Computer Graphics report

Introduction

This report will discuss the thought process undertaken to write the extended code for a ray tracer. The theory will first be discussed, followed by how the code was structured. Simulation results are shown in the appendix.

Reflection and refraction

The current lighting equation as far as lab 4 is described as follows:

$$I = I_a + K_d \sum_{j=1}^{j=ls} (N.L_j) + (K_s(R.V)^n) \quad (1)$$

This lighting model was suitable as a basis for understanding how to represent light. However, it lacked inter-surface reflections between surfaces. This equation therefore needed to be adjusted.

In an article by Whitted, (1980), an extension of the above equation is proposed:

$$I = I_a + K_d \sum_{j=1}^{j=ls} (N.L_j) + K_s S + K_t T \quad (2)$$

where K_s is the specular reflective coefficient
K_t is the transmission coefficient
S is the intensity of light incident from the reflection direction
T is the intensity of light incident from the transmitted direction

With this lighting model, an implementation is also proposed by Whitted, (1980). The advised implementation is similar to prior implementations in labs 3 and 4 with respect to calculating the ambient, diffuse and specular components. The difference with the implementation is that if an object is reflective, a ray will be generated at the intersection position, and the ray tracing function will be called again in the reflection direction. The resulting intensity (I) from the callback function will be scaled by the intersected object's reflection coefficient (K_s) and will then be added to the parent ray's calculation. In this way, a final colour will be calculated.

The direction of the reflected ray will use the following equation:

$$R = I - 2(N.I)N \quad \text{Equation 3}$$

where I is the direction of incident light
N is the normalised surface normal

The R in equation 3 is the same R that is used to calculate the specular component in equation 1. In order to avoid reflection acne, the ray position needs to be adjusted by a bias. This bias is in the direction of R (the direction of the reflected ray).

The structure proposed for the reflection implementation will naturally be recursive. In order for the recursion to eventually terminate, depth recursion is used. The recursion will eventually stop at a certain depth or it will stop when no object is intersected. The $K_s S$ term of the lighting model is covered. The results for reflective objects can be seen from Image 1 to 3 in the appendix.

With an implementation of reflection completed ($K_s S\ t$) , an implementation for the transmission ($K_t T$) term is also discussed. Much like the reflected ray case, the transmitted ray will follow a similar pattern of recursion. The transmitted ray, however, will follow a different path compared to the reflected ray. The equation for the direction of the transmitted ray (Tray) is shown below:

$$\text{Tray} = (1/\eta)I + \left(\cos\theta_t - \frac{1}{\eta}\cos\theta_i\right)N \quad \text{Equation 4}$$

where $\eta = \eta 1/\eta 2$
$\eta 1$ is the index of refraction(ior) for the origin medium
$\eta 2$ is the ior for the destination medium
I is the direction of incident light

$$\cos\theta_t = \sqrt{1 - (\frac{1}{\eta^2})(1 - cos^2\theta_i}$$

N is the normalised surface normal
$\cos\theta_i = N.I$

Similar to reflection, a bias will need to be added as well to avoid refraction acne. The bias will shift the ray point a small distance along the direction of the refracted way. In this way refraction acne is avoided.

There are a few points to note with the equation above. The equation assumes that the direction of the normal (N) and the direction of the incident light (I) are different to one another. Therefore, the direction of the normal needs to be reversed in the case that light travels from the destination medium to the medium of origin. This will also require $\eta 1$ and $\eta 2$ to be reversed.

Another consideration is that the value of $N.I$ needs to be positive. As a result, if the dot product is negative, the absolute value will be taken as an input to calculate the direction of the transmitted ray.

The final consideration is to deal with the case of total internal reflection. This case is covered by checking if the value within the root of $\cos\theta_t$ is an imaginary number. If it is imaginary, the reflection coefficient is required to be set to 1 and the intensity of the ray in the reflection direction will be calculated using the recursive algorithm previously mentioned. Otherwise, the callback function at the intersection position with light travelling in the transmission direction will be called. This completes refraction.

To generate a more realistic image, the values of K_s and K_t are not set by the user. Instead, the Fresnel equations below are utilised in order to accurately calculate the reflection and transmission coefficients. Note that the previous symbols introduced in equation 4 are present in equations 5 and 6.

$$r_{||} = \frac{\eta\cos\theta_i - \cos\theta_t}{\eta\cos\theta_i - \cos\theta_t} \quad \text{Equation 5}$$

$$r_{\perp} = \frac{\cos\theta_i - \eta\cos\theta_t}{\cos\theta_i + \eta\cos\theta_t} \quad \text{Equation 6}$$

The amount of light reflected is hence

$$k_s = \frac{1}{2}({r_{||}}^2 + {r_{\perp}}^2) \quad \text{Equation 7}$$

By conservation of energy the amount of refracted light is:

$$k_t = 1 - k_s \quad \text{Equation 8}$$

When calculating these coefficients, the same considerations for refraction need to be applied to the implementation using the Fresnel equations excluding the reversal of the normal vector. With the theory covered, the structure of the code, coding details and the challenges can be described.

Using the code template provided, the render scene function included calculations for the diffuse component, specular component and ambient component. This was followed by 2 if statements: one for calculating the intensity of reflective objects and one for calculating the intensity of transparent objects with preset coefficients. The if statement structure was kept the same with the addition of another if statement to account for calculating the intensity of transparent objects using the Fresnel equations. Material types were added to the material class that served as the conditions to the if statements. With this consideration, non-reflective, reflective and transparent objects could be rendered within the same scene.

The calculations for the reflection direction, refractive direction and the k_s term using the Fresnel equations were coded in the Vector class as the directions would return a vector and the calculations for k_s were considered as a utility function.

One difficulty encountered when coding transparency was recognising if the effect was being simulated correctly. A suggestion that greatly helped was to change the background colour from black to another colour such that the transparent object could be observed more easily (Suffern, 2016).

One of the problems when dealing with refraction was that the generated image would contain a black ring around it when the ior of the transparent material was not set to 1. Image 4 in the appendix shows a transparent sphere with ior = 1, while image 5 shows the error when the ior is not set to 1. It was later discovered that the reason for such an error was because of the total internal reflection. The problem was that the k_s value was not being set to 1 when total internal reflection occurred (Suffern 2016). To rectify this, if total internal reflection were to occur, k_s would be set to 1. This was also handled in the k_s function using the Fresnel equations. Image 6 in the appendix illustrates this.

This completed the simulation of transparent as well as reflective objects.

