# Photon mapping

Informática gráfica

January 2019

*721057* Izquierdo Barranco, Sergio
*716185* Guerrero Viu, Julia

# Contents

# Chapter 1

# Introduction

This report depicts the development of a render engine based on Photon Mapping algorithm. Photon Mapping is a two-pass global illumination algorithm developed by Henrik Wann Jensen that approximates Render Equation. The first main basic idea consists on shooting photons from the light source instead of from the camera and storing them in a photon map, hence its name. Second pass traces rays from the camera and estimates global illumination with a radiance estimation based on the k-nearest neighbours on the map. Photon Mapping algorithm is specially interesting because it can simulate difficult effects to capture with path tracer, such as caustics, or effects that are usually computationally very expensive such as participating media.

For the implementation of the render engine, the provided basic code has been used, adding different extension classes to it. Basically, this engine includes planes, spheres and triangle meshes as geometries, and lambertian, Phong, perfect specular and refractive as types of materials. Light sources are modelled with puntual lights and radiance estimation is calculated using the Uniform box kernel. This basic version is explained in chapter 2 together with an explanation of the main challenges faced and how they were solved.

In addition, two extensions including participating media (ambient approximation and homogeneous isotropic media) and an analysis of different kernels are explained in chapter 3.

Finally, some conclusions of the project and an analysis of workload are included in last two chapters.

# Chapter 2

# Basic Photon Mapping

## 2.1 Photon Map Generation

### 2.1.1 Light sampling

**Question 1.1.** ***When tracing photons, you need to sample an omnidirectional point light source from which you will start the random walk; describe the procedure you are using for obtaining the origin and direction of a photon random walk.***

To sample the lights, two methods have been implemented and will be analyzed: Rejection sampling and solid angle sampling. Both methods aim to sample the unit sphere around the light uniformly, what means that all differential points from that sphere have the same probability of being chosen.

#### Rejection sampling

Firstly, a simple rejection sampling was implemented. The idea of this method is to include the sphere inside a unit cube, uniformly sample inside that cube (three random numbers in range $[-1, 1]$, one number for each dimension) and then reject points that are not inside the sphere, what means points whose distance to the center is greater than 1. This way of sampling is simple and correct but a bit "dumb" in the sense that we are calculating points that are not used in the end. In terms of efficiency, however, is not almost appreciated in comparison to solid angle sampling, as even some points are discarded, calculations are simpler and sampling time is insignificant versus rendering time so the general efficiency of the engine is not affected.

#### Solid angle sampling

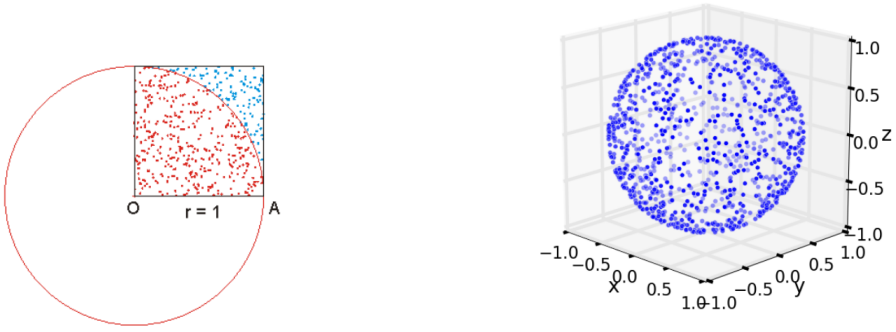
Secondly, solid angle sampling has been implemented and it is the way of sampling that is used in the algorithm by default. To sample the sphere, two random angles, $\phi$ and $\theta$, should be chosen in ranges $[0, 2\pi]$ and $[-\pi, \pi]$, respectively. However, generating just two random values in those ranges would not be uniformly distributed in the sphere, as, for example, in the poles all values of $\phi$ will correspond to the same point while in the equator every value corresponds to a different point. The idea of this way of sampling is to avoid that problem by using the $cos(\theta)$ as the *pdf* (probability density function) in $\theta$. This way, the *cdf* (cumulative density function) is given by the $sin(\theta)$, which is going to be uniformly sampled in the range $[-1, 1]$. Putting all together, next sample ray is calculated as follows based on the two angles:

1: $r_1 = uniform\_random\_number([-1, 1])$
2: $r_2 = uniform\_random\_number([0, 1])$
3: $\theta = arcsin(r_1)$
4: $\phi = 2\pi \cdot r_2$

Finally, for proper normalization, it is important to consider that integrating the solid angle over the sphere, the result is not 1 but $4\pi$, so it has to be taken into account when saving the photons to assure energy conservation.

$$\int_0^{2\pi} \int_{\frac{-\pi}{2}}^{\frac{\pi}{2}} cos(\theta) d\theta d\phi = 4\pi$$

This means that if one light source has, for instance, intensity of 5 Watt per differential solid angle, it will have a total flux equal to $5 \cdot 4\pi$ in the whole sphere, so if we shoot, for example, 100 photons from it, each of the photons will start their path having energy equal to $\frac{5 \cdot 4\pi}{100}$.




(a) Rejection sampling (2D)

(b) Solid angle sampling (3D)

Figure 2.1: Graphical representation of two ways of uniform sampling the sphere. Rejection is represented in 2d although in our case the 3d version is done.

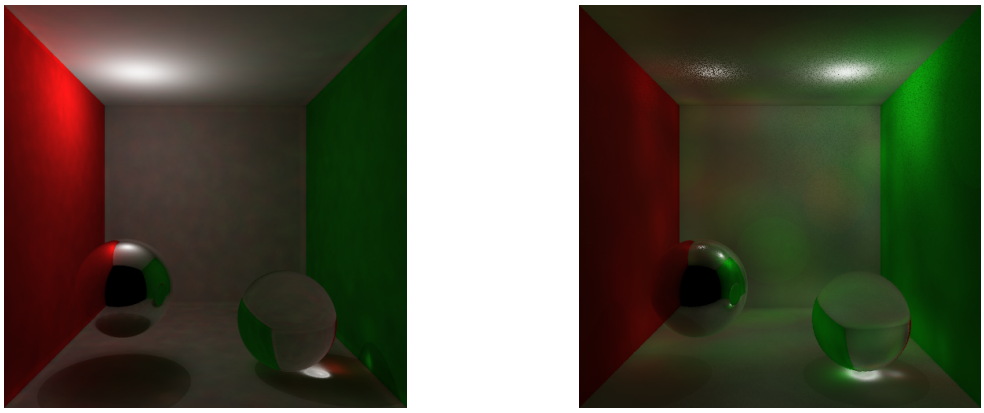
### 2.1.2 Light Russian Roulette

To be able to model differences between intensities when having more than one light source and avoiding to store photons with very different levels of energy, Russian Roulette between lights has been implemented.

This way, every time shooting a photon, a light source to shoot from is chosen, based on their intensities: in a scene with two lights, having $l_1$ intensity equal to 5 Watt per differential solid angle and $l_2$ intensity equal to 15 Watt per differential solid angle, there will be a probability of 25% of choosing $l_1$ and 75% of choosing $l_2$.

Without this Russian Roulette, if you shoot always the same number of photons from each of the lights, photons from very different intensity light sources will be stored with very different energies, and the algorithm will perform worse. Another option could be to choose manually the number of photons shot from each of the lights sources, having too many different parameters and depending too much on the scene rendered.

Russian Roulette between lights is also applied when computing direct illumination in the second pass of the algorithm.



(a) One puntual light source of power 5W/sr on the left top corner.

(b) Two puntual light sources. Left source has power of 5W/sr and right source has power of 15W/sr

Figure 2.2: Comparison of same scene with one or two light sources. Although the second scene has not converged yet, the different intensity of lights can be clearly seen, and the effect of two light sources on the shadows, making them softer. Both images have been redered shooting 10 000 photons global and 10 000 caustic, and doing radiance estimation with 200 nearest neighbours. The applied tonemapper (which bases the maximum luminance with the maximum of the scene) makes impossible to distinguished that second scene has more total energy.

### 2.1.3 Area lights

Area lights have not been finally implemented. To add them to the algorithm, the first thing to be taken into account should be how to sample them uniformly. It would depend on the geometry of the area or volume light. Apart from that, for the second pass of the algorithm, next event estimation with this type of lights should be done (it is not trivial) if wanted to calculate direct illumination with ray tracing. Another option would be to calculate both direct and indirect illumination using the photon map.

### 2.1.4 Photon random walk

**Question 1.2.** ***Note that in the provided ray tracer the photon random walk is already implemented (function PhotonMapping::trace_ray in PhotonMapping.cpp, together with a KD-Tree to accelerate photons' look-ups. If you choose to use the provided code, you will need to explain thoroughly what the function trace ray does and why. The more descriptive, the more you will show that know what's going on.***

The trace_ray function firstly updates the total number of shot photons, to guarantee that no more than the maximum (set by *m_max_nb_shots* parameter) are shot. Then, it stores in variable *energy* the energy of the photon, got by a parameter of the function and shifts the ray to avoid numerical errors in intersections. After that, an infinite loop starts to calculate the random walk that can only end under three circumstances: the ray does not hit anything (*!it.did_hit()*), the Russian Roulette chooses absorption (*epsilon2* > *avg_surf_albedo*) or after the $20^{th}$ bound (*photon_ray.get_level()* > 20), just in case absorption was not chosen before and because the contribution of the next bounds is likely to be insignificant.

Inside that loop, it first calculates next intersection given the ray. After that, there are different cases: if intersected material is delta (perfect specular or refractive), it just updates a boolean flag (*is_caustic_particle*) without saving the photon, as in a delta. This flag will be used in next iterations to distinguish between photons to be stored in either the global or the caustic map. If it is not a delta material, it checks firstly whether the photon comes directly from the light or not, and in this case, excepting the "direct" flag is true, the photon is not stored, as direct illumination will be calculated in the second pass using ray tracing. When it has to be stored, the caustic flag is checked to determine which map to use, and the size of the map is also checked to assure that no more than the maximum number of photons of each type (given by the *m_nb_caustic_photons* and *m_nb_global_photons* parameters) are stored.

