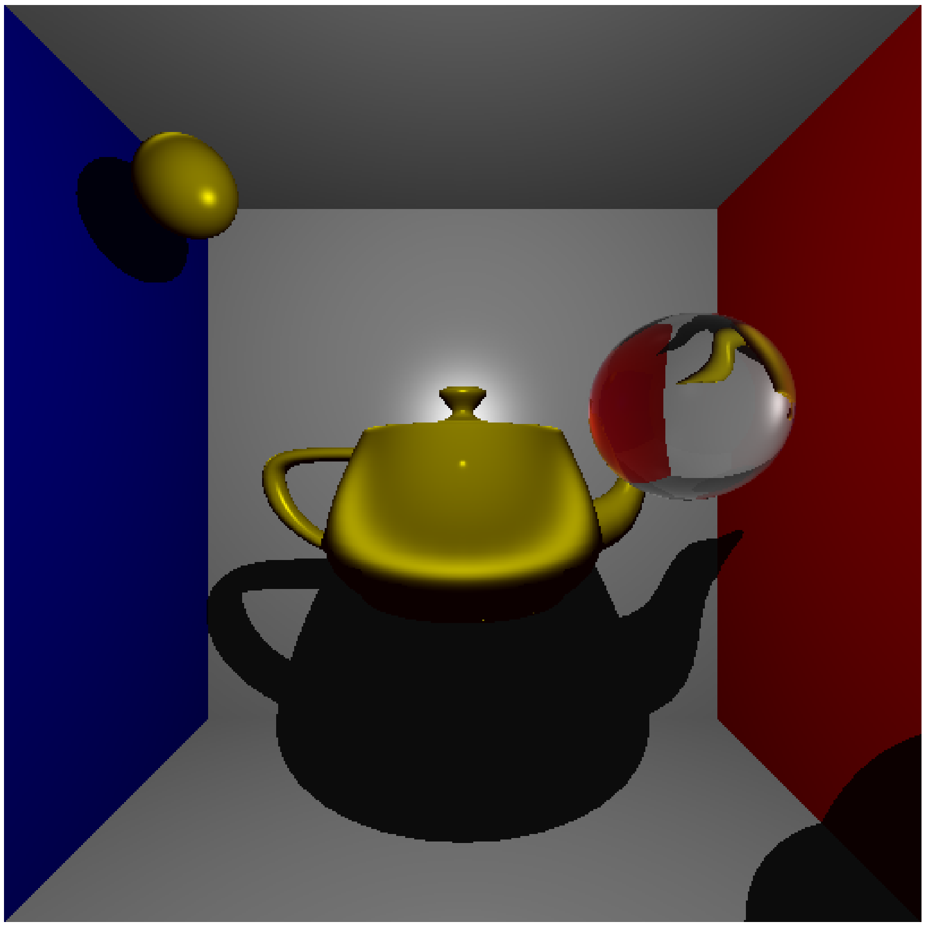

Figure 5: Scene with refractive sphere and Fresnel

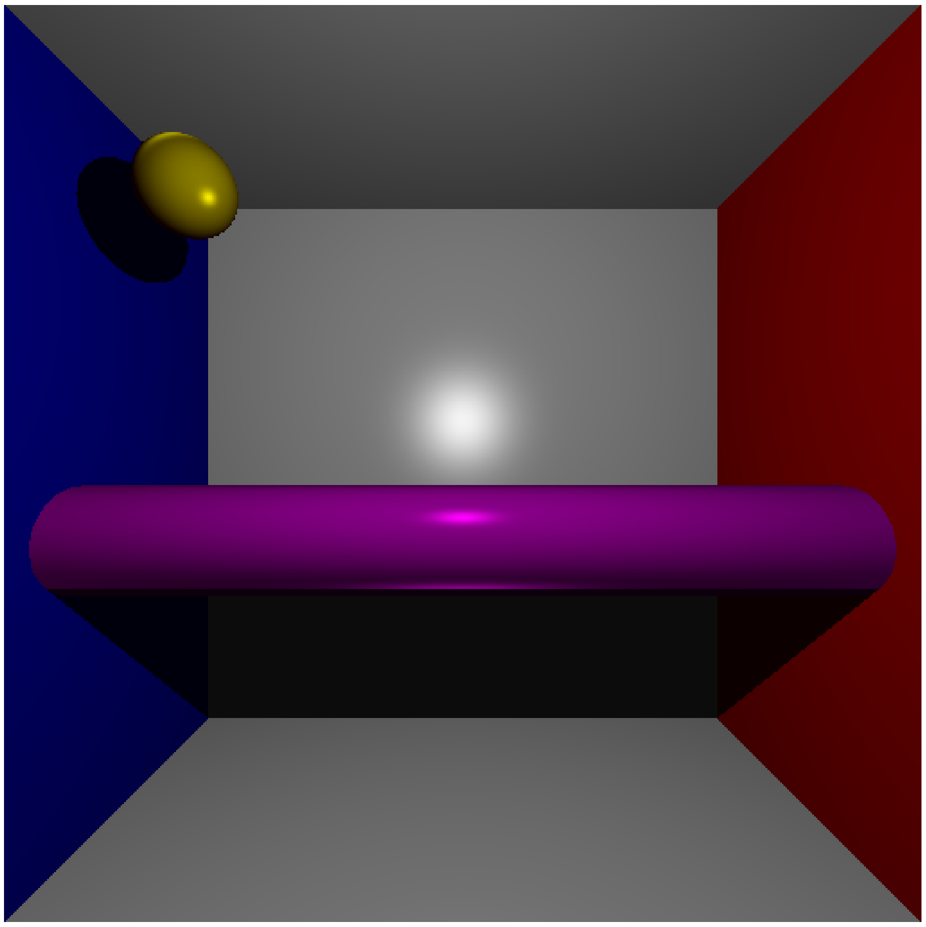

Figure 6: Cylinder quadratic object

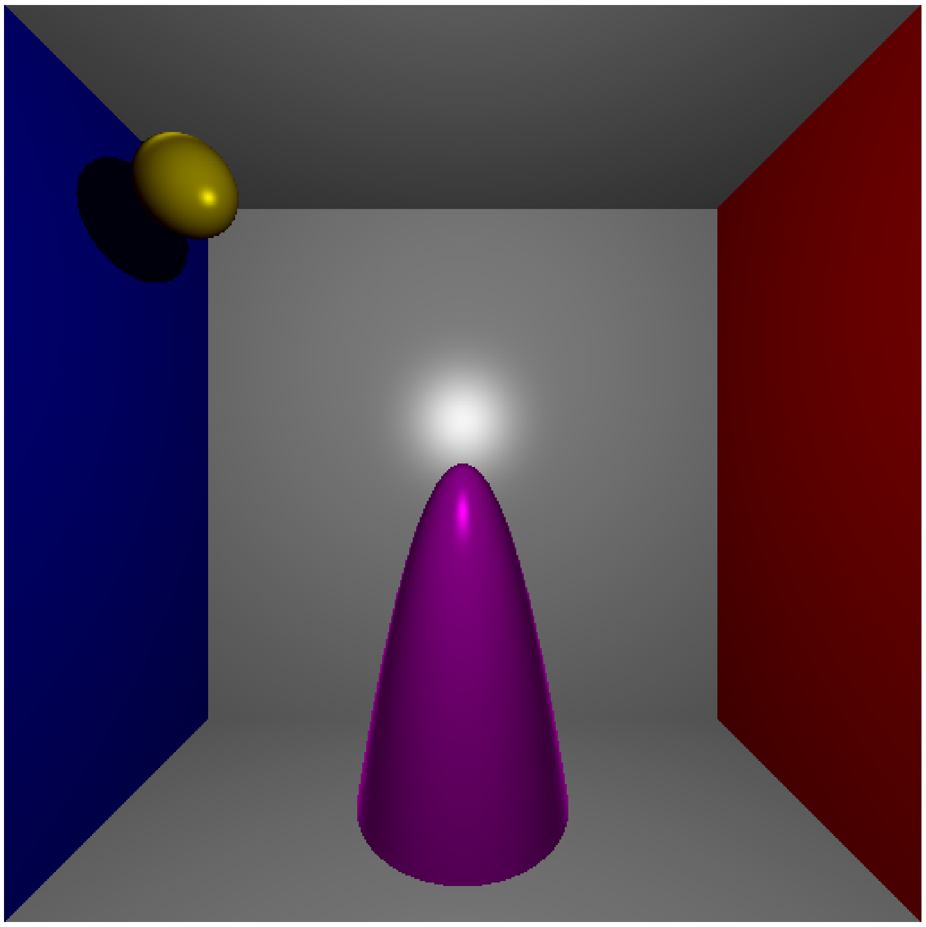
