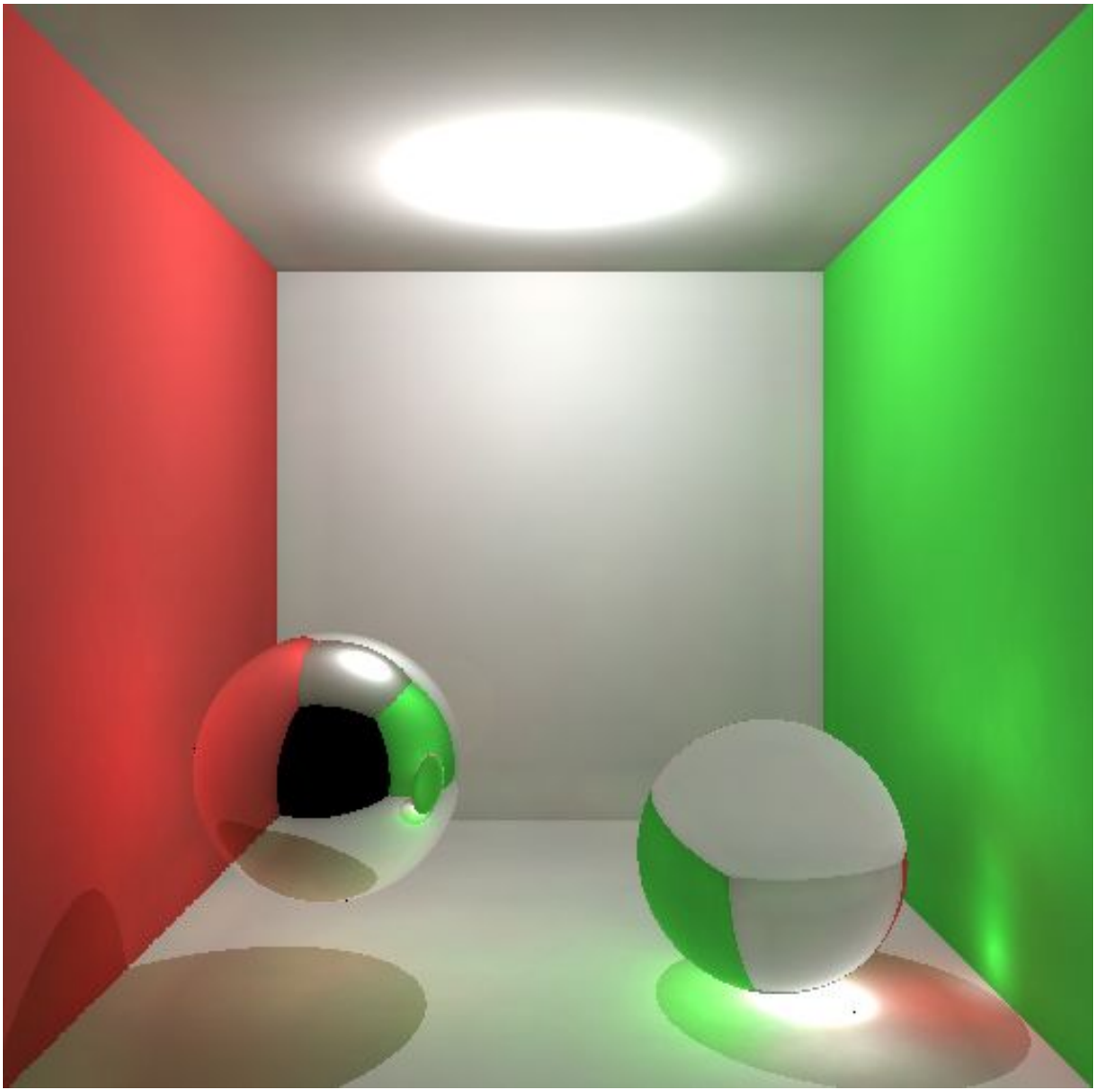


Photon Mapping

Alejandro Gutiérrez Bolea 735089

Daniel Cay Delgado 741066

Informática Gráfica 2019-2020

1. Introducción 2

2. Generación del Mapa de fotones 3
   2.1 Photon random walk 4

3. Estimación de radiancia 5
   3.1 Reflexión especular 6
   3.2 Iluminación directa 6
   3.3 Iluminación indirecta y cáusticas 7
   3.4 Filtro de cono 7

4. Cuestiones 8
   4.1 Pruebas cambiando número de fotones almacenados 11
   4.2 Pruebas cambiando el número de vecinos 13

5. Conclusiones 17

6. Bibliografía 18

1. Introducción

Photon Mapping es un algoritmo capaz de renderizar imágenes de geometría compleja con iluminación global basado en dos fases y que fue desarrollado por Henrik Wann Jensen.

La primera fase consiste en lanzar fotones desde las fuentes de luz, donde se almacenan las intersecciones con la materia en un mapa de fotones. Se han utilizado dos mapas de fotones, uno para luz indirecta y otro para cáusticas.

En la segunda fase, se trazan rayos desde la cámara y se estima la radiancia a partir de los k fotones más cercanos almacenados en los mapas.

Además, se hace un pase adicional de ray tracing para la luz directa.

Las ventajas de Photon Mapping son la capacidad de simular fenómenos visuales complicados como las cáusticas, que son costosas de capturar con algoritmos como Path tracing, además de la eliminación del ruido y el menor tiempo de cálculo. Como desventaja aparece el sesgo.

Para la implementación se ha usado el código proporcionado por los profesores.

2. Generación del Mapa de fotones

Se trata de la primera fase del algoritmo que consiste en emitir fotones desde las fuentes de luz y trazarlos a través de la escena almacenando las intersecciones con la materia. Esta será la técnica usada para construir el mapa fotónico. Dicha implementación se encuentra en la función *preprocess*.

En la primera etapa, emisión de fotones, se generan aleatoriamente fotones que saldrán de las diferentes fuentes de luz, esto se hace en forma aleatoria. Un enorme número de fotones son emitidos desde cada fuente de luz y con una dirección aleatoria seleccionada mediante muestreo por rechazo. Esta técnica se basa en dar la misma probabilidad a todos los puntos de la esfera situada en torno a la fuente de luz puntual, con el fin de que la dirección elegida para lanzar el fotón sea de verdad aleatoria. La técnica consiste en introducir la esfera en un cubo, de manera que se generan 3 valores aleatorios en el rango [-1,1], uno por cada una de las 3 dimensiones. Si el punto resultante está a una distancia del centro de la esfera mayor a 1, quiere decir que el punto no pertenece a la esfera (está en la zona del cubo fuera de la esfera), por lo que se desecha el punto y hay que generar uno nuevo.

