# Ray Tracer

Juan Lorente Guarnieri

# Render Equation

The **Render Equation** describes how the interaction of light in a three-dimensional scene is simulated.

$$L_o(\mathbf{x},\omega_\mathbf{o}) = \int_\Omega \underbrace{L_i(\mathbf{x},\omega_\mathbf{i})}_{\text{INCOMING LIGHT}} \; \underbrace{f_r(\mathbf{x},\omega_\mathbf{i},\omega_\mathbf{o})}_{\text{BRDF}} \; \underbrace{|\mathbf{n}\cdot\omega_\mathbf{i}|}_{\text{GEOMETRY}} \; d\omega_\mathbf{i}$$

$$\underset{\text{Outgoing light}}{\boxed{L_\mathrm{o}(X,\hat\omega_\mathrm{o})}} = \underset{\text{Emitted light}}{\boxed{L_\mathrm{e}(X,\hat\omega_\mathrm{o})}} + \int_{\mathbf{S}^2} \underset{\text{Incoming light}}{\boxed{L_\mathrm{i}(X,\hat\omega_\mathrm{i})}} \; \underset{\text{Material}}{\boxed{f_X(\hat\omega_\mathrm{i},\hat\omega_\mathrm{o})}} \; \underset{\text{Lambert}}{\boxed{|\hat\omega_\mathrm{i}\cdot\hat n|}} \; d\hat\omega_\mathrm{i}$$

$L_\mathrm{o}(X,\hat\omega_\mathrm{o})$ It represents the outgoing radiation (or light) from $X$ (a point in the scene can be seen as the point you are looking at) to $\widehat{w}_o$ (Exit Direction)

$L_\mathrm{e}(X,\hat\omega_\mathrm{o})$ In my case, I have omitted the emitted radiation because it is assumed that there is no direct emission at point X.

$L_\mathrm{i}(X,\hat\omega_\mathrm{i})$ Given a point and a direction, what light comes from that direction

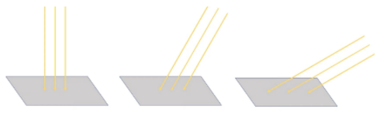
P For incident light, simply iterate through all the lights to check if there is direct light from the intersection point:

$$\sum_{i=1}^{n} \frac{p_i}{|\mathbf{c_i}-\mathbf{x}|^2}$$

It is important to mention that only direct light is added, if it is a diffuse type object, otherwise it is not taken into account.

$f_X(\hat\omega_\mathrm{i},\hat\omega_\mathrm{o})$ Reflectance function: describes how incident light $w_i$ is reflected in the direction $w_o$ at point $X$ depending on the type of surface.

$|\hat\omega_\mathrm{i}\cdot\hat n| \; d\hat\omega_\mathrm{i}$



It models how light is distributed over the surface depending on the angle of incidence.

For the material properties (BRDF), different constants are stored in each geometry to represent the percentage of each variable being red the diffuse component, green as the specular component, and blue represents the refraction component.

$$\boxed{f_r(\mathbf{x},\omega_\mathbf{i},\omega_\mathbf{o}) = k_d\frac{1}{\pi} + k_s\frac{\delta_{\omega_r}(\omega_\mathbf{i})}{\mathbf{n}\cdot\omega_\mathbf{i}} + k_t\frac{\delta_{\omega_t}(\omega_\mathbf{i})}{\mathbf{n}\cdot\omega_\mathbf{i}}}$$

And finally, for the geometry, the cosine term between the normal and the exit ray is used.

$$\boxed{f_r(\mathbf{x},\omega_\mathbf{i},\omega_\mathbf{o})|\mathbf{n}\cdot\omega_\mathbf{i}|}$$

In general, our **path-tracer**:

1) Starts a loop to go through each pixel (parallelization by rows).
2) In each pixel, it launches multiple rays (anti-aliasing). When launching rays, using Monte Carlo estimation implies adding multiple random trajectories. The more trajectories we use, the better our approximation of the integral, which translates into a more accurate result.
3) It performs the intersection of the ray with the geometries in the scene, finding the nearest intersection.
4) If there is an intersection, it determines the type of bounce (diffuse, specular, etc.), for this Russian Roulette has been used.
5) It calculates the contribution of light, taking into account the type of bounce.
6) If there is a bounce and the multiplication of the BRDF is significant, it makes a recursive call to continue tracking the ray's trajectory. It must be taken into account that for the path tracer it is necessary to accumulate the product of the BRDF and also the product of the cosine terms for the secondary ray that is carried out.

For **Photon Mapping**, which is a past algorithm, the following is done:

1. <u>Trace the virtual photons from the light source, their scattering on surfaces, and the cache</u>

During this phase, virtual photons are simulated emanating from the light sources (the number of emitted photons is proportional to the power of the light source) and scatter on the surfaces of the scene. These virtual photons carry information about their position, direction, and luminous flux. This information is stored in a photon map, which serves as a kind of "cache" of indirect lighting information.

2. <u>Ray tracing of the scene and use of photons to calculate indirect illumination</u>

During this phase, conventional ray tracing is performed in the scene. When a ray hits a surface, the photon map generated in the first pass is used to calculate the indirect illumination at that point. The photons stored in the map contribute to the global illumination based on their position, direction, and luminous flux. In other words, the incident radiance, unlike the path tracer, comes from the information stored in the photons, so the Render Equation will require finding the k nearest neighbor photons (the closest ones) to each point being illuminated, and use them to compute the reflected radiance at that point.

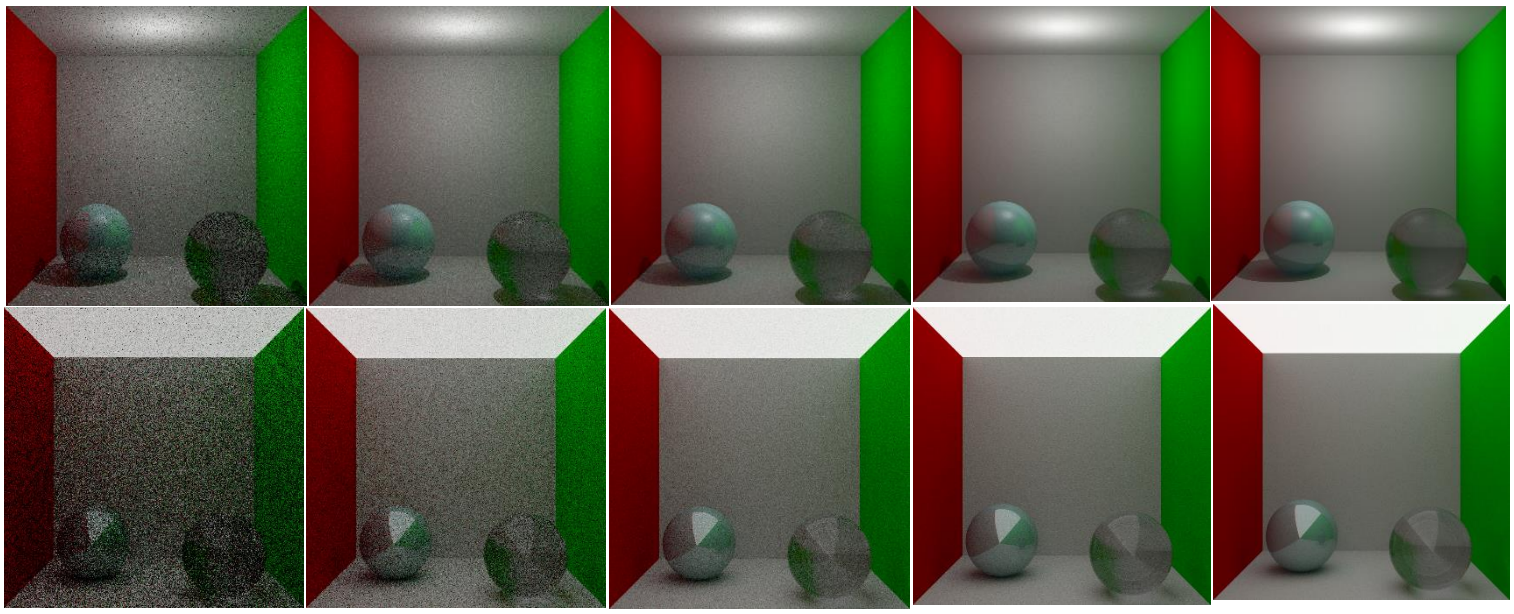
$$L_o(\mathbf{x}, \omega_\mathbf{o}) \approx \sum_{p=1}^{k} f_r(\mathbf{x}, \omega_\mathbf{p}, \omega_\mathbf{o}) \frac{\Phi_p}{\pi r_k^2}$$

There are multiple cases where photons should not be stored, nor should the neighboring photons in the photon map be searched for.