Figure 7: Cone quadratic object

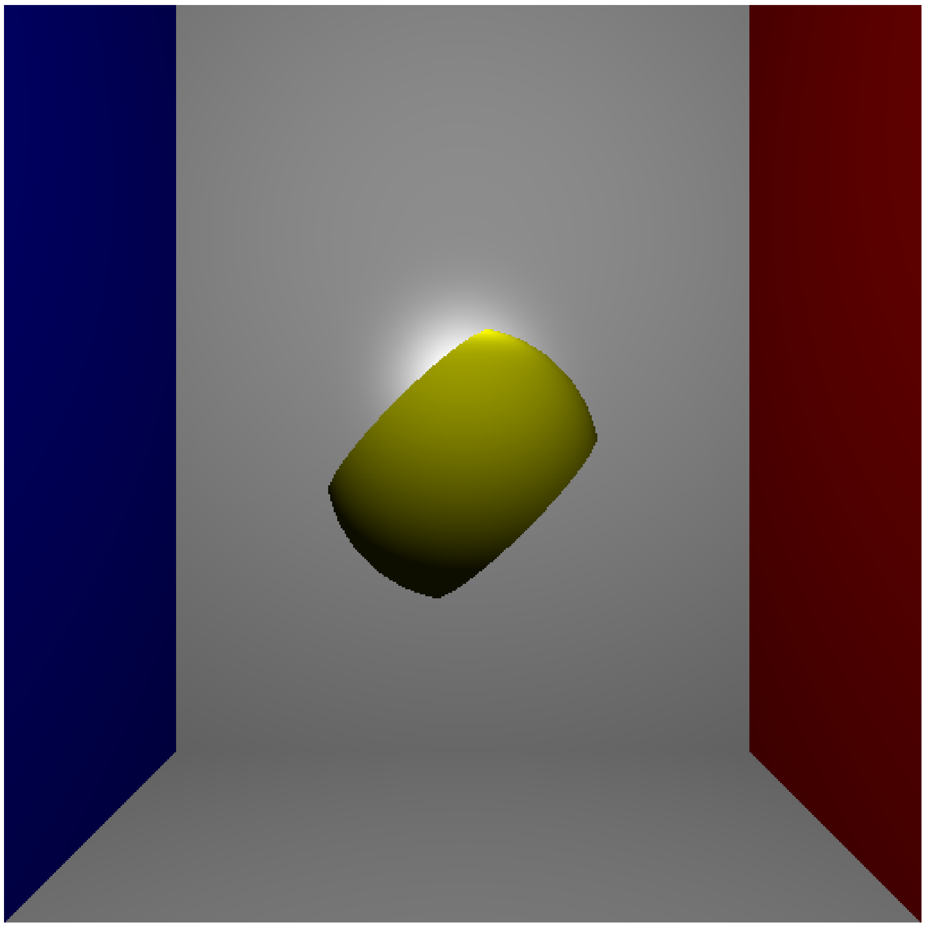

Figure 8: CSG of two spheres taken out of a third

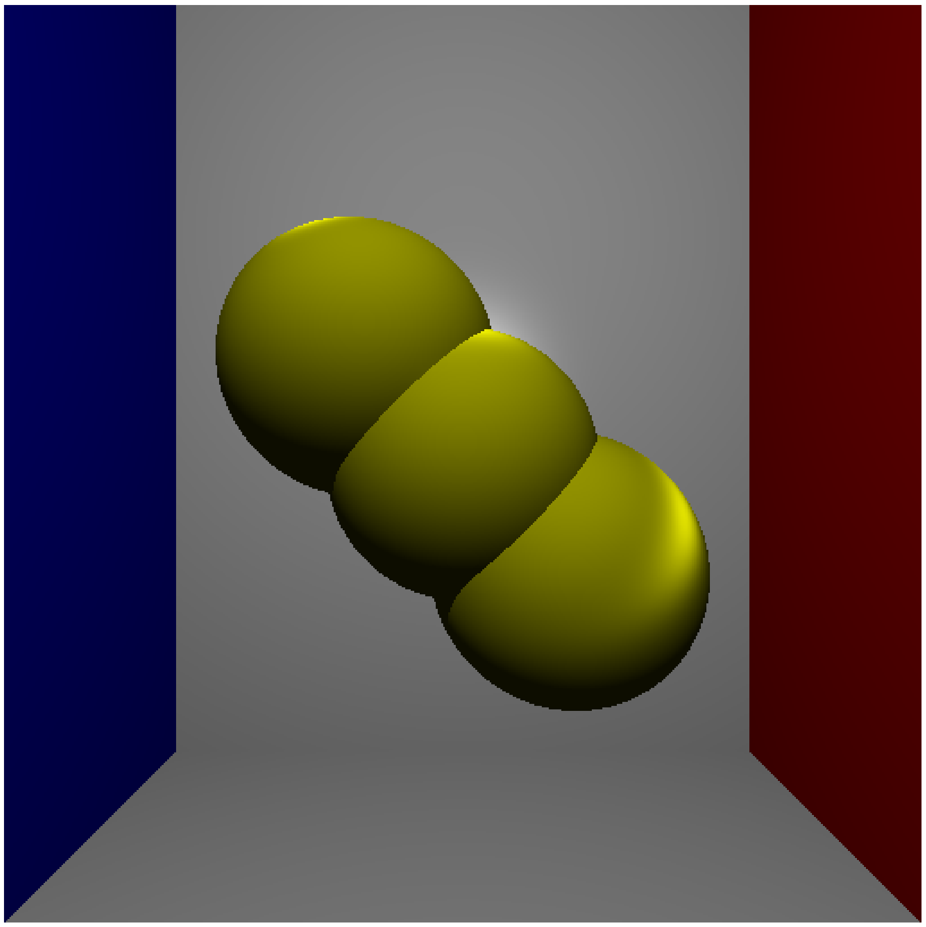

Figure 9: Three spheres from the CSG rendered to make the CSG more clear.

