PATH TRACER INFORMÁTICA GRÁFICA

David Solanas (738630)

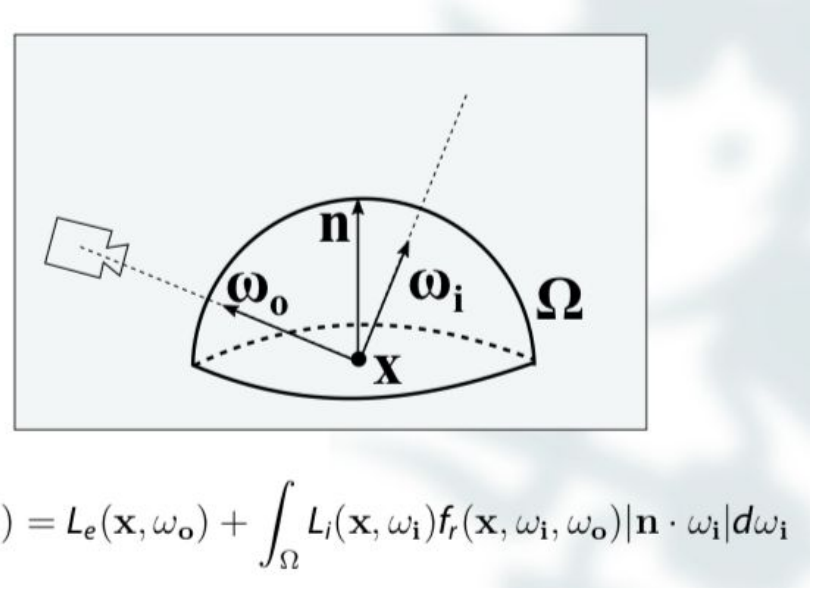
Introducción

En este trabajo se pide realizar la implementación del algoritmo de *Path Tracing* estudiado en la asignatura, para ello se ha desarrollado en C++ el conjunto de funcionalidades necesarias de forma progresiva con las distintas tareas intermedias, consiguiendo al final un Path Tracer que genera imágenes en formato HDR, además se ha implementado un Tone Mapper para convertir esas imágenes en HDR a formato LDR [7].

Implementación

El algoritmo clásico de *Path Tracing* consiste en reducir el número de rayos secundarios trazados en *Ray Tracing* a un único camino que va rebotando por la escena y por cada intersección se calcula la iluminación en ese punto, tanto indirecta como directa.

Para el cálculo de la iluminación en un punto se hace uso de la ecuación de render [1]:



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

Donde *Lo* es la luz que sale del punto *x* en la dirección *wo*, *Le* es la luz que emite el objeto intersectado en el punto x (en esta implementación este término se ignora ya que se considera que un objeto que refleja luz no emite y viceversa). El término *Li* es la luz que incide en el punto *x* con dirección *wi*; *fr* que es la función de distribución de reflectancia bidireccional que modela las propiedades del objeto, determinando así su color, esta función depende del material del objeto y de cómo incide y sale reflejada la luz. Por último se tiene el término del coseno |*n* · *wi*| que modela el ángulo incidente de la luz en el punto *x*, puesto que no es lo mismo que

llegue luz a un objeto de forma que *wi* = *n* (llega el máximo de luz) que con un ángulo de 85º respecto de la normal. Esta ecuación se integra en toda la hemiesfera con centro *x* para obtener así la luz que llega a dicho punto por todas las direcciones posibles. Como calcular esta ecuación computacionalmente es imposible (tiempo infinito debido a infinitas direcciones *wi*), se realiza una aproximación probabilista basada en la técnica de Monte Carlo de forma que tenemos la siguiente ecuación: $\frac{1}{N}\sum_{i=1}^{N} Lo(x,\ wo)$, donde N es el número de caminos (paths) por píxel que se lanzan. Esta función se evalúa para cada píxel de la imagen obteniendo finalmente una matriz de tuplas RGB que corresponde con los colores de la imagen. Todo este proceso se realiza con la función *render_image* que recibe como parámetro (entre otros) el número de caminos por píxel. Cabe destacar que a mayor *N* especificado, más se aproxima la solución a la correcta a costa de un mayor tiempo de ejecución, con *N* suficientemente grande se pueden obtener buenos resultados, como los obtenidos en la figura 1.