La potencia de la luz será dividida entre todos los fotones emitidos por igual, y por eso cada fotón transportará una fracción de la potencia de la fuente de luz inicial. Es importante remarcar que la potencia de los fotones será proporcional al número de fotones emitidos y no al número de fotones almacenados en el mapa de fotones.

Una vez se tiene la posición, dirección y la potencia del rayo, se traza el fotón a través de la escena almacenando las intersecciones del camino aleatorio del fotón con la materia en dos listas diferentes (indirecta y cáusticas).

Tras recorrer todas las luces y obteniendo las dos listas, se construyen los mapas de fotones global y el cáustico. y se realiza un balanceo.

Para estos mapas de fotones se ha utilizado la estructura kd-Tree balanceada ya que se necesita una estructura de datos que sea rápida en encontrar los fotones vecinos más cercanos en 3 dimensiones a una posición dada. La complejidad para encontrar un fotón en un Kd-Tree balanceado es de O(log N), donde N es el número de fotones en el mapa de fotones. La simplicidad de los Kd-Trees permite implementar un algoritmo sencillo y eficiente de búsqueda para encontrar los k vecinos más cercanos.

Una vez construidos los mapas de fotones se utilizarán para calcular la luz radiada.

2.1 Photon random walk

Question 1.1

La función *trace_ray* traza un rayo a través de la escena almacenando las intersecciones con la materia en dos listas de fotones, globales y cáusticas.

A dicha función se le pasan como parámetros, el rayo trazado y la dirección aleatoria obtenida anteriormente, así como la energía de fotón y las dos listas que guardan los fotones globales y las cáusticas.

Primero actualiza el número total de fotones disparados, para garantizar que no se disparen más del máximo establecido (parámetro global m_max_nb_shots). Luego, almacena en la variable *energy* la potencia del fotón, obtenida por un parámetro de la función y desplaza el rayo (*shift*) para evitar errores numéricos en las intersecciones. Después, se encuentra un bucle infinito que empieza a calcular el camino aleatorio que sólo puede terminar con cuatro circunstancias:

1. El rayo no golpea nada (*!it.did hit()*).
2. La ruleta rusa elige absorción y el fotón muere (*epsilon2* > *avg_surf_albedo*).
3. El rayo llega al límite de rebotes (*photon_ray.get level()*> *20*).
4. Variable booleana *direct_only* verdadera, por lo que solo se almacena la luz directa.

Dentro de este bucle, primero se calcula la siguiente intersección del rayo. Después de eso, hay diferentes casos:

Si el material interceptado es delta (especular perfecto o refractivo), simplemente actualiza, sin guardar el fotón, una variable booleana cáustica para determinar que se trata de un objeto delta, ya que dicha variable se usará en las próximas iteraciones para distinguir entre los fotones que se almacenarán en el mapa global o en el cáustico.

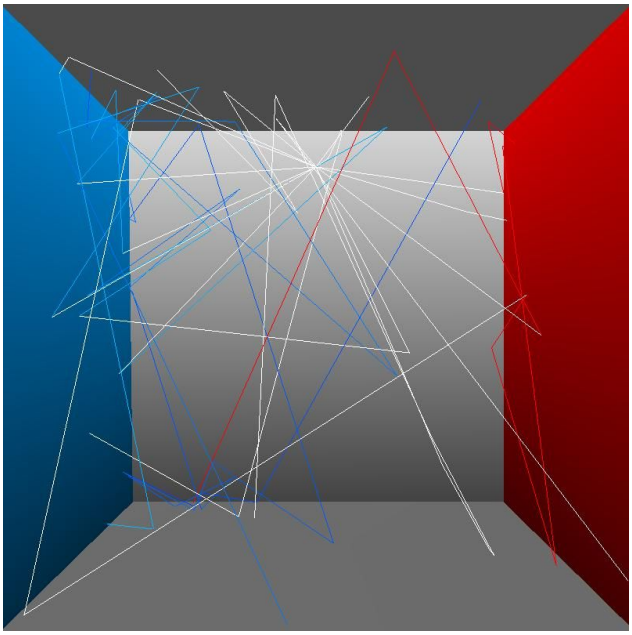
Si no es un material delta, primero verifica si el fotón proviene directamente de la luz o no, y en este caso, salvo que la variable *direct* sea verdadera, el primer rebote no se almacena, ya que la iluminación directa se calcula en la segunda fase usando trazado de rayos. Cuando tiene que almacenarse, se compara con la variable cáustica para determinar qué mapa usar, y también se verifica el tamaño del mapa para asegurar que no se almacenen más del número máximo de fotones de cada tipo. Serán almacenados la posición del fotón, la dirección incidente y la potencia del mismo.

Una vez que se almacena el fotón (o no si es luz directa o delta), se aplica la Ruleta Rusa (epsilon2) en función del albedo de la superficie, para decidir si el camino aleatorio termina (el fotón se absorbe) o no. Como el albedo es en realidad un color de tres canales (RGB), se usa el promedio entre los tres canales. La Ruleta Rusa es útil para almacenar fotones con energías similares, en lugar de reducir su potencia cada vez que golpean una superficie.

Después de eso, si el fotón aún está vivo, se debe muestrear un nuevo rayo desde ese punto (*get_outgoing_sample_ray,* obtiene el rayo saliente) y se debe actualizar la energía con respecto a las propiedades del material para la próxima iteración. De acuerdo a la ecuación de render, la energía debe ser actualizada multiplicándola por el coseno del ángulo entre la dirección del nuevo rayo y la normal del punto interceptado, además de dividirla para π con el fin de realizar la normalización oportuna. Además, puesto que se aplica muestreo por importancia para calcular la dirección del nuevo rayo, hay que dividir la energía también para la pdf, que es devuelta por la función *get_outgoing_sample_ray.* Por último, el resultado es multiplicado por el albedo de la superficie, y dividido para la media de este (es la razón de la pequeña diferencia entre los rangos de energía que contienen los distintos fotones almacenados, mejorando así la estimación de la radiancia).