1. Perfectly specular and transmissive surfaces.
2. Direct light (in case of using next-event estimation)

## Path Tracer

The following shows the example of the Cornell Box with 2, 8, 32, 128, and 512 samples, for point light and area light:



**a) How quickly does path tracing converge with respect to the number of paths per pixel?**

From 128 samples per pixel, it begins to converge in such a way that noise starts to be barely detectable. This is due to the noise generated by Monte Carlo being due to the great variability of paths that can be caused by hitting a diffuse surface.

If we define a desired reduction in noise (for example, halving the visible level of noise), we can estimate the necessary number of samples. Suppose that with N samples, we have a noise level R. To reduce this noise to R/2, we would need approximately 4N samples.

$$\hat{I} = \frac{1}{N}\sum_{i=0}^{N}\frac{f(x_i)}{p(x_i)} \qquad \mathrm{Var}\left(\sum^{N} aX\right) = a^2\sum^{N}\mathrm{Var}(X)$$

$$\mathrm{Var}(\hat{I}) = \mathrm{Var}\left(\frac{1}{N}\sum^{N}\frac{\mathrm{f}(X)}{\mathrm{p}(X)}\right) = \frac{1}{N^2}\sum^{N}\mathrm{Var}\left(\frac{\mathrm{f}(X)}{\mathrm{p}(X)}\right)$$

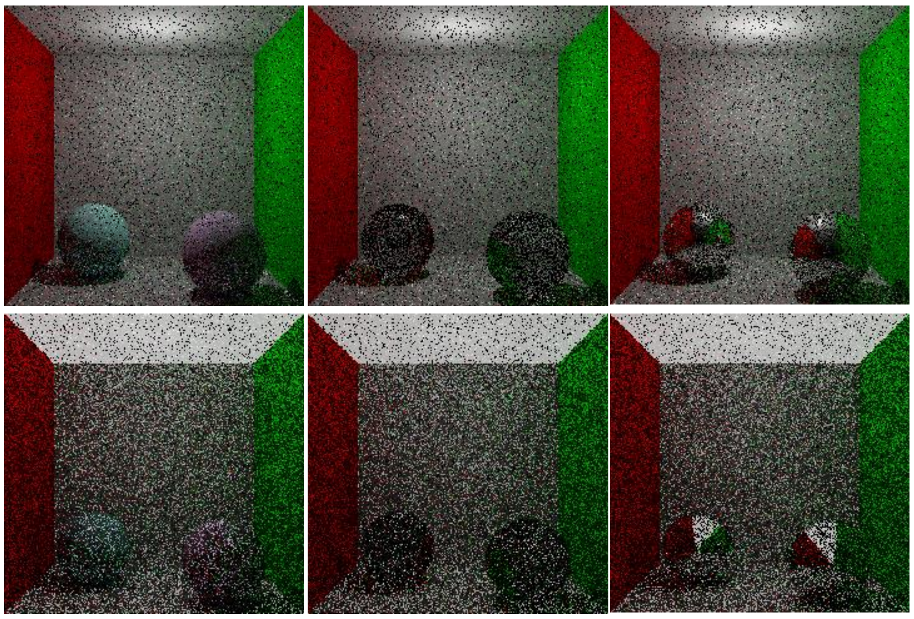
$$= \frac{1}{N^2}N\,\mathrm{Var}\left(\frac{\mathrm{f}(X)}{\mathrm{p}(X)}\right) = \frac{1}{N}\,\mathrm{Var}\left(\frac{\mathrm{f}(X)}{\mathrm{p}(X)}\right)$$

As the number of samples (N) increases, the accuracy of Monte Carlo improves, as the variance decreases.

**b) Of the implemented materials, which ones make convergence slower? In what circumstances? Why?**

Refractive materials, since light rays can experience multiple refractions and reflections within the transparent material, which requires more ray paths to converge. In addition, they can cause significant changes in the direction of light rays depending on the angles of incidence and refraction, which can cause some rays to follow more complicated paths, thereby increasing the number of paths needed to obtain an accurate estimate.

In contrast, diffuse materials tend to converge more quickly, as they reflect light uniformly in all directions, which facilitates the capture of light information from the scene.

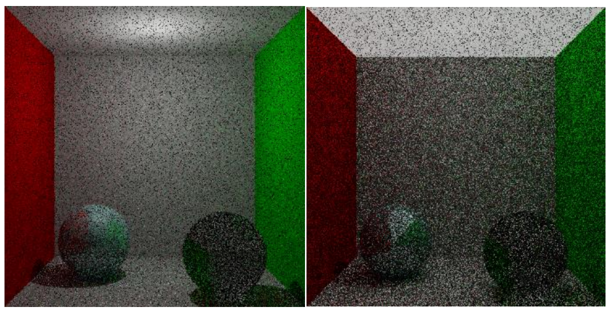


In these examples, it can be seen how using the same number of pixels, only varying the material of the spheres, great differences between them can be observed.

**c) What light sources make path tracing slower? Area lights or point lights? In what circumstances? Why?**

Area lights tend to make path tracing slower, this is because they have a finite size, which means they can emit light over a larger area, that is, more paths must be followed to trace the light from an area source than from a point source.

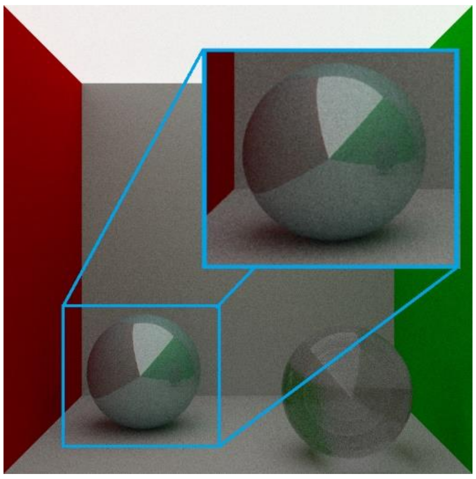
This is more noticeable when area lights are large, as the larger the area they illuminate. Or when they are close to objects as it can bounce off the object and reach the camera from different directions.



To demonstrate this, the following images show the same scene, on the left using point light and, on the right, using area light. As can be seen in the image of the area light, there is much more noise compared to the other.

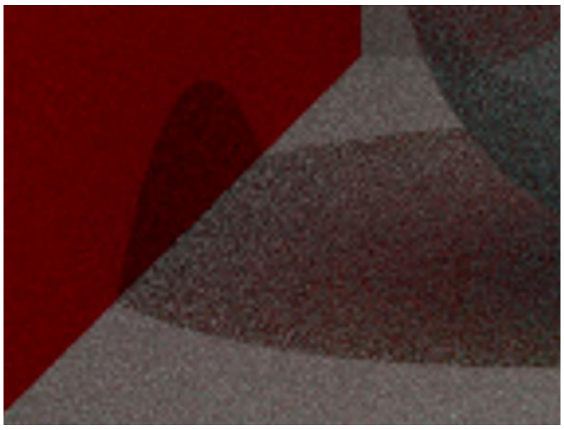
## Effects easily obtained with path tracing::

○ **Soft Shadows**: Path tracing excels in creating soft shadows, as it realistically simulates light scattering. This is achieved by calculating the path of light from multiple points, from an area light source, resulting in softer and more blurred shadow edges.



