Path tracing

Asignatura: Informática Gráfica

Fecha: Enero 2020

Autores: Andrés Gavín Murillo, 716358

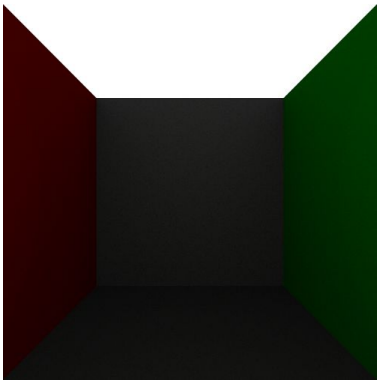
Abel Naya, 544125

Índice

Introducción

En este documento se detalla el trabajo realizado sobre path tracing para la asignatura de informática gráfica. Se detalla la forma en la que se ha implementado el trazado de rayos por montecarlo, junto a los detalles que se consideran importantes, así como las respuestas a las preguntas planteadas por los profesores.

Se muestran los detalles y resultados sobre los distintos componentes, así como imágenes comparativas que ayudan a entender el proceso implicado. Salvo que se indique lo contrario, las imágenes han sido generadas únicamente con el código desarrollado, mediante el uso de un ordenador con procesador intel core i9-9900K 3.6GHz (16 threads de paralelismo) con tamaño 500x500 y 500ppp. La escena por defecto consta de 5 planos formando una caja (abierta en dirección contraria a la cámara) con el plano superior emisor blanco, la pared izquierda difuso rojo, la pared derecha difuso verde, y los otros dos planos difusos grises. Cada escena además es post procesada mediante clamping y/o gamma, ya sea de manera automática o manual. Se explicarán cada uno de estos conceptos en los apartados correspondientes.



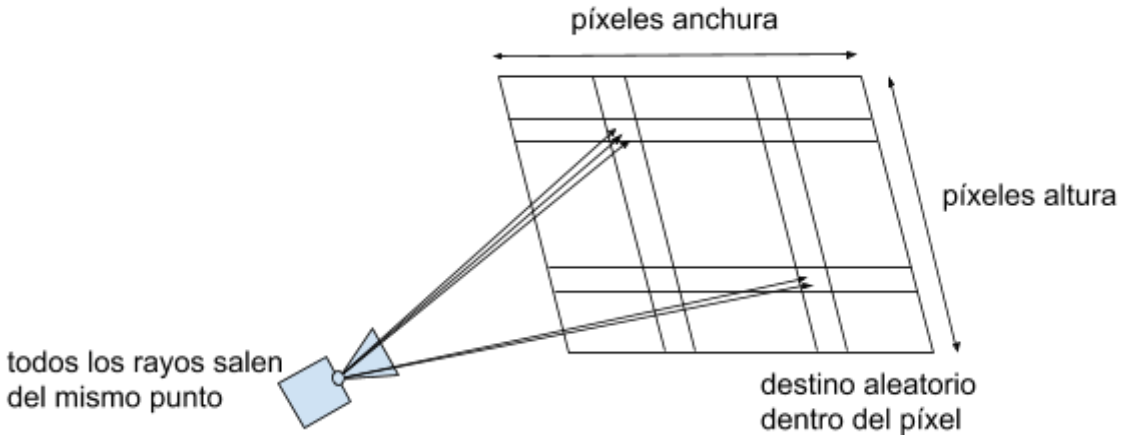
Path tracing

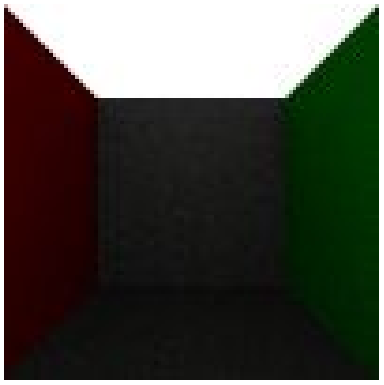
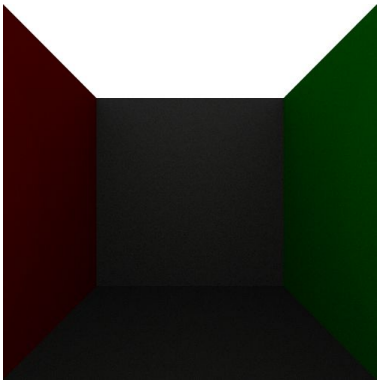
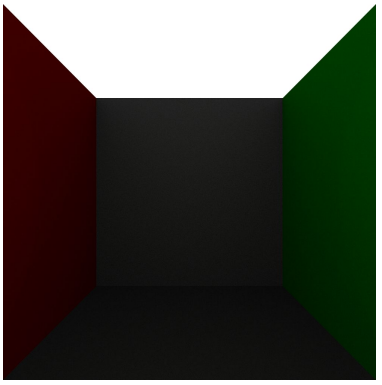
El path tracing consiste en renderizar una escena mediante el trazado de rayos que parten de la cámara y van 'rebotando' entre los diferentes objetos que se encuentran. La característica principal respecto a ray tracing es que en cada intersección en lugar de lanzar varios nuevos rayos (lo que hace aumentar exponencialmente el número de éstos) se lanza uno solo, o incluso ninguno, elegido de forma aleatoria entre los diferentes eventos que el material tenga asociados; y en su lugar se lanzan varios rayos desde la cámara, que se median para obtener el color.
Los diversos componentes implicados en el proceso son:

