- TP 2 -

Photon Mapping

Informática Gráfica

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

755584 Sergio García
758803 Daniel González

Índice

Introducción

Tras haber completado en el anterior trabajo el desarrollo del algoritmo de path tracer, falta buscar maneras de completar la imagen generada con elementos faltantes que el propio algoritmo no es capaz de encontrar. Ésto motiva a buscar una solución que evolucione más allá del algoritmo propuesto y de esa motivación surge la necesidad de implementar el algoritmo de photon mapping.

El algoritmo de photon mapping desarrolla la idea de que los caminos que surgen de las propias fuentes de luz dejan a través de la escena elementos a los que llamaremos fotones que contienen la potencia que recibe un objeto en el punto de intersección de esos caminos con los objetos. Esto lleva a que se generan nuevos efectos relacionados con la interacción de la luz con los objetos, cosa que no veíamos con el path tracer.

1. Uso del código opcional

Para este trabajo vamos a partir del código proporcionado por el profesorado de la asignatura. Los motivos detrás de esta decisión son el tiempo limitado que le podemos dedicar a este trabajo, además de omitir la exposición al riesgo que un código que no puede ser óptimo generado a partir de nuestro path tracer. En el código proporcionado podemos ver que está, entre otros, implementado un árbol KD para acelerar las búsquedas de fotones y la función que calcula el camino aleatorio de fotones.

Esta función es ***PhotonMapping::trace_ray***, la cual calcula los fotones trazando el rayo *r* desde la fuente de luz a través de la escena y almacenando las intersecciones con la materia en las listas *global_photons* y *caustic_photons*, almacenando los fotones difusos (globales) y cáusticos respectivamente. Por eficiencia, ambos se calculan al mismo tiempo, ya que calcularlos por separado daría como resultado la pérdida de varias muestras marcadas como cáusticas o difusas. El parámetro booleano *direct*, especifica si los fotones directos (de la luz a la superficie) se almacenan o no, más adelante veremos la diferencia entre calcular la iluminación directa con Photon Mapping o con Ray Tracer. El fotón trazado inicial tiene energía definida por el parámetro *p*, que explica la potencia emitida por la fuente de luz. La función devolverá verdadero cuando haya más fotones (cáusticos o difusos) para disparar, y falso en caso contrario.
A esta función hubo que añadirle un par de retoques con el fin de poder decidir qué intersecciones queremos almacenar en esas listas. El booleano *direct* pasa a ser un enum triestado que contiene los dos anteriores además de incluir la opción de que solo se almacenen los primeros fotones que chocan contra la materia (solo luz-superficie, ver Fig1).

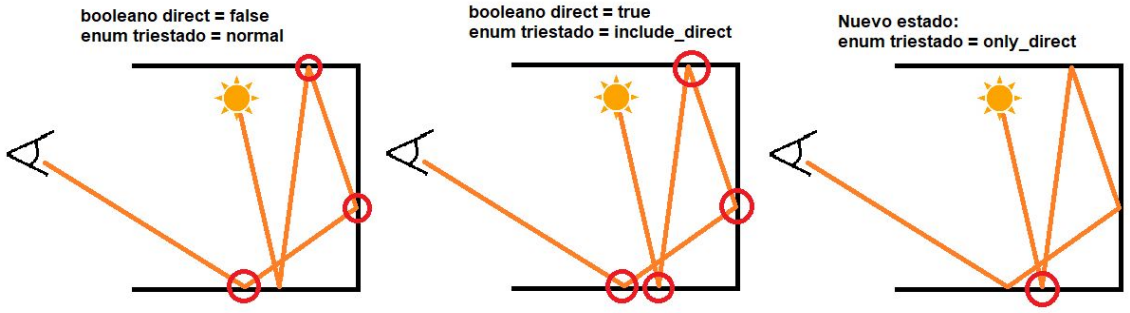


Figura 1: en rojo los fotones almacenados dependiendo la variable direct / triestado

2. Muestreo de fotones

Para construir los mapas de fotones de la escena, los usuarios necesitarán muestrear una luz en la escena para comenzar la dispersión de fotones. Esto significa seleccionar una luz de la escena y muestrear una posición y dirección de la luz elegida, en concreto utilizamos un **muestreo esférico uniforme**. El rayo definido por la posición y la dirección muestreadas será el origen de la dispersión aleatoria, mientras que la energía muestreada de la fuente de luz será la potencia inicial. En cada intersección de la dispersión aleatoria, se almacenará un fotón en el mapa de fotones, en concreto se guarda su posición, su dirección y su flujo, para poder utilizarlo posteriormente en el cálculo de la imagen final.