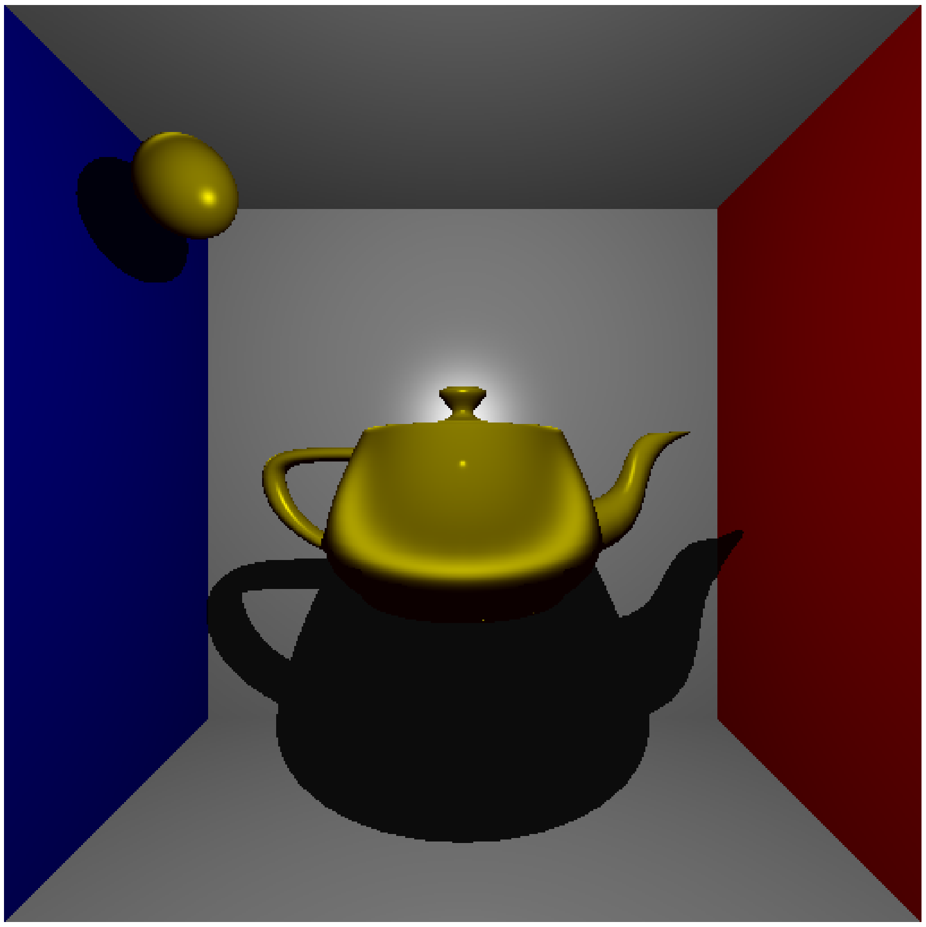

Figure 10: Teapot without bounding sphere shown

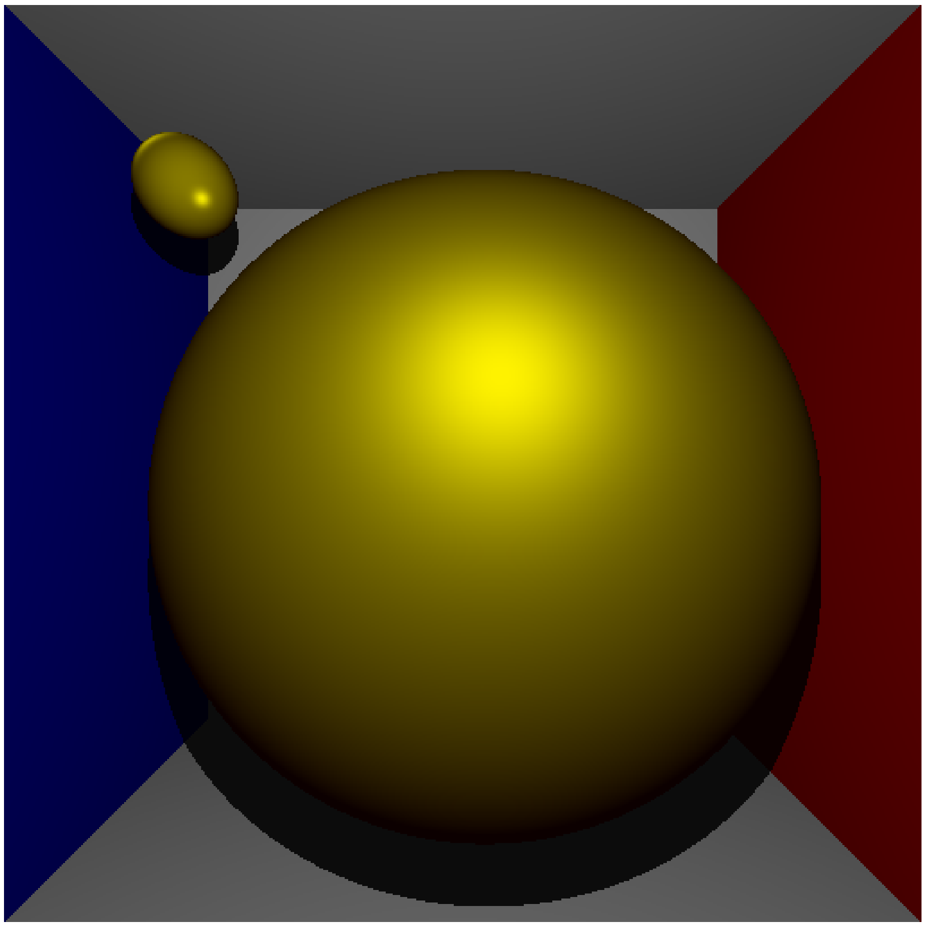

Figure 11: Bounding sphere for teapot polymesh

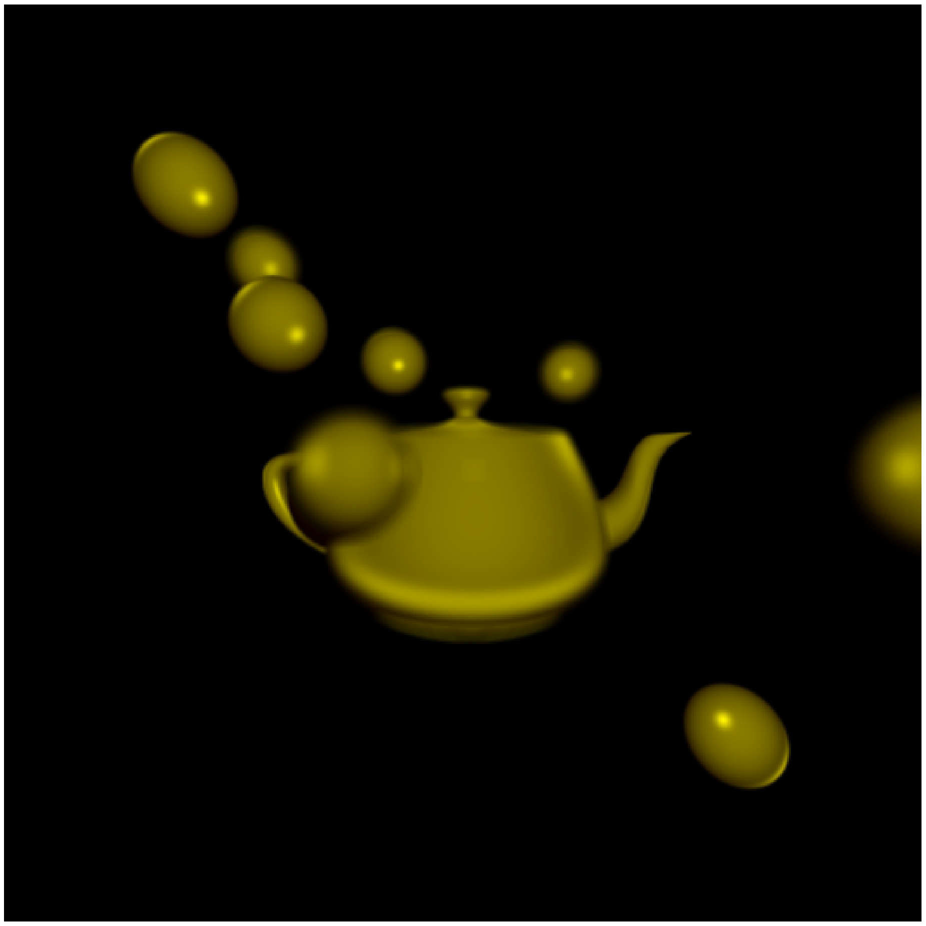

Figure 12: Raytracer camera focus on teapot

## B Photon Mapping Photon Appendix

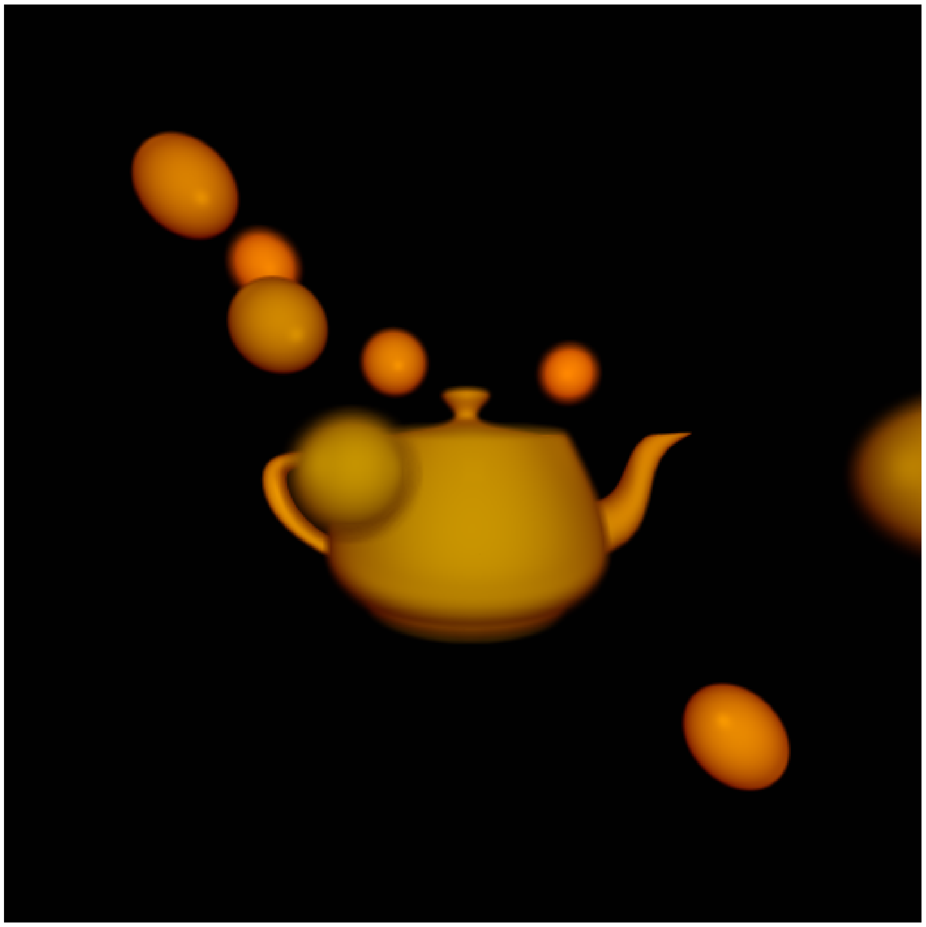