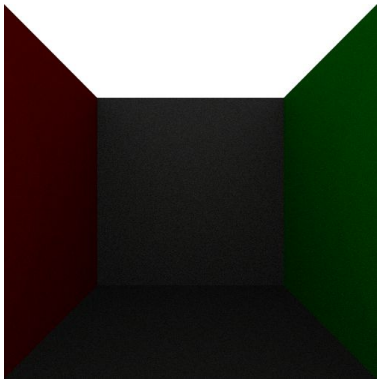
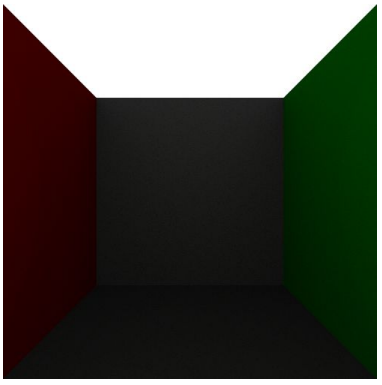
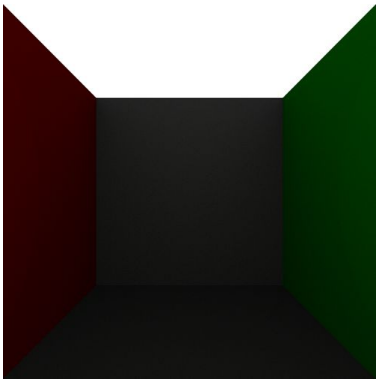
Cámara

El modelo de cámara implementado es un pinhole, es decir todos los rayos salen de un mismo punto de la escena hacia direcciones diferentes. Estas direcciones son hacia un rectángulo a poca distancia, de forma que cada rayo se lanza hacia uno de los píxeles de

este rectángulo (se puede especificar el número de píxeles por parámetro). Al lanzar un rayo a un pixel se toma aleatoriamente un punto dentro de éste, de forma que al hacer la media no se produzca ruido provocado por coger puntos equidistantes. A más píxeles mayor tiempo de renderizado aunque mejor definición.

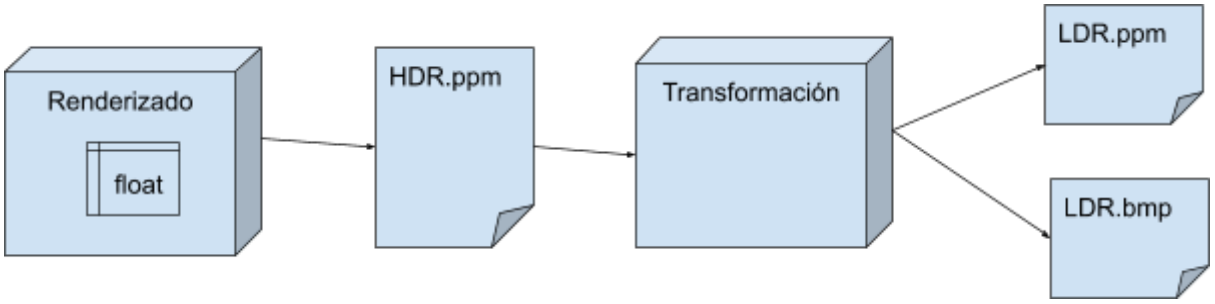


		
100x100px, 250ppp 0,193s	500x500px, 250ppp 3.578s	1000x1000px, 250ppp 14.275s

		
500x500px, 100ppp 1,497s	500x500px, 500ppp 7,227s	500x500px, 1000ppp 14,227s

Guardar/Cargar imágenes

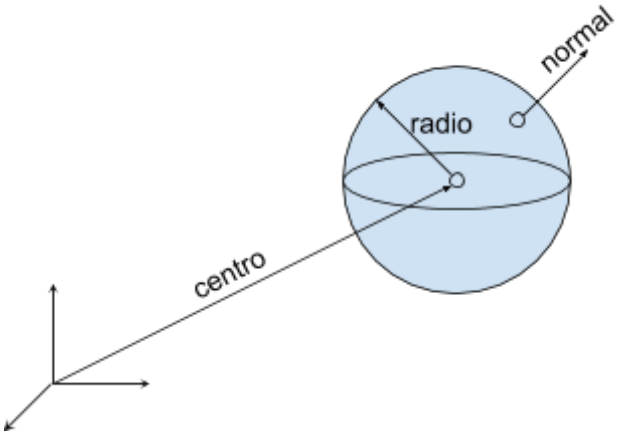
Por indicación del profesorado las imágenes generadas como números positivos float se almacenan con resolución HDR (High definition range, números enteros entre rango variable) y un segundo programa lee estos ficheros y los convierte a LDR (formato estándar ppm o tipo imágen bmp) en función de los parámetros que se proporcionen.



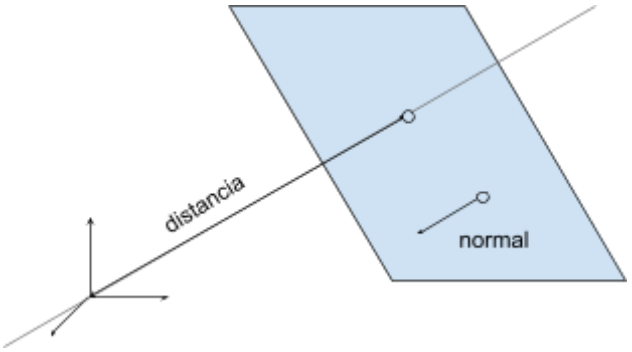
Geometrías: Esferas y planos

Las primitivas originales utilizadas que fueron requeridas han sido esferas y planos.

- Una esfera está definida con el centro (punto en coordenadas de la escena) y el radio. Esto también permite definir una normal, que apunta en dirección opuesta al centro.

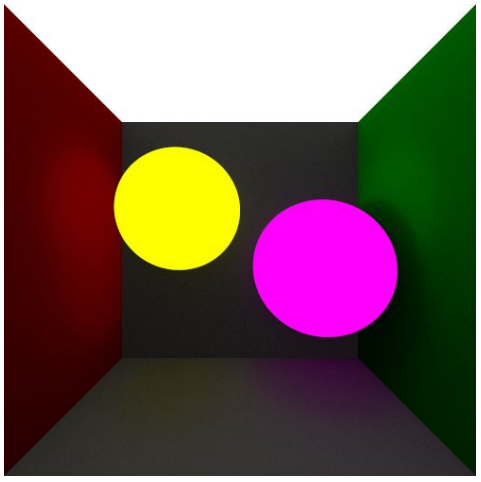


- Un plano está definido con su normal (es constante en todos sus puntos, por lo que ya no hace falta calcularla) y la distancia al origen (que coincide con el módulo del vector paralelo a la normal que pasa por el origen y corta al plano).