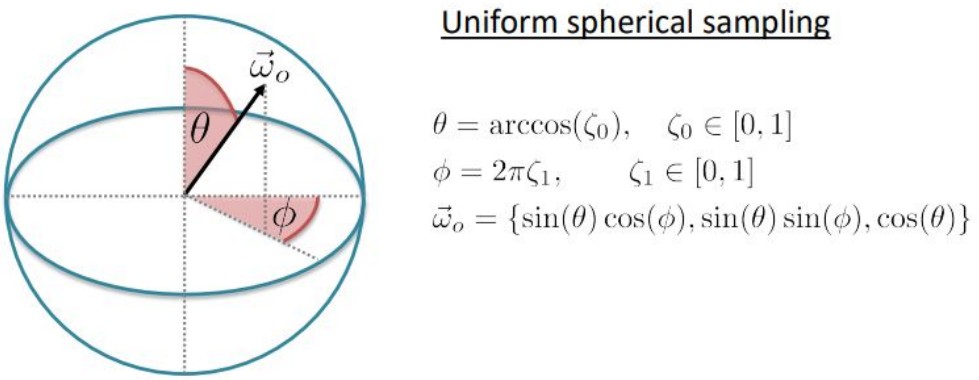


Figura 2: Muestreo esférico uniforme

En lo que respecta a código, nos encontramos en la función *PhotonMapping::preprocess*, donde realizamos el muestreo mencionado, y se almacenan en árboles KD que se balancean en esa misma función.

3. Estimación de radiancia

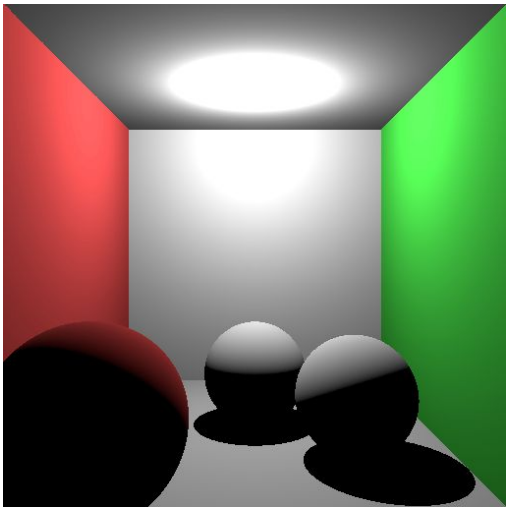
A partir de los mapas de fotones que habremos generado en la función *PhotonMapping:: preprocess*, podemos empezar a calcular las sombras en un punto. El cálculo de la radiancia la hacemos de la siguiente manera: primero buscamos la energía que reciba el punto comprobando la luz directa que le llega desde todas las luces que se encuentren en la escena, y después, procederemos a leer los mapas de fotones.

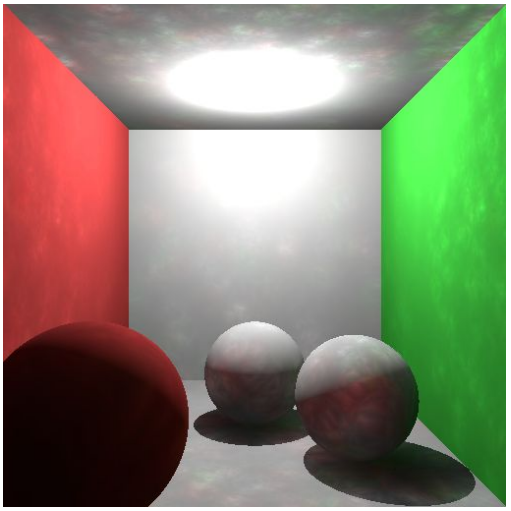
3.1 Iluminación directa

Para la iluminación directa, tenemos las posibilidades de ejecutar dos algoritmos:

- Si optamos por la opción de no usar los mapas de fotones, el algoritmo se nos queda en un ray tracing. En nuestro caso simplemente se calcula la iluminación directa que será la única que contribuya a la radiancia de la superficie alcanzada.
- Si por el contrario usamos los mapas de fotones, estaremos usando el photon mapping que se ha implementado.