Once the photon is stored (or not if direct or delta material) Russian Roulette is applied (*epsilon2*) in function of the albedo of the surface, to decide whether the random walk ends (photon is absorbed) or not. As albedo is actually a color of three channels (RGB), the average between the three channels is used. Russian Roulette is useful here in order to store photons with similar energies, rather than reducing its power whenever they hit a surface. This is simply explained as follows: Giving an initial power of 1000 photons of 1W each and a surface albedo of 0.7, shooting 1000 photons of 0.7W is equivalent to shooting 700 photons of 1W, and the second option makes the algorithm to perform better.

After that, if the photon is still alive, a new ray has to be sampled from that point (*get_outgoing_sample_ray*) and energy has to be updated regarding the properties of the material for next iteration. To account for Render Equation, diffuse materials should be multiplied by the cosine of the angle formed by the next ray and the normal of the surface, and for proper normalization it is divided by $\pi$. In addition, as importance sampling is being applied when getting the next ray, result should be divided by the *pdf* using when sampling, that is returned by the function (*get_outgoing_sample_ray*). We can check that importance sampling, for both diffuse and delta materials, matches the BRDF distribution of energy, because the value of the *pdf* for lambertian materials is equal to $\frac{cos(w_i \cdot n)}{\pi}$, conserving exactly the same energy, and *pdf* for delta materials is equal to 1, also conserving energy. Finally, the result is multiplied and after divided by the albedo of the surface. This is important to account for color bleeding effects, and is the reason of the small range differences in the energies stored by each of the photons. If the scene were monocromatic, multiplying and dividing by the same value would not alter the result, so energy will be conserved perfectly. However, as albedo is actually a color, the energy of the photon is multiplied by it and divided by the average of the three

channels, which ends in changing the color, and hence the energy of that photon.

Finally, if all of the maps are already full, random walk stops as there is no point in continuing shooting photons that will not be stored. In other cases, a new iteration of the loop starts, with the new sampled ray and the modified energy.

In addition, in order to know the exact number of photons shot (not stored) to fill every map and hence knowing exactly the energy of every photon, counters have been added (one for each map and light source) and have to be updated any time starting a new random walk, so these calculations are also done inside this function. Due to the extension for participating media, a new photon map (volumetric photon map) has also been added, together with a flag to distinguish participating media particles.

**Question 1.3.** ***Imagine that in your scene you have three light sources, each with power 1 Watt per differential solid angle. Assuming that you have a monocromatic scene (so you have perfect importance sampling and Russian roullette). In such scene, you are tracing 1000 photons. What is the range of energies stored by the photons?***

As previously discussed in 2.1.1, the total flux emitted by the lights would be $3 \cdot 1 \cdot 4\pi$, so assuming that you are tracing 1000 photons in total (for the 3 lights) and that they are equally distributed among the three light sources, each photon stored would have energy equal to $\frac{3 \cdot 1 \cdot 4\pi}{1000}$. As the scene is monocromatic and with perfect importance sampling, all photons would have exactly the same energy, making the algorithm converge faster.

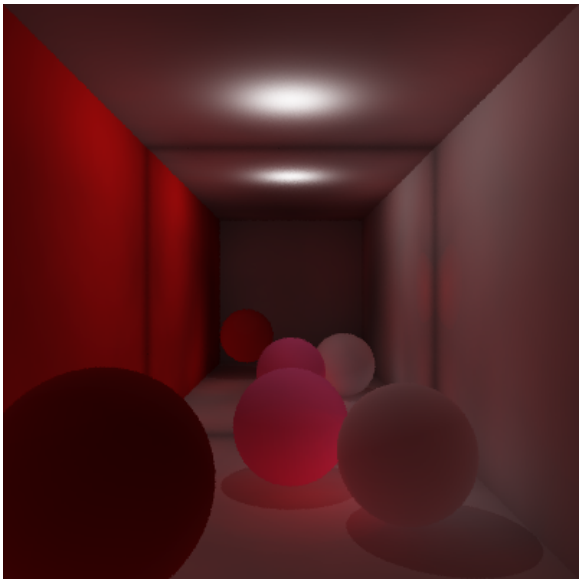


Figure 2.3: Example of an image created with Photon Mapping: Cornell box rendered with 10000 global photons, 20000 caustic photons, and 400-nearest neighbours. Back wall is a mirror, all everything is diffuse.

## 2.2 Radiance estimation

The radiance estimation is performed in the *shade* function. A ray is cast to the scene and once intersected direct and indirect light is computed. Direct light is computed accurately with classic next even estimation. Indirect light is calculated (approximately) with the radiance estimation (Equation 2.2), resulting in although pleasuring, biased images.

### 2.2.1 The Rendering Equation

**Question 2.1.**

***Discuss how photon mapping approximates the rendering equation, comparing their two forms. What terms are different, and why? What would be needed to compute the volume rendering equation (in participating media) with photon mapping? What would be the main differences?***

Photon Mapping and Path Tracing propose two different forms to calculate incoming radiance in the render equation, $L_i(x, \omega_i)$. Whereas path tracing precisely computes the radiance at a point $x$ calculating the incoming radiance by tracing rays from the point to its hemisphere, photon mapping uses the flux to perform an estimation, making use of photons map where flux information is stored.

$$L_o(x, \omega) = \int_\Omega L_i(x, \omega_i) f_r(x, \omega_i, \omega) |n \cdot \omega_i| d\omega_i \tag{2.1}$$

The approximation performed by photon mapping is based on the relation between flux and radiance, by substituting the term $L_i(x, \omega_i)$ with the incoming flux. This incoming flux is approximated by expanding a sphere from the point $x$ until it contains $n$ photons. Each of them is projected onto the surface and the density estimation is calculated using the area of the circle (projected sphere). With all of this, the render equation mutates to the following form:

$$L_r(x, \omega) \approx \frac{1}{\pi r^2} \sum_{p=1}^{N} f_r(x, \omega_p, \omega) \Delta\Phi_p(x, \omega_p) \tag{2.2}$$

This estimation is only valid for surfaces, as it is projecting the photons, assuming that all the photons are located on surfaces. In participate media, photons are stored in the whole scene, and volumetric estimations are needed to compute the in-scattered radiance. To tackle these issues the density estimation must divide by the volume of the sphere rather than by the circle's area. In this equation, the $f(x, \omega_p, \omega)$ has been substituted with the function phase, $p(x, \omega_p, \omega)$.

$$L_r(x, \omega) \approx \frac{1}{\sigma(x)} \sum_{p=1}^{N} p(x, \omega_p, \omega) \frac{\Delta\Phi_p(x, \omega_p)}{\frac{4}{3}\pi r^3} \tag{2.3}$$

### 2.2.2 Conservation of Energy

**Question 2.2.**

***How are you ensuring you are conserving energy during density estimation? Compare the amount of energy in the scene using only direct illumination and including global illumination: what is the ratio between them? Why?***
When photons are traced their energy is moduled according to their albedo, that is to say, the photon's energy will be multiplied by the albedo and divided by its average. The absorption chances are modelled with Russian Roulette, surfaces with high albedo will have more chances of survival for their photons. This will result in proper conservation of the energy in monochromatic scenes but it could make the energy to increase or decrease in scenes with materials with different color channels. Although the overall energy can vary slightly this approach is done with the aim of make all the photons to have similar energies, what will significantly improve the performance of the radiance estimation.

In the other part of the algorithm, the render itself, the energy is safely conserved by dividing the photons' flux by the number of emitted photons (taking into account the light source of each photon) and with the calculation of the density. When estimating the radiosity, we are estimating the density, dividing the summed flux by the area of the projected circle. No matter if a photon is picked several times by several spheres, as we are considering the density, the calculations will ensure that the energy is not being added twice.

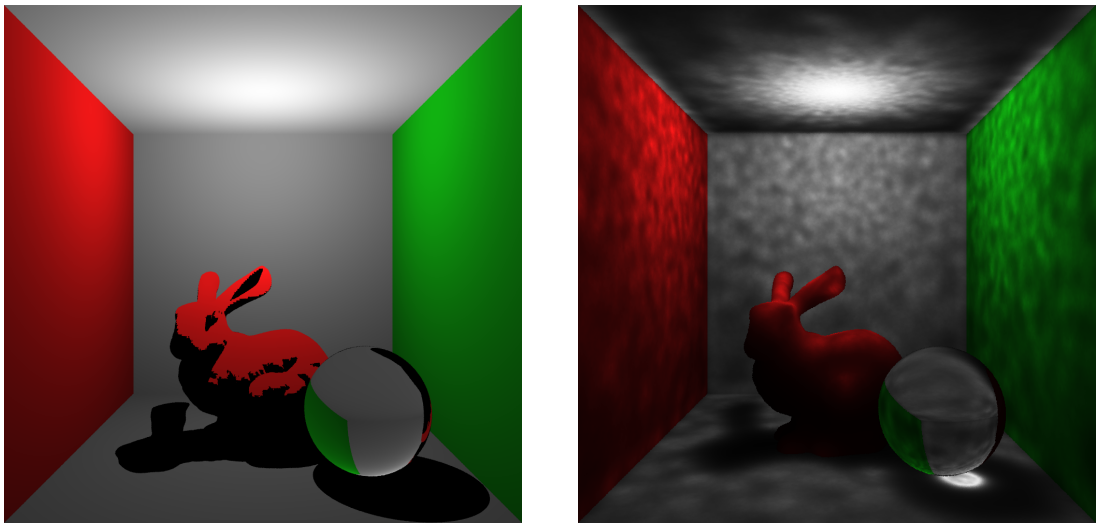
The overall energy of the image has been calculated as the summation of each pixel energy average, $r = \frac{\sum_i L_{D_i}}{\sum_i L_{I_i}}$ obtaining a rate of around 1.25, being direct light slightly bigger than indirect. In our opinion, the rate should be bigger, as, in images, the contribution of direct light is more important, but in the algorithm, and depending on the scene, this rate can change considerably due to for example a high albedo and its difference between channels.

### 2.2.3 Photon Mapping Bias

**Question 2.3.**

***Render two images of a scene with only direct illumination (DI), each image where DI is computed with ray tracing and with photon mapping respectively. What are the differences between ray traced direct illumination or direct illumination computed using photon mapping? Which one do you think is more accurate and why?***

Images are shown in Figure2.4. Ray-traced direct light is accurate whereas radiance estimation is obviously biased due to the fact that the expanded sphere is including photons from different places and even different surfaces. This clearly visible in the shadows. As only direct light is computed, shadows should be hard. Ray-traced image, Figure 2.4a, displays the shadows correctly. In contrast, with photon-based direct illumination, shadows exist but are much softer, Figure 2.4b. This is because the only difference between shadows and illuminated points is the size of the sphere (the density). Also the photon-mapped direct light renders a caustic (because caustic photons are followed until they reach a diffuse material) while it is impossible for the ray-tracer to do it.