Material emisor

El tipo básico de material que se requería. Simboliza un material que emite luz en todas las direcciones desde todos sus puntos. La luz es una tupla RGB por lo que un material emisor solo tiene asociado una tupla con estos parámetros. Notar que cuando un rayo choca con un objeto emisor se considera el fin del camino, tomando el color de dicho objeto para el cálculo del color del camino completo.

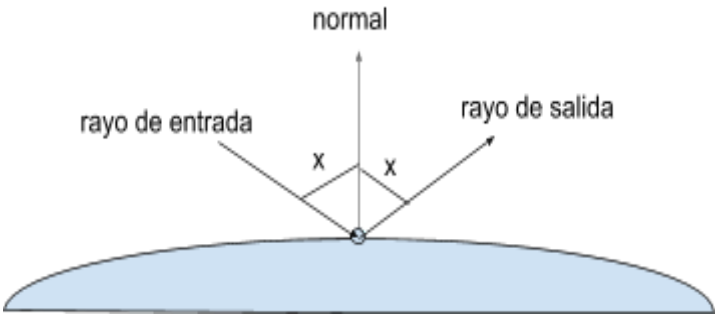


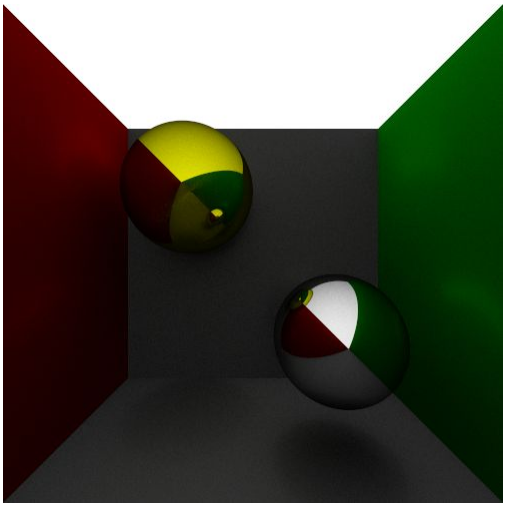
Dos esferas emisoras

Material reflectante

Los objetos no emisores reflejan la luz (no se han considerado objetos con ambas propiedades) y lo hacen en función de varios tipos de propiedades, que cada objeto puede tener independientemente del resto. Estas propiedades, definidas como colores que son multiplicados a lo largo del camino, son:

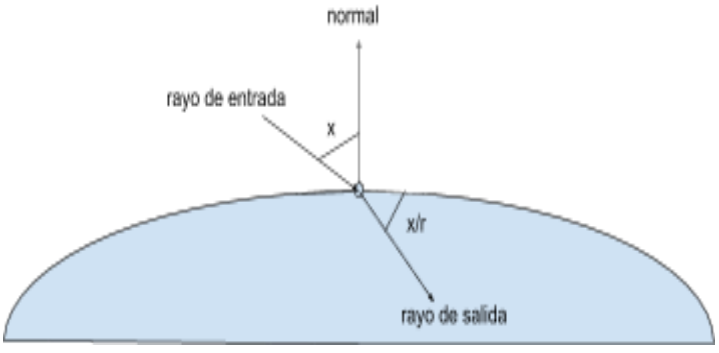
- Especular perfecto: Representa un espejo en el que cada rayo es rebotado en la dirección simétrica respecto a la normal.
  El factor correspondiente a dicho rebote es el color del objeto en dicho punto. Aunque lo normal es darle un color blanco para que casi todos los rayos sean rebotados tal cual, es posible especificar otros colores para obtener espejos tintados.

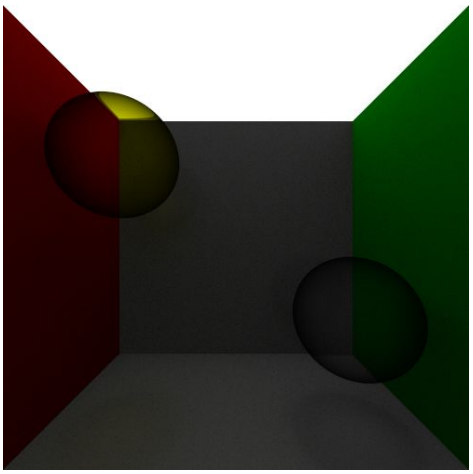




Esferas especulares perfectas

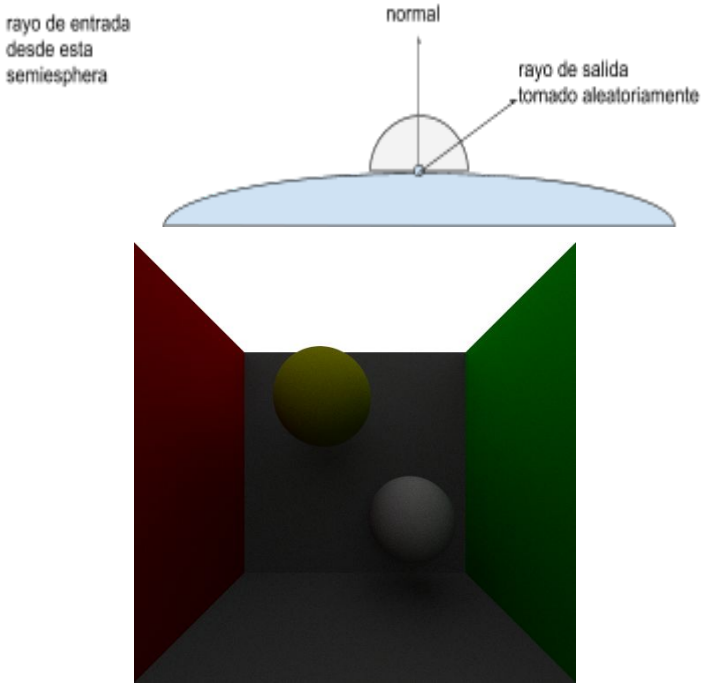
- Refracción perfecta: representa un objeto transparente (cristal, agua, ámbar,...) en el que los rayos son refractados, haciendo que 'atraviesen' el objeto con algo de desviación. Para el cálculo de la refracción se ha utilizado la ley de Snell dándole a cada objeto y a la escena en general un índice de refracción para desviar más o menos el ángulo de refracción, de tal manera que para calcular el ratio de refracción hay que dividir el índice de refracción del medio (escena u otro objeto) del que parte el rayo de entrada entre el índice de refracción del objeto refractante (el ratio para salir del objeto refractante se calcula como el inverso del ratio de entrada).
  Al igual que antes se le puede dar un color blanco para simular objetos translúcidos, o alguno diferente para simular tintes (por ejemplo tonos azulados para simular agua).