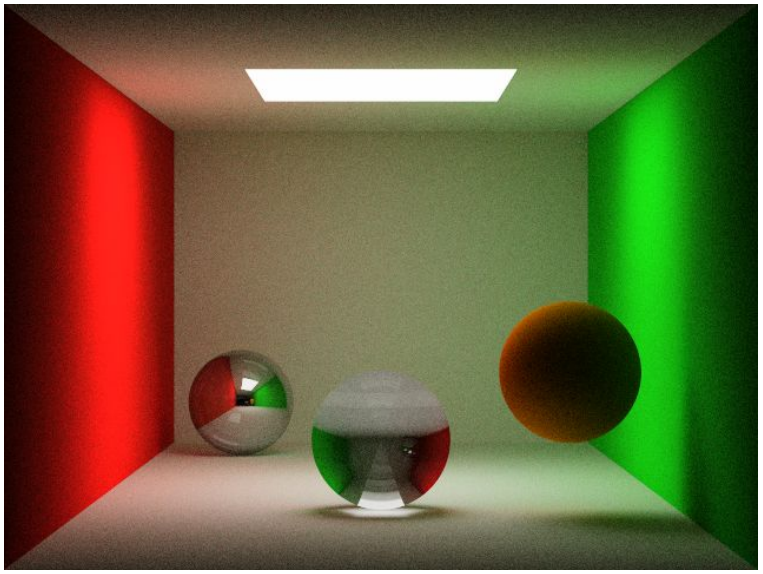


Figura 1: *N = 5K, tiempo de render -> Aproximadamente 1 día.*

El algoritmo implementado sigue el siguiente comportamiento:

1. Para cada píxel de la imagen se genera un rayo (camino), que tiene como origen el origen de la cámara y dirección una dirección aleatoria dentro del rango del píxel (para obtener mejores resultados con la aproximación de Monte Carlo). Se ha hecho de tal forma que los píxeles tengan lado 1 por lo que para calcular dicha dirección inicial se parte de la esquina superior del píxel (<0,0> por ejemplo, por lo que el límite del píxel será <1,1>) y se eligen dos números aleatorios en el rango [0,1] de forma que el punto en el píxel será el punto resultante de sumar esos valores aleatorios a la esquina

superior (para el <0,0> y los valores 0.3 en el eje x y 0.7 en el eje y, el punto resultante será <0.3, 0.7>). Finalmente la dirección (normalizada) del camino será (punto calculado - origen cámara).

2. Con el primer rayo generado, se llama a la función *trace_path* que, recursivamente, va calculando todo el camino del rayo por la escena.
3. El primer paso en *trace_path* es calcular la primera intersección de ese camino con la escena (en el caso de que haya fuentes de luz de área también se comprueba si intersecta alguna de ellas), para ello se comprueba si dicho camino intersecta con cada objeto y de aquellos intersectado elige el que esté más cerca. En este punto de la función se pueden dar tres casos:
    a. El camino no intersecta con nada, por lo que ya no se comprobará nada más y devolverá como resultado el color de fondo de la escena (por defecto negro).
    b. El camino intersecta con una fuente de luz de área, en cuyo caso se devuelve directamente la intensidad de la fuente de luz.
    c. El camino intersecta con un objeto que refleja luz, en ese caso se procede a realizar los pasos a partir del 4.

    Cabe destacar que si en la escena no hubiera fuentes de luz de área pero sí hubiera fuentes de luz puntuales, nunca se daría el caso b en el paso 3.
4. En este punto se sabe que ha habido una intersección con un objeto de la escena el cual no es una fuente de luz. Se obtiene el punto de intersección y la normal a la superficie del objeto en ese punto.
5. Se calcula el nuevo camino con la función *get_outgoing_sample_ray*, para ello se ha implementado el algoritmo de ruleta rusa que determinará la dirección del nuevo camino con origen el punto de intersección, o si dicho camino debe dejar de ser muestreado. El algoritmo de ruleta rusa generalizado está explicado en las transparencias de la asignatura [2] (a partir de la diapositiva 53). Se presentan varios casos debido a las distintas BRDF/BTDF implementadas:
    a. En el caso de la BRDF Lambertiana y de Phong, la dirección del nuevo camino se obtiene mediante el muestreo por coseno [1][2], para ello se hace un cambio de base a coordenadas locales (donde la normal es <0, 1, 0>) y se muestrean dos ángulos θ y ϕ aleatoriamente, se eligen dos números aleatorios comprendidos entre 0 y 1 $\xi\theta, \xi\phi$ de forma que $\theta = arcos\sqrt{1-\xi\theta}\, y \;\; \phi = 2\pi\xi\phi$, con estos ángulos se calcula el vector dirección y se realiza un cambio de coordenadas globales para obtener la nueva dirección correcta. Una vez obtenida la nueva dirección, se comprueba el evento a considerar según la ruleta rusa, en el caso de los objetos lambertianos, si el valor supera el máximo *kd* de los 3 canales RGB del objeto, dicho camino se “mata”, en caso contrario se devuelve el valor calculado de la BRDF. Para la BRDF de Phong se sigue el mismo procedimiento que para la Lambertiana, sin