(a) Direct light using ray tracing (b) Direct light using Photon Maps

Figure 2.4: Comparison between direct light calculated with ray tracing and using photon maps

**Question 2.4.**

***Why is photon mapping biased? What does that bias mean? What is the effect of that bias in the final image? Please discuss using the images rendered in the previous question.***

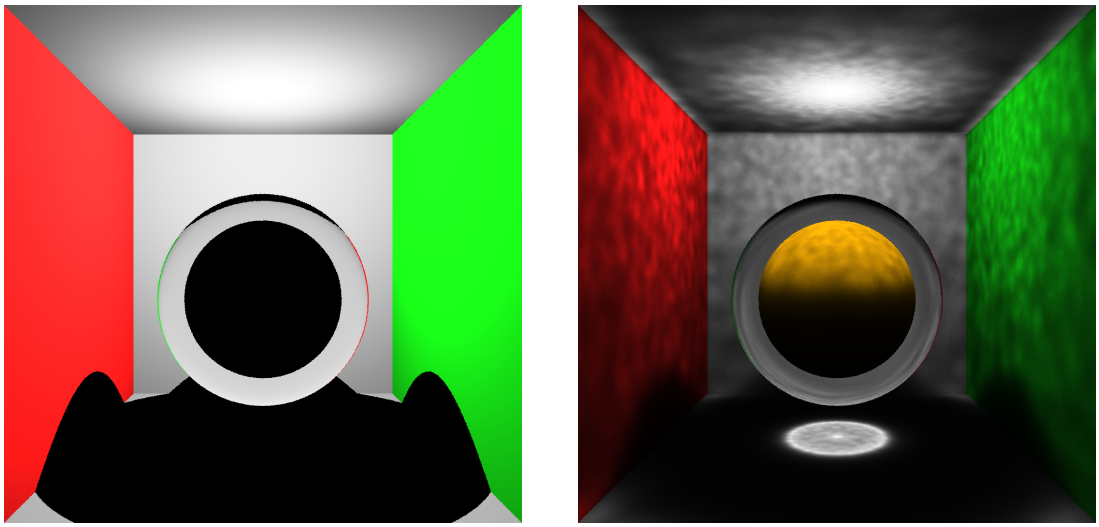
The algorithm is biased because it is assuming that the result returned by the radiance estimation belongs to the infinitesimal area at point $x$. Nevertheless, the expanded sphere contains points that could be far from x, and moreover, it may contain points belonging to other materials and surfaces, what is especially noticeable in the edges of the figures. The accuracy of the results will directly depend on the number of shot photons and on the number of photons used in the estimation. The more photons are shot the better the estimation will be.

This bias can be observed in the images in the shape of wrong color bleeding. For instance, it is more prominent on the edges of the walls, where a small line of fake color is noticeable.

**Question 2.5.**

***In the scene shown in Figure 1 you can see a diffuse torus inside glass, illuminated by a point light source. What would be the difference (in a converged scene) between rendering the direct illumination using ray tracing and using photon mapping? Render scene #4 (a simpler version of the scene of the figure with a diffuse sphere enclosed into a refractive sphere) with both ray traced and photon-based direct illumination and discuss the result.***

In this case, photon mapped direct only illumination will achieve a result closer to the global illumination as ray traced direct illumination fails to give light to the torus. This is due to the fact that, in ray tracing, the ray will be cast to the torus, but after it, the point will be marked as a shadow as the ray intersects with the glass cube in its path to the light. In contrast, photon mapped light can illuminate the torus as there are stored caustic photons on it (even with direct only flag when a traced photon interacts with a delta material its path is followed until it reaches a diffuse).



(a) Direct light using ray tracing

(b) Direct light using Photon Maps

Figure 2.5: Comparison between direct light calculated with ray tracing and using photon maps

In Figure 2.5 a similar situation is rendered with spheres. As expected, inner ball is absolutely black with the ray-traced light whereas it is illuminated in the photon-mapped one.

### 2.2.4 Performance of Materials

**Question 2.6.**

***Discuss how do you think photon mapping performs with very specular materials. For example, what would be the result of rendering a sphere shaded using Phong wiht varying degrees of specularity? Photon mapping is better for computing the rendering equation in diffuse or specular materials? Bonus point if answered with renders.***
It is way better for diffuse materials. Firstly, it is more efficient, obviously because you just shoot one ray and it ends, with specularity, you have to continue the path until you arrive at a diffuse material. Furthermore, a few photons are enough on a diffuse surface, as illumination on different points is slightly different. To represent highly specular materials, a big number of photons are required because two points, although close to each other, could have very different incoming luminances.

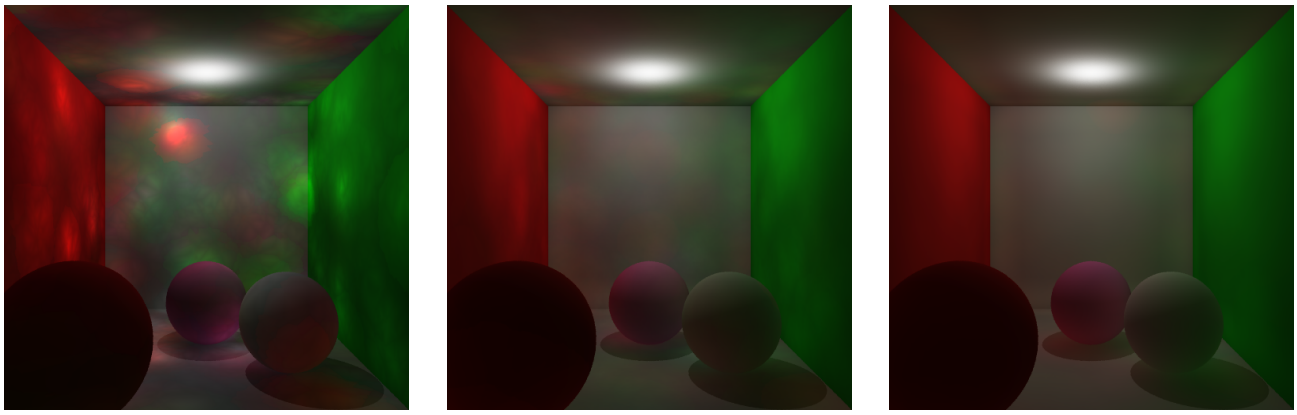
With Phong, it would perform worse. Firstly, if you do not do importance sampling considering Phong lobulus, your photons will have different energies, so the algorithm will perform worse. Even if you do so, you will have a degree of specularity, and therefore unless a considerable number of photons are shot, the algorithm will not model the complexity of the light in the specular materials.

## 2.3 Parameters analysis

In this section, an analysis of the effect of the different parameters of the algorithm is conducted and illustrated with renders. Although it is impossible to give the perfect parameters to use in general, because it is strongly dependent on the concrete scene (light, materials, position of the geometry...), knowing the effects and contribution of each of them is essential to understand the way the algorithm performs.

### 2.3.1 Number of photons

The number of photons stored by the maps is one of the main parameters to control convergence of the algorithm. More photons shot will make the image to converge more, as they will be more accurately distributed along the scene and, when estimating radiance, nearest neighbours are more likely to be closer than in a scene with less photons. The influence of the number of photons on a diffuse Cornell box (as well as the number of neighbours, maintaining the same ratio between them) can be seen in Figure 2.6.



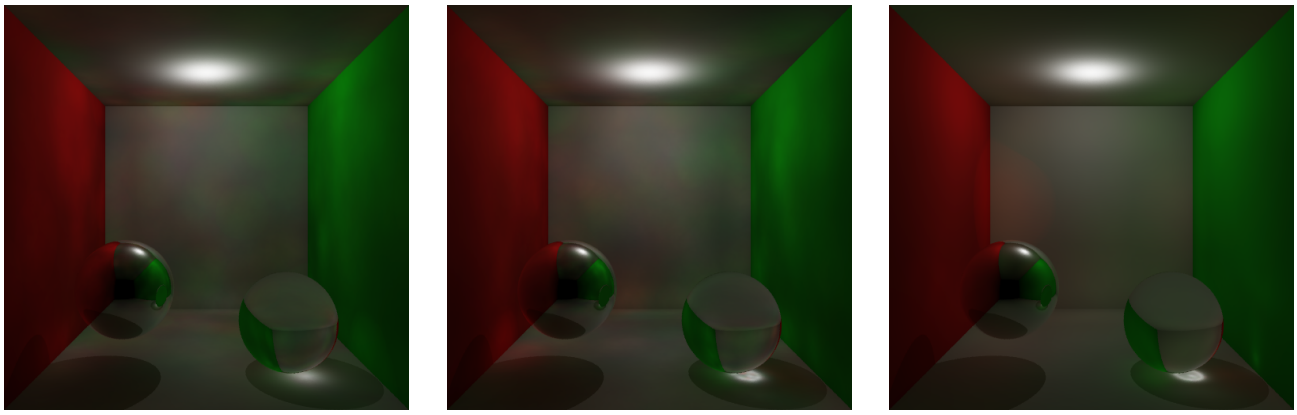
(a) 1000 photons stored. 10-nearest neighbours

(b) 10000 photons stored. 100-nearest neighbours

(c) 100000 photons stored. 1000-nearest neighbours

Figure 2.6: Cornell box with different number of photons. Illuminated with one puntual light and walls and spheres are all diffuse.

Another aspect to remark is, in a scene with both global and caustic photons, the relation between both of the numbers of photons. As it is shown in Figure 2.7, caustics will be better captured if many photons are stored, but a balance should be found between them because trying to get too many caustic photons can make the algorithm very slow.



(a) 1000 caustic photons: 10 times less the number of global ones. Notice caustic is too smooth.

(b) 10000 caustic photons: same number of global ones. Notice caustic is clearly observed.