Figure 13: Photon mapping camera focus on teapot, the difference between this and the is mainly due to tweaking of parameters, the ambient colour of this has been given more weight in my photon mapping implementation and since the ambient colour is red it gains a more red hue.

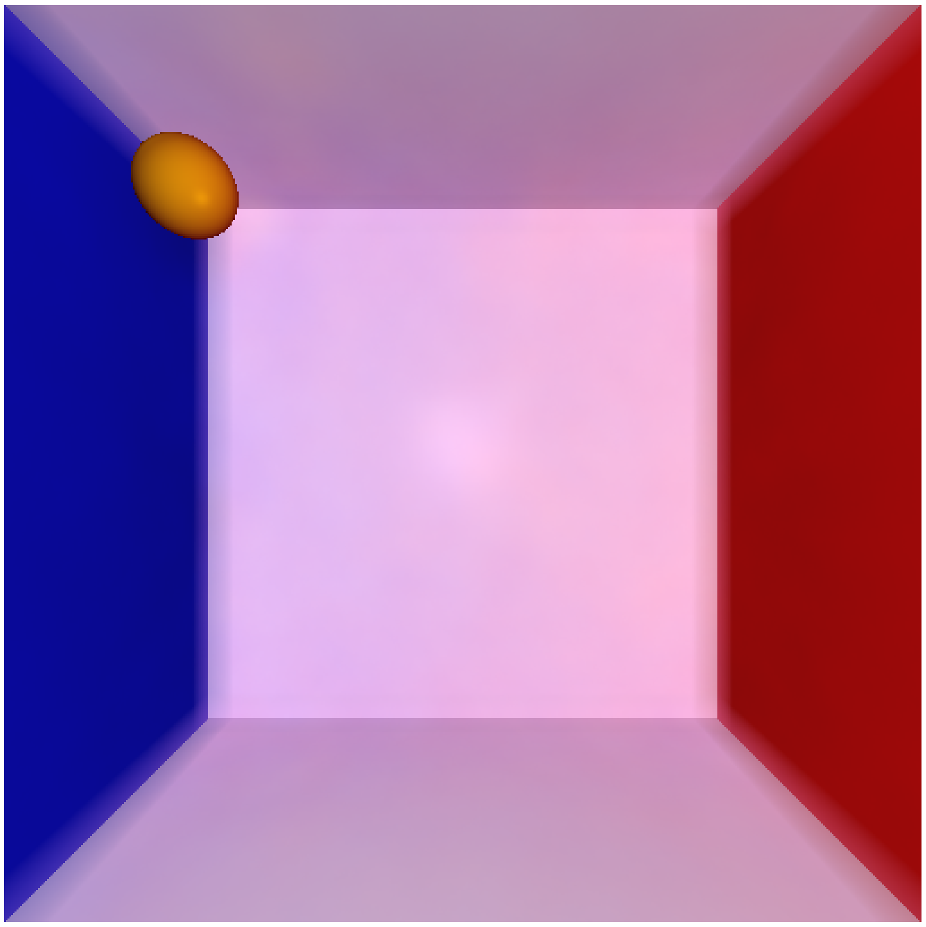

Figure 14: Scene where I scaled up minor bleeding effect caused by slight chance for any material