Path tracing

Informática gráfica

4º curso, Grado de Ingeniería en Informática, 2021-2022

Enero de 2022

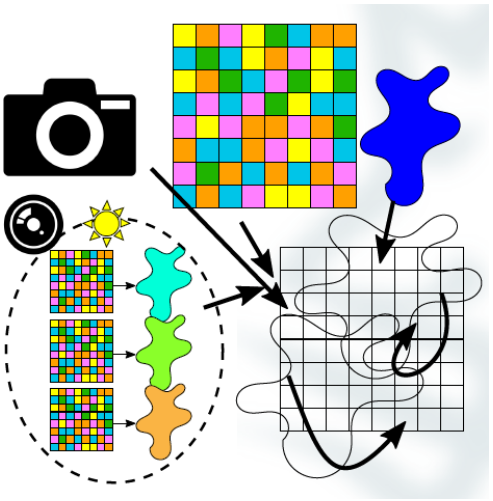
Diego Caballé Casanova, 738712

Patricia Briones Yus, 735576

Índice de contenidos

1. Introducción

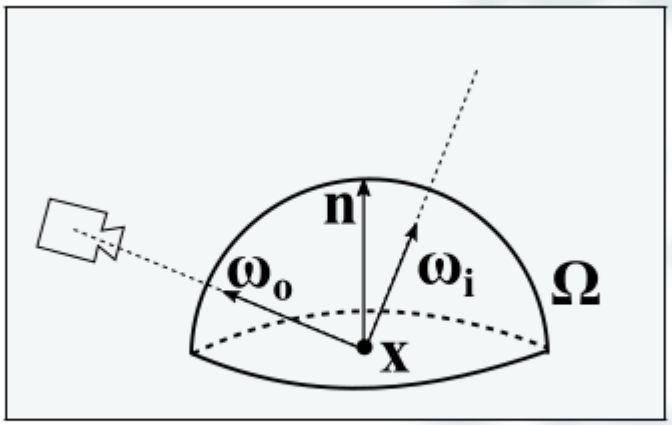
En este documento se explica el algoritmo implementado de generación de rayos (path tracer) desarrollado en el lenguaje C++ para la asignatura de Informática gráfica. Este genera imágenes en formato “.ppm” mediante los colores RGB obtenidos por cada uno de los rayos lanzados a una escena previamente creada. El único requisito del usuario para ejecutar el algoritmo es introducir el número de rayos que se desea lanzar por cada píxel a la imagen, de manera que esta contenga un mayor o menor ruido.



Los rayos se originan desde un punto, el cual será la ubicación de la cámara, y son dirigidos a un plano de un ancho y alto (establecidos en el código por defecto), lo que condiciona el tamaño de la imagen final obtenida. En cuanto a la escena, está compuesta por distintas figuras geométricas, capaz de distinguirlas correctamente y apreciar sus distintos materiales, así como la incidencia de áreas de luz o luces puntuales en distintas ubicaciones.

2. Diseños de implementación

La base de nuestra implementación gira entorno a la ecuación de render:



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

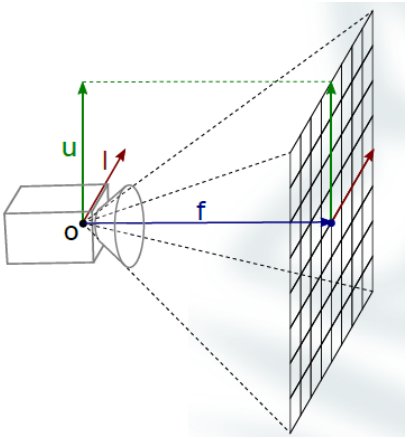
Esta ecuación la podemos separar en varias partes.
La primera es que el color visto desde la cámara es el color de la luz de un emisor más el color del objeto sobre el que rebota. Después, entramos en el color del objeto. Para conocer el color de un objeto, necesitamos calcular lo que nosotros llamamos rebote. Este rebote consta de 3 partes, la luz que tendrá el rayo de salida, por el cálculo de la BRDF, por el coseno del ángulo que forma la normal y el nuevo rayo, aquí calculado como el producto escalar.
Estas 3 partes para calcular el color del rebote también están reflejadas en nuestro código de una forma explícita, aunque la mayor diferencia es la integral de la semiesfera omega.
Computacionalmente es muy costoso calcular la integral de todos los rayos de la semiesfera, así que cambiamos la integral por un productorio, lo que nos lleva a diferir de la ecuación de render. En vez de calcular la integral de todos los rayos que salen de un punto, calculamos el aporte de un rayo en un punto.
Esta diferencia, nos obliga a hacer varios cambios, como un método para elegir un ángulo de salida aleatorio, en nuestro caso ruleta rusa + muestreo por coseno, y “olvidarnos” del aporte de luz siguiente, como hacemos un productorio, podemos asumir que el siguiente rebote nos dará esa información. Así, terminamos utilizando esta otra ecuación:

$$L_o(\mathbf{x_1}, \omega_\mathbf{o1}) \approx L_e(\mathbf{x_n}, \omega_\mathbf{on}) \prod_{j=1}^{n} \frac{f_r(\mathbf{x_j}, \omega_\mathbf{ij}, \omega_\mathbf{oj}) |\mathbf{n_j} \cdot \omega_\mathbf{ij}|}{p(\omega_\mathbf{ij})}$$

Podemos ver que en esencia son muy similares, pero la mayor diferencia es la idea de rayo y camino.

2.1 Cámara

La cámara se localiza en un punto del espacio formado por 3 coordenadas (x,y,z) y se establece hacia qué sentido se lanzarán los rayos a la escena para obtener los colores de las intersecciones con las figuras. Para esta implementación se ha elegido el eje “z” en sentido positivo. Los valores escogidos para las coordenadas de la cámara han sido (0,0,0). Y por tanto, el sistema de coordenadas de la cámara será para el eje X (1,0,0), Y (0,1,0) y Z (0,0,1). La configuración de los ejes de esta manera hará que posteriormente nos facilite a la hora de lanzar los rayos.



2.2 Figuras geométricas

Hemos implementado las 2 figuras geométricas obligatorias, una esfera y un plano. El plano consta de un punto y una normal que apunta hacia donde mira. La esfera consta de un radio y el centro de la esfera. Ambos objetos heredan de una clase padre Figuras geométricas, que define las operaciones necesarias para ambos hijos (si un rayo intersecta o no, coger valores de estos) y que contiene información para ambos hijos, en nuestro caso, la tupla kd de tipo RGB, que define el color del objeto, el tipo de material del objeto definido por los valores kd, ks y kt que son valores de 0 a 1 y si el material es un foco o no, para poder tener luces indirectas.
Las distintas ecuaciones de intersección son, para el plano y la esfera, respectivamente:

$$t = -\frac{c + \mathbf{o} \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}} \qquad t^2|\mathbf{d}|^2 + 2t\mathbf{d} \cdot (\mathbf{o} - \mathbf{c}) + |\mathbf{o} - \mathbf{c}|^2 - r^2 = 0$$

Donde según el valor de “t” se podrá saber si el rayo no intersecta, intersecta o atraviesa dicha figura. En el caso de la esfera, si a la hora de hacer la ecuación de segundo grado, el discriminante (lo de dentro de la raíz cuadrada) tiene un valor de 0, se diría que el rayo es tangente a la esfera, si su valor es > 0 la atraviesa y si es negativo no la intersecta. Para los planos sería aplicar las mismas condiciones.

2.3 Trazado de rayos

Un rayo, como se ha comentado anteriormente, está compuesto por un punto origen y una dirección. Como la creación de todos los rayos son desde el ojo de la cámara, el punto origen de todos ellos inicialmente serán por tanto las coordenadas de esta. A la hora de crear la dirección del rayo, se calcula primero la coordenada del píxel del plano al que se quiere lanzar. Para ello, al estar los planos compuestos solo por las 2 coordenadas (x,y), se puede fijar el punto al que se quiere llegar (píxel) como (x,y,ancho), siendo “x” e “y” dos valores calculados (aleatoriamente) dentro de las dimensiones de ese píxel y “z” el valor del ancho del plano. Las dimensiones de los píxeles variarán según el número de rayos totales que se quieran realizar (dependiendo del tamaño de la imagen).