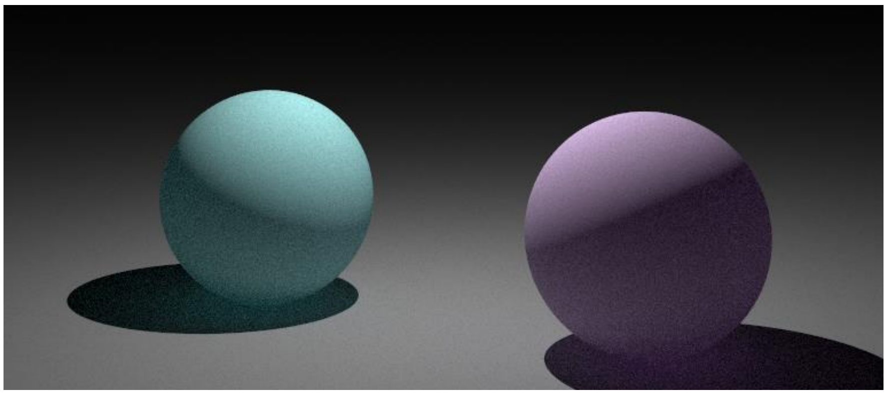
In this render, the shadow of the sphere is very softened due to the presence of area light (the ceiling).

○ **Color Bleeding**: This effect occurs when light reflected from a colored object illuminates and tints neighboring objects. Path tracing captures this phenomenon naturally by simulating how light scatters and mixes in an environment.



In this zoomed-in render of the Cornell Box with point light, it is noticeable how the floor is tinted red due to the adjacent wall bouncing part of the light that hits it.

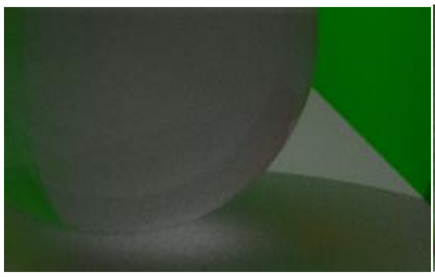
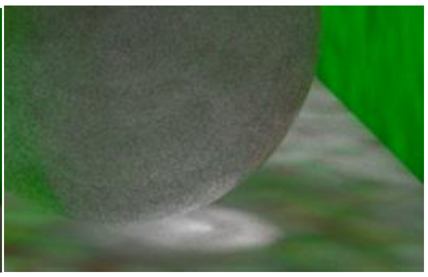
○ **Hard Shadows**: Hard shadows are created when a point light source illuminates an object, and in path tracing, simulating a precise point light source, although not perfect, can be achieved as long as there isn't much color bleeding from nearby objects.



In this render, the hard shadows originating from the only point light present are better appreciated.

## Effects difficult or slow to obtain with path tracing:

○**Caustics**: Although path tracing can generate caustics (patterns of concentrated light created when light passes through transparent or reflective surfaces), it often lacks precision and efficiency. This is due to the need to correctly trace specific and concentrated light paths, which often requires a large number of samples and, therefore, longer convergence time.

**Path tracer** **Photon Mapping**

It can be noted that caustics are much more easily obtained with other techniques.

## Scenarios

**For each of these effects, different scenarios are required:**

○**Soft Shadows:**
Required Scene: Area light sources, for example, on a ceiling to make it noticeable, or in a sphere surrounding the scene.
Material Properties: Opaque materials that can block part of the light (with null refractive component, to prevent light passage and affect visualization).
Geometric Distribution: Objects positioned so that part of their surface is illuminated while another part is in shadow.

○**Color Bleeding:**
Required Scene: Brightly colored objects near neutral surfaces.
Material Properties: Diffuse materials that reflect light in a scattered way.
Geometric Distribution: Objects placed so that light reflected from a colored object can reach other objects, like a bright-colored surface illuminated next to a surface without direct light.

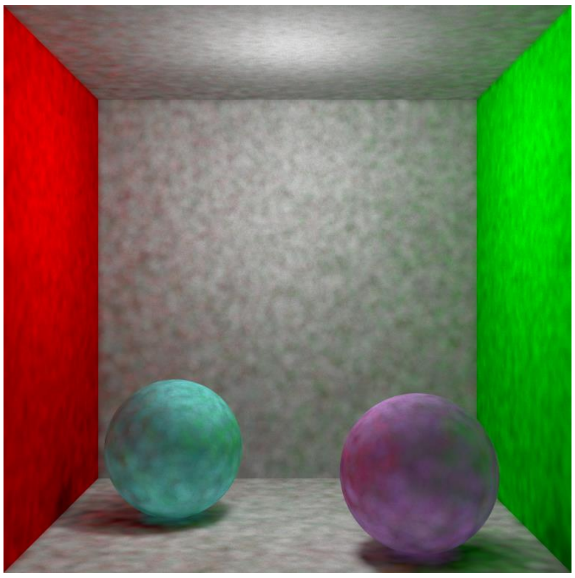
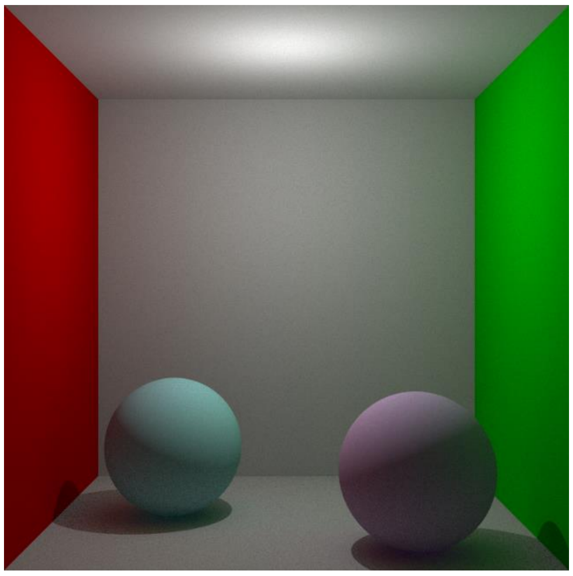
○**Caustics**:
Required Scene: Transparent or reflective objects (like water or glass).
Material Properties: Materials that significantly refract or reflect light.
Geometric Distribution: A light source that can pass through or reflect in these materials to create concentrated light patterns, such as a sphere.

○**Hard Shadows:**
Required Scene: Direct and point light source.
Material Properties: Any material that blocks light.
Geometric Distribution: Object and light source aligned to create a defined and clear shadow.

## Photon Mapping

Comparison of direct illumination obtained with next-event estimation (shadow rays to point lights) and with photons stored in the photon map.

What effects are easily obtained with the photon map? What problems does the photon map present when used to estimate direct illumination?

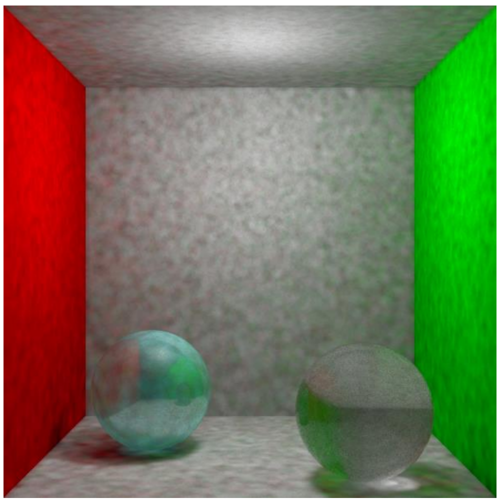
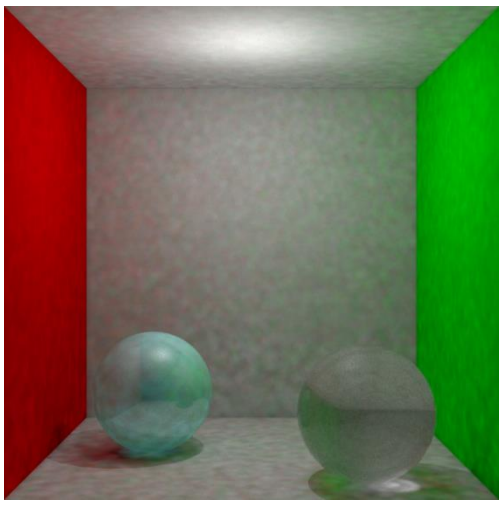



Photon Map Next Event Stimation

Direct illumination is the light that reaches a point in a scene directly from a light source. It can be estimated in two ways: through Next Event Estimation or using a photon map.