to reflect, as you can see the left half is blueish and right is reddish, I scaled the chance up to make it more obvious.

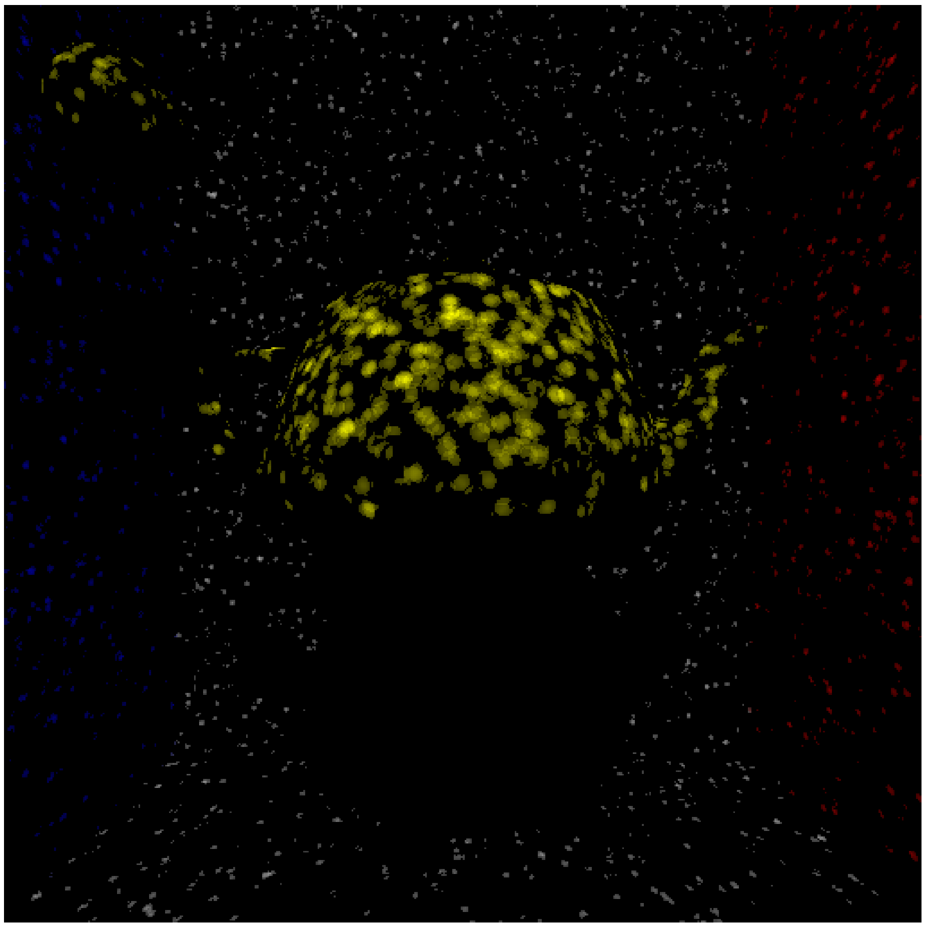

Figure 15: Photon map with 10,000 global photons visualised, it should be noted that a lot of photons miss entirely as the light is outside of the box hence the sparse look for such a high number of photons fired.