Depth of Field

The next feature to be explained is depth of field (DOF). DOF is used to focus particular objects like that of a camera. Objects outside of this focal point will be blurred. In order to simulate this effect, the thin lens camera model needs to be examined.

The thin lens camera model is composed of 2 planes: the image plane and the focal plane. A point on the focal plane has a corresponding point on the image plane. Rays from the point on the image plane pass through a thin lens located between the focal plane and the image plane such that all of the rays will end will end up at the same point on the focal plane. Hence, this point is said to be in focus. This concept is illustrated in Figure 1 below:

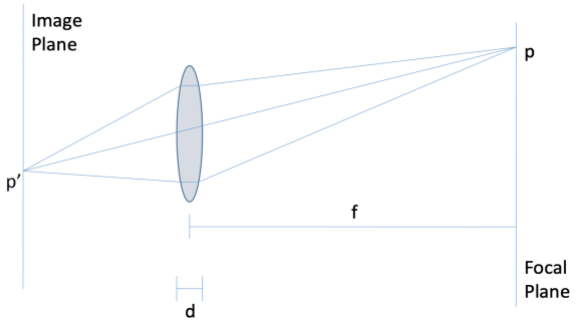

Figure 1: Thin lens model

However, some of the points may be out of focus. This is outlined in Figure 2. This occurs because multiple rays from a point that is not on the focal plane will generate multiple points on the image plane. This is what is referred to as a circle of confusion. This is the intended effect to simulate in order to generate blurry images. A more detailed description of this effect can be found in Suffern, (2016).

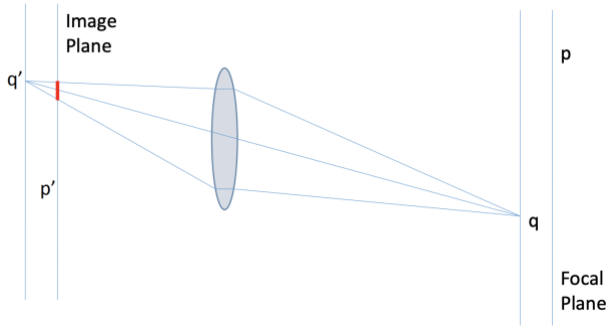

Figure 2: point q out of focus resulting in q'

In order to code this from the basic ray tracer in labs 3 and 4, a separate camera class called ThinLens was created to hold all of the code used to render a scene using the thin lens model. This class comprised of the exact same data members as in the pinhole camera class (eye,u,v,w,lookAt and d) with the addition of the radius of the lens, the distance to the viewing plane from the lens and the distance of the lens to the focal plane.

With the data members set up, two constructors were coded. One constructor to take in the distances and radius that had the default vectors set up and another constructor that allowed all of the data members to be customised. This was done to allow for a flexible scene configuration.
In the chapter about DOF by Suffern, (2016), 2 assumptions can be made that simplify the simulation of DOF. The lens in the thin lens model can be assumed to have a thickness of 0. The second assumption mentioned is that no refraction has to be considered. This therefore meant that the lens could be represented as a disk centered about the eye.

Simulating DOF comprises of the following process:
A point p on the focal plane must be generated from a ray from the center of the lens. Next, the point p and a sample point on the lens excluding the center of the lens are used to compute a direction for the ray to be used in ray tracing. Finally, the ray tracing occurs (Suffern, 2016). This coupled with the assumptions in the previous paragraph provided the formulation of methods to be added to the ThinLens class.

Instead of firing a ray from the center of the lens, a simplification was made to generate a point on the viewing plane as done when dealing with the pinhole camera model. To generate a sample points on a disk of radius r given a center point, a method was set up that would return sample point within the radius of the disk. These points were used with the sampled pixel point to calculate the point p on the focal plane as well as the direction of the ray. Following this, a method that calculated the direction of the ray given a sample point and a pixel point was coded. Equations 9, 10 and 11 are used in this method. One thing to be cautions of is to change the starting ray position to the sample point on the lens as not doing so will lead to erroneous outputs.

$p_x = p_{sx}(f/d)$ Equation 9
$p_y = p_{sy}(f/d)$ Equation 10
direction = $(p_x - l_{sx})u + (p_y - l_{sy})v - fw$ Equation 11

where $(p_{sx,}ps_y)$ is a point on a pixel (the view plane)
f is the distance away from the focal plane
d is the distance from the view plane
$(p_{x,}p_y)$ is a point on the focal plane
$(l_{sx,}l_{sy})$ is a sample point on the lens
$u, v\ and\ w$ are normalised vectors in the camera model.

After the ray direction was calculated, the ray would then be traced into the scene using a render scene method. When the intensities were summed for each of the sample points on the lens, an average colour was recorded. This process would then be called for each pixel of the image.

To improve the quality of the image generated, an anti-aliased option of rendering was coded. To perform anti-aliasing in this method, a parameter is added to the render function that includes the square root of the sample size. The square root was used as an input as the parameter because jittered sampling was used. Jittered sampling generates multiple sample points per pixel, such that the points are randomly scattered.

The anti-aliased version of the code meant that the number of sample points on the pixel was equal to the number of sample points on the lens. If no anti-aliasing were to occur, the number of sample points on the lens was set to a default of 64 (8 as a parameter). This completed the simulation of depth of field. Images 7, 8 and 9 in the appendix show the results of simulation

Photon mapping

The methods for generating photo realistic images thus far have been using traditional ray tracing in order to render scenes. Using Herckbert notation the paths LD(S)*E have been satisfied by the current ray tracer. However, this does not simulate L(D|S)*E paths. Diffuse inter-reflections in the traditional ray tracer are represented using the ambient term. This term, however, is an inaccurate approximation. There are better methods of approximation that can be utilised.

Radiosity has been used to obtain a better approximation leading to the simulation of L(D)*E paths, but this does not account for the specular reflection between objects. An accepted method of an approximation for L(D|S)*E paths is photon mapping described in Jensen (1996, 2001). The process of photon mapping from these 2 papers will be summarised.

Photon mapping is an algorithm comprising of 2 passes: 1 pass for the generation of photons and another pass for rendering the scene. Photons can be conceptualised as light particles. In the first pass, a large number of photons start at a light source and are emitted. The path that they take is dependent on the type of light source. One thing to note is that the intensity of the photons is uniformly distributed amongst emitted photons.

A photon map is a data structure that will store photons. In the literature by Jensen (1996,2001), this structure is a KD tree as it allows for a uniform distribution of photons. Photons will be stored in the photon map when they intersect with a diffuse surface. When a photon intersects with an object, its trajectory is determined. This trajectory is calculated by the Russian roulette algorithm. The algorithm will generate a random number between 0 and 1. This number will determine if the photon is reflected, transmitted or absorbed. Photons that are stored within a photon map will have the following attributes: The hit point, the incoming direction to the hit point, the power of the photon and the type of photon.