Una vez fuera del bucle, si los mapas ya están llenos, el camino aleatorio se detiene ya que no tiene sentido continuar disparando fotones que no se almacenarán. En otros casos, comienza una nueva iteración del bucle, con el nuevo rayo muestreado y la energía modificada.

Muestra de los diferentes rebotes del rayo:



3. Estimación de radiancia

Se ha realizado la estimación de la radiación producida en un punto de la escena mediante estimación de densidad. Se trazan rayos desde la cámara hacia la escena y se estima la radiancia a partir de los k fotones más cercanos (vecinos) almacenados en los mapas de fotones. Además, se hace un pase adicional de ray tracing para la luz directa, ya que gracias

a que la luz es aditiva, se calcula por un lado la iluminación directa, indirecta y las cáusticas. Dicha implementación se encuentra en la función *shade*.

El término que cambia con respecto a la ecuación de render en path tracing es la luz incidente, ya que para photon mapping se aproxima mediante el mapa de fotones [1].

Ecuación de radiancia reflejada

$$L_r(\mathbf{x},\omega_r) = \int f_r(\omega_i,\omega_r) L_i(\mathbf{x},\omega_i)(N\cdot\omega_i)d\omega_i$$

Radiancia por flujo

$$L_i(\mathbf{x},\omega_i) = \frac{d^2\Phi_i(\mathbf{x},\omega_i)}{(N\cdot\omega_i)d\omega_i dA_i}$$

Sustituimos

$$L_r(\mathbf{x},\omega_r) = \int f_r(\omega_i,\omega_r)\frac{d^2\Phi_i(\mathbf{x},\omega_i)}{dA_i}$$

Numéricamente

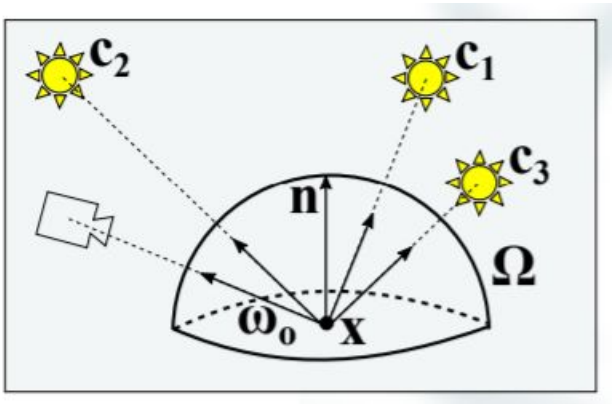
3.1 Choque con material delta

En caso de que el rayo lanzado desde la cámara colisione con un objeto con material delta, se ha de trazar el rayo de salida hasta la siguiente colisión. Este procedimiento se sigue hasta que el rayo encuentre una superficie difusa, donde se procederá a aplicar la iluminación directa de las luces puntuales empleando la ecuación de render. El coeficiente W, que comienza con valor 1, se multiplicará por el albedo del material interceptado y se dividirá por la pdf oportuna debido al muestreo por importancia. Esta operación ocurre con cada choque hasta encontrar un material difuso.

3.2 Iluminación directa

El camino ”directo” se calcula perfectamente con trazado de rayos estándar (*next event estimation)*.

Ecuación de Render [2]:



$$L_o(\mathbf{x}, \omega_o) = \sum_{i=1}^{n} \underbrace{L_i(\mathbf{x}, \omega_i)}_{\text{INCOMING LIGHT}} \underbrace{f_r(\mathbf{x}, \omega_i, \omega_o)}_{\text{BRDF}} \underbrace{|\mathbf{n} \cdot \omega_i|}_{\text{GEOMETRY}}$$

Siendo *Li* el aporte de la luz, calculado mediante la potencia de la luz entre la distancia en el punto x (función *get_incoming_light*), *fr* las propiedades del material, y el producto escalar de la normal y la dirección de la luz hacia el punto. Dada la propiedad aditiva de la luz, se suman los resultados de cada luz puntual y se obtiene el aporte total de la luz directa.

En el cálculo de la *fr* se ha implementado la posibilidad de crear materiales difusos y de Phong.

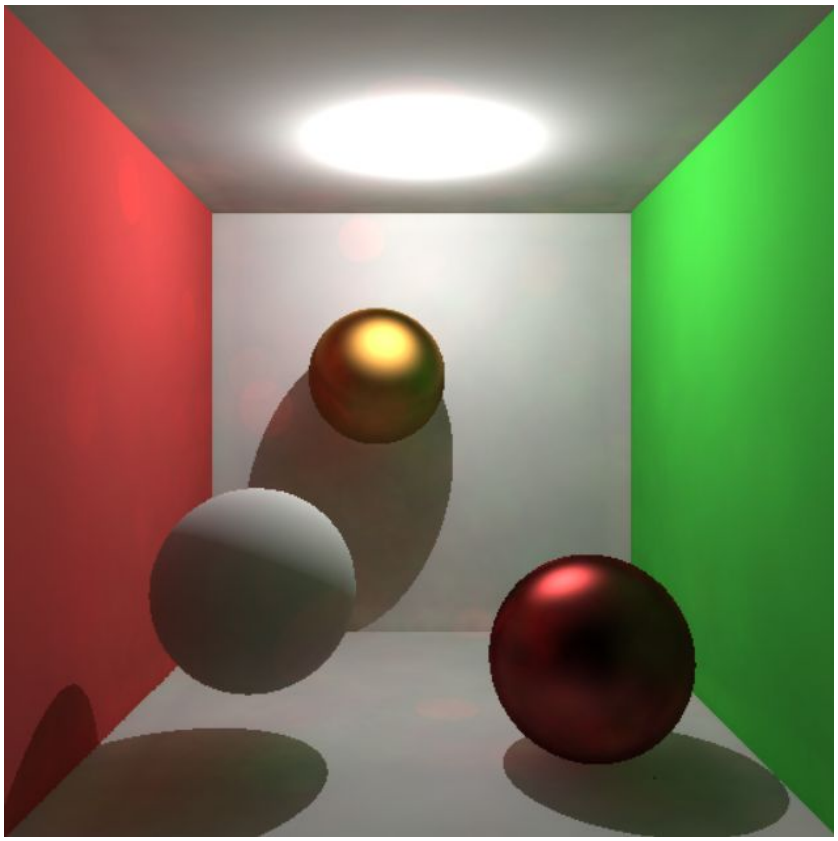


Figura 1: esfera blanca lambertiana con albedo (.85,.85,.85) / esfera roja de Phong con albedo (.85,.085,.085) y alpha=20 / esfera naranja de Phong con albedo (.85,.40,.085) y alpha=3