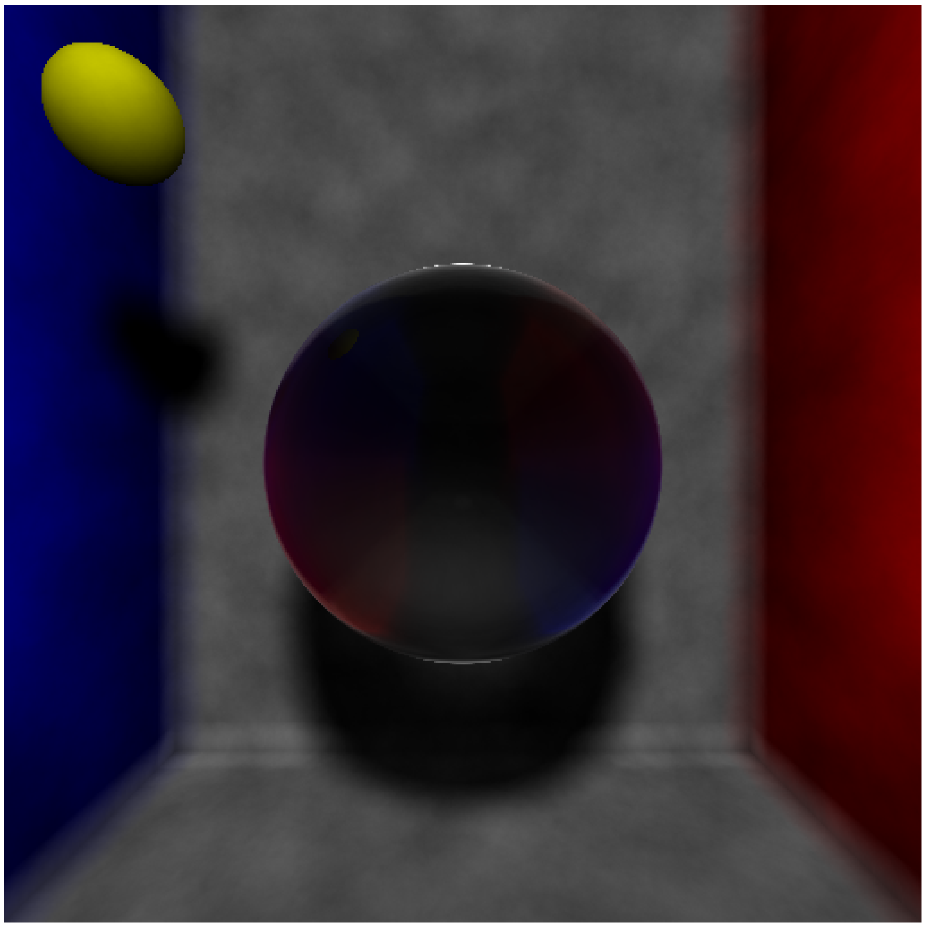

Figure 16: Scene after the addition of global lighting

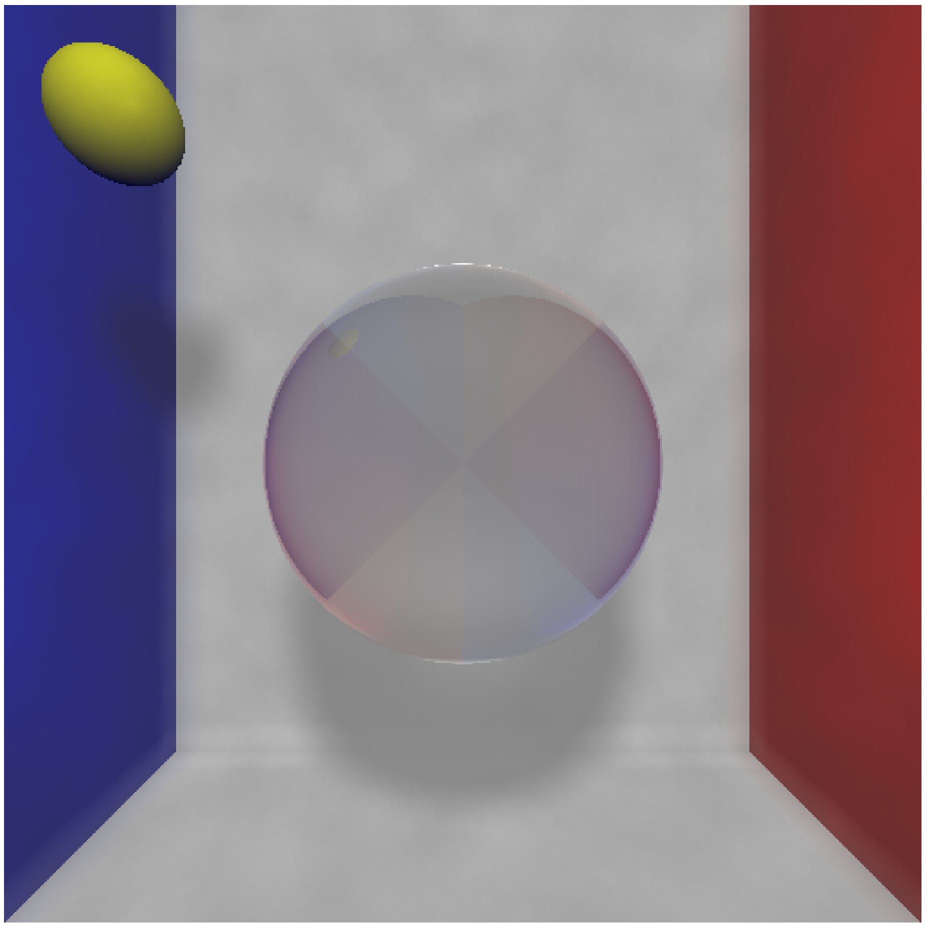

Figure 17: Scene after the addition of specular lighting

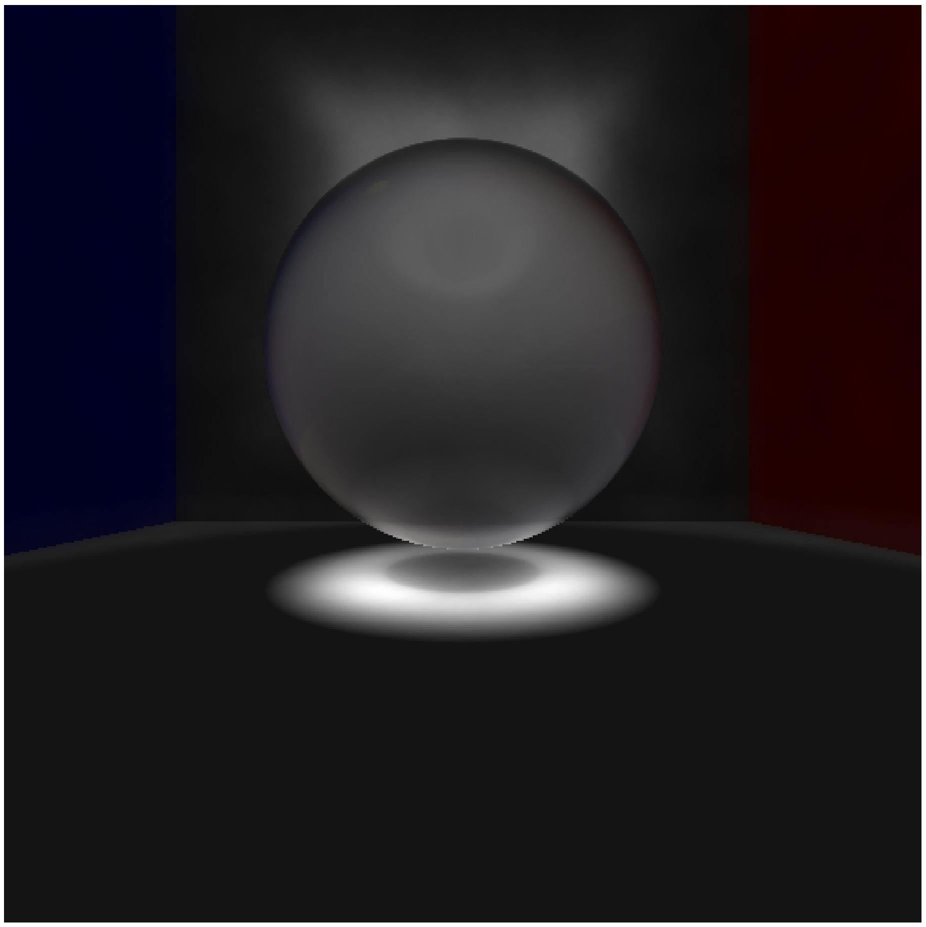

Figure 18: Close up of caustic lighting, I have scaled up the caustic lighting to make it more obvious, moving the light closer