En el caso de que queramos calcular la iluminación directa, es sencillo. Para el ray tracing solo tenemos que excluir del cómputo aquellos valores que no pertenezcan al algoritmo y por ende que no contribuyan a la radiancia al finalizar la función. En el caso del photon mapping, esos valores tendrán que estar descomentados (es decir, tienen contribución) y en la generación de esos mapas tendremos que indicar de que solo queremos computar la luz directa (como hemos visto en apartados anteriores, eso se consigue con un valor triestado que se pone a DIRECT_ONLY).





Figuras 3 y 4: Cómputo de luz directa en ray tracing y photon mapping respectivamente

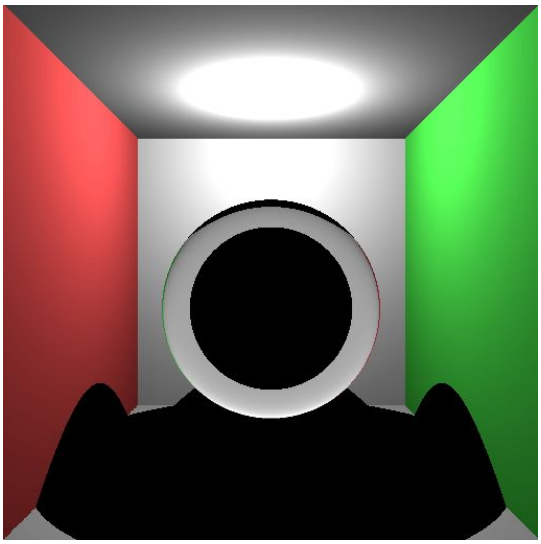
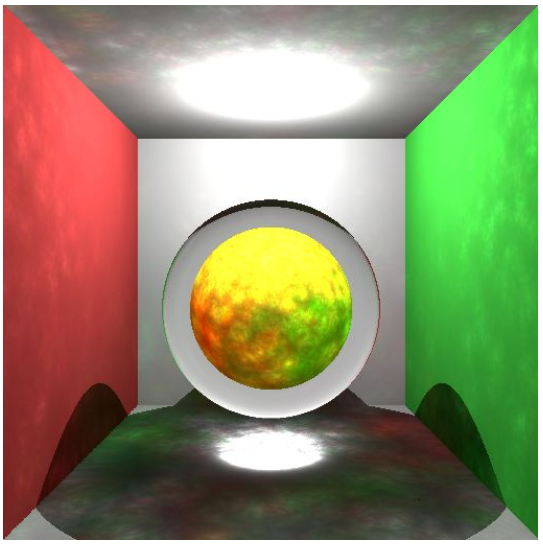
La escena 2 es una cornell box formada por tres esferas, paredes laterales de color y una luz puntual en la parte superior. Si ejecutamos el ray tracing obtenemos la imagen superior izquierda (ver Fig. 3), mientras que si renderizamos la escena con photon mapping obtenemos la imagen superior derecha (ver Fig. 4).

Si nos fijamos en las diferencias que existen entre ambas figuras, lo más perceptible es la comparación de ruido que ha generado el photon mapping con esa especie de manchas mientras que el ray tracing tiene todas las superficies con unos degradados uniformes en términos de luminosidad. Otra diferencia crucial son las sombras de las esferas, donde podemos apreciar que en ray tracing se pierde la información de la escena mientras que en la de photon mapping puede distinguirse en la sombra el contorno de la esfera respecto al suelo. Esto se debe entre otras razones por la aproximación de los fotones vecinos.

Sobre la precisión está claro que el photon mapping le gana a ray tracing. El mero hecho de no perder la información en las sombras de las esferas es un claro ejemplo de que algoritmo es más preciso. Aunque el uso de este algoritmo genera cierta cantidad de ruido para los valores por defecto que tiene el programa, veremos cómo al aumentar ciertos valores como el número de fotones o disminuir el radio de media hará que el ruido desaparezca y que, por tanto, la calidad aumente.

En la escena anterior, comprobamos los materiales difusos, pero, ¿en superficies delta?. La escena siguiente (escena 4) se compone de una esfera de cristal en medio de la cornell box, que a su vez tiene una esfera naranja dentro de la de cristal.