As can be seen in the image, photon noise is more evident in soft shadows and on diffuse surfaces. This is because soft shadows require a large number of photons to be accurately represented, just like diffuse surfaces, which also require a large number of photons to produce a uniform color.

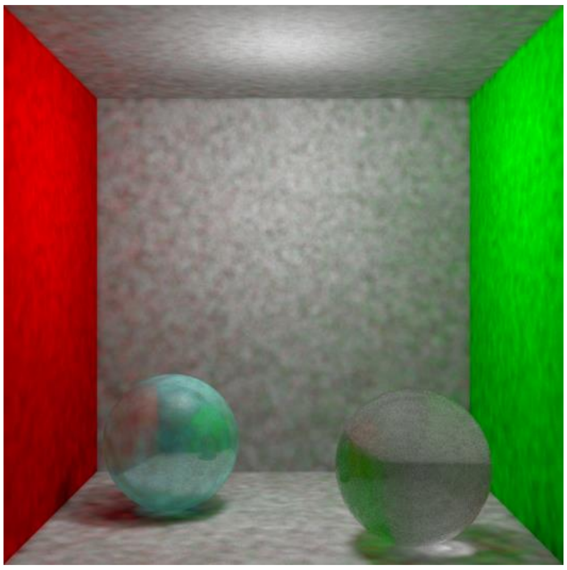
It consists of not storing the first collision of the photons, so that the light to be calculated in the rendering is the direct light from the first intersection (using next-event-estimation) and the secondary rays with photon mapping:

1st Intersection with Photon Mapping 1st Intersection Next-Event-Estimation

As can be seen in the images, when we use next-event-estimation, we get more defined shadows, as it does not rely exclusively on photons stored in the photon map, but directly incorporates light from light sources. This allows capturing the direct contribution of light to the scene, especially in areas where there may be few or no photons stored.

3. **Analyzing the effects of delta BSDFs in Photon Mapping. What lighting and appearance effects come from photon mapping? What effects come from the ray tracing part of the algorithm?**



Delta BSDFs are a type of BSDF that only have one output value, regardless of the direction of light incidence. This means that light is reflected or refracted in one direction, regardless of how it hits the surface.

The lighting and appearance effects that can be obtained with delta BSDFs in photon mapping are:

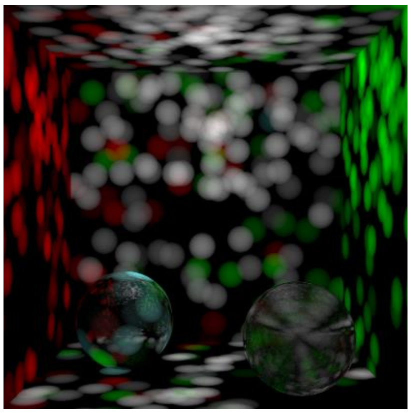
- Caustics: Efficient for simulating caustics, such as patterns of bright and focused light in specific areas of the scene.
- Color Bleeding: Occurs when light from one object reflects onto another. Photon mapping is particularly suitable for handling this effect because the algorithm reflects photons from one surface to another based on the bidirectional reflectance distribution function (BRDF) of that surface, and thus the light from an object hitting another is a natural result of the method. In the image, you can see how the wall reflects a reddish color onto the floor.

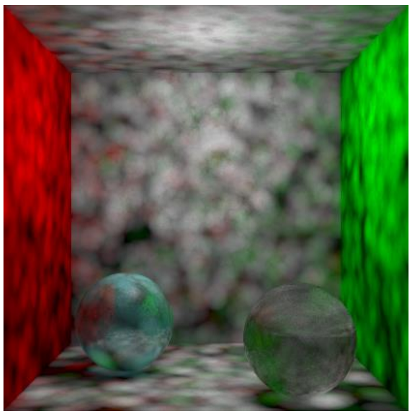
The lighting and appearance effects that come from the ray tracing part of the algorithm are:

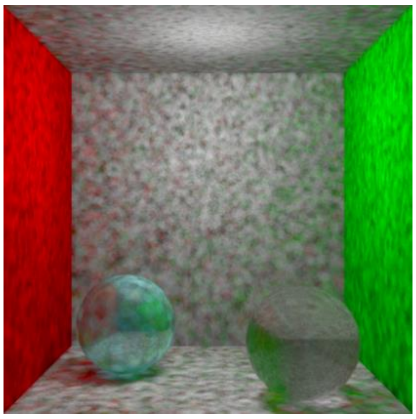
- Direct Illumination: The ray tracing part of the algorithm, especially when using next-event estimation, contributes to the direct illumination coming from point light sources.

4. **Results obtained, in terms of cost, image quality, and convergence.**

**K=10**

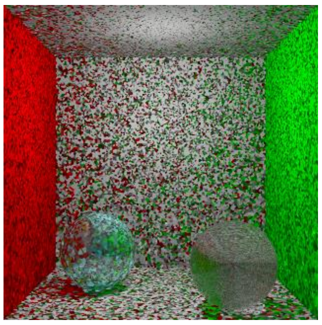

N=1k : 68s

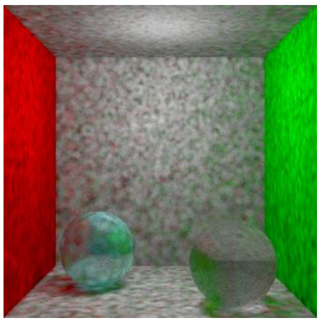

N=10k : 161s

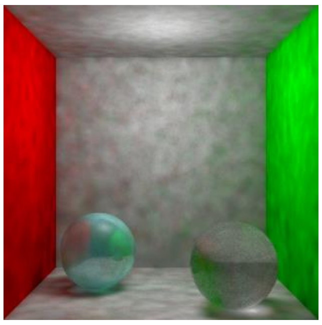

N=100K : 272s

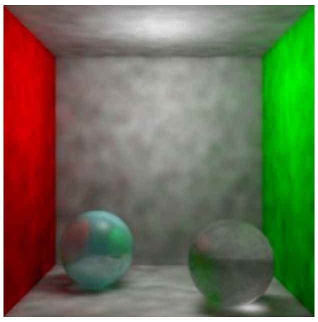
Increasing the number of photons (N) in the scene with a fixed number of nearest neighbors (k) in radiance estimation can improve the quality of the image, as it provides a more accurate representation of global illumination, but not significantly, as the number of neighbors remains small. However, increasing N also implies a higher computational cost, as more photons must be tracked and stored. This approach tends to improve convergence and reduce noise in the image, but the cost can be very high.

**N=100k**


K=1 : 99s


K=10 : 272s


K=50 : 721s


K=100 : 809s

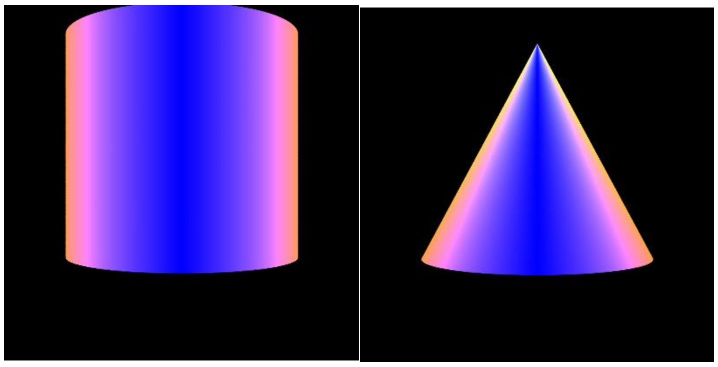
It is observed that a too low value of k results in a lot of noise due to the lack of information from nearby photons. As k increases, the quality of the image improves, but so does the computational cost (as seen in the time). It can also be seen that if k continues to increase, the image will not improve but will start to worsen, as it captures local details and noise instead of more general lighting patterns.