After creating a photon map, the second pass is ready to be called. In the second pass, the intensity towards the eye is generalised by the rendering equation.

$L_o = L_e(x, w) + L_r(x, w)$ Equation 12
where L_o is the outgoing radiance
$L_e(x, w)$ is the emitted radiance
$L_r(x, w)$ is the reflected radiance

The term $L_e(x, w)$ is only used when the source is a light source. Next, $L_r(x, w)$ can be rewritten as follows:

$L_r(x, w) = \int_{\Omega_x} f_r\,(x, w', w) L_i(x, w') |N \cdot w'| dw'_i$
Equation 13

where L_r is the reflected radiance at x in the direction w, Ω_x is the hemisphere of incoming directions, f_r is the BDRF at x and L_i is the incoming radiance.

One of the problems encountered here was that equation 13 appeared to be abstract, hence some apprehension was experienced when trying to understand a concrete method of implementation for this equation.

In Jensen, (2001) a simplification is made to the formula that treats the integral as an average of n photons in an area. The reflected radiance $L_r(x, w)$ can thus be calculated by finding the n nearest photons in the photon map within a given radius, then summing up their radiance and scaling the individual radiance by the area of the circle that was used to gather the photons.

With an estimate of $L_r(x, w)$, the calculations of the second pass can proceed. In Jensen, (1996,2001), $L_r(x, w)$ can be split into four components: Direct illumination, Specular and glossy reflections, caustics and multiple diffuse reflections. The computations are split into accurate and estimate. Accurate calculations are used when the point is seen directly from the eye or little specular reflection. Approximate calculations are used if the ray has been reflected diffusely since leaving the eye or if the distance between the origin and the object is below a certain threshold.

Direct illumination is computed accurately as the eye is sensitive to it, whereas the approximate estimate is used by estimating the radiance in the photon map. Specular and glossy reflection can be calculated by using the traditional ray tracing approach to generate reflective rays and transmissive rays depending if the object is transparent. The radiance of the reflected rays will be dependent on the object's surface. Diffuse inter-reflections can be calculated by reflected radiance estimate at the surface of the object. Caustics can be accurately calculated using a separate caustic photon map that fires photons towards specular objects and it can be approximated using the global photon map. Using these insights, the coding was ready to proceed.

To implement the photon map data structure, a struct for a photon was created in the class responsible for rendering the scene (scene class). The first pass and second pass methods were added to scene class as well. In doing so, this ensured that the first pass could be done in main program and then the second pass could be conducted in the pinhole camera class for every pixel in the image.

To emit photons, the first pass method included a loop to ensure that all of the lights in the scene were iterated. The emit function was stored in the light classes. This meant that different lights could distribute photons with respect to their light distribution. The emit method would take in an array of rays. This was included as an argument because this would allow photons to be traced after the different emission directions were calculated. The starting position of the ray was set to the coordinates of the light source. In this implementation, only the point lights included the emit function. This is because these were the simplest to implement.

After the ray array for the photons was setup, a method that traced photons into the scene was called. This method was

essentially the same as the render scene function. The main difference was that photons were stored in a KD-tree when they hit a diffuse surface. Also, when photons were to hit a surface, there the Russian roulette method to choose a random number based on the material constants would be called. This generated number would determine the trajectory of the photon. There were 3 cases of materials to consider in the photon tracing method: diffuse materials, reflective materials and transmissive materials.

One challenge encountered was the storage of photons in a KD tree. To store photons in a KD-tree, the nanoflann library was utilised as it provided radius search and examples on KD creation with an adaptor class. An adaptor class was required because as opposed to storing points, the photon map had to store the photon information that was previously mentioned. Therefore, photons could be stored in a PointCloud structure when they hit a diffuse surface. After the photon tracing was called, a simple buildIndex() function was called to create the KD tree. This completed the first pass.

The second pass method's code was the same as the render scene method, with the addition of radiance estimates for indirect diffuse-interreflections. The method of radiance estimation first used the nanoflann library method to locate photons within a given radius. After that, a loop to iterate all of the returned points was coded. This would ensure that a sum for an estimate could be computed. The challenging part about the estimation method was that I did not realise that the BRDF scaling had to occur in the first pass as well.

Another challenge with this method was that scaling factors had to be added on top of the calculations to ensure that the reflected light to the eye did not exceed 1. This was done by adding a hard-coded scaling factor to the diffuse scaling calculation. For the BRDF calculation, the Phong model was utilised. The calculated BRDF calculation would then be divided by the area of the circle of search and the next point would be addressed. This ensured that a colour for indirect interreflections could be computed. This finished the implementation of the estimate for indirect diffuse reflections. The results for this can be seen in images 10 and 11 in the appendix.

For caustic estimation, a second photon map was created to store caustic photons. To create this photon map, the photon trace method was used. Caustic estimation requires many photons to render caustics effectively. It was therefore a decision to replicate the photon emission for caustics after the global photon emission had taken place. This emission would be enclosed in a while loop to ensure that there were enough photons that were caustic before proceeding with the caustic map creation. The photon trace method dealt with this by checking if a photon had the type 'S' when it hit a diffuse surface. If the photon had that type meant that it had been secularly reflected. It could therefore be added to the caustic photon map. One mistake made was adding more photons to the global photon map if they were not considered caustic. This mistake can be seen in image 12 in the appendix. In order to solve this issue, an integer was added to this method. This integer would be set to 1 in the global photon map creation phase. It would allow paths of L(D|S)* to be accounted for. The integer would then be set to 2 to ensure that only L(S)+ (specular reflections or transmissions) would be stored in the caustic photon map. The caustic photon map was then constructed using the buildIndex() function.

The estimation of caustics was then placed after the estimate of diffuse-interreflections. A separate estimation method was chosen. This was done in order to use the caustic photon map for point gathering. The cone filter was applied to the scaling as well because in the literature by Jensen, 2001, a cone filter can reduce the blur of caustics. The creation of a separate method allowed the BRDF calculation to be scaled by a different value as well. This ensured that caustic estimates were accounted for. Images 13 and 14 in the appendix show the results.

Concluding remarks

This report has detailed the taken approach to extending the ray tracer by adding reflection, refraction, more accurate lighting using photon mapping and blur effects using depth of field. The limitations with this current implementation are that shadow caching with respect to photon mapping was not utilised. More complex methods of photon emittance for different light sources was not used. Furthermore, the shadow effects for the point lights did not work in the cornell box. I therefore left out the shadow ray calculations in my implementation. If project were to be continued, more complex algorithms for directing light into the scene for more accurate caustics would be researched.