Al estar la coordenada del píxel en un sistema local, habrá que pasarlo al global haciendo un cambio de base usando el sistema de coordenadas de la cámara previamente

comentado. Por lo tanto, una vez hecho esto, se puede calcular la dirección final del rayo mediante la resta del punto del píxel (global) y el origen (cámara).

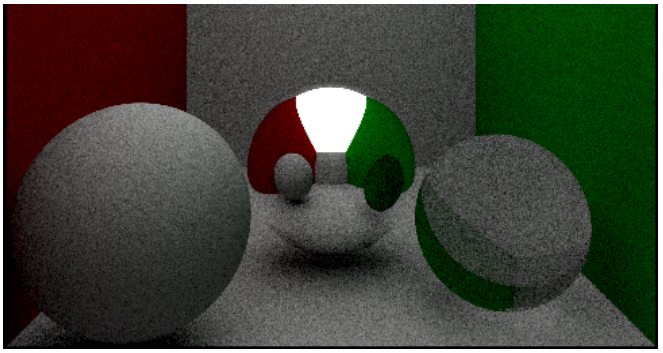
Una vez se haya lanzado el rayo, caben dos posibilidades: intersecte con una figura geométrica o no. En caso de no hacerlo, el throughput de ese camino valdrá 0 y su trayectoria habrá terminado. En cambio, si sí que ha chocado contra un objeto, se evaluará, mediante la ruleta rusa, la siguiente estimación de evento. La estimación de evento podría matar el rayo porque “no hay evento” o porque el porcentaje de absorción de ese rayo es 1 (en cada rebote se incrementa en 0.5) y devolver un throughput 0.

Si el evento obtenido no mata el rayo, se obtendrá una nueva dirección según el tipo de material y se creará un nuevo rayo rebote a raíz de esta y con origen en el punto de intersección del rayo inicial con la figura. Se irán lanzando rebotes de esta manera hasta que muera o intersecte con un área de luz. Este proceso se realiza de manera iterativa, lo que hace que la renderización de la imagen sea más rápida.

2.4 Materiales

Hemos implementado los 3 materiales obligatorios, difuso, espejo y cristal, en el caso del difuso, cuando un rayo intersecta, sale rebotado en función del muestreo coseno, una operación que escoge un ángulo aleatorio de la semiesfera de la ecuación de render. Además, el rayo almacena en su throughput el aporte de color del material difuso.
En el caso del espejo y del cristal tenemos grandes diferencias con el difuso, en el caso del espejo, es parte difuso y parte reflectante y en el caso del cristal es parte reflectante y parte refractante. Para implementar estas interacciones realizamos la ruleta rusa, donde llamaremos evento a elegir una de las partes del material.
En la ruleta rusa, elegimos de forma aleatoria cómo se va a comportar el material en un rebote, por lo que algunos rayos que intersectan con el cristal serán reflejados y otros refractados. Esta elección aleatoria afectará al aporte del rebote, el cual será dividido por la probabilidad de elección del evento ocurrido. Esta probabilidad viene dada por unos valores arbitrarios y constantes que le hemos dado a cada material en sus constantes kd, ks y kt. De esta forma, podemos tener un reflejo leve sobre una esfera de cristal y una refracción fuerte para poder diferenciar los distintos aportes.
Además del aporte de color, la refracción y la reflexión son diferentes, puesto que el rayo rebotado sale de formas diferentes. En el caso de la reflexión sigue las leyes de la reflexión, y en el caso de la refracción las leyes de Snell.

A continuación se muestra una imagen de los 3 materiales implementados:



2.5 Luces directas e indirectas

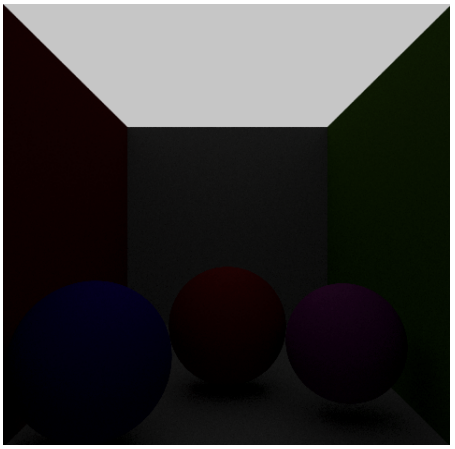
En las escenas creadas pueden existir 2 tipos de luces, directas e indirectas. Las directas son capaces de iluminar una escena entera, ya que son de mayores dimensiones, mientras que las indirectas son puntos de luz situados por la escena de manera que iluminan, dependiendo de su posición, sólo un pequeño área.
Para que las luces indirectas funcionen no hace falta hacer nada especial, cuando un camino intersecte con una luz indirecta o se muera, el camino terminará y aportará la luz del emisor.
Para las luces directas necesitamos implementar next event estimation, puesto que un rayo intersectando con una luz puntual es imposible. Así, lo que necesitamos es trazar rayos de sombra desde el rebote hacia todas las luces puntuales para conocer el aporte de cada luz puntual sobre ese punto. La suma de los aportes de luz de las luces directas se llama radiancia. Para calcular la radiancia, utilizamos la ecuación de render, pero sólo las 3 partes del aporte de color, la luz del emisor * la BRDF del objeto intersectado * n*wi. Para que la luz puntual se atenúe en función de la distancia a la que esté el emisor del punto, hemos dividido su aporte por la distancia al cuadrado. También, a la hora de usar las luces puntuales, tenemos que tener en cuenta que el aporte de su luz es tan importante como el de las luces indirectas, por lo que, al tener un algoritmo iterativo, tuvimos que crear una terminación nueva, donde sólo utilizamos el valor de la radiancia para dar color al pixel.

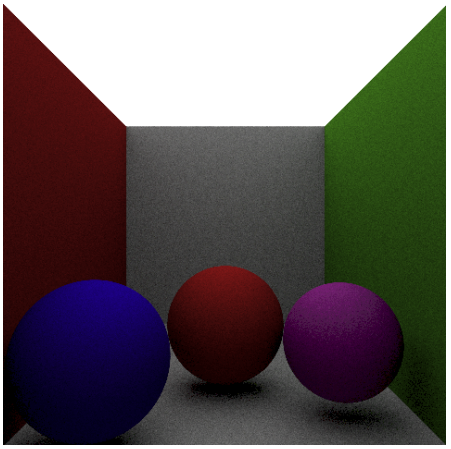
2.6 Fichero .PPM

A la hora de crear el fichero resultante “.ppm” de este algoritmo, que contiene los datos de la imagen, el color de cada píxel se irá escribiendo a medida que el camino de los rayos lanzados haya terminado, por lo que estos valores no se almacenan en ningún tipo de lista o matriz.

Para obtener una menor cantidad de ruido en la imagen, se utiliza un número introducido por el usuario de rayos lanzados por píxel, lo que hará que se obtenga un mejor color ya que al tener más muestras se obtiene una mejor media entre todos ellos.
Pero de nada sirve lanzar muchos rayos por píxel si todos los rayos que se lanzan son a la misma coordenada dentro del tamaño que abarca el píxel, esto hace que se produzca un trazado escalonado de los contornos de los objetos.
Esto lo evitamos utilizando la técnica de antialiasing, y es que a la hora de lanzar cada uno de los rayos, calculamos 6 números aleatorios para el valor de la “x” y otros 6 para el valor de la “y” y se hace la media de cada uno para establecer el punto final por el que se va a lanzar el rayo, lo que hace que los rayos por píxel lanzados sean más variados y puedan aportar un color más exacto.

Por último, una vez se ha creado el fichero “.ppm”, se utiliza la técnica de “tone mapping” para enriquecer un poco los tonos colores obtenidos, implementada en la práctica 2. A continuación se muestran 2 imágenes, sin “tone mapping” y con “tone mapping”:

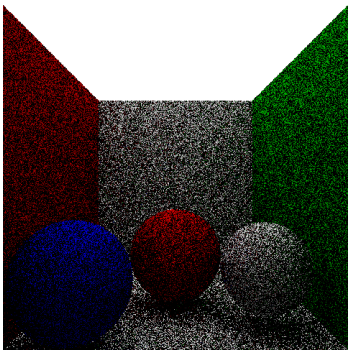




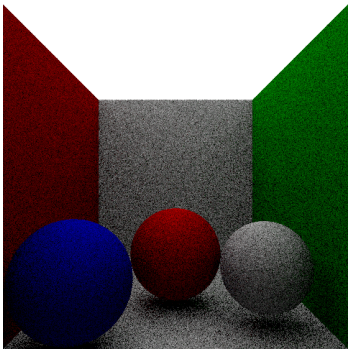
3. Evaluación de resultados

How fast does path tracing converge with respect to the number of paths per pixel? Illustrate with renders. Bonus point if a numerical analysis is included.