embargo aquí además del evento difuso hay un evento especular por lo que habrá que tenerlo en cuenta a la hora de determinar el evento a procesar para el nuevo camino. La dirección del nuevo camino se muestrea igual que para los lambertianos (muestreo por coseno), en este punto, si la ruleta rusa decide que el evento es difuso devolverá como resultado el cálculo de la BRDF para la parte difusa, si decide un evento especular para la parte especular y si no se decide ninguno de los dos eventos anteriores se “mata” el camino.

b. En el caso de las BTDF (reflexión y refracción), las direcciones se calculan con la ley de reflexión [1] y la ley de Snell [3][4]. Para estos casos no hace falta realizar el cambio de base a coordenadas locales, pero para la refracción hay que tener en cuenta si el camino se encuentra en el interior del objeto que refracta o fuera, dicha comprobación se realiza con el producto escalar de la dirección y la normal en el punto de intersección (si es positivo $\Rightarrow$ dentro del objeto, en caso contrario está fuera). La ruleta rusa decidirá qué evento se producirá para cada BTDF, para ello se elegirá el máximo de los 3 canales RGB de sus respectivos coeficientes de reflexión/refracción, si el de reflexión/refracción o el de “matar” el camino, en el caso de que el camino sea reflejado/refractado se devolverá el cálculo de la BTDF.
c. Materiales dieléctricos (reflejan y refractan luz de forma especular al mismo tiempo). Para obtener los correctos coeficientes de reflexión y refracción para determinar el evento a procesar por la ruleta rusa, se aplican las ecuaciones de Fresnel [4][5]. Con dichos coeficientes calculados se procede a determinar el evento a procesar, si se elige reflexión se calcula el nuevo camino con la ley de reflexión y se devuelve el resultado de evaluar su BTDF, si se elige refracción se aplica la ley de Snell y se devuelve el resultado de su BTDF, si por el contrario no se elige ningún evento de los anteriores se “mata” el camino.
d. Se ha implementado una clase *Material* que es una combinación de todas las BRDF/BTDF mencionadas anteriormente de modo que los coeficientes para determinar el evento por la ruleta rusa serán el máximo de los 3 canales RGB de cada k_x especificada por el usuario, dependiendo del evento elegido se calculará la nueva dirección y se devolverá el valor de su respectiva BRDF/BTDF tal y como se ha explicado en los puntos a, b y c.

Cabe mencionar que al devolver el valor de la BRDF/BTDF del objeto se considera también el término del coseno de la ecuación de render y se realizan directamente las simplificaciones “a mano” para reducir posibles errores por redondeo y tiempo de cálculo, así como por simplicidad del código desarrollado. Cuando la ruleta rusa decide “matar” un camino

devuelve la tupla <-1, -1, -1> para indicar que no hay que seguir procesando ese camino.

6. Una vez calculado el nuevo camino y evaluado la BRDF y el término del coseno en ese punto, se comprueba si dicho camino no ha sido “matado”, en caso de que la ruleta rusa indique que no hay que procesar ese camino se termina el cálculo de dicho camino y devuelve como resultado el color de fondo de la escena.
7. Si el camino no ha sido matado, se calcula la contribución de la luz directa en ese punto pero únicamente de las luces puntuales de la escena ya que estas no pueden nunca intersectar con un camino, en el caso de las luces de área el resultado de evaluar la función *get_incoming_light* será 0.
    a. La función *get_incoming_light* iterará por todas las fuentes de luz de la escena para calcular la contribución de la misma en ese punto.
    b. Si la luz es de área no se realiza el cálculo.
    c. Si la luz es puntual se obtiene cuánta luz llega a dicho punto, calculando por un lado *Li* y por otro el término del coseno, el resultado a devolver será *Li * término coseno* [1].
8. Con la luz directa calculada, se calcula ahora la luz indirecta que llega a ese punto, para ello se llama recursivamente a la función *trace_path* especificando el nuevo camino obtenido en el paso 5.
9. Finalmente (tras la convergencia de ese camino ya bien porque no ha rebotado con nada, ha sido “matado” o a intersectado con una fuente de luz), se suman la luz directa e indirecta (debido a que la luz es aditiva) y se devuelve el resultado de multiplicar la luz incidente por la BRDF y el término del coseno, así como de dividir por la función de densidad de probabilidad (estos tres últimos términos son calculados en el paso 5).
10. Este proceso (pasos del 1 al 9) se realiza tantas veces como especifique el usuario (parámetro de caminos por píxel), finalmente tras haber acumulado todos los resultados (se suman los valores devueltos por la función *trace_path*), el valor a almacenar en el píxel [i, j] será la división de dicho resultado por el número de caminos por píxel.