Si ejecutamos los dos algoritmos, obtendremos las imágenes de las figuras 5 y 6. ¿Qué está pasando en la de ray tracing? La explicación es la siguiente: al estar la esfera naranja dentro de la de cristal y solo al contemplar la luz directa, esta esfera no será visible, de ahí el gran contraste entre el halo gris y el círculo negro. En lo referente a la sombra del suelo es prácticamente idéntico a lo anterior, no es directo ya que la esfera de vidrio interfiere.

Figuras 5 y 6: Cómputo de luz directa en ray tracing y photon mapping respectivamente

En cambio, ¿por qué en la imagen renderizada con photon mapping sí que aparece la esfera naranja central? Los fotones se guardan en los mapas una vez tocan algún material difuso, y por eso permiten ver a través del vidrio, los rayos lanzados desde la cámara serán los que recojan esa energía y lo que de color a la esfera interior. Pasa exactamente lo mismo con las sombras y las cáusticas: a través del vidrio, los fotones se dispersan en la zona inferior y se recoge la energía desde los mapas de fotones.

3.2 Reflejos, Cáusticas y superficies delta

Si recogemos lo anterior y le añadimos los mapas de fotones, obtendremos una imagen completa muy aproximada a lo real , pero, ¿la ecuación de render?. Si nos acordamos del path tracer, veíamos que la ecuación de render tenía la siguiente forma:

$$L_o(x,\vec{\omega}_o) = L_e(x,\vec{\omega}_o) + \int_\Omega L_i(x,\vec{\omega}_i) f_r(x,\vec{\omega}_i,\vec{\omega}_o)|n\cdot\vec{\omega}_i| d\vec{\omega}_i$$

Igual que en el trabajo anterior, la probabilidad de lanzar un rayo y que está acierte en una luz puntual es 0. Esto en el photon mapping se traduce en que no toda la escena tiene los fotones necesarios para cubrir una superficie con una radiancia determinada, es por eso que también tenemos que buscar la luz para saber su contribución directa (con n=inf sería posible).
Ahora son fotones, no rayos, y es por eso que la ecuación de render es diferente: la radiancia que incide en un punto ya no es otra ecuación de render, sino que ahora procedemos a calcular la radiancia estimada en un área para calcular la energía final:

$$L_o(x,\vec{\omega}_o) = L_e(x,\vec{\omega}_o) + \int_\Omega f_r(x,\vec{\omega}_i,\vec{\omega}_o)\frac{\Phi}{dA\cos(\theta_i)d\vec{\omega}_i}\cos(\theta_i)d\vec{\omega}_i$$

$$L_o(x,\vec{\omega}_o) = L_e(x,\vec{\omega}_o) + \sum_{photons=1}^{N} f_r(x,\vec{\omega}_i,\vec{\omega}_o)\frac{\Phi_i}{\pi r_k^2}$$

En la figura 7 podemos observar la imagen generada al renderizar la escena 1, la cual es una cornell box en la que hay dos esferas, ambas de materiales delta. La esfera de la izquierda está formada por un material de espejo, por lo que refleja toda la luz que llega a su superficie, es por ello que su sombra es muy similar a la sombra de un material difuso. La esfera de la derecha, en cambio, está formada por un material de cristal, dieléctrico, que refleja y refracta la luz que llega a su superficie, la refracción de la luz que viene del punto de luz atraviesa la esfera y genera el área circular iluminada que se observa en dicha figura.