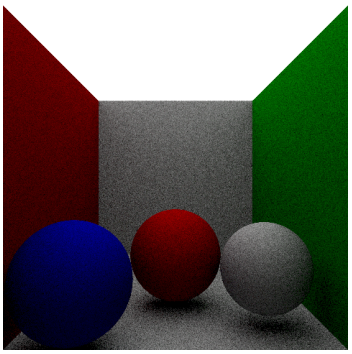
Como se ha comentado previamente, la calidad de la imagen en cuanto a ruido se habla, depende del número de rayos que se lancen por píxel (rpp).



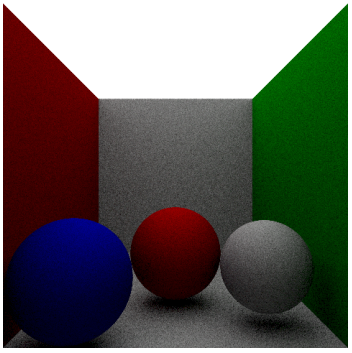
1 rpp - 3.686 seg



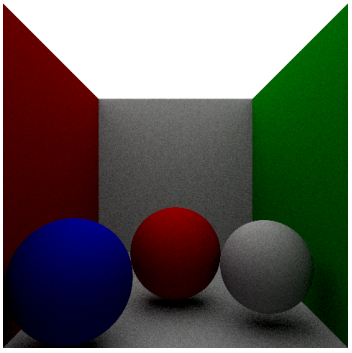
10 rpp - 38.757seg



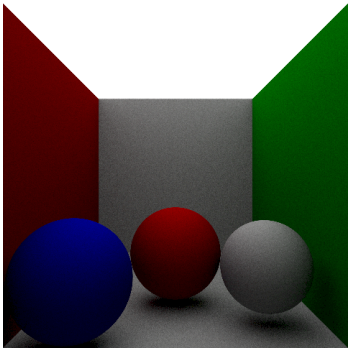
30 rpp - 120.039 seg



50 rpp - 210.624 seg

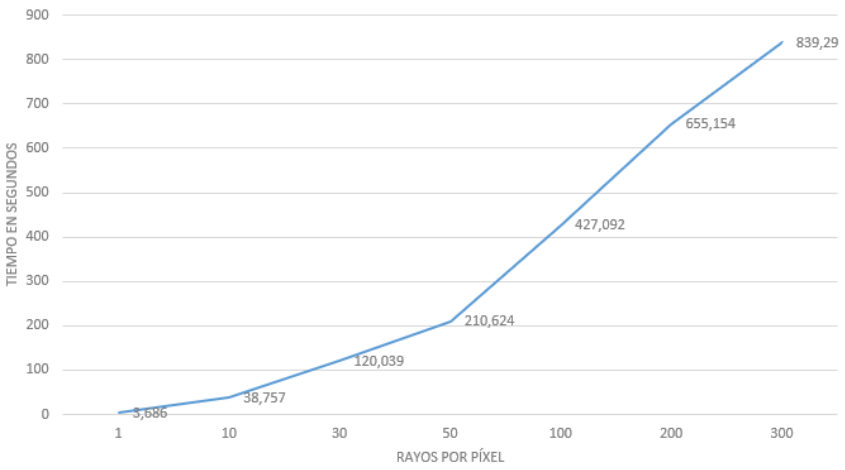


100 rpp - 427.092 seg



200 rpp - 655.154 seg

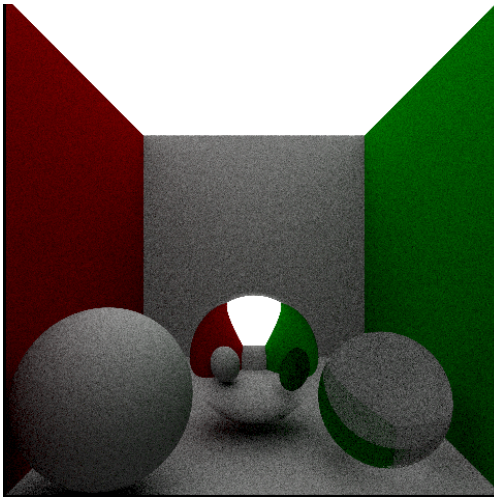
Según los tiempos obtenidos en renderizar cada una de las imágenes, se puede comprobar que el tiempo crece de forma lineal según el número de rayos establecidos.