Se han implementado las texturas en los objetos, para ello se ha usado la técnica de UV mapping de forma que cada tipo de objeto tiene su propia forma de calcular las coordenadas *u, v*. Para poder implementar esto, se ha añadido una matriz a la clase objeto (si el objeto no tiene textura la matriz no será inicializada) que guarda los valores RGB correspondientes (en el rango 0-1). Con las coordenadas *u* y *v* calculadas se hace un mapeo al espacio de píxeles de la textura para obtener el color correspondiente en la textura (representado como una tupla RGB). Para integrar la textura en la imagen lo que se hace es en el paso 9, antes de devolver el resultado se comprueba si el objeto tiene textura o no, si no la tiene no hay ninguna modificación en el paso 9; pero si el objeto tiene textura, se obtiene el

color correspondiente de la misma y al resultado a devolver en el paso 9 se le multiplica esa tupla RGB (el valor de la textura).

En la figura 2 se muestra un render con texturas añadidas, se han lanzado 4500 caminos por píxel con un tiempo de render de 22.5 horas aproximadamente. En la escena se encuentran varias figuras geométricas, para cada figura geométrica se ha implementado su correspondiente mapeado UV para poder añadir texturas. La esfera azul es una combinación de Phong + Reflexión, el cilindro es de cristal (cabe destacar que necesitaría más caminos por píxeles para converger a una mejor solución), los demás objetos (exceptuando el espejo) son Lambertianos y tengan o no textura.

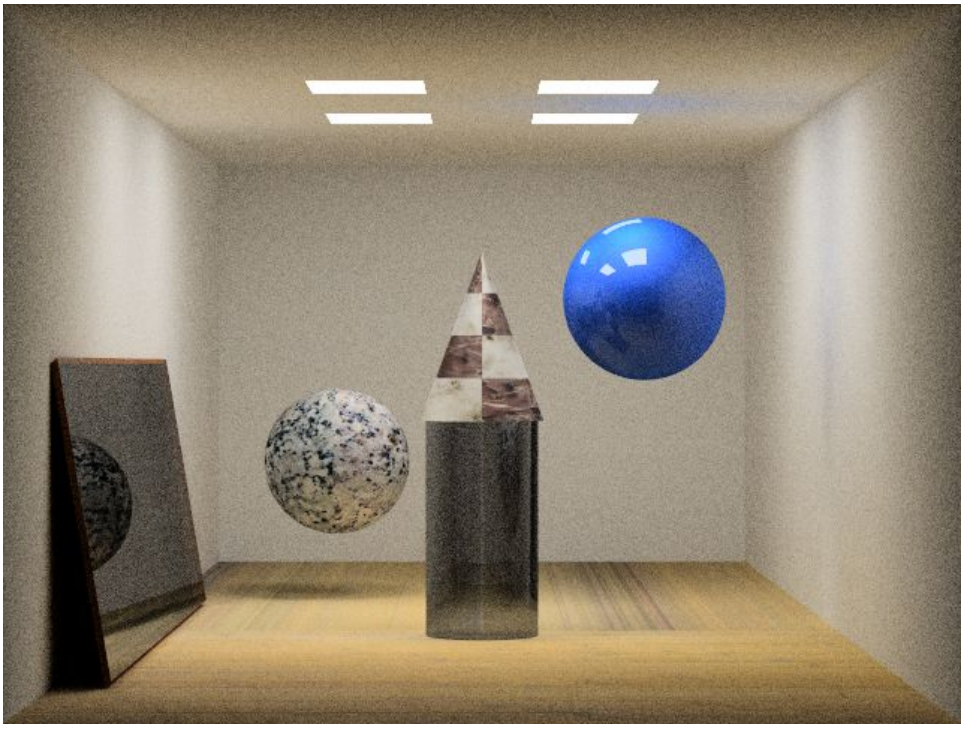


Figura 2: *N = 4.5K, tiempo de render -> Aproximadamente 22.5 horas.*

En la figura 2 también se aprecia el efecto de color bleeding en la pared derecha y en el techo principalmente, aunque también un poco en el suelo. Esto es debido a la iluminación indirecta que llega en esos puntos (techos/paredes), por ejemplo partiendo de la luz de la derecha que está más cerca habrá un camino que rebote en la esfera azul y vaya directamente al techo de modo que en ese punto recibirá como luz indirecta el azul de dicha esfera, provocando color bleeding.