and above also causes more global photons to act the way caustic photons act making the patch more bright, a sphere causes this shape.

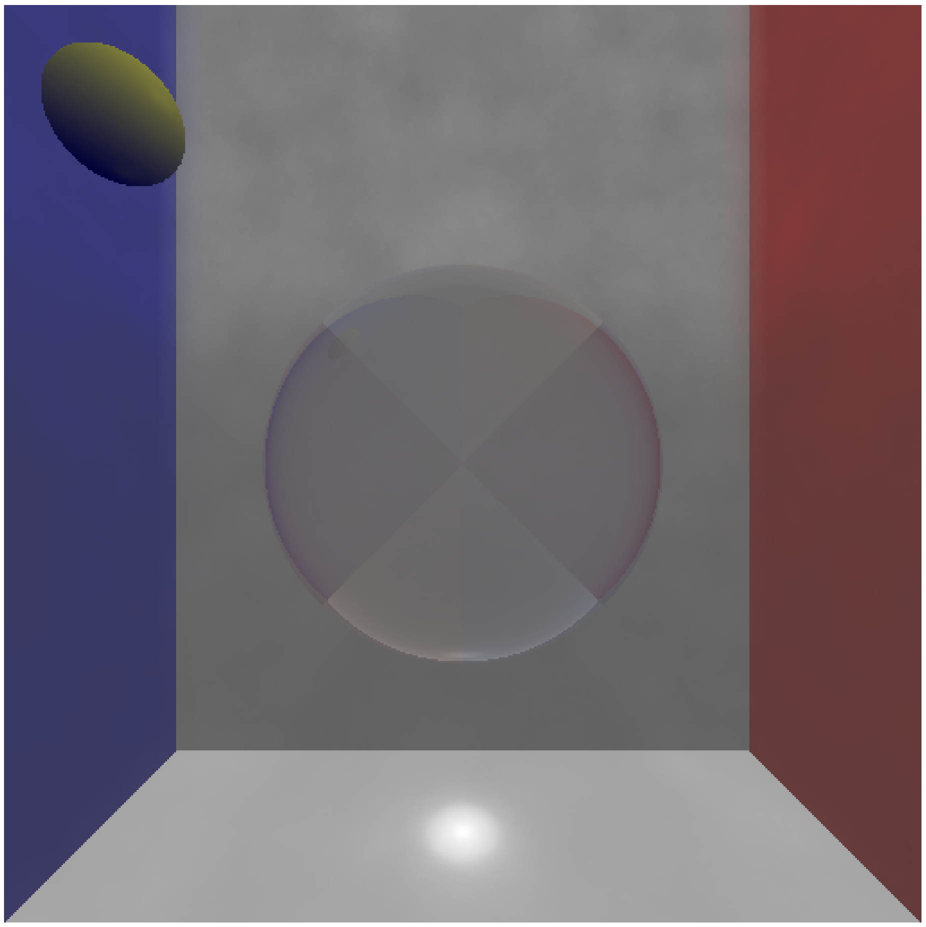

Figure 19: Scene after the addition of caustic lighting, I moved the light above so some other lighting changed but the caustic effect can more clearly be seen on the bottom this way.