3.3 Iluminación indirecta y cáusticas

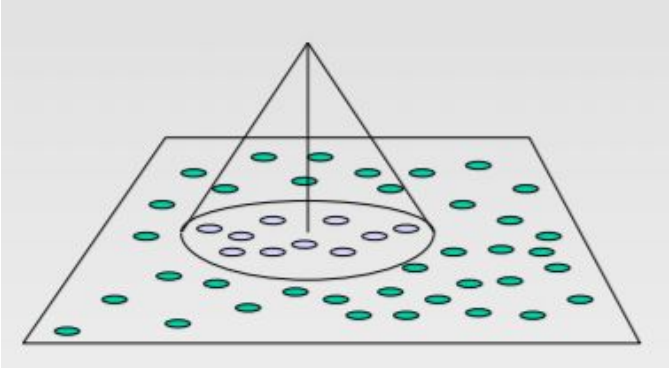
Se calcula la estimación de radiancia con los mapas de fotones generados tanto para la luz indirecta como para las cáusticas.

Se trata de realizar el sumatorio del flujo de los k fotones más cercanos dividido por el área de la geometría que los contiene. Es por ello que primero se realiza la búsqueda de los fotones vecinos a la intersección en una superficie haciendo uso de los mapas de fotones calculados anteriormente.

Dado que los k fotones encontrados no deberían tener igual peso, ya que los fotones más cercanos deben ser considerados con mayor importancia a la hora de calcular la radiancia, se ha usado uno de los filtros por distancia (del fotón a x) como es el filtro de cono.

3.4 Filtro de cono

En la implementación se ha optado por el filtrado de cono.



Para calcular la radiación *Lr*, que sale de un punto de intersección *x* en una superficie con BRDF *fr*, basándonos en que cada fotón *p* representa el *flujo* que llega a *x* desde la dirección *Wp*, se puede integrar toda esta información dentro de la ecuación de Render como [1]:

Results in a cone filtering of photons

$$L_r(\mathbf{x},\omega_r) \approx \frac{1}{(1-\frac{2}{3k})\pi r^2}\sum_{p=1}^{n}\left(1-\frac{\|\mathbf{x}-\mathbf{x}_p\|}{kr}\right) f_r(\omega_p,\omega_r)\Delta\Phi_p(\mathbf{x}_p,\omega_p)$$

Donde *r* es la distancia máxima del fotón más lejano al punto y *k* es el parámetro del filtro de cono.

Se realiza la suma de luz directa, indirecta y cáusticas, y se multiplica por el coeficiente W, que en el caso de que no haya pasado el rayo por ninguna superficie delta será 1.

4. Cuestiones

Question 2.1 *Render two images of Scene2 (using -scene 2) with only direct illumination (DI), each image where DI is computed with ray tracing and with photon mapping respectively. You can compute direct illumination with photon mapping setting the parameter direct=true in PhotonMapping::trace ray. What are the differences between the direct illumination computed using ray tracing photon mapping? Which one do you think is more accurate and why?*

La luz directa calculada mediante ray tracing es más precisa que la estimación de la radiación de photon mapping, ya que esta segunda está obviamente sesgada debido al hecho de que el filtrado de cono incluye fotones de diferentes lugares e incluso diferentes superficies. Esto es claramente visible en las sombras, las cuales al solo calcular luz directa deberían tener mucho contraste con las zonas no sombreadas. La *Figura 1* y *Figura 3* basadas en el ray tracer muestran las sombras correctamente.

Con la iluminación directa basada en photon mapping, las sombras existen pero son mucho más suaves. Esto se debe a que la única diferencia entre las zonas de sombra y los puntos iluminados es el tamaño del cono, ya que una zona de sombra tiene pocos fotones mapeados, por lo que hay que buscar los vecinos más lejos (*Figura 2*).

Otro punto a añadir es que la luz directa calculada con los fotones permite recrear las cáusticas (porque los fotones cáusticos son seguidos hasta que se llega a un material difuso), efecto que con la luz directa de ray tracing no se puede simular (*Figura 4*).