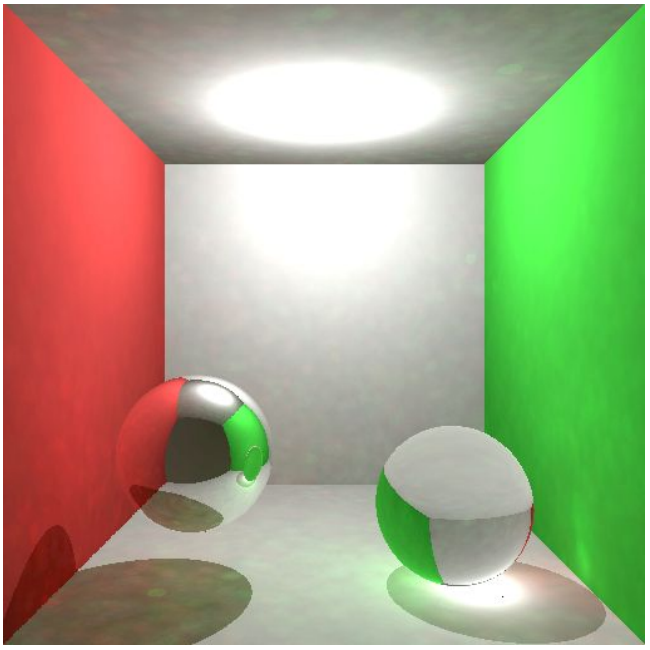


Figura 7: Sombras de materiales delta

3.3 Phong (Opcional)

En los archivos proporcionados, se ha visto que existe la posibilidad de que un objeto tenga como material el que se modula a través de la BRDF de Phong. En las escenas creadas no aparece ningún material Phong, todos los materiales de las escenas son de tipo delta o Lambertianos. Hemos creado una escena con esferas Phong, a la izquierda en la imagen, y esferas Lambertianas, a la derecha de la imagen (ver Figura 8).

Para que los materiales Phong tengan diferente iluminación a los materiales Lambertianos, hemos modificado la función *PhotonMapping::shade*. Al calcular la iluminación directa, y la estimación de radiancia tanto en fotones globales como en fotones cáusticos, detectamos que es Phong mirando que el coeficiente especular del material sea distinto de cero. Una vez comprobado aplicamos el cálculo de BRDF característico de los plásticos en vez del de los materiales Lambertianos.

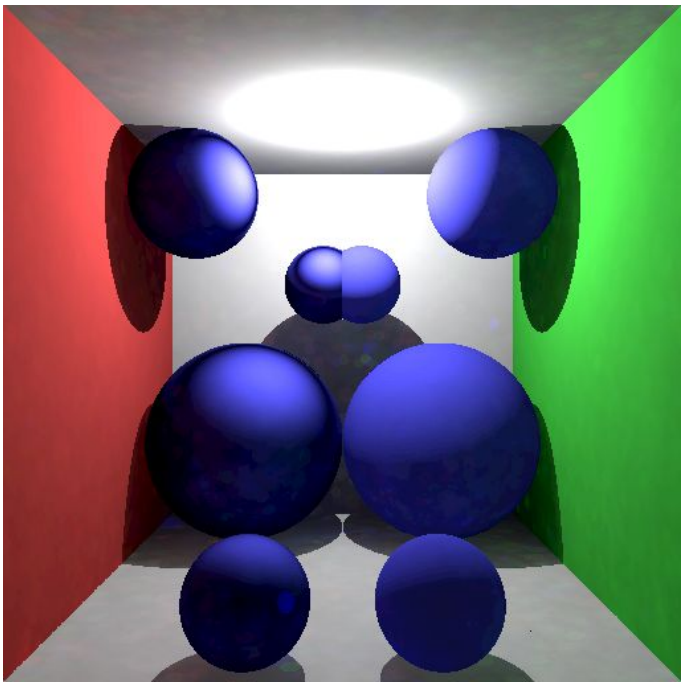
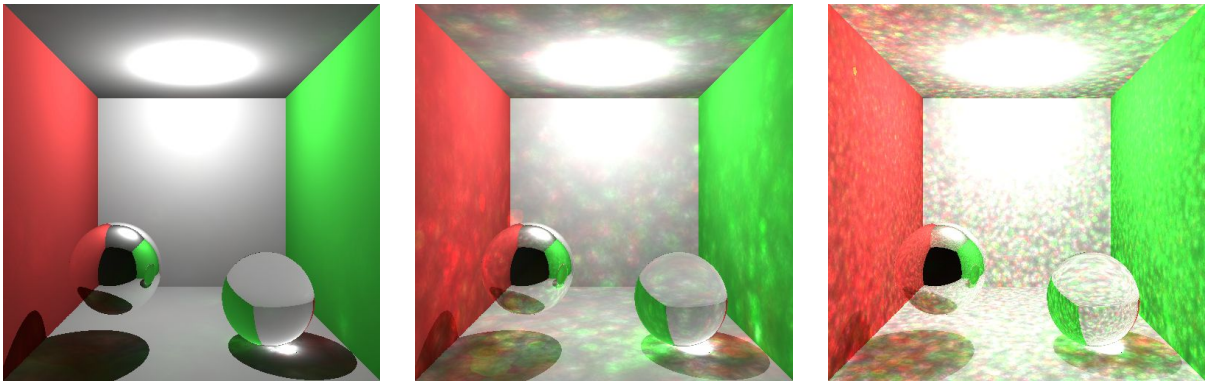


Figura 8: Material Phong

4. Parámetros de renderizado (n y k)

Una vez explicados los fundamentos en los que se basa el photon mapping, queda ver cuales son las interacciones entre el usuario y el algoritmo, en concreto las variables que controlan la calidad del cómputo del algoritmo. Una de esas variables en las que el algoritmo se apoya es el número de fotones vecinos respecto del punto de intersección para determinar la radiancia. Otra de las variables es el número de fotones que pueden muestrear las luces de la escena.