References

Jensen, H.W., 1996, June. *Global illumination using photon maps*. In Eurographics workshop on Rendering techniques (pp. 21-30). Springer, Vienna.

Jensen, H.W., Christensen, P.H., Kato, T. and Suykens, F., 2001. *A practical guide to global illumination using photon mapping*. ACM SIGGRAPH 2001 Course Notes CD-ROM, 10(1103900.1103920).

Suffern, K., 2016. *Ray Tracing from the Ground up*. CRC Press.

Whitted, T., 1980. *An improved illumination model for shaded display*. Commun. ACM 23, 6 (June 1980), 343-349. DOI: https://doi.org/10.1145/358876.35888

Appendix

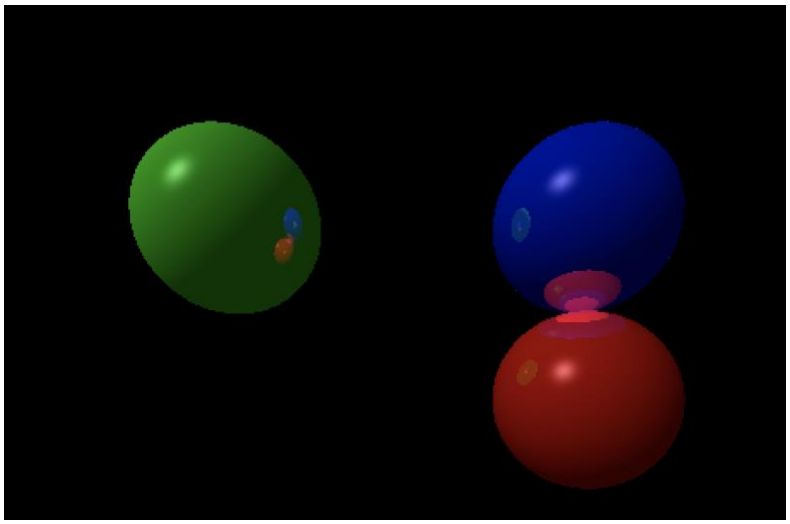

Image 1 Reflection where coefficients are not equal

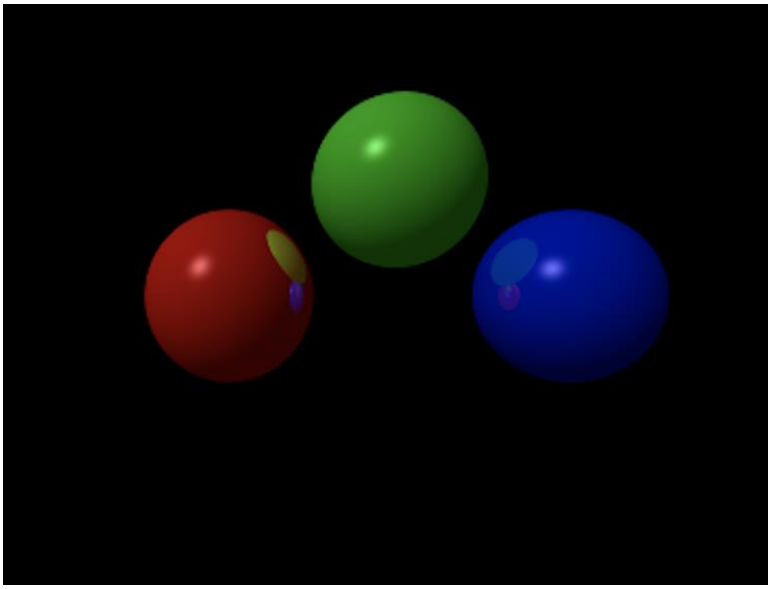

Image 2 Not all objects are reflective