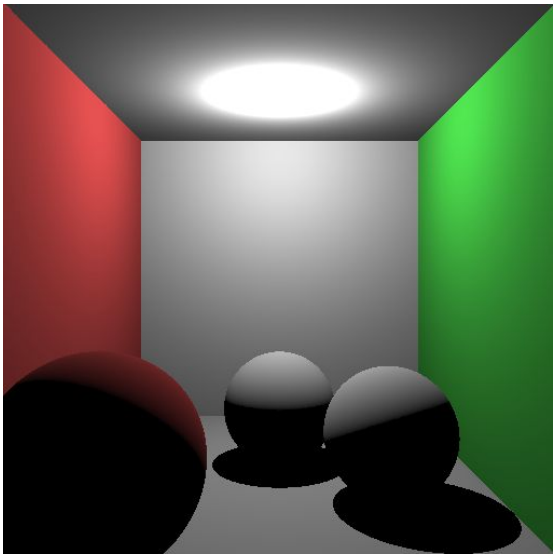


Figura 2: Iluminación directa Ray tracing

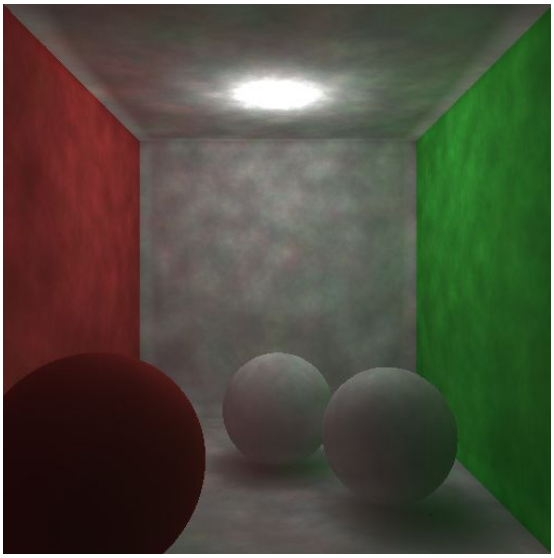


Figura 3: Iluminación directa Photon Mapping

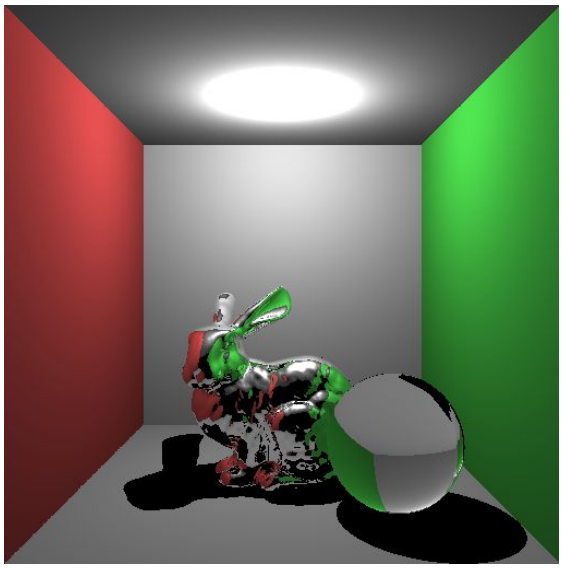


Figura 4: Iluminación directa Ray tracing

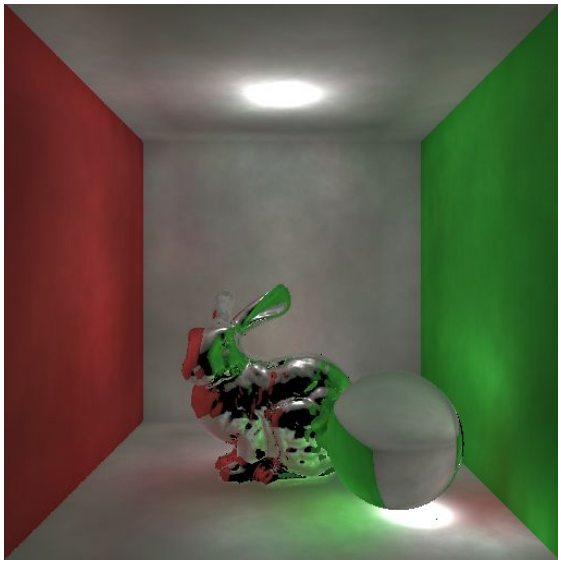


Figura 5: Iluminación directa Photon Mapping

Question 2.2 *Render Scene1 (using -scene 1) using photon mapping. This scene has two spheres with delta BSDF. Describe how do you shade each of these spheres and why.*

La esfera de la izquierda es especular perfecta, mientras que la de la derecha es refractora perfecta (materiales delta). Cuando se lanza un rayo desde la cámara y choca con una de estas esferas, para saber el color que va sobre el punto interceptado, no se puede calcular la luz directa e indirecta en ese punto y sumarlas como se haría con una superficie difusa.

La forma de proceder consiste en cambiar la dirección del rayo, la cual pasará a ser una u otra según si se trata de la esfera de la izquierda (el rayo rebotará según la ley de reflexión) o de la derecha (el rayo se refractará a través de la esfera según la ley de Snell). El rayo seguirá rebotando por la escena hasta que se intercepte una superficie difusa o se alcance el número máximo de rebotes. Si ocurre lo primero, ya se podrá proceder a calcular la iluminación total en la superficie del material delta. Hay que tener en cuenta que el número de rebotes hasta encontrar una superficie difusa afecta al cálculo del color, ya que el color del punto cortado por el rayo lanzado desde la cámara es igual a L*W, siendo L la suma de la luz directa, indirecta y cáusticas, y W un coeficiente cuyo valor se va modificando con cada rebote del rayo hasta encontrar una superficie difusa.

Además, unos pocos fotones son suficientes en una superficie difusa, ya que la iluminación en diferentes puntos es ligeramente diferente. Para representar materiales delta BSDF, un gran número de fotones son requeridos porque dos puntos, aunque cercanos entre sí, podrían tener luminancias de entrada muy diferentes.

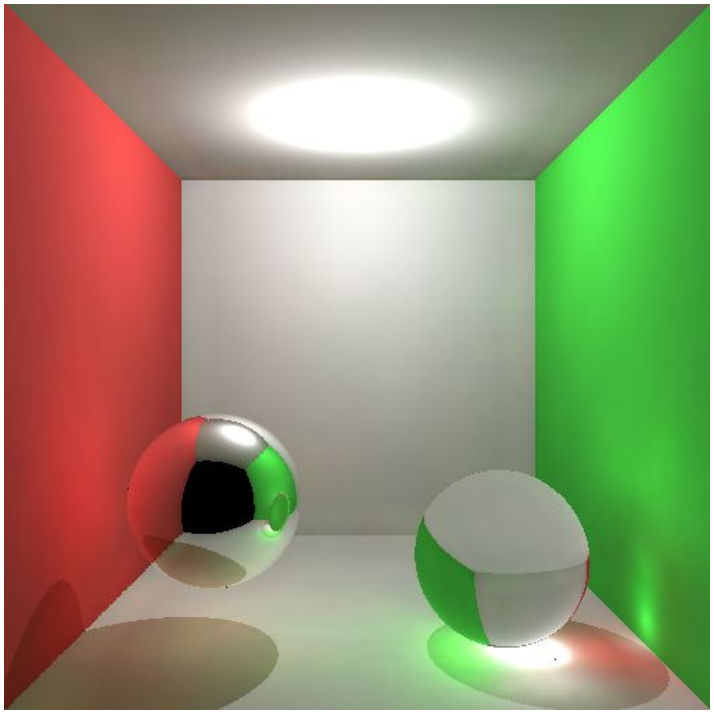


Figura 6: Escena 1 con 1.000.000 fotones, 100.000 fotones almacenados y 500 vecinos.

Figura 7: Escena 3 con 1.000.000 fotones, 100.000 fotones almacenados y 500 vecinos.

Question 2.3 *Render Scene1 using an increasing number of photons (e.g. n = {1K, 10K, 100K}), fixing the number of nearest neighbors in radiance estimation to k = 10. Now, render the same scene with n = 100K photons, and increasing the nearest neighbors density estimation kernel with k = {1, 10, 50, 100}. Discuss the results obtained, in terms of cost and image quality.*

Con Photon Mapping el número de fotones que se especifican para ser lanzados y el número de vecinos a encontrar para estimar la radiancia en un punto de la escena repercute directamente a la calidad de las imágenes obtenidas pero también al tiempo de cálculo. Conforme mayor es el número de fotones lanzados, más se aproxima la solución a la realidad (se reduce el sesgo) y habrá una mejor iluminación en la escena, pero aumenta el tiempo de ejecución de la generación del mapa de fotones, ya que hay más fotones que guardar.