# Extensions

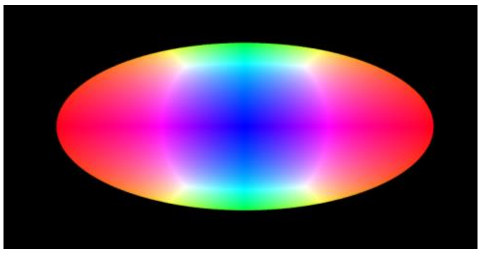
## Other primitives

**Cones, Cylinders, and Ellipsoids**

- **Cones and Cylinders**: These were implemented by defining mathematical equations for their surfaces and solving ray intersections with these shapes. Efficiency in intersection calculation and accuracy in surface definition were considered.

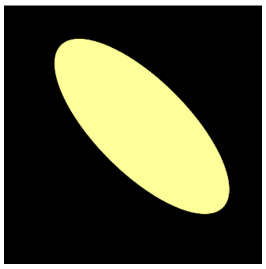


- **Ellipsoids**: Similar to cones and cylinders, ellipsoids were represented using parametric equations, allowing a precise description of their three-dimensional shape and facilitating the calculation of intersections.

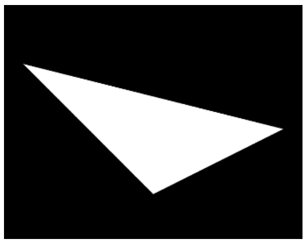


**Discs, Triangles, and Toroids**

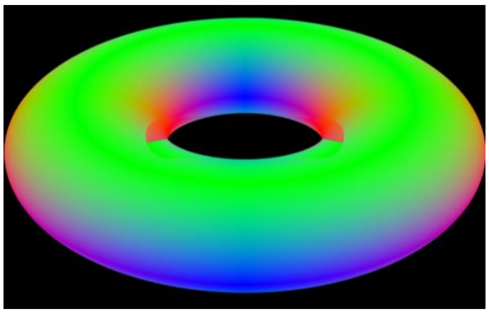
- **Discs**: Implemented as a variant of planes, with a simple equation for ray intersections.



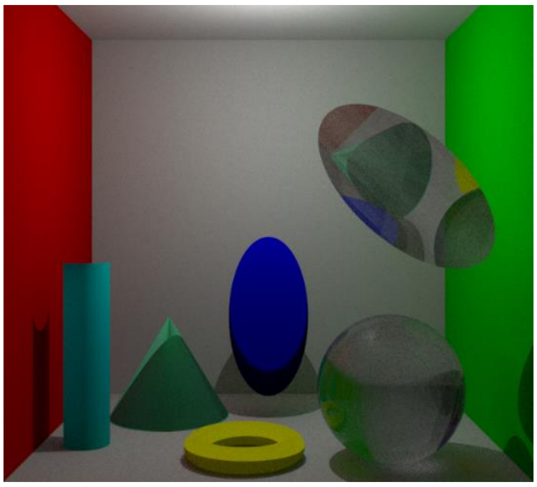
- **Triangles**: Fundamental in 3D graphics and already included in the basic model of the path tracer. However, their implementation was optimized to improve efficiency, as they were later used for another extension (3D models).



- **Toroids:** Represented with complex parametric equations. Their implementation was challenging due to the difficulty of efficiently solving ray intersections, so external sources were used to resolve quadratic equations and various websites were consulted for guidance on calculating intersection normals.



The scene similar to the Cornell Box with the added figures is shown below:

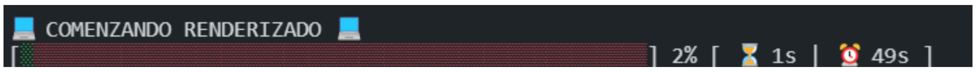


## Parallelization

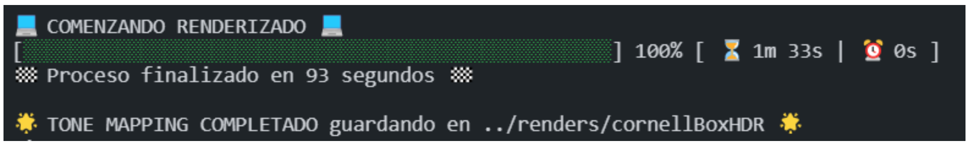
The image was parallelized by rows, with multiple rays traced per pixel for anti-aliasing and accumulating the results in the pixel vector. This allows multiple rows of the image to be processed simultaneously.

The main intention of using parallelization is to increase processing speed by utilizing the system's resources, such that depending on each computer's properties (like the number of cores), processing speed increases. However, since rendering uses the maximum available resources, it can slow down user requests, requiring the computer to finish rendering before further use.

A simple monitor was implemented to track render progress and estimate remaining time, showing a progress percentage based on the number of pixels processed:



```
💻 COMENZANDO RENDERIZADO 💻
[▒░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░] 2% [ ⏳ 1s | ⏰ 49s ]
```



```
💻 COMENZANDO RENDERIZADO 💻
[▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒] 100% [ ⏳ 1m 33s | ⏰ 0s ]
🏁 Proceso finalizado en 93 segundos 🏁

🌟 TONE MAPPING COMPLETADO guardando en ../renders/cornellBoxHDR 🌟
```

This extension is related to Ray Tracing Optimization, explaining generally how to use parallelization to optimize render times. Thanks to this, the rendering time decreased considerably, but increased again with the addition of 3D models despite parallelization.

## Bounding Volumes Hierarchy (BVH)

BVH is a technique used to optimize the rendering process in computer graphics. BVH involves organizing scene objects (triangles in a 3D model in our case) in a tree structure. Each tree node encapsulates a bounding volume containing a subset of scene objects, and each tree level further divides these objects.

The construction of a BVH starts with the scene and its bounding box, forming the root node. We recursively subdivide this node using a division plane that separates the primitives into two. It's crucial not to split the space itself, preventing primitives from being divided or assigned to both sides.

Centroids are assigned, the root node is initialized with all primitives, and then the BVH is recursively subdivided to optimize the structure and improve intersection test efficiency.

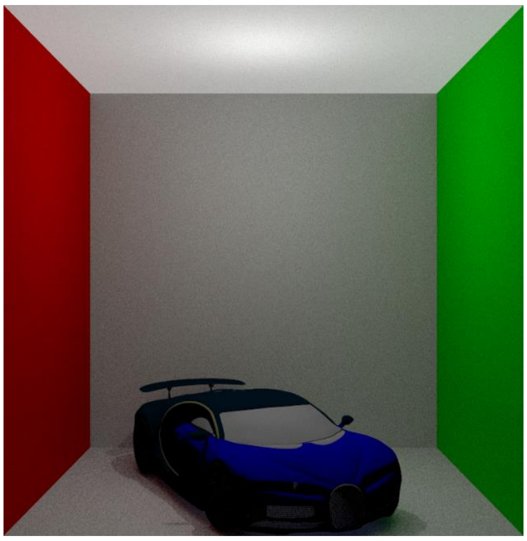
This subdivision involves choosing a division plane, partitioning the primitive list into two groups, and creating child nodes for each group. This is done efficiently, without sorting triangles, and using the size of the undivided list to optimize the operation in place. Once the left and right child nodes are generated, their bounds are updated, and we recursively dive into the child nodes. We generally end when we have 2 or fewer primitives in a leaf.

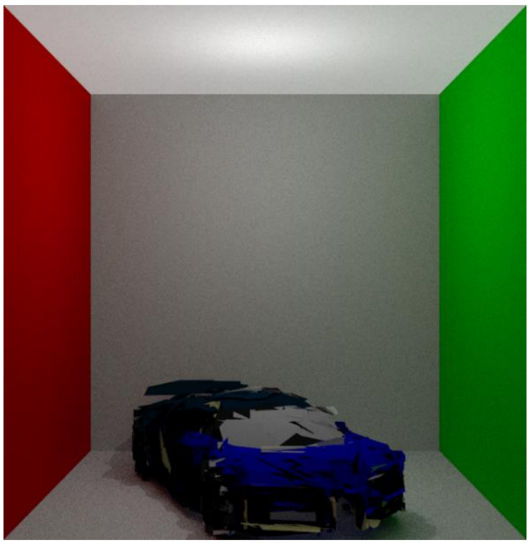
Adjusting the range of values for dividing the AABB over centroids can improve the tree's performance.