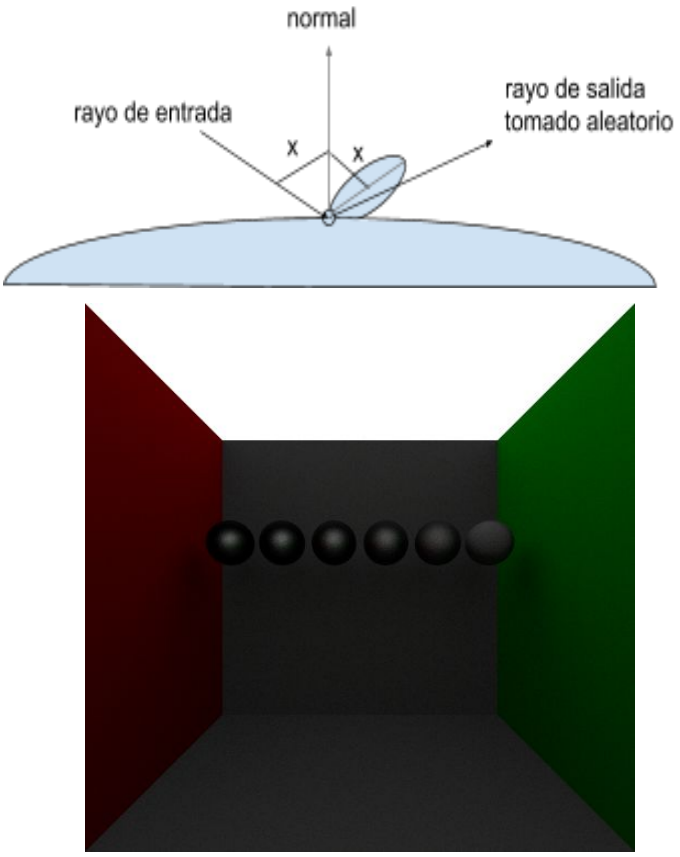
Esferas refractantes

- Phong Difuso: Un objeto difuso refleja la luz en todas las direcciones de la semiesfera positiva y con la misma intensidad. No tiene en cuenta la dirección de entrada (salvo para saber por qué lado del objeto viene), así que el cálculo de la dirección de salida se realiza tomando un vector aleatorio en la semiesfera definida por la normal.
  El coeficiente de dicho rebote equivale a dividir el color del objeto por la constante Pi.



Esferas difusas

- Phong especular: A diferencia del phong difuso en esta componente si se tiene en cuenta la dirección de entrada, pues un objeto de este tipo refleja principalmente en dirección al reflejo del rayo de entrada. Consta además de un parámetro s que permite aumentar o disminuir cuánto se desvía de la reflexión perfecta.



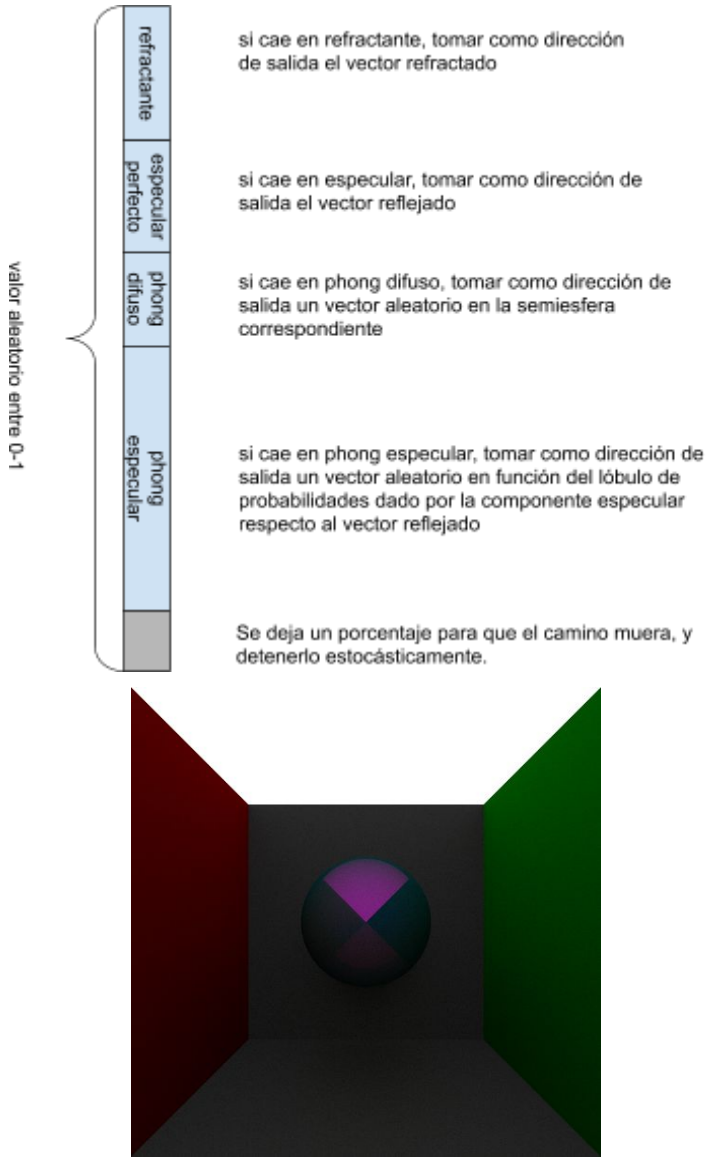
Esferas especulares con distinto ratio s