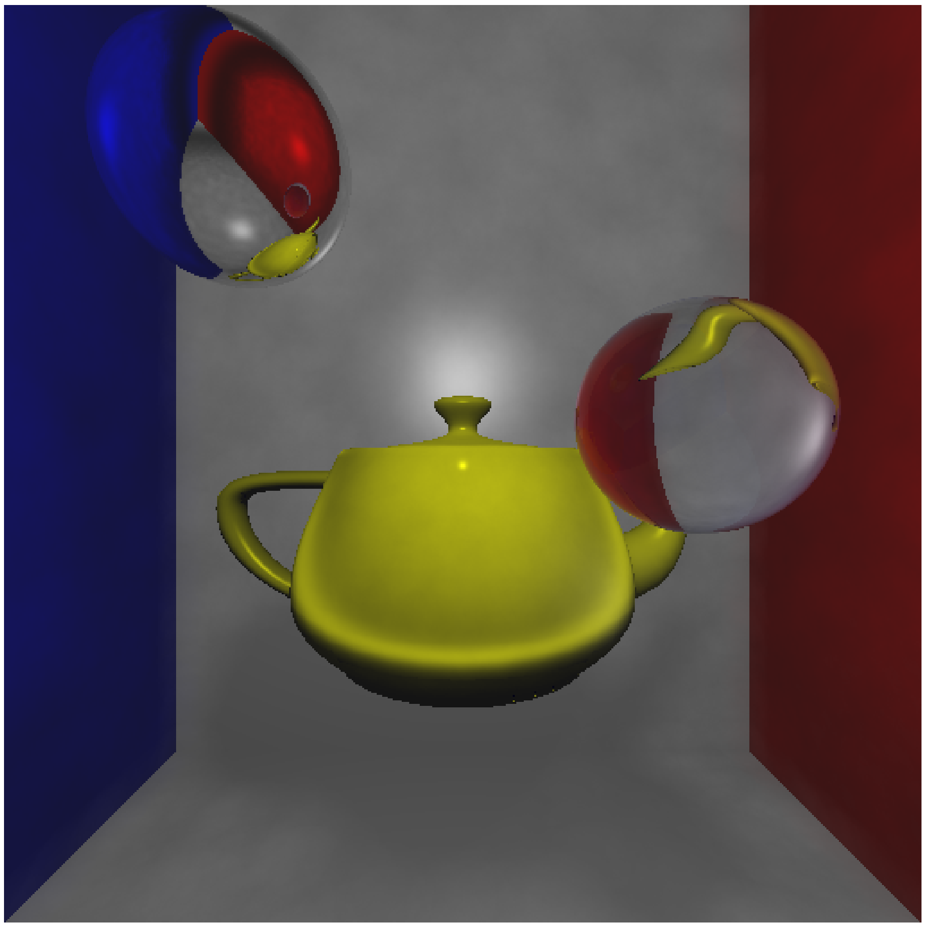

Figure 20: Teapot in room, produced with photon mapping

## C Graph Appendix

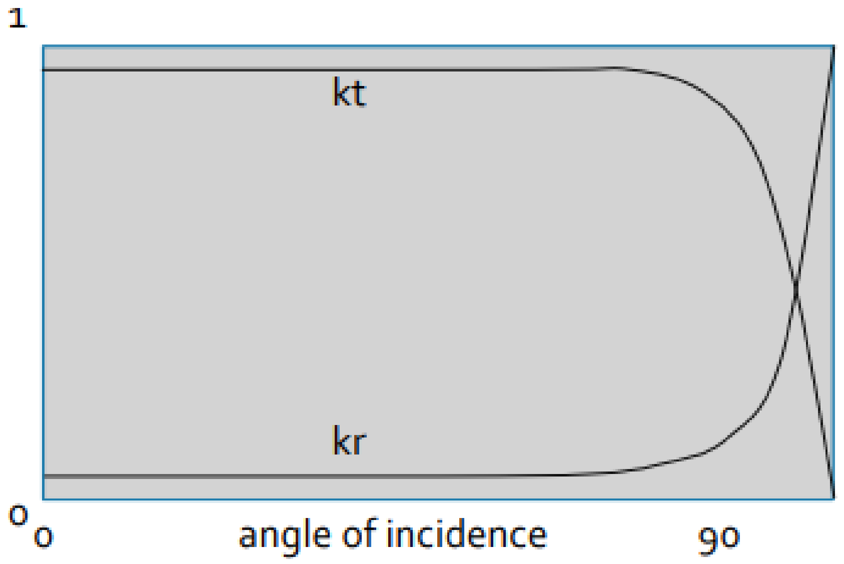

Figure 21: Graph showing reflection and refraction in a refractive material relative to angle of incidence

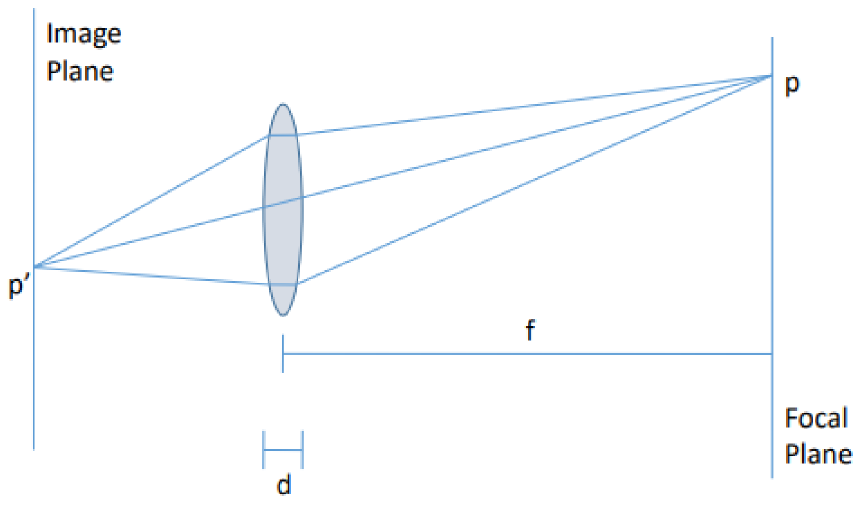

Figure 22: Diagram of thin lens model