- **Choice of Bounding Volume:** We chose axis-aligned bounding boxes (AABB) for their simplicity and efficiency in intersection calculations.
- **Division Strategy:** Node division is based on a spatial balance criterion, dividing objects into two groups of approximately equal size, resulting in a more balanced tree.
- **Traversal Optimization:** Efficient algorithms were implemented to traverse the BVH tree during intersection searches, minimizing unnecessary calculations.

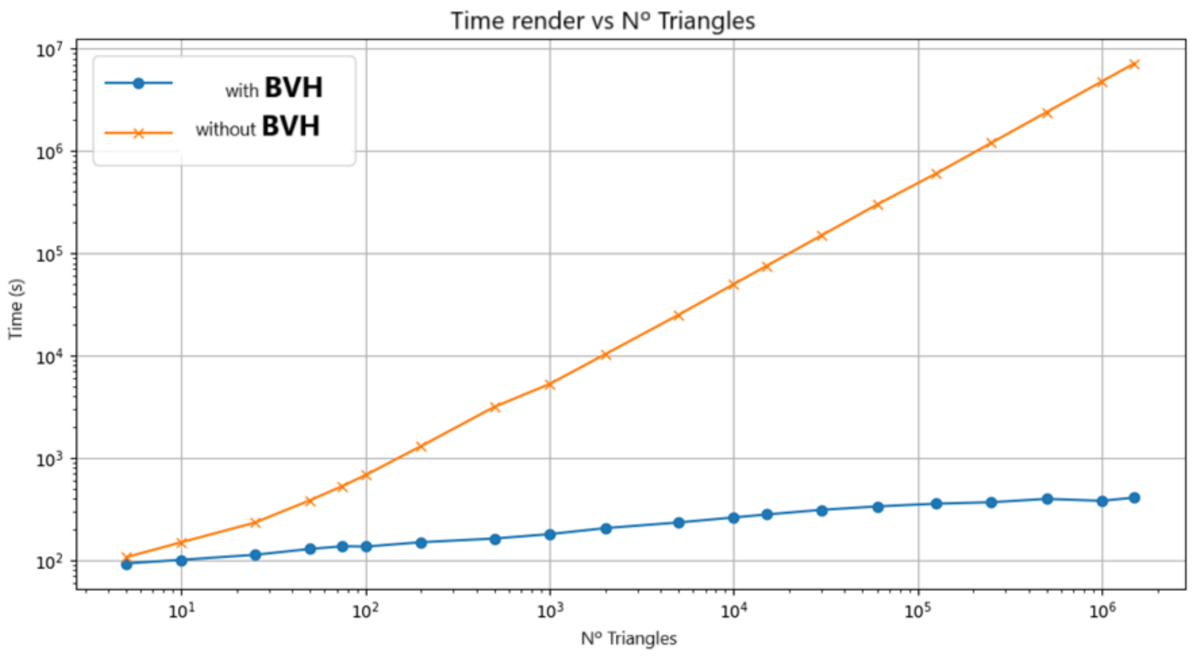
I based the  development on the solution explained in **How to build a BVH – Part 1: Basics - Jacco's Blog** to adapt it to my especific functions and classes.

Once the tree hierarchy was completed, we compared the rendering time of the same 3D model with different numbers of triangles, using or not using BVH:


1.5M triangles


2K triangles

A graph is provided showing the difference in using BVH for different quantities of triangles:



*For values of 200, 500, 1000, and 2000 triangles without using BVH, the approximation from the parallelization execution estimation was used.

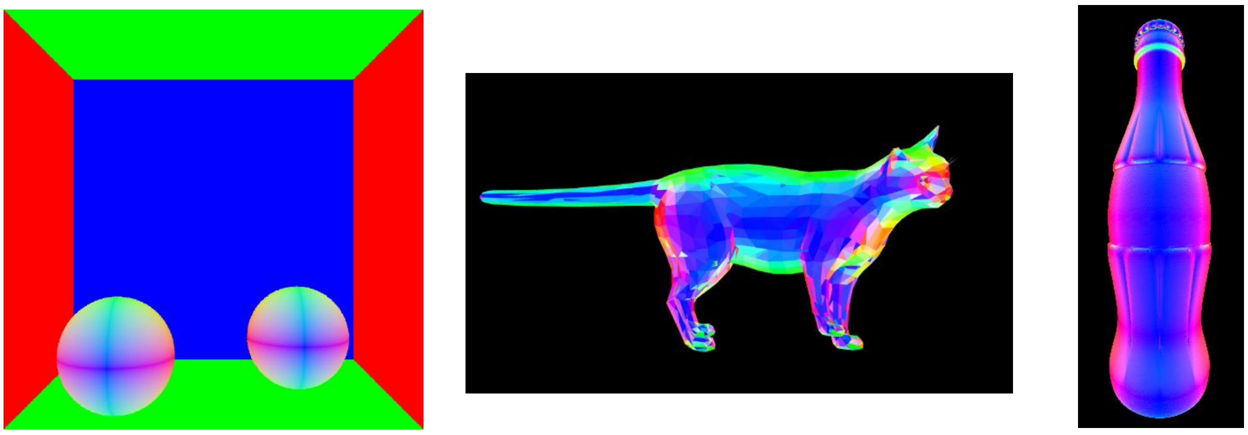
**From 5 thousand triangles without BVH, the approximation was based on simple estimation from previous costs.

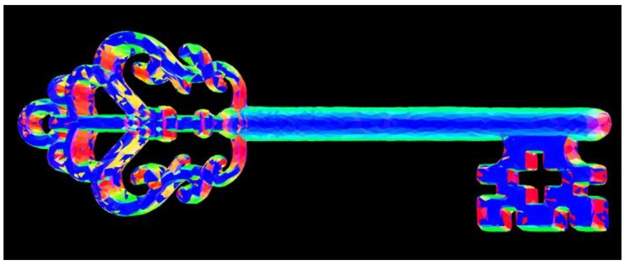
In conclusion, without BVH, time scales linearly with the number of triangles, but with BVH, time scales logarithmically due to the approximately balanced tree structure.

This extension is related to Ray Tracing Optimization, explaining the general idea behind BVHs. Thanks to this, the time for rendering 3D models decreased significantly.

## Normal Visualization

To better visualize object depth when only flat colors (any object's emission value) were visible without any light, we added a way to view objects depending on the orientation of the object's normal, making each dimension show based on RGB values:

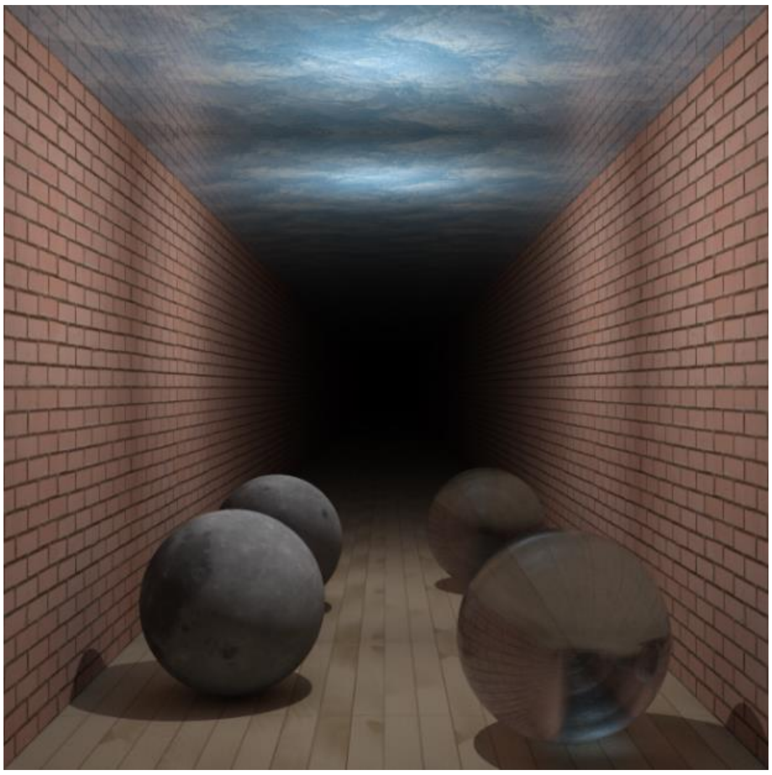




This extension was done because it allowed us to continue the process knowing that we had a solid base. It was done before direct light was implemented to check scene depth.