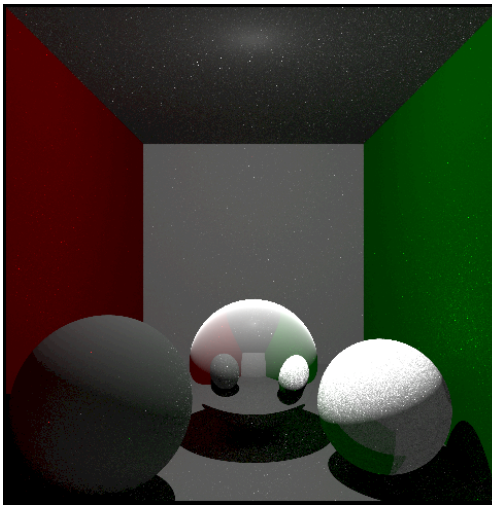
Of the implemented materials, which ones make convergence slower? In which circumstances? Why?

En cuanto al tiempo de generación de imagen según el tipo de material, tras un estudio de varias pruebas comparativas, los objetos dieléctricos tienden a tardar más. Cuando más se nota la diferencia de tiempo es según el número de rayos por píxel que se lanzan. Esto se debe a que el camino que tiene que recorrer el rayo que atraviesa los materiales dieléctricos es más largo y por tanto en cada rayo lanzado se acumula ese tiempo.

Which light sources make path tracing converge slower? Area lights or point lights? In which circumstances? Why? Illustrate it with renders.



Area light - 91 segundos

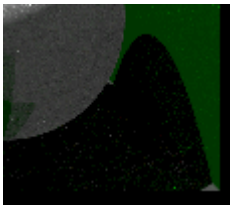


Point light - 254 segundos

Cuando utilizamos luces puntuales necesitamos dibujar rayos de sombra, de lo contrario nunca encontraríamos la fuente de luz. Además, la única terminación con luces directas es la absorción de la Ruleta Rusa, aumentando el nº de rebotes posibles. Por ejemplo, en el caso de la luz de área, un rayo que rebota 3 veces, llega a la luz de área y termina. En el caso de la luz puntual ese rayo rebotaría hasta que la ruleta rusa diera no evento y, además, de cada rebote salen rayos de sombra a todas las luces puntuales.

Which of these global illumination effects (hard shadows, soft shadows, color bleeding, caustics) are easily obtained with path tracing? Why? Illustrate them with renders and insets (zoom in) to specific areas.
Which scenes do you need to obtain each of these effects? Discuss scene requirements for each of the effects: material properties, light source properties, geometrical distribution of objects…

De las fotos del apartado anterior podemos ver varios efectos:


Hard shadows


Soft shadows


Color bleeding


Cáustica

Tanto hard shadows como soft shadows dependen del tipo de iluminación elegida para verse claramente, el primero necesita luces directas y el segundo luces indirectas.
La razón por la que ocurren depende de los tipos de luz y su lógica asociada. En el caso de las luces directas, si el rayo de sombra intersecta con un objeto no aporta la radiancia de la luz puntual, dando unas sombras negras y muy duras. Por el otro lado, la sombra depende de la probabilidad del rebote, si llega a la luz de área, si se absorbe… Tienen un tono más claro y menos marcado por esa aleatoriedad asociada a la terminación correcta o absorbida del rayo.

Color bleeding ocurre independientemente de la iluminación utilizada. El hecho de controlar el throughput de los caminos hace que la participación de un color sobre otro sea directa, no hay que hacer nada especial, los propios rebotes generan estos efectos.
Para que el color bleeding sea muy visible, es recomendable usar objetos blancos cerca de objetos de colores. Así, el objeto de color le refleja su color al objeto blanco.