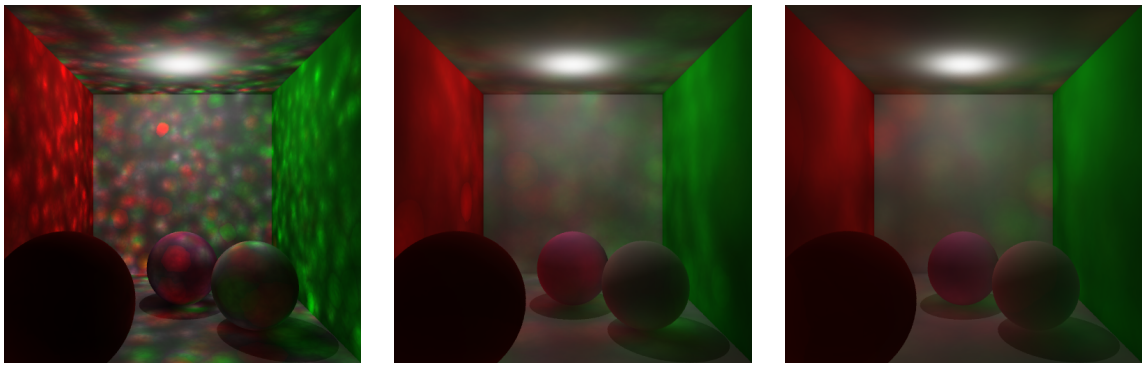
(c) 100000 caustic photons: 10 times more the number of global ones. Notice the bright dot on the right wall.

Figure 2.7: Cornell box with different number of caustic photons. All of them are shot with 10 000 global photons and radiance estimation with 500 neighbours.

Finally, maximum number of photons shot should be added to control the termination of the algorithm, but in general, we have tried to put it in a way that it permits to store all (or almost all) of the photons from both photon maps as in that case we are sure the algorithm is behaving as it should.

### 2.3.2 Number of neighbours

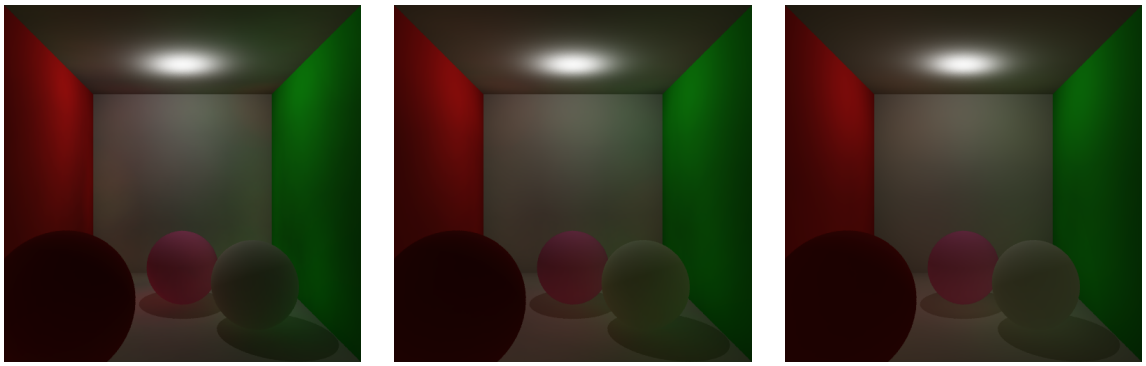
When doing radiance estimation, the number k of nearest neighbours used when taking photons from the map is also very important to be analyzed. It should be highly correlated with the number of photons shot, as in a scene with very few photons stored, it makes no sense to get a lot of them when estimation, because errors would be huge as it would be counting radiance from points that are way far from the point. On the other hand, if using a small number of neighbours when there are many photons in the scene, the radius of the circle would be so small that indirect illumination would be exagerated and hence getting those clearly seen circle areas, shown in Figure 2.8a. The correct k is very dependent on the scene, so it should be estimated for every case, although it is said to be good to use it between 50 and 500 photons. For diffuse Cornell box, an analysis of the neighbourhood effects can be seen in Figure 2.8.



(a) 10-nearest neighbours (not enough neighbours, too many and bright circles)

(b) 50-nearest neighbours (not enough neighbours, clear circles)

(c) 100-nearest neighbours



(d) 300-nearest neighbours

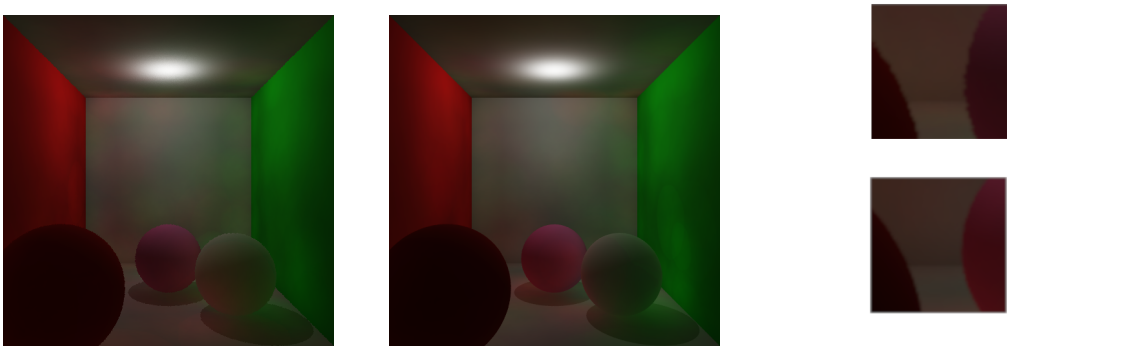
(e) 500-nearest neighbours (too many neighbours)

(f) 1000-nearest neighbours (too many neighbours, softed image)

Figure 2.8: Cornell box using different number of neighbours to compute radiance estimation. All of them are shot with 10 000 global photons.

### 2.3.3 Rays per pixel

Regarding rays per pixel, it only impacts in the second pass of the algorithm, when tracing rays from the camera. This parameter is not so important for the convergence as more rays per pixel does not imply that the algorithm performs better. It could compensate some mistakes from different calculations but could also make them bigger as computing something wrong many times. Tracing more rays per pixel and creating them through a random point inside each pixel helps with aliasing problems, as it can be slightly seen in Figure 2.9. Number of rays per pixel clearly increases rendering time, so it is not recommended to use many of them: 8 rpp has been found to be a good balance for most of the scenes rendered for this report.



(a) 1 ray per pixel. See the aliasing on the spheres

(b) 16 rays per pixel. Notice the softer bounds of spheres (antialiasing)

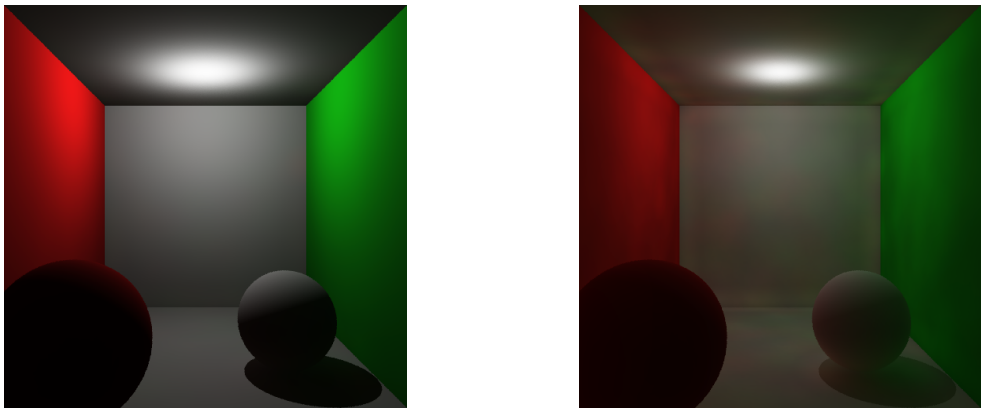
(c) Zoom-in 1rpp above and 16 rpp below

Figure 2.9: Cornell box rendered with 10 000 global photons and 160-nearest neighbours

## 2.4 Main challenges

During the development of the algorithm, many challenges were faced to correctly debug the code and deeply understand the algorithm and its characteristics. Some of the main or more curious ones and how they were solved are explained here, illustrated with renders.

First of all, many problems with conservation of energy were faced. At the beginning, the power of light was not being normalized by $4pi$ factor, and hence direct illumination was too much greater than indirect illumination and almost none of the global illumination effects were appreciated, as it can be seen in Figure 3.6. After it was solved, it was also detected that, when counting the number of photons shot to divide light energy by, maximum number of shots was used, which was not correct as not always so many photons were needed. To calculate it, a counter was added to the algorithm that is updated any time starting a random walk. It should also be taken into consideration that there should be different counters for each of the maps. The reason for that can be seen when, for instance, trying to find caustics in a scene which global photon map is already full, there, a lot of photons will be shot which will not be stored even if they had to, so they just should be counted as caustics shots but not global ones.

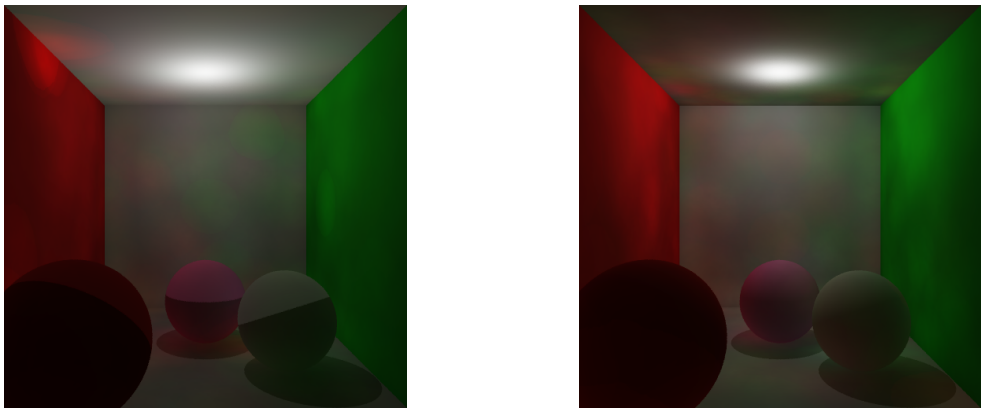


(a) Too much direct illumination. Global illumination effects cannot be seen

(b) Proper conservation of energy. Notice color bleeding on the back wall and shadows

Figure 2.10: Comparison of same scene with and without proper normalization. For both images, 100000 global photons are stored and radiance estimation is computed with 500-nearest neighbours.