Monte Carlo

Para realizar un renderizado físicamente correcto se debería hacer la integral en cada rebote, pero esto implica saber la luz que llega desde cada dirección. Lo que ray tracing implementa es un número concreto de rayos que tomar en cada rebote, pero esto hace aumentar el número de rayos de manera exponencial. El proceso que se sigue en path tracing es tomar un único rayo desde cada rebote, y dejar el parámetro para el número de rayos que se lanzan a cada píxel (cada uno a una dirección aleatoria dentro de este).
Aunque solo tengamos un rayo por rebote, el problema del renderizado físicamente correcto sigue estando y al hacer la media de los caminos las integrales deberían ser equivalentes. Si tenemos un objeto especular perfecto o refractante el nuevo vector es fácil de calcular, y es único, pero con phong tenemos varios entre los que elegir.
Con Montecarlo esto se consigue haciendo que la nueva dirección sea aleatoria dentro de las posibles válidas, para que así en el límite es como si se navegaran todas las posibles direcciones, y por tanto el color converja. Para phong difuso por ejemplo se toma un vector aleatorio dentro de la semiesfera, y para especular se toma en función de ese lóbulo de probabilidades.

¿Pero qué pasa si un objeto tiene componentes especulares y difusas, o es parte refractante y parte especular (por ejemplo agua), o tiene los 4 tipos de componentes? En ese caso en el que tenemos la decisión de tomar uno u otro componente, se realiza mediante ruleta rusa, donde cada probabilidad es proporcional a la máxima componente rgb

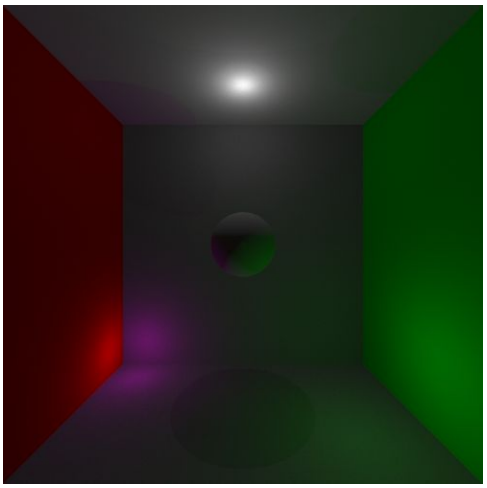
de su color. De esta forma componentes con mayor color (más visuales) serán tomados más a menudo.
Notar además que se deja un porcentaje para que el camino muera (no se lanzan más rayos) permitiendo así además no tener que implementar un número máximo de rebotes, siendo la detección automática.



Esfera con los 4 tipos de propiedades (refracción amarilla, especular morado, phong difuso cyan y phong especular azul)

Luces puntuales

La incorporación de luces puntuales se ha desarrollado generando rayos de sombra desde cada intersección a cada luz puntual, como si de nuevos path se tratase en caso de que la luz sea visible, y mediando entre ellos. Al ser luces puntuales es necesario además tener en cuenta que la luminosidad que genera se reduce conforme te alejas (pues se reparte entre más superficie) por lo que en el caso de los rayos de sombra es necesario además dividir por la distancia al cuadrado desde la luz puntual hasta el primer rebote.



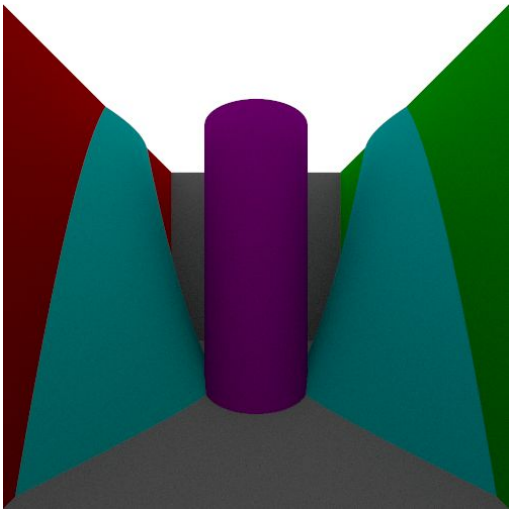
Tres luces puntuales, sin materiales emisores. Blanca en la parte superior, morada en la esquina inferior izquierda y verde en la parte inferior derecha.

Extensiones

Otras geometrías: Cuádricas y triángulos

Se han implementado, además de las geometrías originales, cuádricas y triángulos.

- Una cuádrica está definida con los coeficientes de la ecuación Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fxz + Gx + Hy + Iz + J = 0. Esto permite realizar elipsoides, cilindros o conos, entre otros. Las ecuaciones para implementar estas cuádricas se han obtenido de [1].

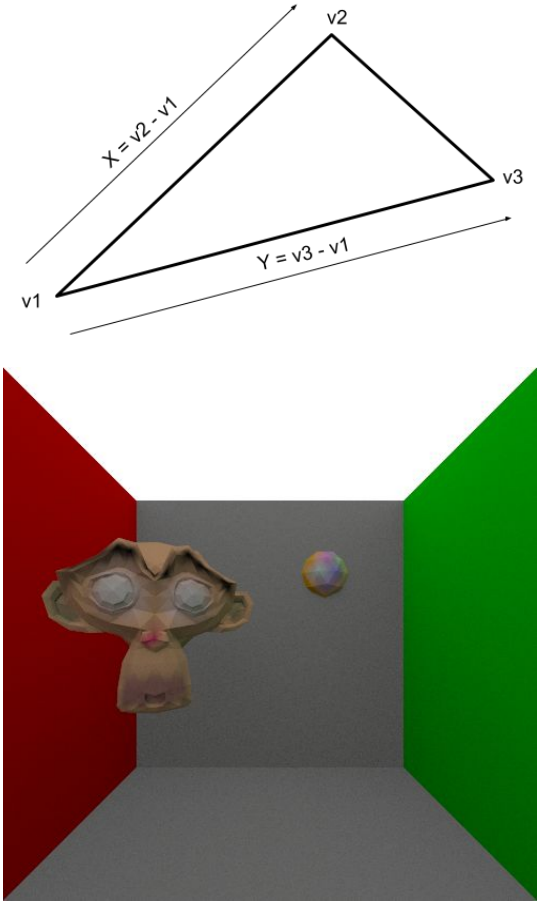


Ejemplo de cuádricas

- Un triángulo está definido con el plano en el que está y sus tres vértices, a partir de los cuales se obtienen los vectores X e Y que lo forman. Gracias a esto podemos