Cáusticas son un poco menos visibles que los demás efectos, pero podemos ver cómo se concentra más luz alrededor de la parte baja de la esfera de cristal, esto es porque, desde ese ángulo, los rayos se concentran a través de la esfera y llegan a la fuente de luz, mientras que el resto de ángulos no, dejando un círculo iluminado y un anillo más oscuro.

Para conseguir este efecto es necesario tener un material dieléctrico, cerca del suelo e iluminado con luces de área.

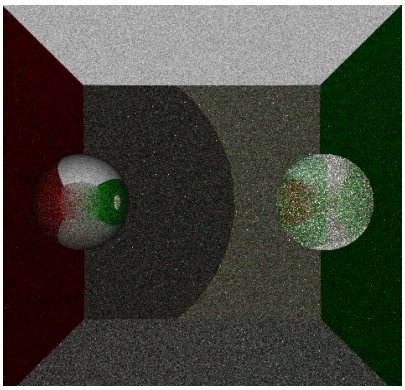
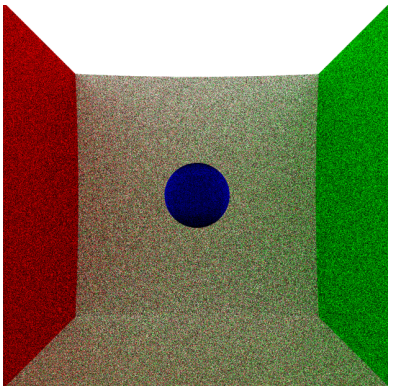
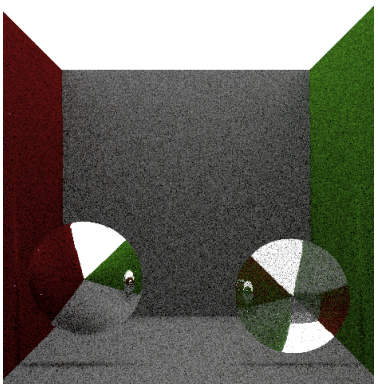
Which of these global illumination effects cannot be obtained (or are very hard to obtain regarding convergence time)? Why?

En la práctica, es imposible encontrar cáusticas con luces puntuales en el algoritmo básico de path tracer. La implementación de Next Event Estimation no permite el cálculo de cáusticas. Como los rayos de sombra van desde el punto de rebote, hasta la luz puntual, sin tener en cuenta la refracción del cristal, no pueden calcular este efecto.

4. Problemas encontrados

Aunque sea un trabajo de una gran extensión el tener que crear un algoritmo “path tracing” desde 0 (y se hayan encontrado un mar de problemas interminables), se podría decir que los problemas que se tuvieron que afrontar en mayor medida fueron:

- La cámara y creación de la imagen. Debido a cómo se superponían los planos y cómo eran las coordenadas de la cámara, se obtenían resultados sin sentido como los siguientes, en los que la sombra solo se ve reflejada en la pared de fondo o las intersecciones entre los planos no llegan a ser del todo rectas.

  O estar tan lejos la cámara y la escena, que las sombras vistas desde la perspectiva de la lente se mostraban visualmente “planas”.

- También hubo problemas a la hora de calcular la radiancia con la estructura que teníamos hecha inicial del código.
- Poder representar visualmente bien los materiales especular y dieléctrico.
- Normalización de vectores.

5. Conclusión

Se han comprendido todas las matemáticas y procesos que conlleva detrás el algoritmo de “path tracing”. Ha sido interesante saber cómo se crean los diferentes tipos de materiales de los objetos, así como la representación de sus formas según el trazado de un rayo. También la diferencia entre iluminación global, iluminación indirecta, luces puntuales y la creación de sus respectivas sombras.

La ecuación de render parece muy complicada al inicio, pero una vez se estudia con detenimiento, y algunos cambios para llevarla a la práctica, nos proporciona imágenes muy reales en poco tiempo.

6. Referencias bibliográficas

- Contenido de la asignatura proporcionado por los profesores en Moodle.
- https://www.scratchapixel.com/
- https://stackoverflow.com/