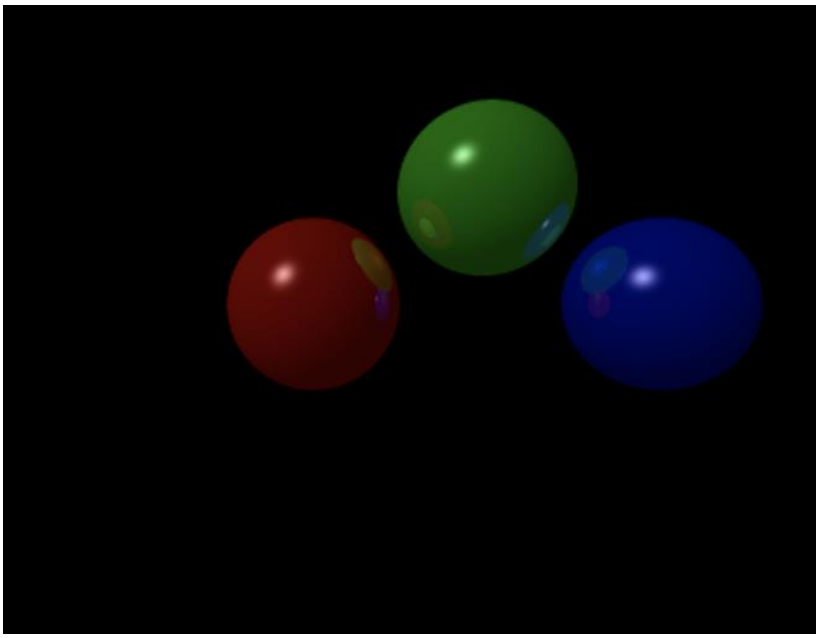

Image 3 Reflective coefficients are equal ks = kr

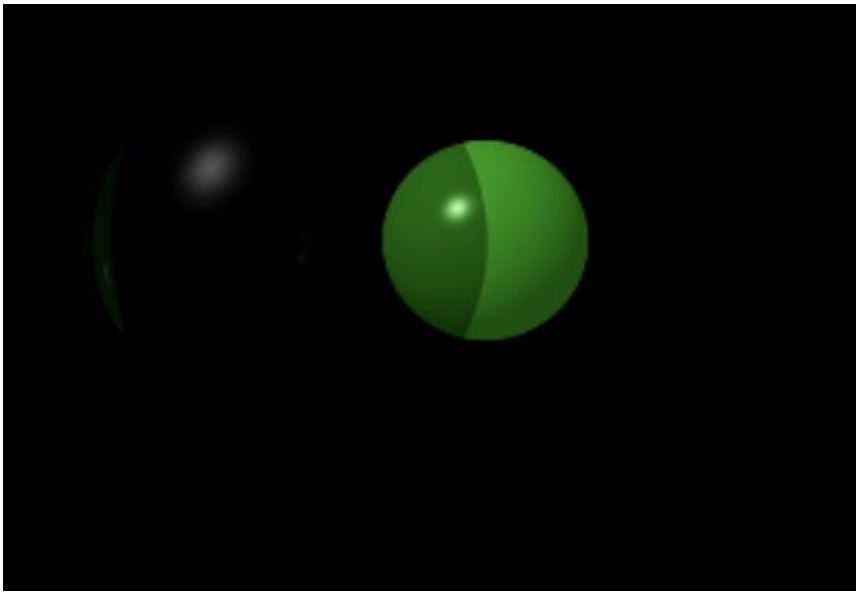

Image 4 Transparency ior = 1.0

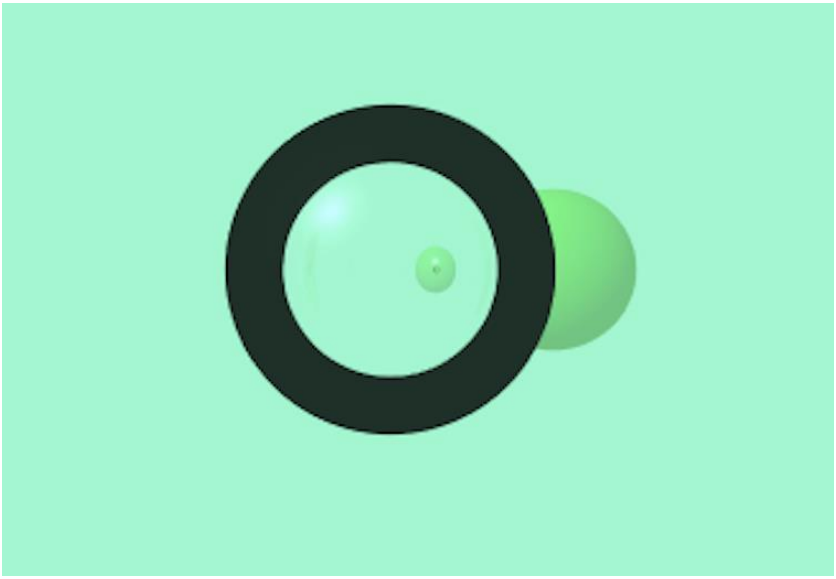

Image 5 Incorrect transparency ior = 1.5

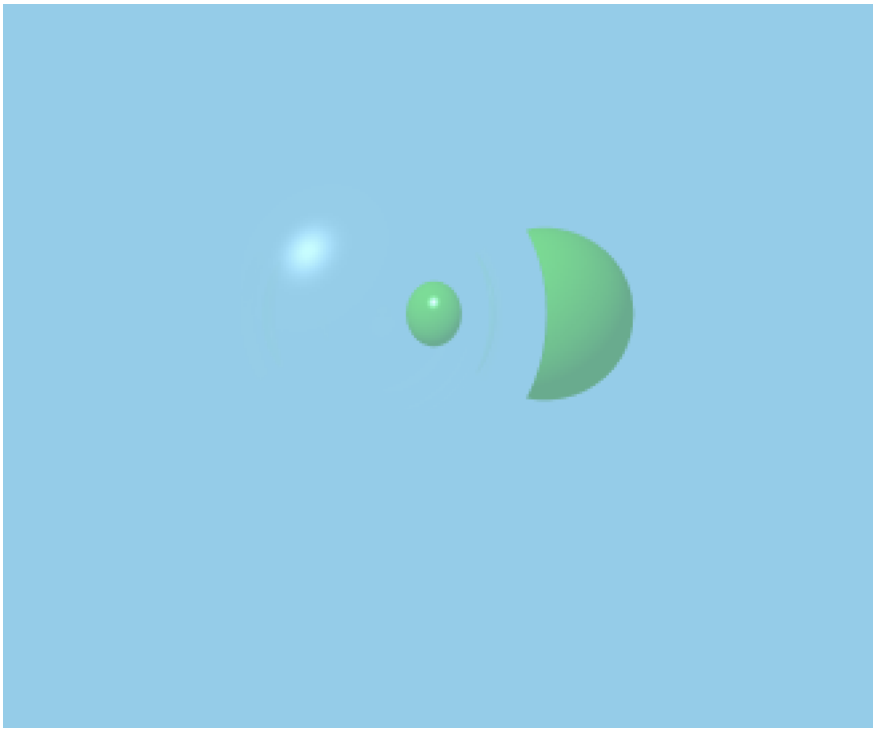

Image 6 Corrected transparency ior = 1.5

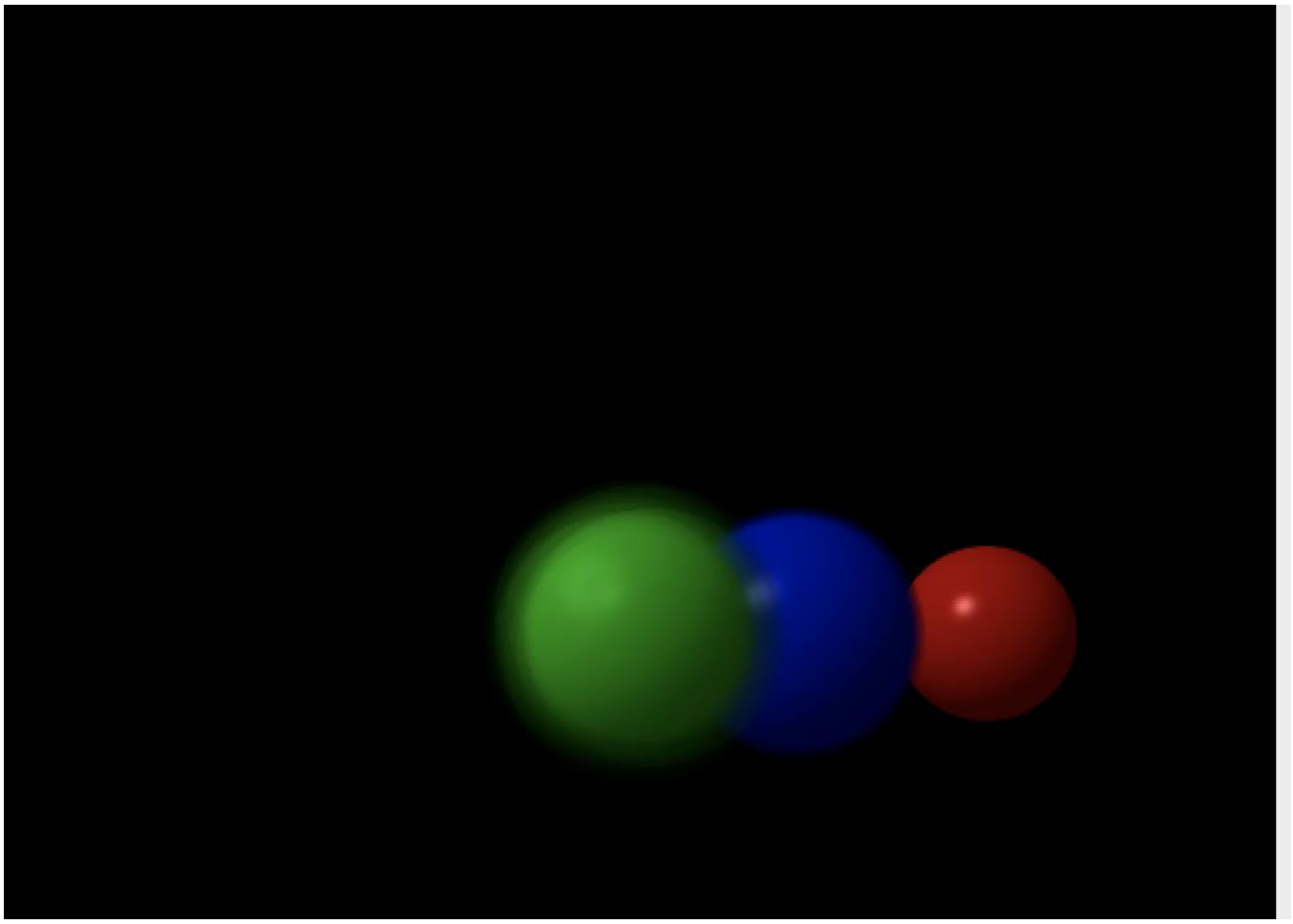

Image 7 Depth of field: red sphere in focus