[1] http://skuld.bmsc.washington.edu/people/merritt/graphics/quadrics.html

saber si un rayo intersecciona con él mirando que interseccione con el plano que lo forma y que el punto caiga dentro del área formado por sus dos vectores. La utilidad de los triángulos es poder formar mallas de triángulos para representar objetos complejos, para lo cual también se ha desarrollado la carga de objetos en formato ply.



Dos ejemplos de figuras ply formadas por triángulos.

Estructuras de aceleración

Para los objetos formados por mallas de triángulos (ficheros .ply) se han implementado esferas que agrupan todos los triángulos del objeto (bounding volume hierarchies). Así, se consigue acelerar el path tracing ya que, en lugar de comprobar si un rayo intersecta con cada uno de los triángulos que forman el objeto hasta dar con el más cercano (si se produce la intersección), primero se comprueba si el rayo intersecta con la esfera que recubre la malla de triángulos y sólo si intersecta con la esfera se hace la comprobación para cada triángulo.

Para generar esta esfera, al leer un objeto del fichero .ply se guardan los puntos mínimos y máximos de cada uno de los ejes x, y, z del conjunto de triángulos que lo forman, es decir, *xMin*, *yMin*, *zMin*, *xMax*, *yMax*, *zMax*. De esta manera, se obtienen los puntos *min* y *max* que representan los límites del diámetro de la esfera, por lo que el centro de la esfera es el punto intermedio a estos dos puntos y el radio de la esfera es la mitad del módulo del vector que forman estos dos puntos.

Paralelización

Para acelerar el proceso de renderizado se ha paralelizado la imágen dividiendo el eje y (la altura) en partes equitativas. Esto permite una disminución en tiempo de renderizado ya que cada hilo de ejecución (thread) realiza el path tracing correspondiente a cada píxel que le haya tocado calcular. Por ejemplo, para una imagen 10x10px, 2ppp y 10 threads, a cada thread le corresponde calcular el path tracing para 10 píxeles, es decir, cada thread calculará 20 caminos (10px, 2ppp). El número de threads en los que se paraleliza se elige a la constante hardware_concurrency() de la biblioteca thread, que indica el número de hilos que el ordenador hardware admite.

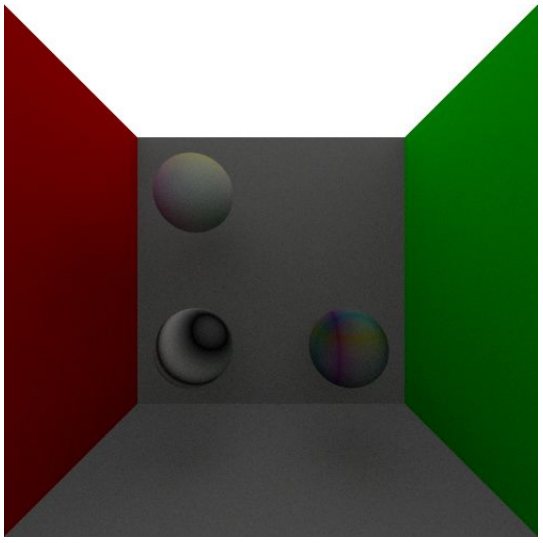
Tiempos de renderizado de la escena por defecto (ver apartado cámara) con tamaño 500x500 y 500ppp. El valor de hardware_concurrency() es 16.

threads	1	2	4	8	16	32	64
tiempo	1m 6,693s	38,335s	20,164s	10,202s	7,419s	8,374s	12,852s

Texturas

Se han implementado diferentes tipos de texturas que parten de colores sólidos, donde la textura básica es únicamente un color sólido rgb. En el caso de los triángulos también existen colores asociados a los vértices (vertex color), donde el color dada una posición del triángulo es proporcional a la distancia a los vértices (color del vértice más cercano o media ponderada de los colores de los vértices en base a la distancia a estos).

Además de estas dos texturas básicas, se han definido tres tipos de texturas 2D que modifican el color correspondiente a una posición multiplicándolo por el seno, el seno cuadrado o el seno por el coseno de dicha posición.



Esferas con diferentes tipos de texturizado basados en posición

Preguntas

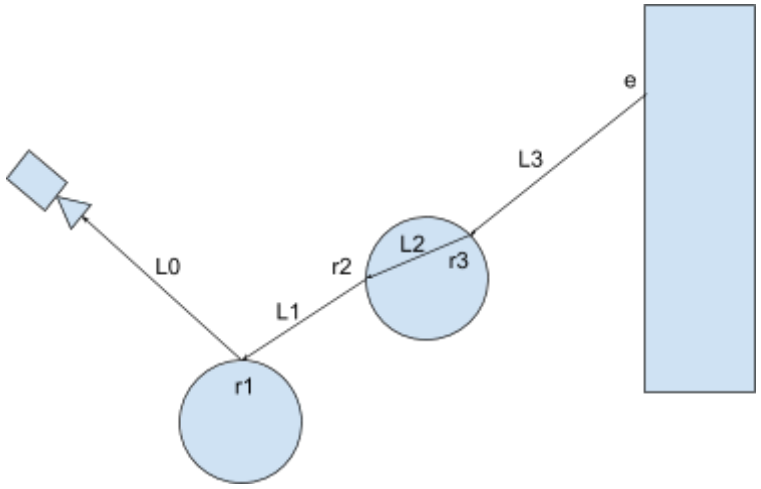
Ecuación de render

Se ha partido de la ecuación de Render original, la cual es una integral en toda la semiesfera de la luz que sale de un punto determinado *x*, las propiedades del material y el ángulo de incidencia de cada rayo ω_i. Esta integral corresponde a la luz que refleja cada rayo que le llega, en dirección del de salida.