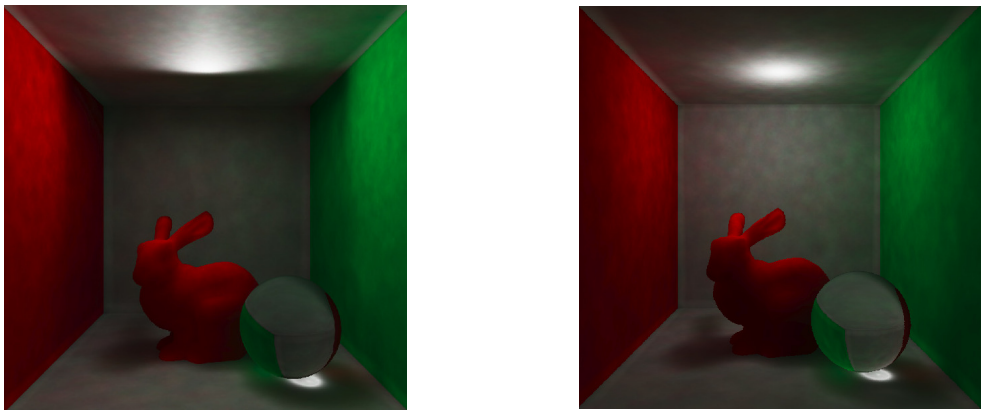
Another important error encountered was that, when calculating direct light, proper normalization and cosine of the angle between the normal and the ray from light source should be taken into account. It is just the application of Render Equation but in the first versions of the code, that cosine was forgotten and caused shadows to be harder and steep. It was solved by looking at the images and thinking about the possible causes of that appearance, to finally discover the bug. In Figure 2.11 a comparison between wrong and correct lambertian spheres is done.



(a) Wrong illumination (b) Correct illumination

Figure 2.11: Comparison of same scene with and without calculation of $cos(n \cdot w_i)$ for diffuse spheres with direct light. Notice the softer blurred shadows and the darker upper wall when doing correctly. For both images, 10000 global photons are stored and radiance estimation is computed with 100-nearest neighbours.

One of the main problems faced was related to correctly uniform sample the sphere. It was challenging because it was difficult to discover it, as images were acceptable but always seemed to be darker than expected at the background. Although something strange was detected it was not after a long time when, thanks to the visualization of only direct light, the error was discovered and solved, as it was just a bug when creating the ray from the angles in the solid angle sampling. Having rejection sampling also implemented served as a ground truth to compare and assure that sampling was finally correctly done. An example of a wrong result is shown in Figure 2.12.



(a) Wrong uniform light sampling (b) Correct uniform light sampling

Figure 2.12: Comparison of same scene with only direct illumination from Photon mapping, with a good and a bad sampling of the puntual light source. Notice the dark area created in the background, due to a lack of photons shot in those directions.

Finally, a simple but curious and time-consuming challenge that was faced was related to numerical errors. When calculating the second pass of the algorithm, while doing the loop that follows delta materials, ray obtained by sampling was not shifted. It was necessary to do so as, without it,

ray was continuously hitting itself because of numerical approximations. This caused a lot of rays to be ended by the maximum number of bounces, without contributing to the final illumination and hence creating those black points on the mirror sphere that is shown in Figure 2.13.

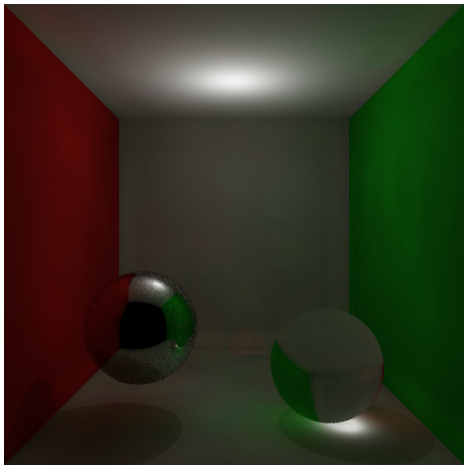


Figure 2.13: Scene with numerical errors caused by infinite intersection inside a delta material. Notice the black points on the mirror sphere.

## 2.5 Tonemapper

The available tone mapper included in the code did not work very well in many cases. In order to include our own tone mappers operators, the *Film* class has been overwritten.

The following operators have been implemented (copied from the Path Tracer report):

**Clamping:** Each luminance is at maximum a value $v$.

$$L'_i = \min(L_i, v)$$

**Equalization:** Each luminance is scaled with reference to the maximum luminance at the image.

$$L'_i = \frac{L_i}{\max L_j - \min L_j}$$

**Equalize and Clamping:** Mix the two previous approaches.

**Gamma curve:** Each luminance is transformed by a *gamma* function with parameter $\gamma$ after being equalized ($L_eq$).

$$L'_i = {L_{eq}}^{\frac{1}{\gamma}}$$

**Clamp and gamma curve:** The same gamma curve with a previous equalization and clamping (instead of just a previous equalization).

**Reinhard 02:** A tone mapper operator based on [4]. First, it computes the average luminance, $\bar{L}_w$, divides each luminance by this average and then applies a sigmoid-like function to correct the maximum luminance ($L_{max}$).

$$\bar{L}_w = \frac{1}{N} exp\left(\sum_i log(\delta + L_i)\right)$$

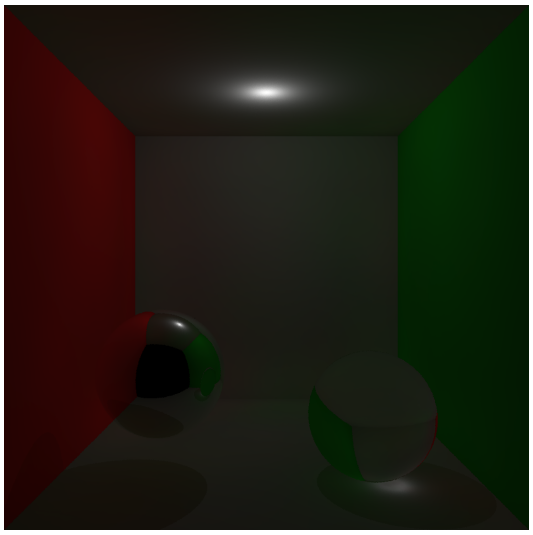
$$L'_i = \frac{a}{\bar{L}_w} L_i$$

$$L''_i = \frac{L'_i\left(1 + \frac{L'_i}{L^2_{max}}\right)}{1 + L'_i}$$

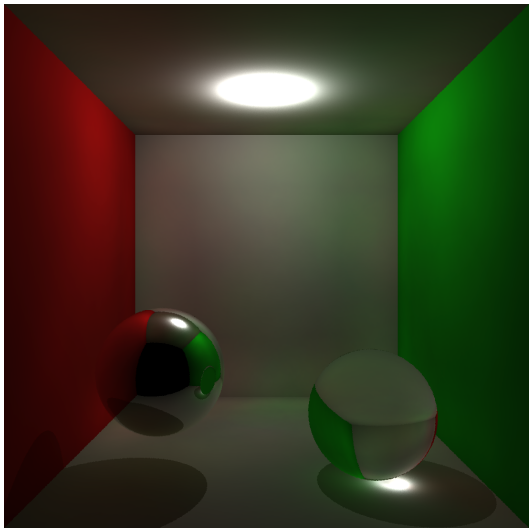
The value of $L_{max}$ in the last correction can be set manually, or set to the maximum scene luminance. This operator fit very well in the vast majority of the scenes, and for this reason, it is used by default in our render. More complicated scenes could require to tweak the $L_{max}$ value, but in most of the cases, the highest luminance works like a charm.

In Figure 2.14 different images of the same rendered scene but with different tone mapping operators are shown. It is a difficult scene as the light is directly visible, there are caustics, and light reflections on the glasses.

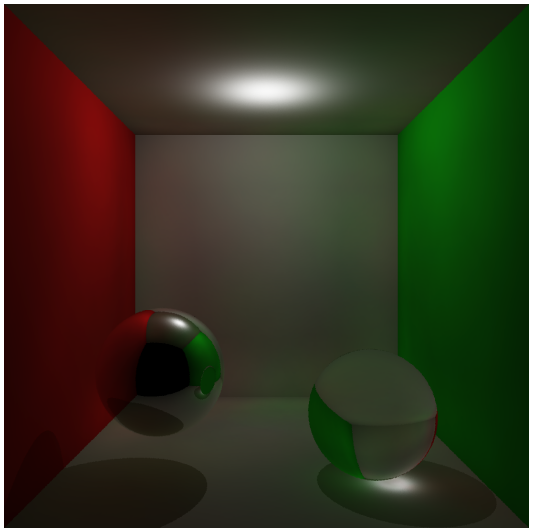
(a) Equalization



(b) Gamma, $\gamma = 2.2$



(c) Reinhard, $L_{max} = 1.5$



(d) Reinhard, $L_{max}$ the highest luminance

Figure 2.14: Comparison of tone mapper operators

# Chapter 3

# Extensions

## 3.1 Participating media

In this section the development of an extension of the algorithm to be able to render participating media is explained. Two approaches have been implemented to discuss and compare between them: homogeneous ambient scattering (*fake* fog used sometimes in videogames to simulate it) and a more realistic one using volumetric photon mapping, which accounts for homogeneous isotropic media.

Before explaining each of the approaches, a summary of the basics of light transport in participating media should be remarked, as well as notation that will be used below. When light travels through a participating medium the radiance may change as a result of four different types of interactions or events: absorption, emission, out-scattering and in-scattering, as shown in Figure 3.1.