Pruebas cambiando número de fotones almacenados

Todas las pruebas se han realizado con 1.000.000 fotones lanzados.

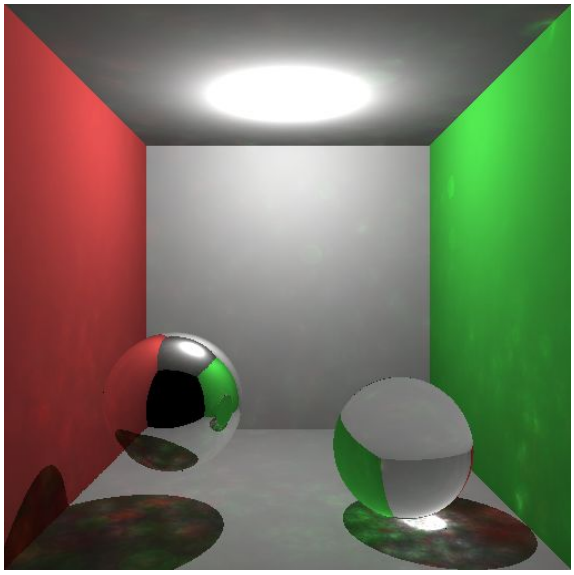


Figura 8: 1.000 fotones almacenados

10 vecinos

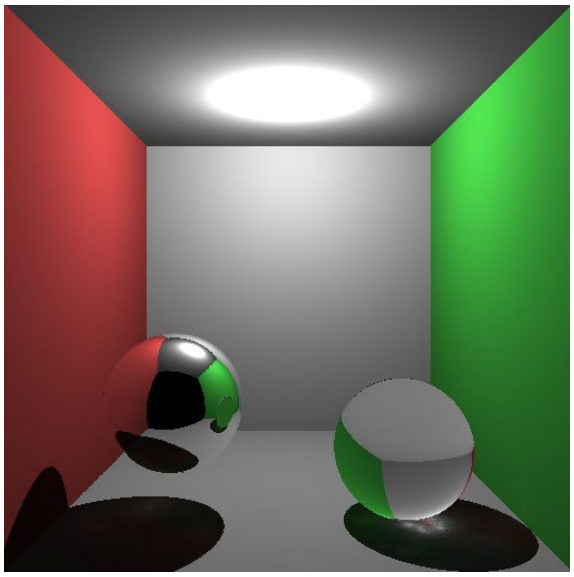


Figura 9: 10.000 fotones almacenados

10 vecinos

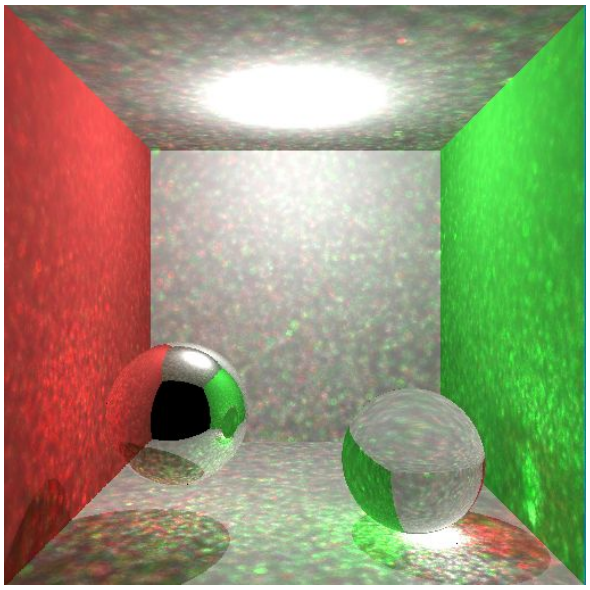


Figura 10: 100.000 fotones almacenados y 10 vecinos

Coste temporal

Figura 8

```
alejandro@DESKTOP-9G2FUPF:/mnt/e/Universidad/4Cuarto/PrimerCuatri/InforGrafica/Practicas/INFORMATICA-GRAFICA/Sma
llPM-linux/Ux$ ./smallpm -scene 1 -film-name Q2.3_n1K_K10 -pm-total-photons 1000 -pm-nb-nearest-neighbor 10
Prepared scene to render: ered..[0:0:7.7e-05]
Photons Shot: ns[0:0:0.199809]
Rendering ...[DONE]: :  [0:0:10.8977]
```

Figura 9

```
alejandro@DESKTOP-9G2FUPF:/mnt/e/Universidad/4Cuarto/PrimerCuatri/InforGrafica/Practicas/INFORMATICA-GRAFICA/Sma
llPM-linux/Ux$ ./smallpm -scene 1 -film-name Q2.3_n10K_K10 -pm-total-photons 10000 -pm-nb-nearest-neighbor 10
Prepared scene to render: ered..[0:0:7.4e-05]
Photons Shot: ns[0:0:1.92394]
Rendering ...[DONE]: :  [0:0:13.1648]
```

Figura 10

```
alejandro@DESKTOP-9G2FUPF:/mnt/e/Universidad/4Cuarto/PrimerCuatri/InforGrafica/Practicas/INFORMATICA-GRAFICA/Sma
llPM-linux/Ux$ ./smallpm -scene 1 -film-name Q2.3_n100K_K10 -pm-total-photons 100000 -pm-nb-nearest-neighbor 10
Prepared scene to render: ered..[0:0:7.6e-05]
Photons Shot: ns[0:0:20.05]
Rendering ...[DONE]: :  [0:0:15.8537]
```

Como se puede observar, un aumento en el número de fotones almacenados debe ir acompañado de un aumento del número de vecinos usados, ya que en este caso, la imagen con 10 mil fotones almacenados se ve mejor que la de 100 mil fotones, puesto que el número de vecinos usados es el mismo en los 2 casos y la diferencia entre 100 mil fotones y 10 vecinos es demasiado desproporcionada.

Pruebas cambiando el número de vecinos

Todas las pruebas se han realizado con 1.000.000 fotones lanzados y 100.000 almacenados para indirecta y cáusticas.