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = \int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}$$

Tal y como ya se ha comentado en apartados anteriores, calcular esta integral completa requeriría trazar infinitos rayos, o en su versión discreta trazar n rayos de forma recursiva. Con Monte carlo y la ruleta rusa esto se convierte en el cálculo de un único rayo, lo que permite concatenar las operaciones en una única ecuación.

Por ejemplo, si tenemos este camino:



El camino sale de la cámara, rebota en una esfera, atraviesa otra, y finaliza en un emisor. En cada uno de estos rebotes se aplica la ecuación de render y por lo tanto:

$$L_0 = L_1 * fr(r1) * g(r1)$$
$$L_1 = L_2 * fr(r2) * g(r2)$$
$$L_2 = L_3 * fr(r3) * g(r3)$$
$$L_3 = e$$

Donde fr representa el factor del material en el rebote y 'g' el factor geométrico del rebote (el coseno del ángulo). Simplificando

$$L_0 = (((e) * fr(r3) * g(r3)) * fr(r2) * g(r2)) * fr(r1) * g(r1)$$

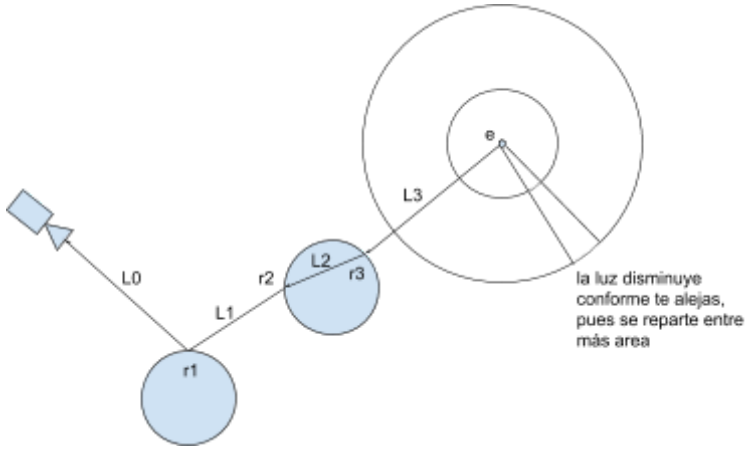
Es decir, el camino se ha convertido en una simple secuencia de multiplicaciones de factores materiales y geométricos. Gracias a montecarlo y a la ruleta rusa la simplificación

de la integral a un único punto es correcta, pues si se lanzan suficientes caminos estos convergerán al valor exacto.

Para el cálculo del componente geométrico cada uno de los eventos tiene unos coeficientes asociados:

- reflexión: se multiplica por el color de reflexión del objeto pues el rayo es único.
- refracción: idéntico a la reflexión (con el color de refracción)
- phong difuso: se multiplica por el color difuso y se divide por □, pues la fórmula para phong difuso es K_d/π
- phong especular: similar a phong difuso, pero con ecuación $k_s * (s + 2) / (2\pi) * |cos(\theta r)|^s$ siendo θr el ángulo de reflexión del ángulo de salida

Para el cálculo de la luz 'e' basta tomar el color emisor de dicho objeto, salvo que estemos calculando un rayo de sombra cuyo final es una luz puntual, en cuyo caso también hay que dividir por la distancia desde la luz al primer rebote al cuadrado, pues las luces puntuales emiten en todas direcciones y conforme te alejas la energía por unidad de superficie disminuye.



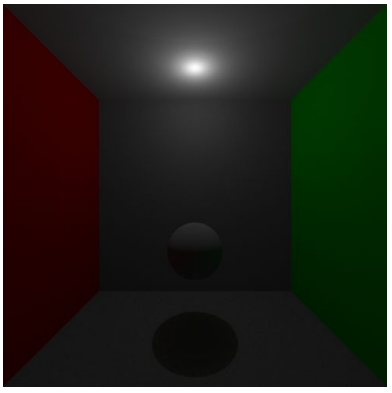
Convergencia del path tracing

- Respecto al número de paths_per_pixel (ppp): Como ya se ha comprobado en el apartado Cámara, la convergencia obtenida al aumentar el número de ppp aumenta, obteniendo mejores resultados, a costa de mayor tiempo de cálculo.
- Respecto al tipo de material: Al utilizar materiales con colores vivos cercanos al blanco el tiempo de renderizado aumenta, pues los caminos tienen menos probabilidad de morir debido a la ruleta rusa. Usando colores oscuros el tiempo disminuye ya que además cuando el color de un camino llega a cero, ya no se sigue (pues no aportaría nada).
- Respecto a tipos de luz: Tal y como se probará en el apartado de sombras suaves, al utilizar objetos emisores de pequeño tamaño la convergencia disminuye drásticamente, pues los caminos tienen poca probabilidad de encontrar la luz, y suelen morir antes. El caso particular son las luces puntuales, que estadísticamente

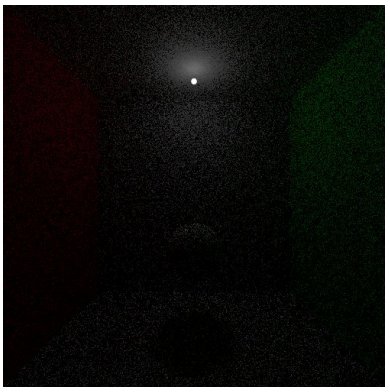
tienen probabilidad casi-nula de ser encontradas, y por eso se utilizan rayos de sombra.

Iluminación global

- Sombras definidas (hard shadows):
  Se obtienen principalmente gracias a las luces puntuales, pues al comprobar desde cada punto si la luz es visible o no se diferencia claramente los puntos visibles de los que no.
  Estas sombras aparecen en escenas con objetos que tapen a otros respecto a las luces puntuales.