## Textures

To increase the realistic visualization of scenes, a feature was added to load PPM images into memory, replacing the default color of the object with a pixel from the texture depending on the intersection point:



It was implemented in all previously mentioned objects, but the main goal is to add textures to 3D models made with triangles.

The color is obtained through texture interpolation: The intersection point of the ray with the object is found, then the texture coordinates at the impact point are obtained to determine the texture to be used.

This extension is related to Textures. It allows us to texture objects, making the final rendered image more realistic.

## Importing 3D Models

To load more complex scenes more easily, an extension was added to import 3D models in .ply format, allowing visualization:

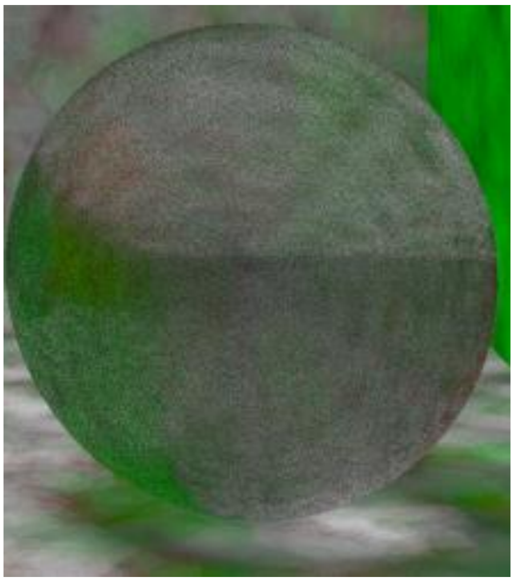
The .ply file format is examined for comments indicating the presence of textures. If textures are found, they are loaded and stored in memory. Then the PLY file header is read, and upon reaching the data section, vertices and faces are read. Triangle objects are created using the vertex and face information, stored in a vector of triangles.

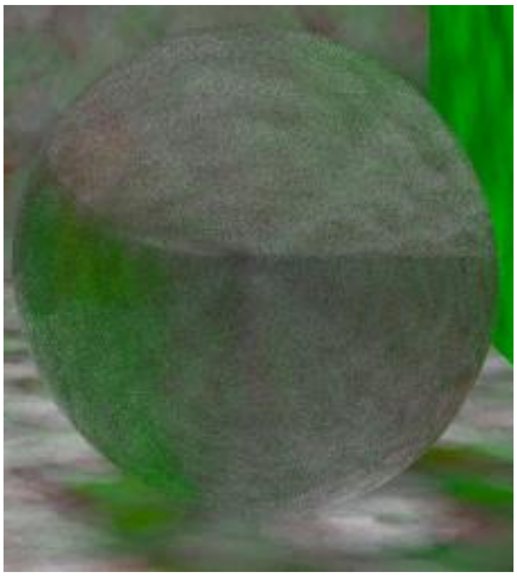
This extension was made to create easily 3D scenes importing from other software as Blender or MeshLab (all the models are from SketchFab)

## Fresnel Effects

This extension focuses on making the Bidirectional Reflectance Distribution Function (BRDF) coefficients more realistic for materials like glass. The uniqueness of this implementation is that the specular part of the BRDF directly depends on the cosine term between the surface normal and the incoming light direction. This design decision ensures that specular reflection varies with the angle of incidence, a phenomenon observed in real life and crucial for accurately simulating transparent and reflective materials. In other words, directional dependence was added to a reflectance coefficient. This extension was explained in class in relation to BRDF coefficients.

Here is a comparative example using Fresnel or not:


Without  Fresnel


With Fresnel

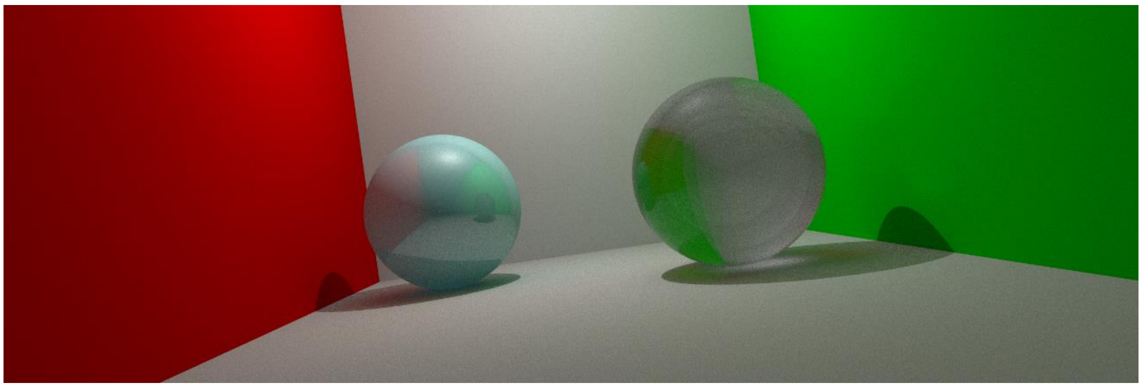
As can be seen in the images, at grazing angles (angles close to being parallel to the surface), the sphere becomes more reflective, while at more perpendicular angles, it becomes less reflective (ks is reduced).

It's important to note that if light passes from one medium to another with a lower refractive index, it can be subject to the phenomenon of total internal reflection. This occurs in the case of materials like glass or water, and must be considered. We do this by calculating, as in the case of reflection, the sine of the refractive angle or $\theta_2$. If $\sin(\theta_1)$ is greater than 1, then we have a case of total reflection. In this specific case, it's not necessary to calculate Fresnel's formulas. It's enough to set FR to 1. As always, you will need to swap the refractive indices if you find that the incident ray is inside the object with the higher refractive index.

This extension is related to the topic of Monte Carlo. Thanks to this, dielectric materials present a more realistic appearance, as the coefficients are chosen using these equations.

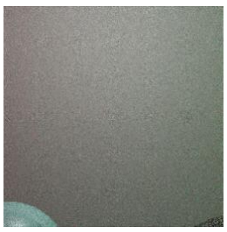
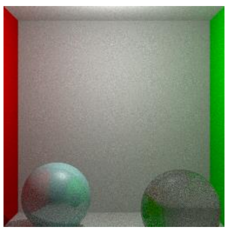
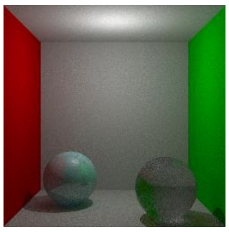
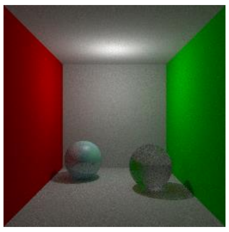
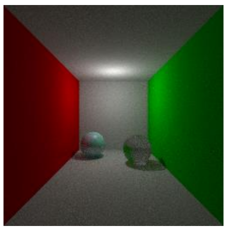
## Camera

This modification in the camera of the path tracer introduces key parameters to increase realism in renders: field of view (fov), focal distance, and aperture. The fov allows control over the extent of the visible scene through the camera, mimicking the "zoom" effect of a real camera. The focal distance affects the sharpness of objects in relation to their distance from the camera, enabling techniques like selective focus. Finally, the aperture controls the depth of field, influencing how much of the scene appears in focus:



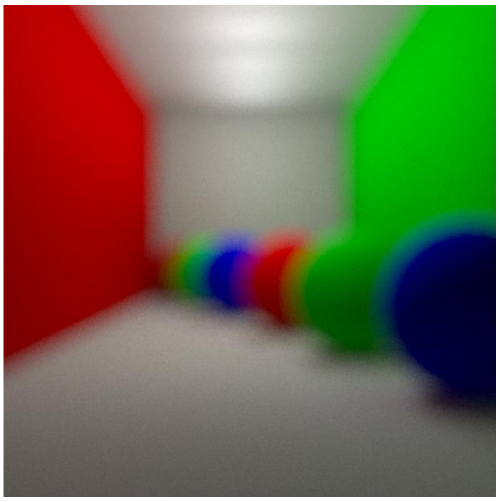
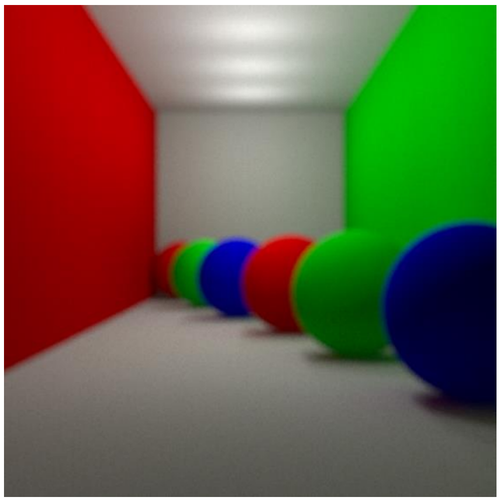
This modification improves the renderer's ability to simulate realistic photographic effects. The comparative renders below show how the inclusion of fov, focal distance, and aperture enriches the visual representation:

**FOV**

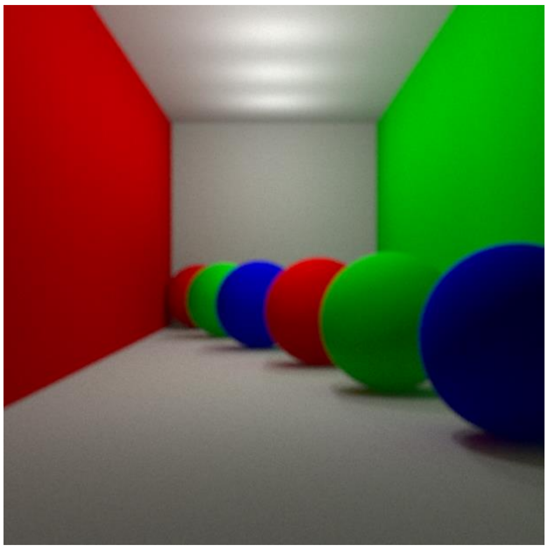






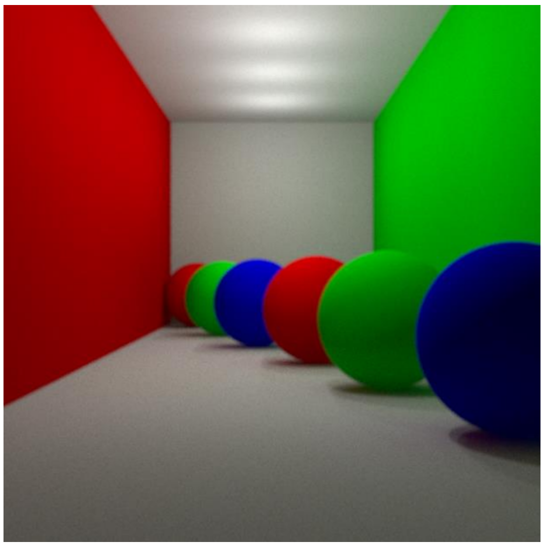
45º 75º 90º 110º 130º

**DISTANCIA FOCAL / APERTURA**

d Focal = 1 | apertura = 0.02 dFocal = 3 | apertura = 0.02


dFocal = 5 | apertura = 0.02

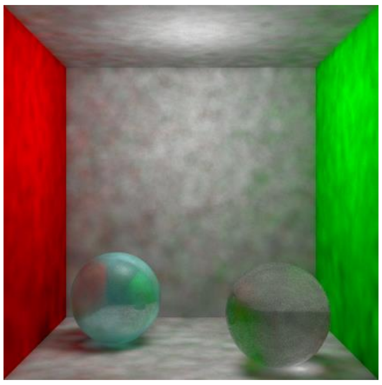

dFocal = 7 | apertura = 0.02

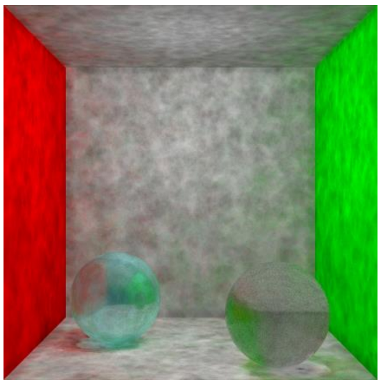
This extension is related to the topic of Cameras. Thanks to it, we have been able to give a different focus to the image, making it look better.

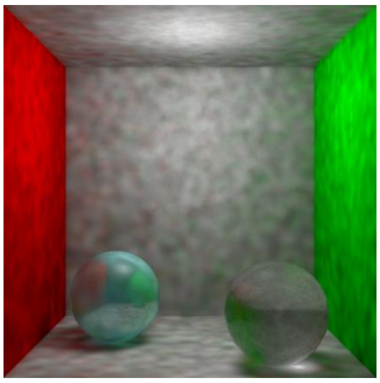
## Diferentes kernels(**cone, Gaussian, Epanechnikov)**

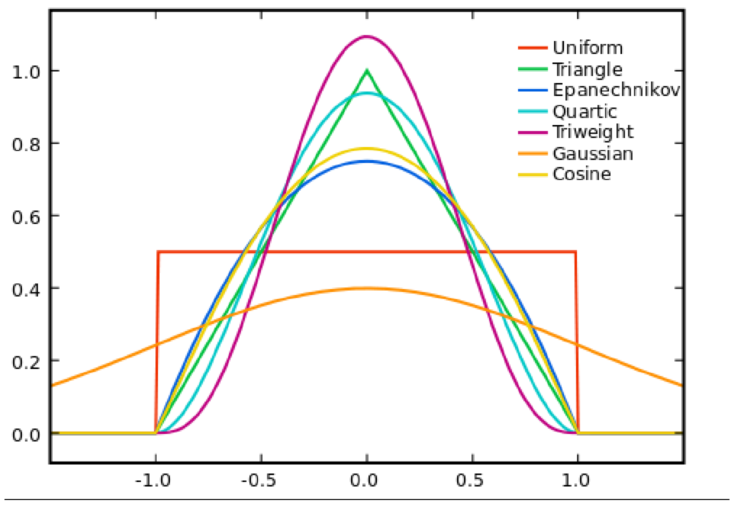
Density estimation involves calculating the density of photons at a particular point in space based on the information of surrounding photons. The kernel acts as a weighting function that assigns weights to nearby photons based on their distance to the point of interest.

In this extension, a variety of kernels – cone, Gaussian, and Epanechnikov – have been integrated for the smoothing process in the path tracer. Each of these kernels has unique characteristics that affect the way pixel values are averaged and, therefore, the final result of the image.


Epanechnikov Kernel


Cone Kernel


Gaussian Kernel



- Epanechnikov kernel: Its function has a hat shape, being zero outside a certain radius and reaching its maximum at the center. This means it assigns a higher weight to closer photons and gradually decreases their contribution as the distance increases.
- Cone kernel: Unlike some kernels that assign uniform weights to all photons within a certain radius, the Cone Kernel assigns higher weights to photons falling within a specific cone defined by its direction and angular aperture. This helps focus the contribution of photons in a particular direction, which can be useful in situations where certain lighting effects are desired.
- Gaussian Kernel: Each nearby photon contributes to the final result with a weight that follows the shape of a Gaussian curve. Photons closest to the point of interest have a higher weight, while those further away have lower weights. The Gaussian distribution provides a gradual smoothing of photon contributions, helping achieve smooth transitions and realistic lighting effects.

It can be observed that using the Cone Kernel, the noise produced by the photons is much more noticeable, while using the Gaussian kernel, there is less color variation at close points. However, the Epanechnikov kernel manages to make the noise less noticeable, similar to applying a blur effect.