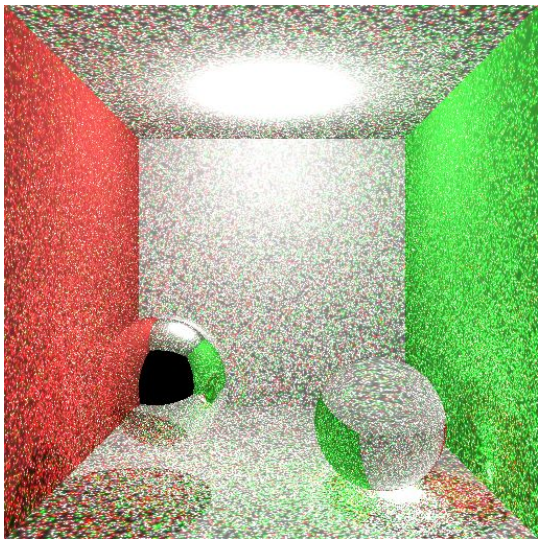


Figura 11: K vecinos = 1

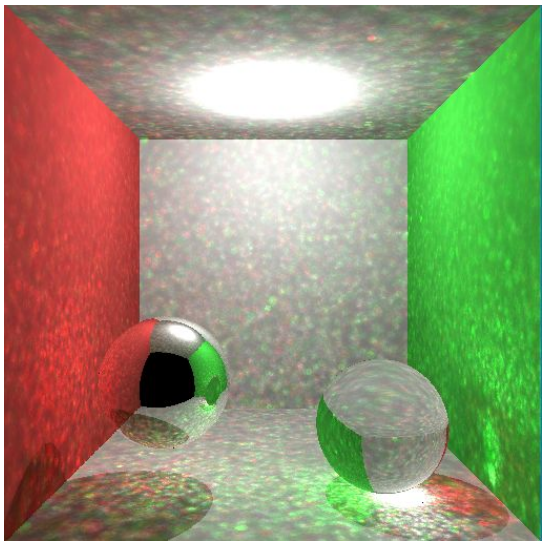


Figura 12: K vecinos = 10

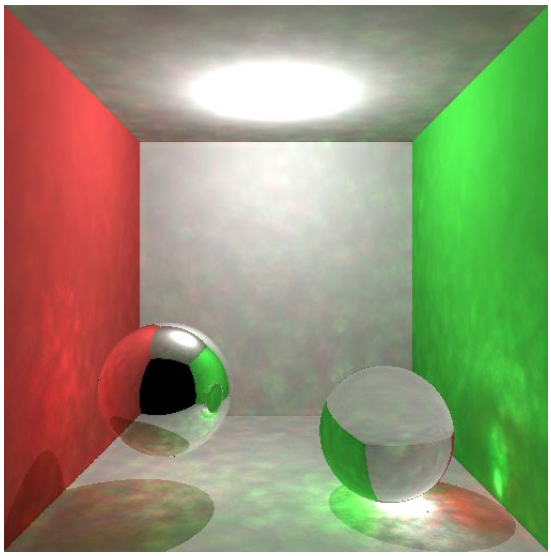


Figura 13: K vecinos = 50

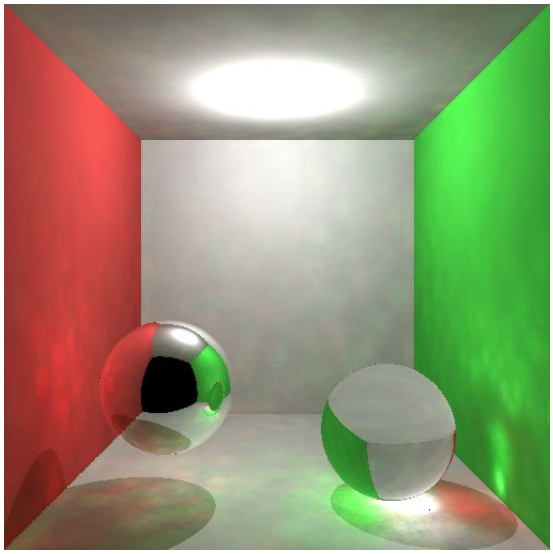


Figura 14: K vecinos = 100

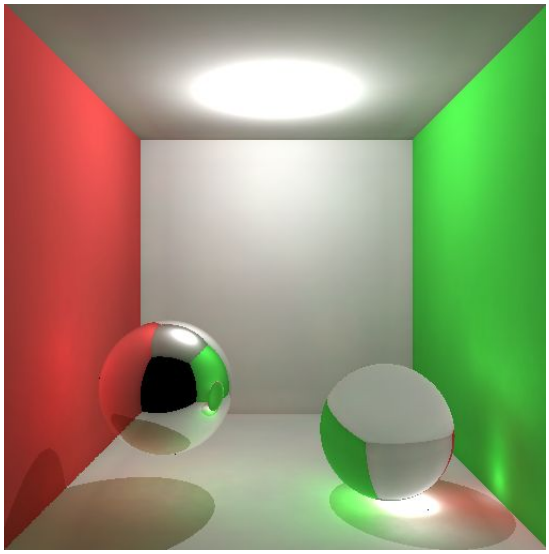


Figura 15: K vecinos = 500

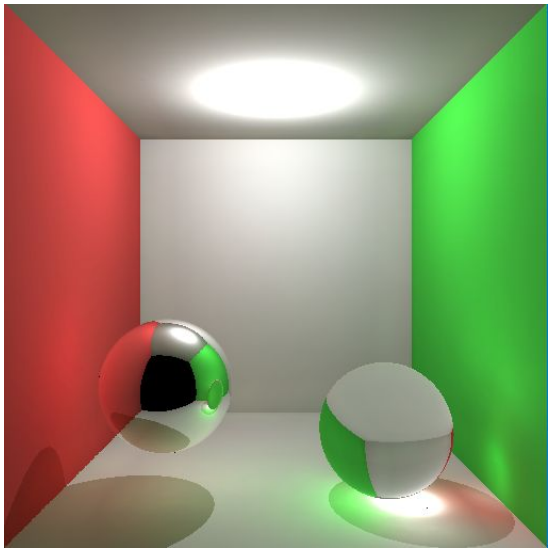


Figura 16: K vecinos = 1000

Coste temporal

Figura 11

```
alejandro@DESKTOP-9G2FUPF:/mnt/e/Universidad/4Cuarto/PrimerCuatri/InforGrafica/Practicas/INFORMATICA-GRAFICA/Small
PM-linux/Ux$ ./smallpm -scene 1 -film-name Q2.3_n100K_K1 -pm-total-photons 100000 -pm-nb-nearest-neighbor 1
Prepared scene to render: ered..[0:0:7.4e-05]
Photons Shot: ns[0:0:20.7031]
Rendering ...[DONE]: :  [0:0:7.19302]
```