Empezando con la última mencionada, se ha mantenido constante el número de fotones mientras que se modificaba los k-vecinos. Para estas pruebas, se va a emplear la escena 1.



Figuras 9,10 y 11: Photon mapping k=10, n=1k, 10k y 100k (de izq a derecha)

Como resultado, podemos observar que a medida que se incrementan los fotones, la energía aumenta y en el último caso puede llegar a observarse que la imagen se quema por exceso de los mismos. Otra observación puede ser que las cáusticas se definen bastante bien con el mínimo de fotones.

Si en cambio dejamos fijo el número de fotones al máximo muestreado del experimento anterior y variamos el número de k-vecinos, obtenemos las siguientes imágenes.

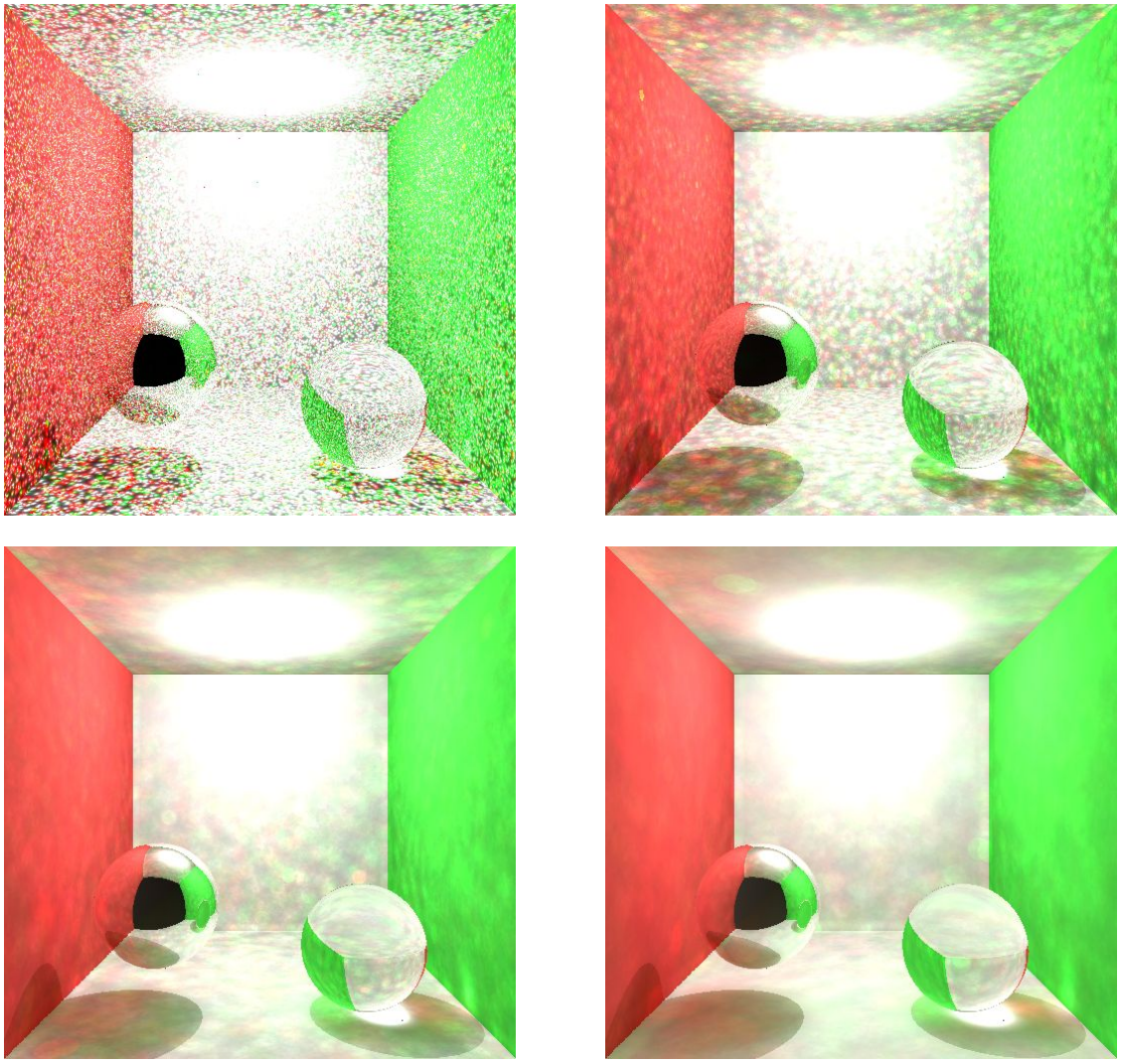


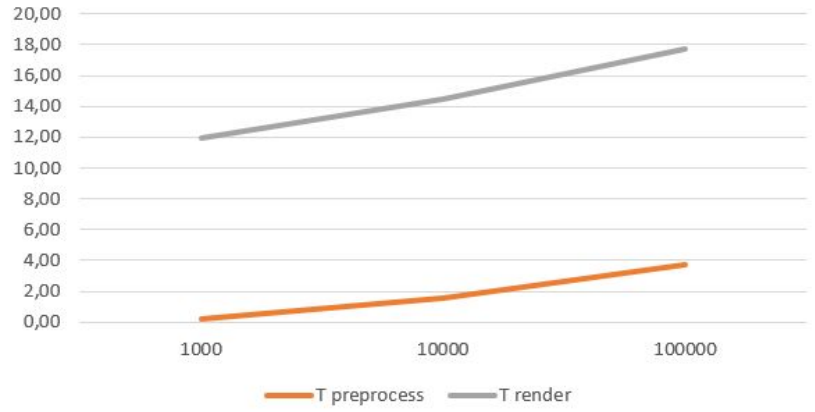
Figura 12-14: Incremento de número de k-fotones vecinos en cada escena, de izq a dcha: 1, 10, 50 y 100