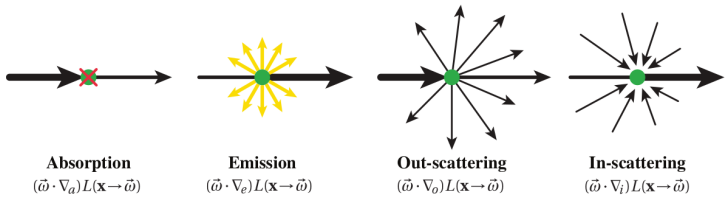



Figure 3.1: Four different events that can happen when interacting to participating media

Hence, the total change in radiance along the ray at a position x can be modelled with the Radiative Transfer Equation:

$$(\vec{\omega}\cdot\nabla)L(\mathbf{x}\rightarrow\vec{\omega}) = -\underbrace{\underbrace{\sigma_a(\mathbf{x})L(\mathbf{x}\rightarrow\vec{\omega})}_{\text{absorption}} - \underbrace{\sigma_s(\mathbf{x})\,L(\mathbf{x}\rightarrow\vec{\omega})}_{\text{out-scattering}}}_{\text{extinction}} + \underbrace{\sigma_a(\mathbf{x})\,L_e(\mathbf{x}\rightarrow\vec{\omega})}_{\text{emission}} + \underbrace{\sigma_s(\mathbf{x})\,L_i(\mathbf{x}\rightarrow\vec{\omega})}_{\text{in-scattering}}.$$

Figure 3.2: Radiative Transfer Equation

We will then distinguish between three different coefficients that characterize the medium: $\sigma_a(x)$ or the absorption coefficient, $\sigma_s(x)$ or the scattering coefficient and $\sigma_t(x)$ or extinction coefficient,

which is defined as $\sigma_t(x) = \sigma_a(x) + \sigma_s(x)$.

Another important concept for participating media is Transmittance. Transmittance gives the fraction of photons that can travel unobstructed between two points in the medium along a straight light an is given by:

$$T_r(x' \leftrightarrow x) = e^{-\tau(x' \leftrightarrow x)}$$

where $\tau$ is the result of integrating the effect of extinction along a line segment

$$\tau(x' \leftrightarrow x) = \int_0^d \sigma_t(x + t\vec{\omega})dt$$

Finally, the concept of phase function should be noticed, as it is the function which describes the angular distribution of light scattering at a point in the medium. In this report we will only consider isotropic phase function which is defined by: $p(x, \vec{\omega}' \leftrightarrow \vec{\omega}) = \frac{1}{4\pi}$, but the second approach used could also manage anisotropic phase functions.

Consequently, the computation of in-scatter radiance involves integrating incident radiance over the whole sphere of directions, multiplying radiance of each direction with the phase function as seen below:

$$L_i(\mathbf{x} \rightarrow \vec{\omega}) = \int_{\Omega_{4\pi}} p(\mathbf{x}, \vec{\omega}' \rightarrow \vec{\omega})\, L(\mathbf{x} \leftarrow \vec{\omega}')\, d\vec{\omega}',$$

Figure 3.3: In-scattering radiance

In general, and for simplicity, we will assume that participating media do not emit light, and hence it can just absorb (by absorption or out-scattering) or in-scatter it. First two events would hence, when having a participating media in front of an object, attenuate the amount of energy that arrives to it. On the other hand, in-scattering radiance will contribute to add energy to it. The final integral, known as the *volume rendering equation* is shown in Figure 3.5

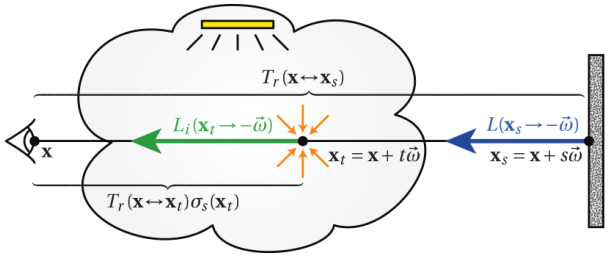



Figure 3.4: Graphical representation for volume rendering equation

$$L(\mathbf{x} \leftarrow \vec{\omega}) = \underbrace{T_r(\mathbf{x} \leftrightarrow \mathbf{x}_s)\, L(\mathbf{x}_s \rightarrow -\vec{\omega})}_{\text{reduced surface radiance}} + \underbrace{\int_0^s T_r(\mathbf{x} \leftrightarrow \mathbf{x}_t)\, \sigma_s(\mathbf{x}_t)\, L_i(\mathbf{x}_t \rightarrow -\vec{\omega})\, dt}_{\text{accumulated in-scattered radiance}}.$$

Figure 3.5: Volume Rendering Equation. Notice that it is a five-dimensional function over 3D positions in the volume and 2D directions over the sphere and, actually, general Render Equation can be seen as a four-dimensional subset of this one.

Taking everything already explained into consideration, it is clear that rendering participating media is very computationally expensive. Whenever a ray is traced to an object which is behind of a participating medium, all interactions with the media along the path should be considered. However, the real problem comes when, for each of the points of that path, interactions from all directions around it should also be considered, and it becomes recursively very expensive. To account for that problem, two approaches have been implemented and are discussed bellow.

### 3.1.1 Homogeneous ambient scattering

First of all, a simple approach was implemented by creating a *fake* fog, assuming in-scattered radiance to be an ambient constant (quite strong assumption). This means that, for all points of the medium, the in-scattered radiance is exactly the same. Consequently, instead of having to integrate over the whole medium, an arithmetic solution can be obtained just by adding an attenuation factor to the scene, which only depends on the distance. This attenuation factor causes the final radiance to be calculated as follows (from class slides):

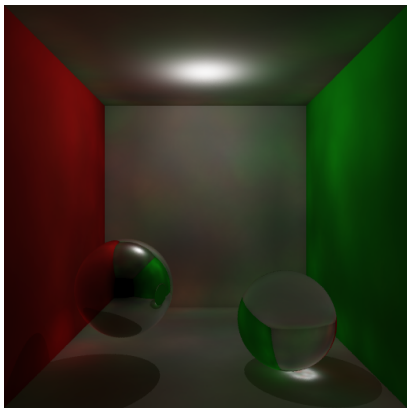
$$L(x, \vec{\omega}) = \sigma_s L_i \frac{1 - e^{-d\sigma_t}}{\sigma_t} + e^{-d\sigma_t} L(x_d, \vec{\omega})$$

where $\sigma_s$ and $\sigma_t$ are the scattering and extinction coefficients of the media (constant $\forall x$) and $L_i$ is the ambient in-scattering radiance.

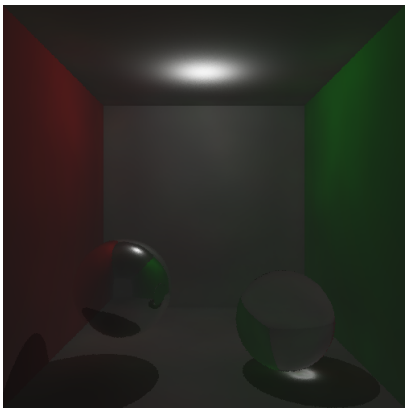
The ambient attenuation should be applied any time calculating energy in the algorithm. This means. firstly, that during photons shot, the energy of each photon should be attenuated taking into account its distance ($d$) to light source. After, when shading, attenuation should be considered first for direct illumination, again computing distance to light, and then for global illumination, considering all distance traversed from point to camera, which includes distance traversed through delta materials, where distance should be accumulated.

Although it has been implemented using Photon Mapping, this approach could also be adopted with other algorithms, such as Path Tracer, as it would just consist on applying the attenuation to every illumination result (both direct and indirect).

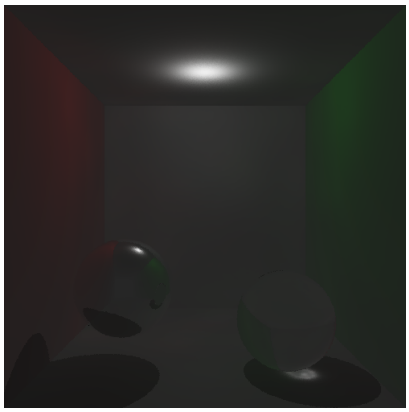
When adding an homogeneous ambient scattering to a normal diffuse Cornell box, resulting images are shown in Figure **??**. As it can be clearly seen, the only effect appreciated is similar to a blurred grey layer on top of the whole image, that is higher on further elements, but does not seem like a realistic fog. This effect, is used sometimes to avoid to compute far points and to hide some errors, especially in real-time applications where rendering time is a problem.



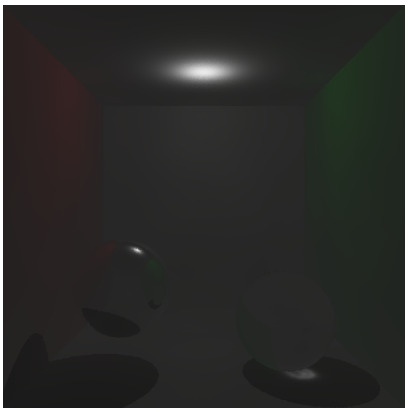
(a) Without participating media



(b) With ambient scattering: $\sigma_s = 0.1, \sigma_t = 0.2$



(c) With ambient scattering: $\sigma_s = 0.2, \sigma_t = 0.3$



(d) With ambient scattering: $\sigma_s = 0.25, \sigma_t = 0.4$

Figure 3.6: Comparison of same scene without and with ambient scattering participating media with different parameters. For both images, 10000 global and 10000 caustic photons are stored and radiance estimation is computed with 100-nearest neighbours.

### 3.1.2 Homogeneous isotropic scattering

Secondly, a more general and better approach has been implemented to compute participating media. It is based on the use of Volumetric Photon Mapping, as explained in [3], and the good point about it is that taking advantage of the use of photon maps, it is clearly more computationally efficient than other approaches such as path tracing. Homogeneous isotropic scattering has been implemented and is the only approach which is going to be detailed, although the same method could also manage heterogeneous media and anisotropic phase functions.

#### Photon tracing

For this phase of the algorithm, the most important issue is that a new map of photons has been added, known as volumetric photon map, which will not store photons from surfaces but from the whole 3D scene. It is simple to add we were already managing 3D photon maps for the basic algorithm.

First of all, photons are shot from light sources, like before, but now a probability of interacting with the participating media before reaching a surface should be added. This has been done by

firstly calculating a distance, given by the following formula, as explained in [2]:

$$d = -\frac{log(\xi)}{\sigma_t(x)}$$

where $\xi \in [0, 1]$ is a uniform random number. Once having defined a point inside the media, the probability of interacting with it is given ([3]) as:

$$F(x) = 1 - \tau(x_o, x) = 1 - e^{-\int_{x_o}^{x} \sigma_t(\xi) d\xi}$$

A random number in range [0,1] hence is used to decide whether the photon should or should not interact.