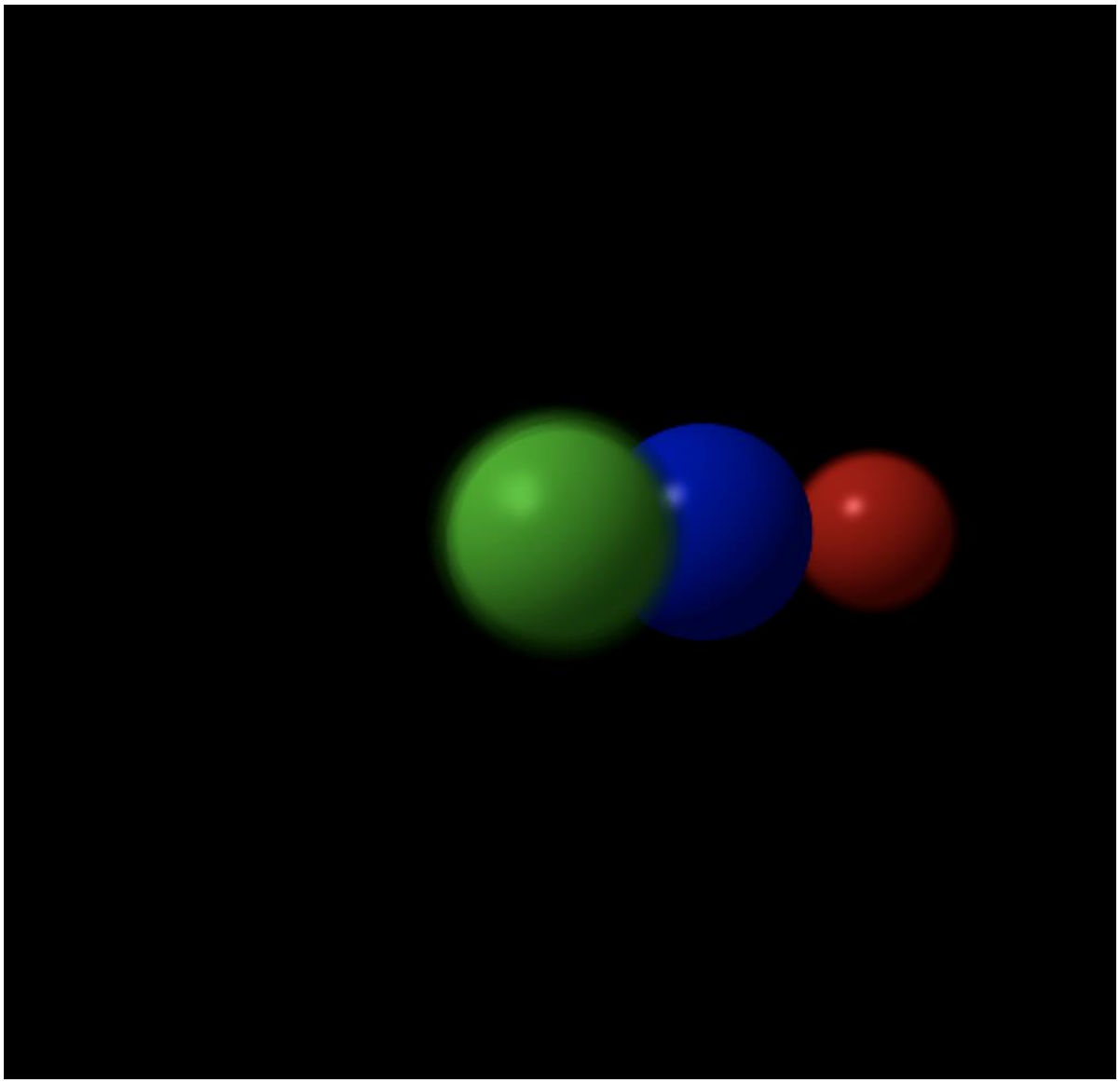

Image 8 DOF: Blue sphere in focus

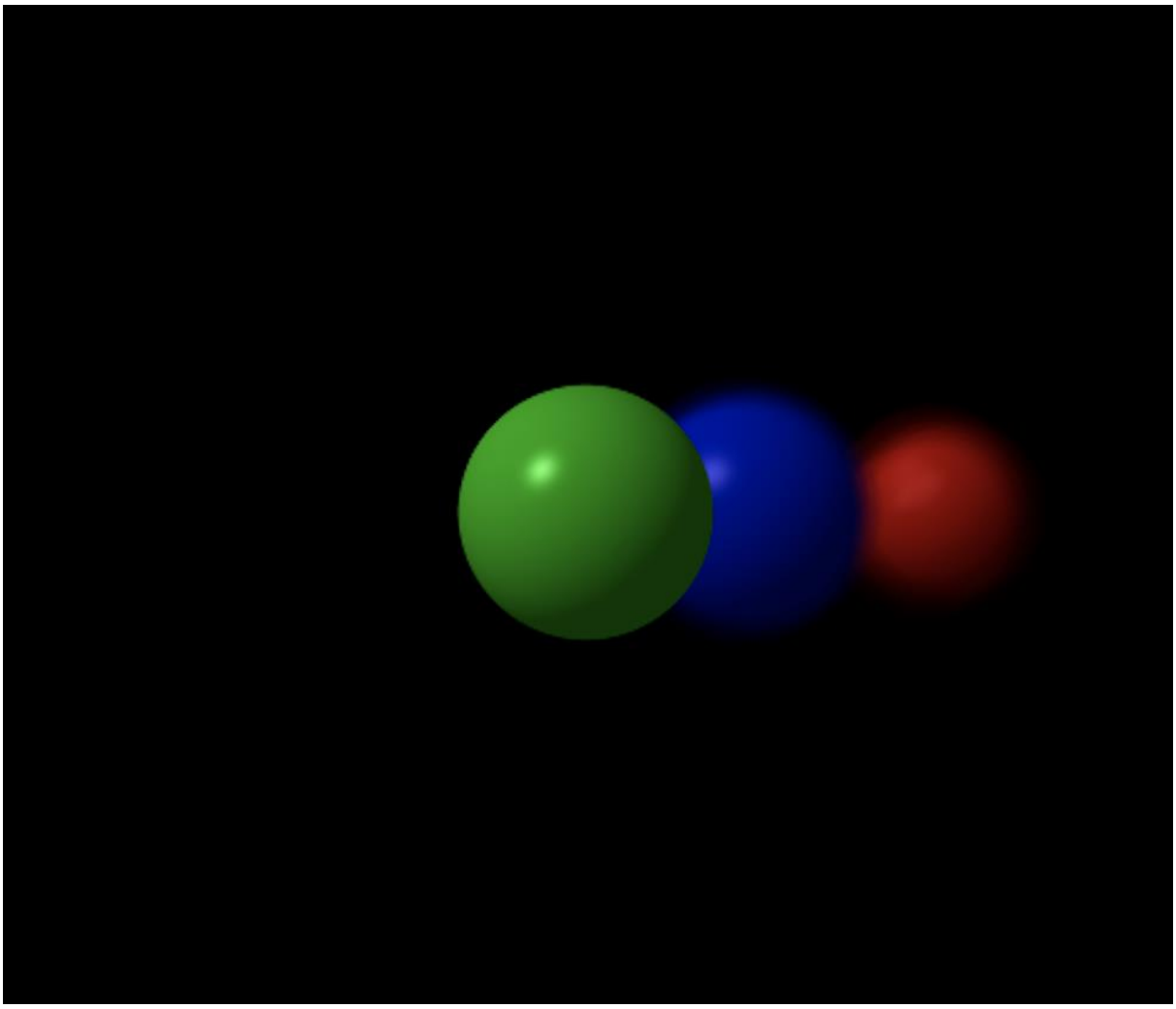

Image 9 DOF: Green sphere in focus

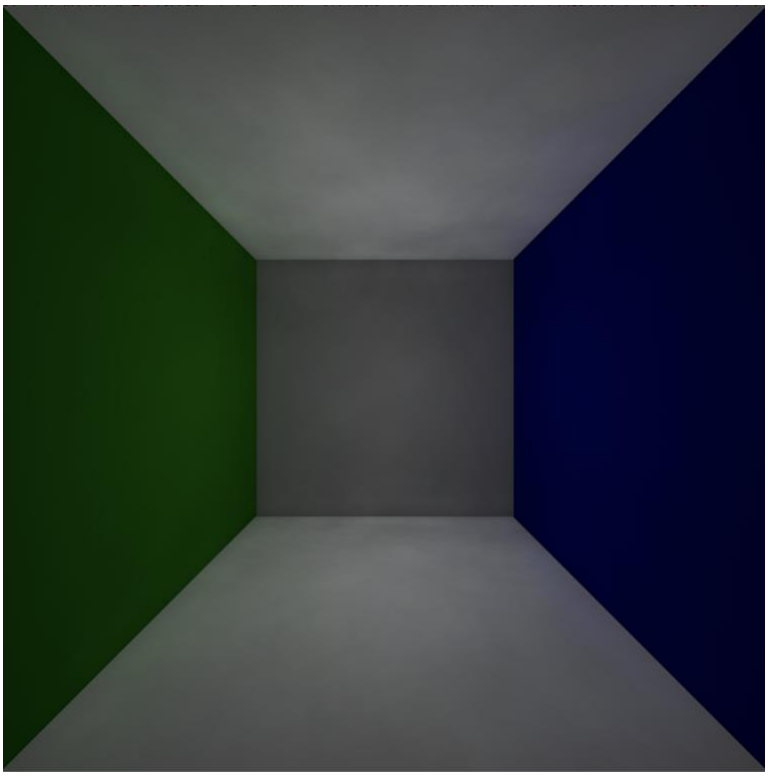

Image 10 Photon mapping N= 50000 Scaling factor = 0.1

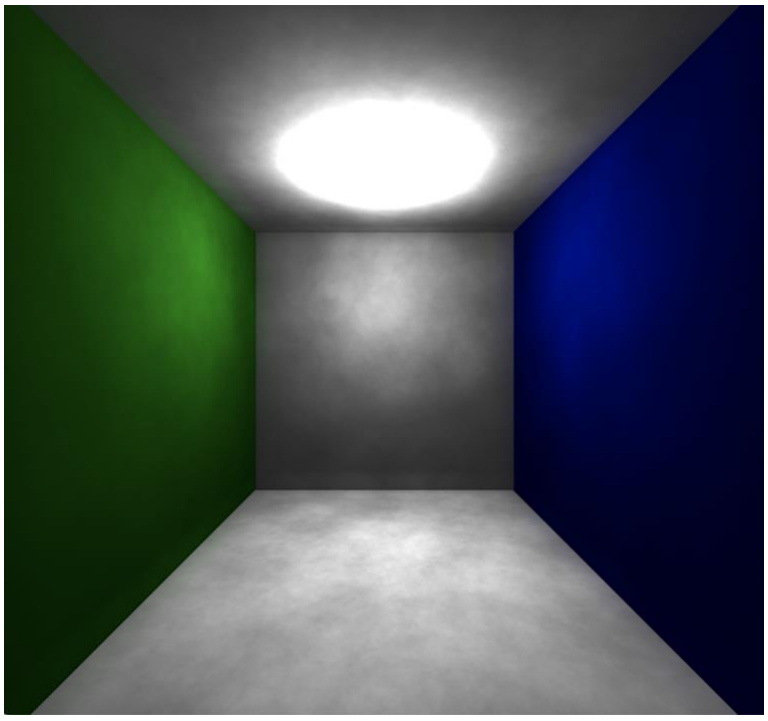

Image 11 Photon mapping Global Photons = 50000 Scaling factor = 0.16