De este experimento, podemos concluir que cuanto mayor sea el número de k-vecinos, es decir la media en un área, más difuminado quedarán los fotones. En especial, resaltar que con el valor de k=1 estamos obteniendo una imagen del calor real de la fuente de luz, viendo que donde más se acumulan los fotones es donde más se quema recibiendo una intensidad mayor que en otras zonas de la imagen.

Si hablamos de métricas de rendimiento, en un ordenador no muy potente se han obtenido los siguientes resultados:

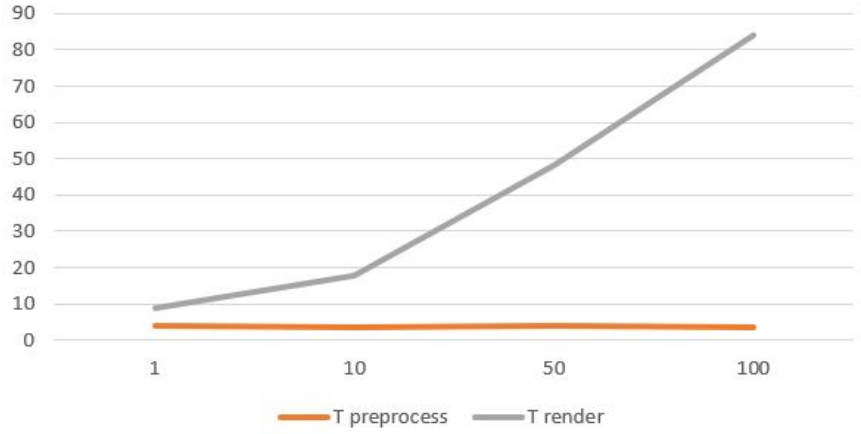
Experimento 1	n=1000	n=10000	n=100000
T preprocess	0.21s	1.59s	3.72s
T render	11.95s	14.44s	17.70s

De forma gráfica:



Experimento 2	k = 1	k = 10	k = 50	k = 100
T preprocess	3.78s	3.72s	3.77s	3.69s
T render	8.66s	17.70s	48.26s	1m 24s

De forma gráfica:



En lo referente al Tiempo de render, podemos ver que para el experimento 1, tenemos un crecimiento lineal no muy pronunciado como lo tenemos en el experimento 2. Para el tiempo del preprocess es muy pequeño incluso para el número tan grande de fotones (3.7s por 100K fotones).