If doing so, Russian roulette decides whether the photon is absorbed or scattered based on the scatering albedo($\sigma_s(x_d)/\sigma_t(x_d)$). If scattered, the photon is stored in the volumetric photon map and a new ray is created from there, by uniformly sampling the sphere, with same method followed in light sampling. Finally, it has been decided to store both direct and indirect photons, as calculating direct light with ray tracing in the case of participating media would be too expensive.

**Radiance estimation**

To compute radiance estimation, ray marching is conducted in the path between the camera and the intersection. To do so, we use a fixed step distance, although a more complex adaptative ray marcher could be done. In each of the steps, in-scattered radiance is computed using the volumetric photon map, as follows:

$$L_i(x, \vec{\omega}) \approx \frac{1}{\sigma_s(x)} \sum_{p=1}^{K} p(x, \overrightarrow{\omega_p}, \vec{\omega}) \frac{\Delta\Phi_p(x, \overrightarrow{\omega_p})}{\frac{4}{3}\pi r^3})$$

As it is seen, it must be divided by the whole volume of the sphere, instead of the projected area from basic photon mapping, as in this case photons are stored in the 3D media.

Then, the radiance from the previous point is attenuated and the already calculated in-scattering contribution is added (emission is not being considered). That in-scattered radiance is approximated as being constant within each step. With this approximation, the radiance at point $x_k$ along a ray is iteratively computed as:

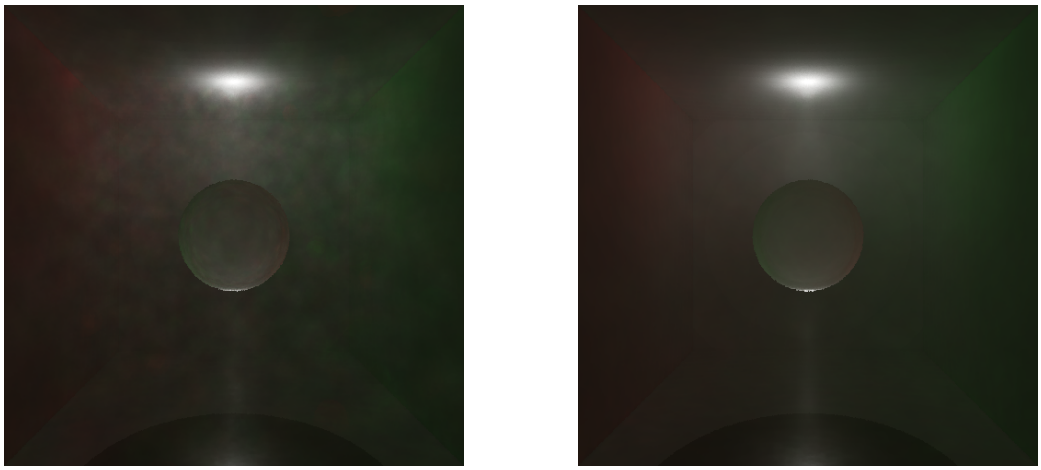
$$L(x_k, \vec{\omega}) = \sigma_s(x_k) L_i(x, \vec{\omega}) \Delta x_k + e^{-\sigma_t(x_k)\Delta x_k} L(x_{k-1}, \vec{\omega})$$

As explained before, direct photons were already included in the map so direct illumination is not calculated aside.

This ray marching is also computed any time hitting a delta material, and the result is accummulated to correctly take into account that traversed paths along the media.

Finally, when calculating direct light with ray tracing, the same ray marching is again computed, to account for media traversed within that path.

Notice that in all the formulas above, as an isotropic phase function, $p(x, \vec{\omega}' \leftrightarrow \vec{\omega}) = \frac{1}{4\pi} \forall x$ and as homogeneous media, $\sigma_t(x)$ and $\sigma_s(x)$ are constant $\forall x$.



(a) Fog using 50 neighbours

(b) Fog using 300 neighbours

Figure 3.7: Foggy scene with homogeneous isotropic media, $\sigma_t = 8$ and $\sigma_s = 0.5$. 1000000 global photons, 1000000 caustic photons and 1500000 volume photons have been shot. Notice the scattering around the light source and the caustic, which did not occur with ambient scattering.

Although it has not been implemented, anisotropic phase functions could be added. To do so, phase function will vary and so will the way of sampling next ray, as importance sampling based on that function should be done. In addition, heterogeneous media could also be managed, requiring more complex equations, as extinction, scattering and absorption coefficients would not be a parameter of the media but a function on each of the points, so would be included in the integral.

### 3.1.3 Challenges

Many challenges were faced during the development of this extension. Debugging participating media has been one of the most time-consuming tasks of this project. It should be remarked that, due to the high execution time and the limited computational capacities of our computers, all scenes modelled with participating media have been created with low resolution.

Firstly, it was attempted to calculate the points of interaction when shooting photons, by using ray marching, as proposed in the original paper. However, after implemented and tested, we realized that, as our ray marching was done with a fixed distance instead of an adaptative one, distances of interactions around a light were always the same, and so do probabilities of interaction (they are based on the distance and the extinction coefficient, which is constant), so a strange effect of concentric circles was created, as it can clearly be seen in Figure 3.8. To solve it, a different approach based on calculating a random distance was used, as explained before.

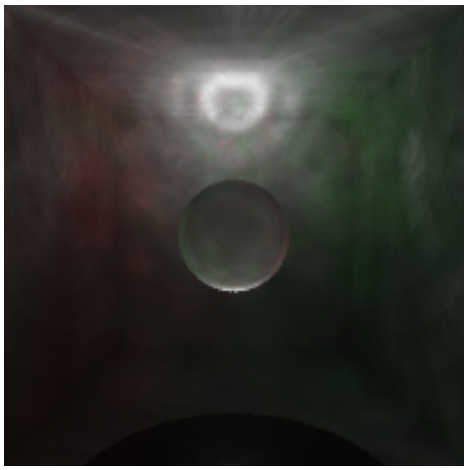
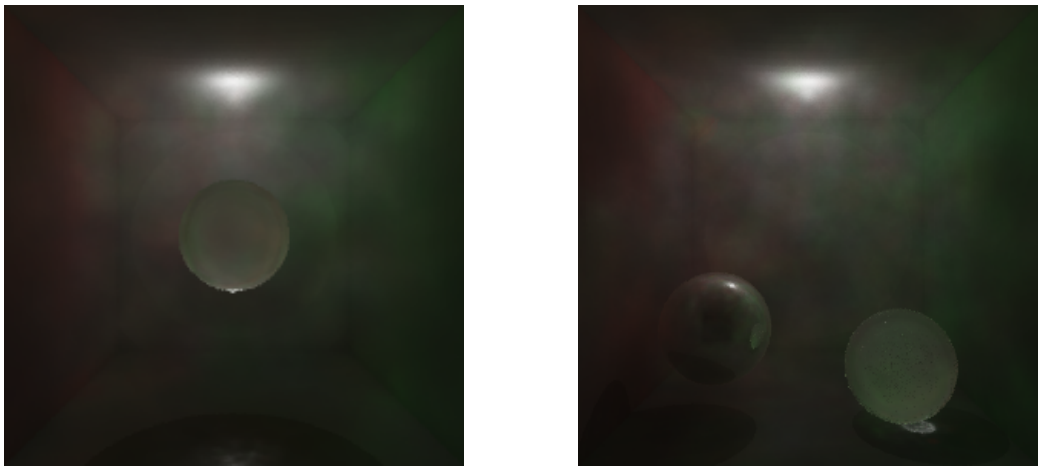


Figure 3.8: Foggy scene shooting photons with ray marching $\sigma_t = 0.9$ and $\sigma_s = 0.4$. 10000 global photons and 15000 volume photons. Notice the wrong concentric circles around light source.

Another remarkable issues can be seen in Figures 3.9a and 3.9b.

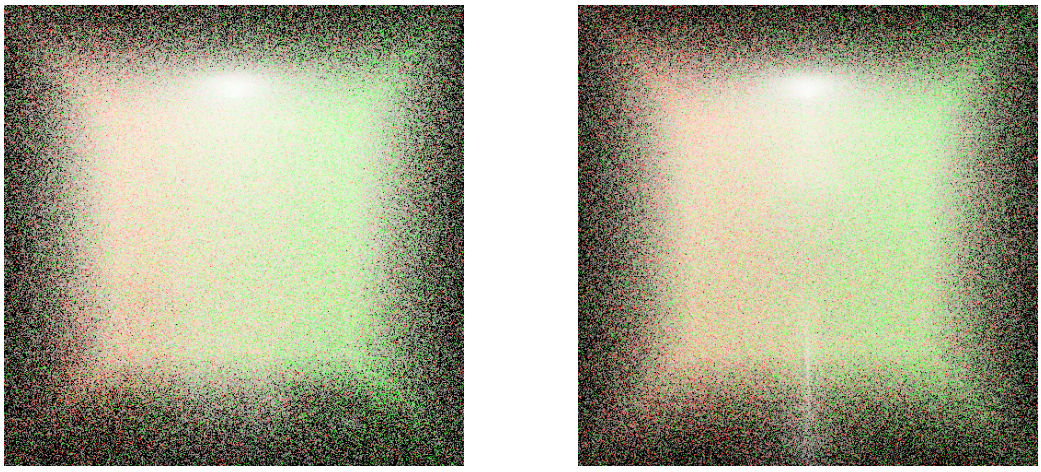


(a) Scattering around the caustic is missing

(b) Glass sphere is too bright

Figure 3.9: Foggy scene with homogeneous isotropic media, $\sigma_t = 1$ and $\sigma_s = 0.6$, 10000 global photons, 20000 caustic photons and 15000 volume photons have been shot

Trying to solve these issues, we implemented the visualization of the photon maps (Figure 3.10), to see if, for instance, in the first image (3.9a) it was a problem with sampling or fewer photons were stored around the caustic, or in the second image (3.9b) the glass sphere was not correct. Observing the maps, we realized we were considering the medium in the whole scene. All. Even inside the glass ball. This is a problem because when rays are inside the glass, if scattered, they will not follow the Delta function path, and therefore, no more photons will be concentred on the caustic area. Moreover, as there are volumetric photons inside the glass, and that ray marching is computed there, a lot of in-scattered light is added, what brings us the bright glass problem.