Figura 12

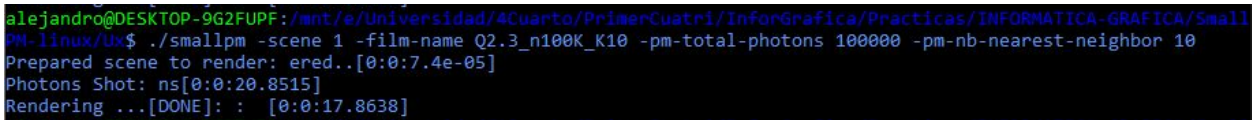


Figura 13

Figura 14

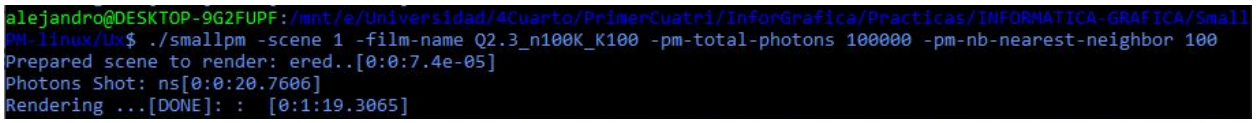
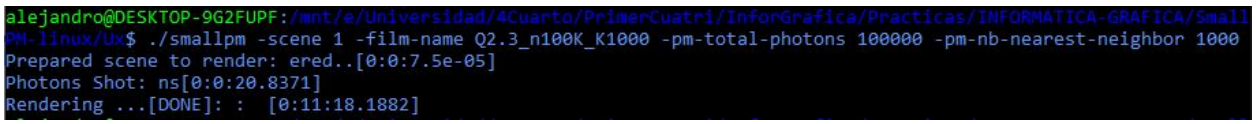


Figura 15

Figura 16



Para estas imágenes, como el número de fotones lanzados es muy alto, conforme se aumenta el número de vecinos usados se ve mejor el resultado final, ya que se está reduciendo esa diferencia excesiva entre ambos valores.

Question 2.4 *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 the Scene4 (using -scene 4) with both ray traced and photon-based direct illumination.*

Cuando se calcula la luz directa con ray tracing el torus se ve de color negro, ya que cuando el rayo lanzado desde la cámara alcanza el torus, la fuente de luz no es capaz de verlo ya que el objeto de material delta que lo recubre se interpone entre ambos (el torus no es visible para la fuente de luz). En cambio, si se calcula con photon mapping el torus queda iluminado, ya que se guardan fotones cáusticos que han impactado con este, porque incluso computando solo la luz directa, cuando un fotón choca con un material delta se sigue su camino hasta que choca con uno difuso.

Figura 17: Luz directa Ray tracing

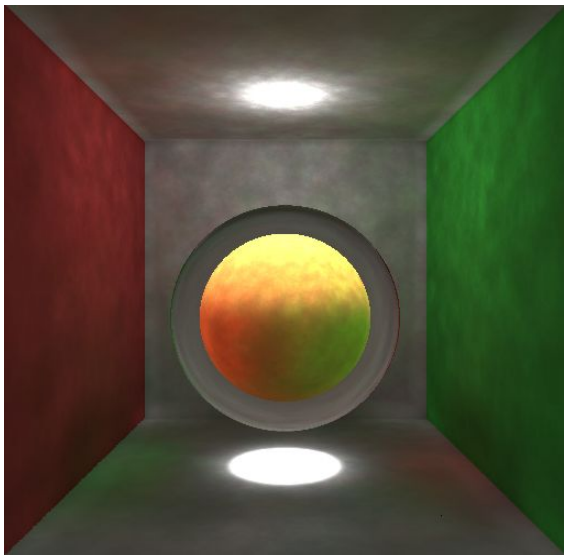


Figura 18: Luz directa Photon Mapping

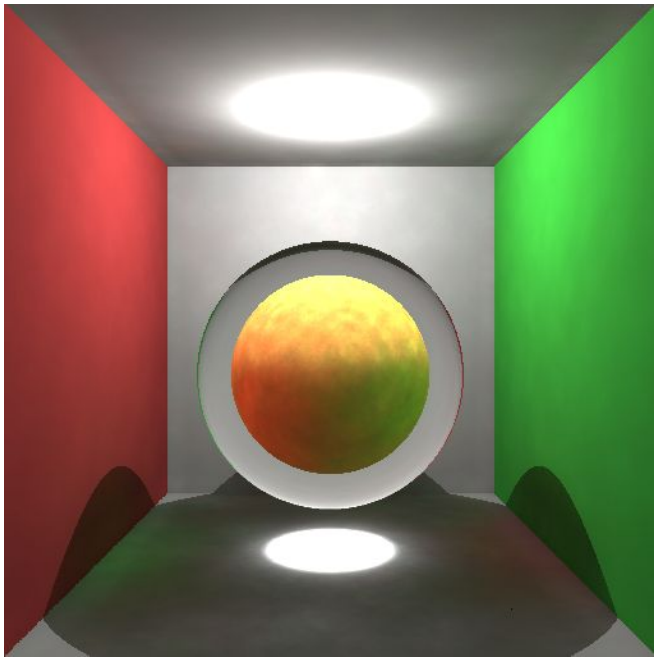


Figura 19: Imagen con Photon Mapping

5. Conclusiones

La realización del algoritmo de photon mapping ha sido útil para observar que existen otros algoritmos que buscan la creación de las imágenes a parte de path tracing.

Además, se han entendido mejor las ventajas que ofrece Photon mapping con respecto de path tracing, como ha sido el menor coste temporal y la mayor facilidad para encontrar el camino de las cáusticas, y los puntos negativos como es la aparición de sesgo.

6. Bibliografía

[1] Graphics and Imaging Lab. Photon Mapping.
https://moodle.unizar.es/add/pluginfile.php/2303601/course/section/356530/40_photon_mapping.pdf?time=1574916009098

[2] Graphics and Imaging Lab. Lighting
https://moodle.unizar.es/add/pluginfile.php/2303601/course/section/356530/03-lighting-handout.pdf