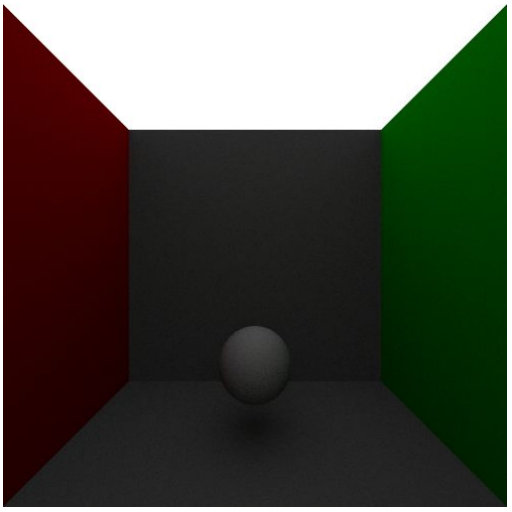


Sombra definida provocada por una luz puntual



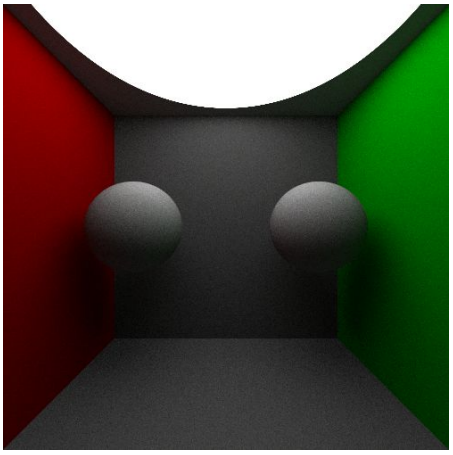
Simulación de luz puntual mediante una esfera emisora pequeña. Para esta imágen el número de rayos por pixel se ha aumentado a 1000, pero aun así la convergencia es casi mínima al no tener importance sampling. De todas formas se puede apreciar la sombra en la parte inferior.

- Sombras suave (soft shadows):
  Se obtienen principalmente gracias a los objetos emisores suficientemente grandes, que hacen que más o menos caminos choquen con él en función del área visible desde el punto, provocando una sombra gradual desde las zonas oscuras hasta las visibles.
  Estas sombras aparecen, al igual que antes, con objetos que tapen a otros respecto a emisores de área.



Sombra suave provocada por el techo emisor

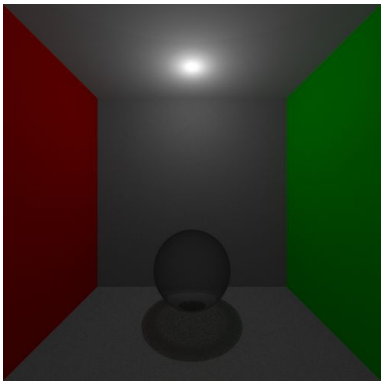
- Transferencia de color (color bleeding):
  Al calcular los caminos e ir multiplicando el factor de color por cada objeto involucrado, si en una zona de la escena (principalmente entre objetos cercanos) los caminos tienden a formar rebotes entre ellos, se provoca que la luz reflejada por uno de ellos se vea en el otro.



Efecto del color bleeding. Se puede apreciar ligeramente en la parte inferior de las esferas, y algo más notable en el techo cerca de las paredes (para esta escena la luz se proporciona por una esfera incrustada en el techo)

- Cáusticas:
  Las cáusticas se forman principalmente cuando la luz atraviesa un objeto refractante (o en ocasiones con objetos reflectantes) debido a los caminos particulares que se generan desde la fuente de luz. Mediante path tracing los caminos se generan desde la cámara hasta encontrar una luz, lo que hace que la probabilidad de encontrar estos caminos sea muy baja, disminuyendo mucho la convergencia. Los rayos de sombra pueden ayudar, pero se trata de rayos directos que no tienen en cuenta los objetos que se encuentran en el camino, por lo que sería necesario calcular los caminos concretos que la luz tiene que tratar (se podría para objetos simples como esferas, pero se complica para objetos más complejos). Una forma de solucionar este problema es mediante caminos que comienzan en las luces (y trazar ‘rayos de

cámara') o mediante un enfoque mixto que lance rayos desde las luces y la cámara. Otra solución similar es utilizar photon mapping, que se trata en el segundo proyecto.



Esfera refractante. Se observa algo de cáustica, pero no llega a converger.

Forma de trabajo

Para las primeras tareas la forma de trabajo que se utilizó fue la de separar el proyecto en partes y asignar unas a cada uno. Conforme el curso avanzaba y dada la distinta disponibilidad de cada uno de los miembros se dejaron indicadas tareas que cada uno iba realizando conforme podía, de forma que ambos miembros estuvieran implicados en el proceso (revisando lo que el otro había hecho) para evitar una separación fuerte y que cada uno aprendiera sólo una parte del proceso. Para el caso de las extensiones por ejemplo cada una sí que ha sido desarrollada por un miembro del equipo.
Para compartir el código y como medio de sincronización se ha utilizado git, en concreto un repositorio privado en Github con el que poder sincronizar de forma rápida lo que se iba realizando.

Conclusiones

A lo largo de esta asignatura hemos visto como todo lo aprendido durante la carrera tiene multitud de usos que no tienen que ver con la informática en sí, es decir, la informática es una herramienta que permite resolver problemas de cualquier ámbito. Para el caso de informática gráfica y en concreto en este trabajo, hemos desarrollado un sistema de simulación de luz físicamente correcto que permite, de manera relativamente eficiente, generar imágenes simulando la interacción de la luz con los objetos y distintas propiedades.

Hemos comprendido que lo importante en la simulación de la luz son las ecuaciones en sí (especialmente la ecuación de render), que poco tienen que ver con la informática en general, pero gracias a esta es posible probar y utilizar dichas ecuaciones. Para ello, es necesario utilizar técnicas que permitan aproximar ecuaciones complejas en otras más sencillas y poder realizar los cálculos en tiempos razonables.