(a) Caustic photon map without a caustic

(b) Volumetric photon map with a caustic

Figure 3.10: Visualization of the photon maps by projecting the stored photons on the focal plane. There is a glass ball on the middle of the scene. Left image is incorrect and the right one is the corrected version. Notice how in the right image there is a concentration of the photons around the caustic area

Fortunately, we discovered the problem and managed to solve it. The solution was as simple as avoiding participating media calculations inside the glasses. A corrected image is shown in Figure 3.7.

Although these main problems have been solved in the end, approaches used are probably not the best. If having more time, participating media would have been deeper tested and debugged.

## 3.2 Kernel analysis

In this section an analysis of the behaviour of different kernels in radiance estimation is included. At the beginning, renders were estimated using a Uniform or Box kernel. However, a simple improvement can be done by considering that not all photons should contribute the same to the final estimation, as closer ones are more likely to be accurate than further points. With this consideration, the flux of each photon should be weighted based on the distance from its position to the point ($w_p$). Four different kernels are analyzed as follows:

### 3.2.1 Cone kernel

Firstly, the simple cone kernel calculates the weight for each photon as:

$$w_p = 1 - \frac{d_p}{kr}$$

with $d_p$ the distance from point $x$ to the photon $p$, $r$ the maximum radius when choosing the nearest neighbours and $k >= 1$ a characteristic kernel parameter. $k = 1$ has been used in the tests.

This kernel has to be normalized by $\left(1 - \frac{2}{3k}\right)$ based on the 2d-distribution, so the final radiance estimation is:

$$L(x, w) = \frac{\sum_{p=1}^{N} f_r(x, w_i, w)\Delta\Phi_p(x, w_i)w_p}{(1 - \frac{2}{3k})\pi r^2}$$

### 3.2.2 Gaussian kernel

Another typical approach is to use a Gaussian curve instead of a triangle. In this case the weight for each photon is calculated as:

$$w_p = \alpha \left(1 - \frac{1 - e^{-\beta \frac{d_p^2}{2r^2}}}{1 - e^{-\beta}}\right)$$

where $\alpha$ and $\beta$ are the parameters of the gaussian, and as discussed in **??** we have used $\alpha = 0.918$ and $\beta = 1.953$. The kernel is already normalized so the only change to the original radiance estimation formula is to multiply photon flux by this weight.

### 3.2.3 Epanechnikov kernel

Epanechnikov kernel is widely used to calculate density estimation in many applications, as for its mathematical properties it is optimal in a mean square error sense (specifically optimal when minimizing AMISE, Asymptotic Mean Integrated Squared Error, as explained in **??**) so other kernels are sometimes evaluated in comparison to it. The equation of the weight for the photons using Epanechnikov kernel is simple and efficient:

$$w_p = 2\left[1 - \left(\frac{d_p}{r}\right)^2\right]$$

As with Gaussian kernel, it is also already normalized.

### 3.2.4 Biweight kernel

Biweight (also called Quartic or Silverman) kernel is closely related to the Epanechnikov, and close to the optimal, but with higher order smoothness, as it is calculated as:

$$w_p = 3\left[1 - \left(\frac{d_p}{r}\right)^2\right]^2$$

### 3.2.5 Comparison

A graphical representation of the 2d-profile of each kernel can be seen in Figure 3.11, which helps to understand the way each of them performs. As seen in the image, Gaussian kernel is the smoothest one followed by Epanechnikov, with main difference between them in weight to further points (Epanechnikov is steeper than Gaussian). Cone and Silverman (Biweight) kernels have a bigger variation between weights for points, being Silverman smoother within those variations.

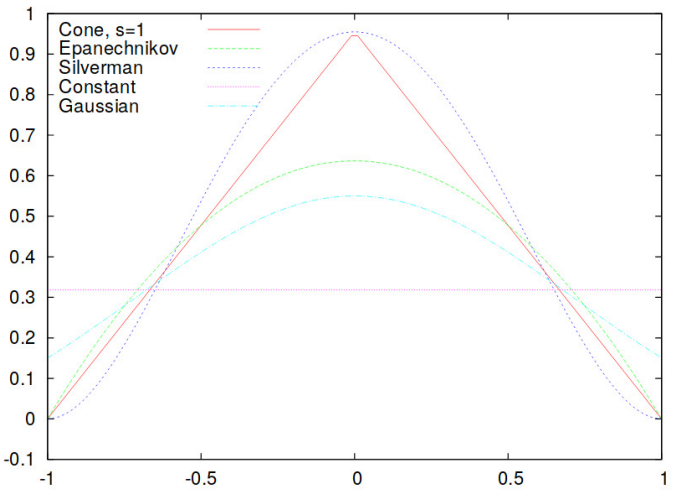
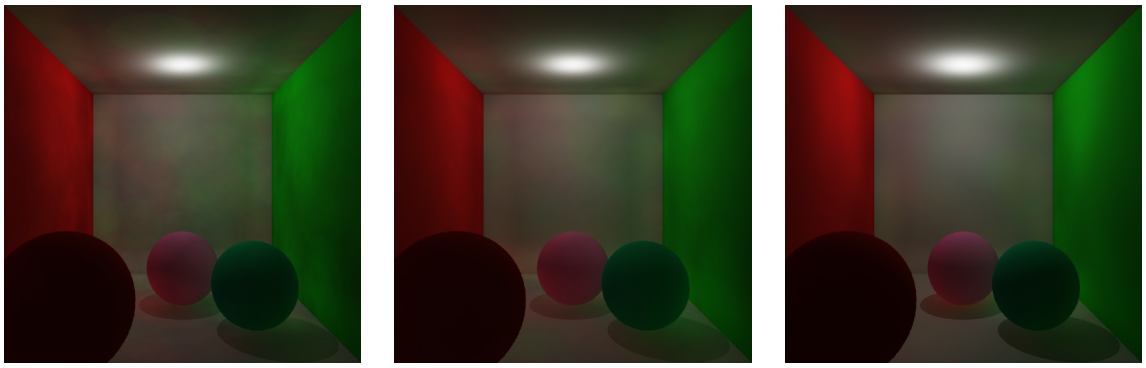
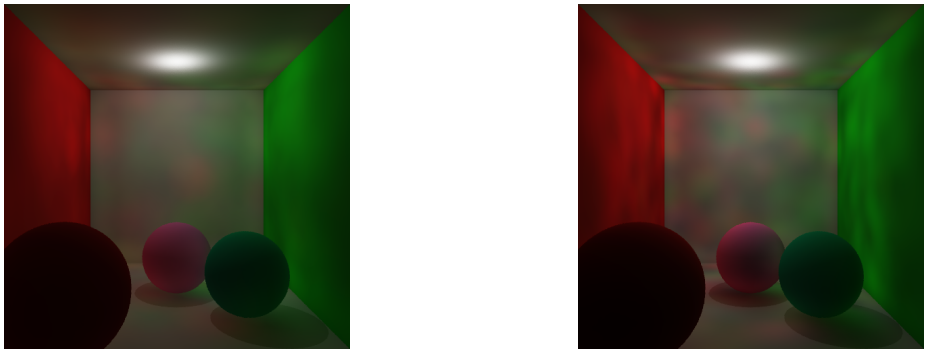



Figure 3.11: 2d-profile of different analyzed kernels. [1]

In Figure 3.12 we present the same scene, a simple diffuse Cornell box, rendered using the different kernels. As seen, differences are very slight and can be specially appreciated in corners. Firstly, it is appreciated that any of them is better than the uniform kernel, as they give more importance to closest neighbours which are likely to be important. With these results, it can be said that smoother kernels (Gaussian and Epanechnikov) better approximate the radiance. Although Epanechnikov filter is theoretically optimal when doing general density estimation, Gaussian kernel seems to perform better, giving a more diffuse, hence realistic, appearance to the image. Regarding Biweight kernel, it seems to have less recognizable patterns (such as for example, bright lines on corners when using box kernel) but does not perform very well, as some recognizable incorrect spots (a bit similar to the effects when using not enough neighbours).



(a) Box kernel (b) Cone kernel (c) Gaussian kernel



(d) Epanechnikov kernel (e) Biweight kernel

Figure 3.12: Cornell box using different kernels when computing radiance estimation. All of them are shot with 10 000 global photons and estimated with 300-nearest neighbours

As a conclusion, differences between kernels, at least in simple Cornell boxes scenes where it has been tested, are not very important, although having a better kernel such as the Gaussian makes bias smaller even if it does not dissapear. More challenging scenarios, with high differences in radiance level closer to each other would be better to develop in order to make a deeper analysis of their performance.

# Chapter 4

# Conclusions

In this assignment, a photon mapping render has been developed to calculate the global illumination of the scenes. Now we can render images having illumination effects that would be very difficult (or even impossible) to achieve with simple path tracing. Photon Mapping permits to generate an approximate result that, although biased, allows to model more complex light events.

This algorithm has the advantage of start being pleasant to the eyes with a smaller execution time than path tracing. With a decent amount of shot photons, it is possible to render images that are close to the reality with the acceptable tradeoff of having a bit of fake color bleeding and some small inconsistencies in global illumination. In contrast, path tracing requires a huge number of cast rays in order to remove the noise, and although correct, it lasts to converge.

During this assignment, efforts have been focused on complex light transport phenomena rather than on the beauty of the rendered scenes (in contrast with path tracing, where different geometries and textures were added, and artistic designs were rendered). For this reason, the included extensions are participating media, simulating the complicated interactions of the photons along the fog, and advanced filters, to improve the results of the radiance estimation.

We must accept that the outcomings do not fulfill all of our expectations, as the time was a limiting factor to our objectives. The rendered scenes are not on a par with the path tracing ones and we would have liked to dig deeply on participating media. Anyway, we consider that after implementing it, we have deeper understood the way photon mapping performs, as well as a wide idea of general rendering algorithms for global illumination.

# Chapter 5

# Workload

As we have made use of the provided code, in this assignment, the harder task was to fully understand the concepts in order to correctly apply them. Once we have implemented it, a big amount of hours were focused to debug, especially participating media. The code was almost fully implemented together.

The hours spent in each part of the assignment are the following:

| Task | Time lasted |
|---|---|
| Understand the basis and smalls trials | 30 h |
| Basic implementation | 60 h |
| Kernels | 6 h |
| Participating media | 40 h |
| Report and renders | 45 h |
| Total time | 181 h |

Figure 5.1: Time spent on each part of the project