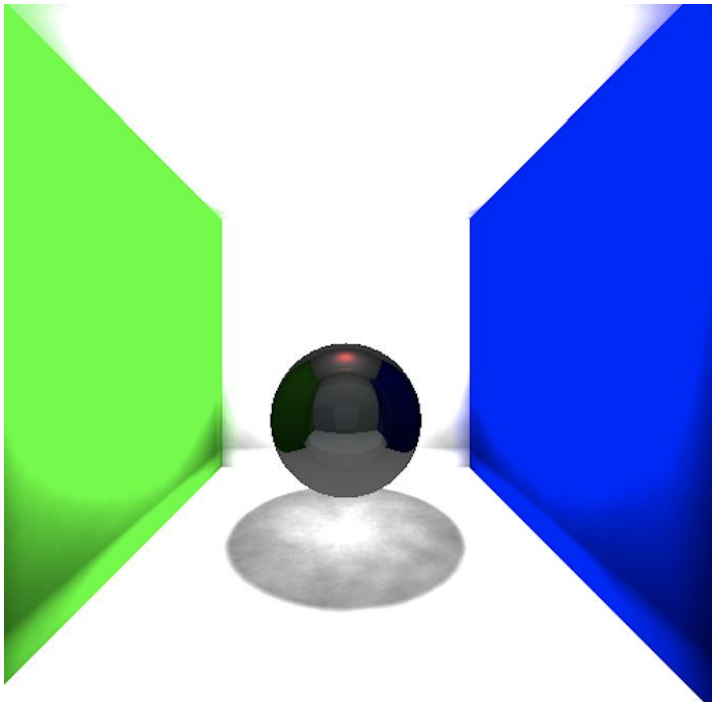

Image 12 Photon mapping: Caustic Photons = 100000, Global photons - 50000. Incorrect scaling

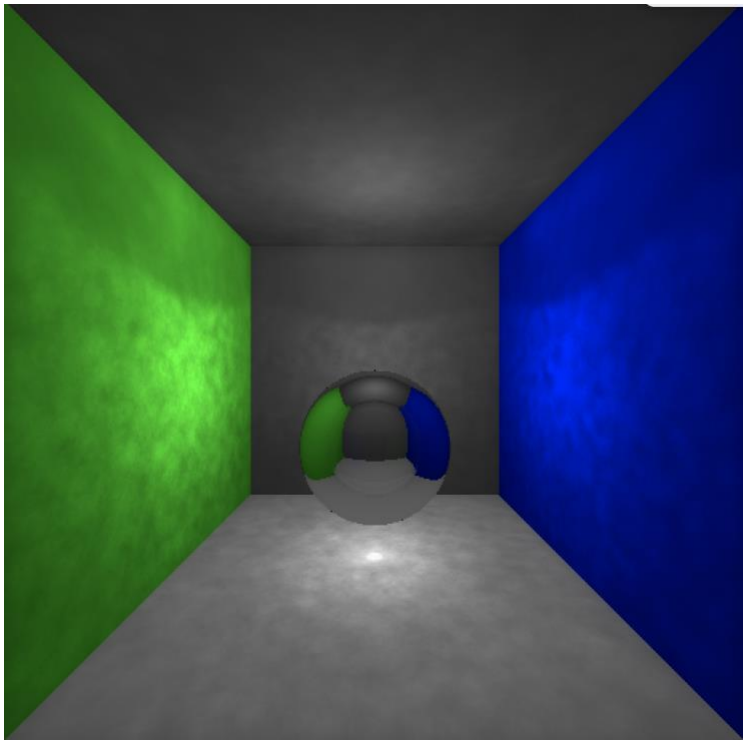

Image 13 Photon mapping: Caustic Photons = 100000, Global Photons = 50000, radius =0.05. Corrected scaling

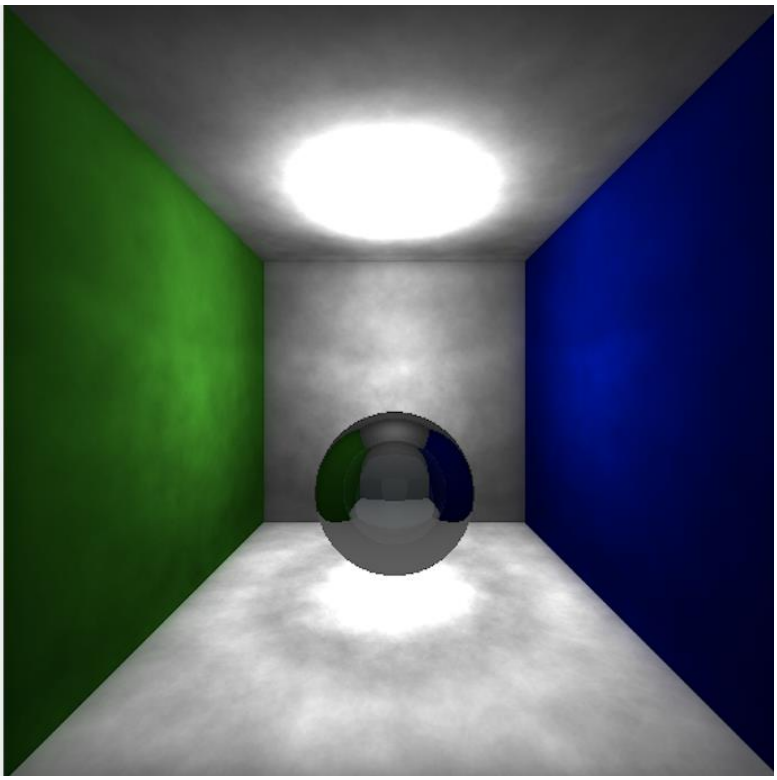

Image 14 Caustic Photons = 100000, Global Photons = 50000, radius =0.1.