5. Progressive Photon Mapping (Opcional)

Durante los renderizados del photon mapping de los apartados anteriores, se puede apreciar como los fotones junto a los k-vecinos forman figuras irregulares que son aleatorios en cada ejecución. Este opcional trata de mejorar la calidad de esas imágenes, tratando de llegar a un resultado donde esas figuras irregulares se vuelven difusas.

Para ello, se ha modificado el *RenderEngine::render* y la clase *PhotonMapping* (también su shade). En la primera función, se itera sobre un número de veces que hemos especificado a través de un nuevo flag. Una iteración completa consiste en la ejecución del preprocesado y de una función interna por pixel que llamará a *PhotonMapping::shade*. En la iteración además se guarda en valor de la máxima distancia a la que está el k-vecino para luego generar un radio. Tras finalizar esa iteración es importante vaciar los KDTree (los mapas de fotones), se escala el radio obtenido y entra en juego la modificación en *PhotonMapping:: shade*, donde las funciones de búsqueda en el KDTree pasan de buscar por vecinos a buscar por el radio.

Como experimento, se ha escogido la escena 1 y se ha propuesto hacer 5 iteraciones del progressive photon mapping. El resultado es el siguiente:

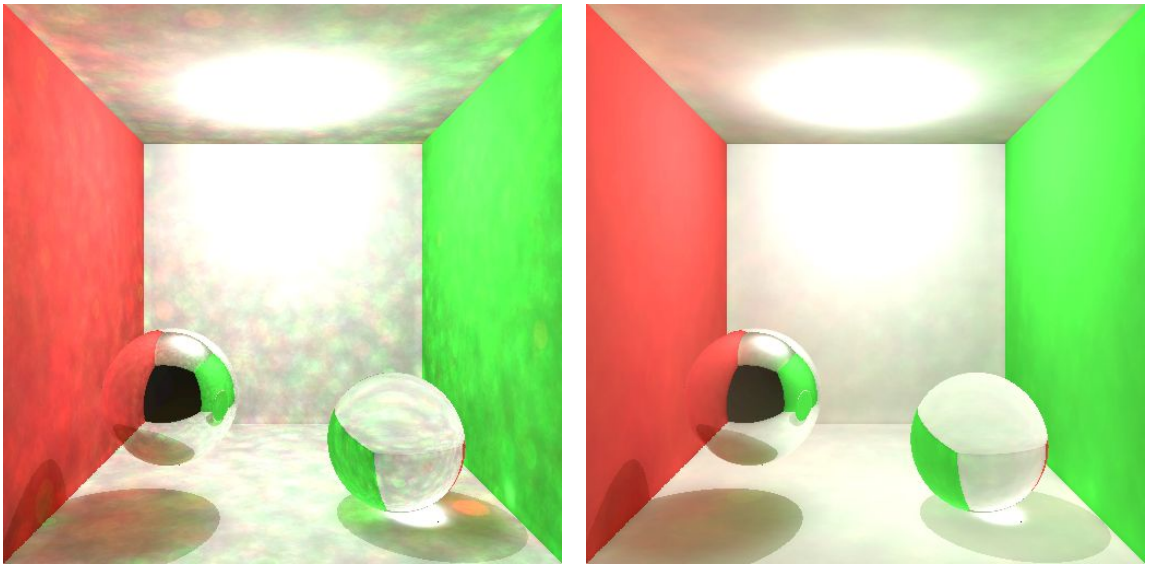


Figura 15 y 16: k=50, n=100k en ambas, iteraciones 1 y 5 respectivamente.

Como se puede observar, una gran cantidad de ruido desaparece y se difumina mucho más la escena, obteniendo buenos resultados. Si hablamos de los aspectos técnicos, la escena ha sufrido 5 iteraciones: 5 preprocess y 5 shade/pixel, esto se traduce en un mayor tiempo de cómputo:

Ttotal	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5
Sin PPM	47.1s	-	-	-	-
Con PPM	46.2s	46.7s	79.9s	189.9s	215.7s

El tiempo total asciende a 578.4s frente a 47s, un 1283% más lento, pero con una calidad altamente superior.

Bibliografía

[1] Diapositivas del curso de Informática Gráfica

[2] Claude Knaus and Matthias Zwicker. Progressive photon mapping: A probabilistic approach. ACM Trans. Graph., 30(3), 2011.