Convergencia de la solución implementada

Para mejorar el tiempo de convergencia se ha paralelizado la ejecución del código, esto es posible debido a que cada camino generado en un píxel es independiente de los demás caminos en los otros píxeles. Por ello la matriz de píxeles (la imagen) se ha dividido en tantas secciones como especifique el usuario (en este caso 4) y para cada sección se asigna un hilo de ejecución. Esto permite que la generación de imágenes sea 4 veces más rápida que con una solución secuencial. Si no se hubiera paralelizado, las figuras 1 y 2 hubieran tardado unos 4 días aproximadamente para alcanzar la misma solución con 5K y 4.5K caminos por píxel respectivamente.

De los materiales propuestos el que converge más lentamente (necesita muchos más caminos por píxel que los demás) es el dieléctrico, esto es debido a que la refracción es el fenómeno más complejo de los implementados ya que se pueden producir ciertos efectos dentro del objeto como la reflexión total interna que dificultan el trazado del camino hacia una fuente de luz. A ésto se le añade que también refleja luz como si fuera un espejo, por lo que la ruleta rusa para determinar el evento generará también más ruido. En la figura 3 se puede observar cómo la esfera dieléctrica (la de más a la derecha) aún necesita más caminos por píxel para converger a una solución realista mientras que las demás esferas y paredes se ven mucho mejor.

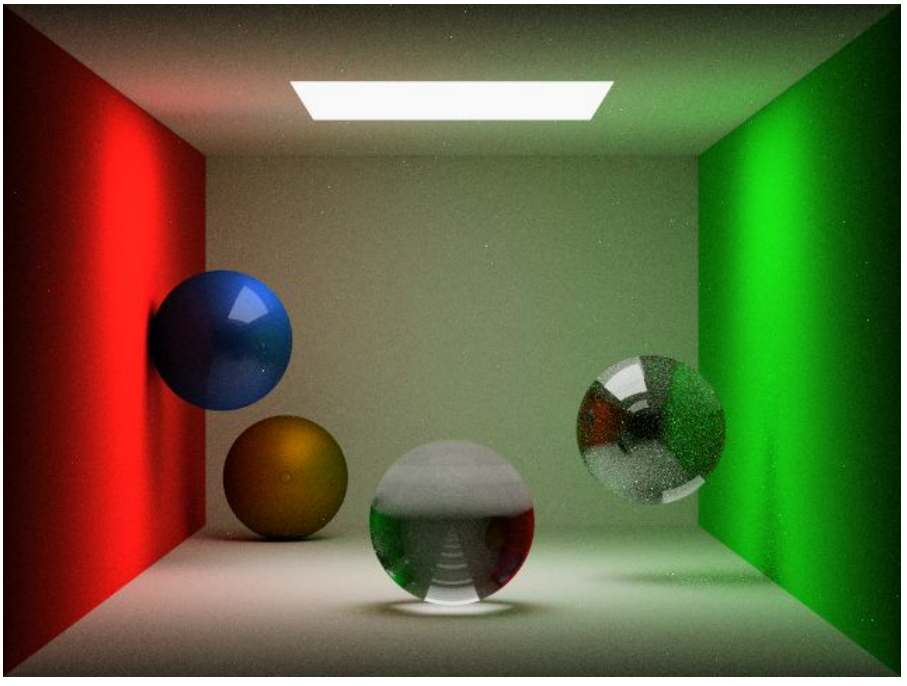


Figura 3: *N = 5.25K, tiempo de render -> Aproximadamente 26 horas*

En cuanto a las fuentes de luz, la convergencia de las escenas con luces puntuales es más rápida que con luces de área, esto se debe a que el cálculo de la luz directa de las luces puntuales es más rápido que el estar comprobando si el camino intersecta o no con las luces de área. Además con luces puntuales la solución necesita menos muestras para obtener un resultado parecido al de la misma escena con luces de área, esto es debido a que con las luces puntuales se calcula la contribución de la luz directa en cada intersección del camino (hasta que no intersecta con nada o la ruleta rusa lo “mata”), sin embargo si en una escena un camino no encuentra la luz de área, el valor devuelto en dicho camino es <0, 0, 0> por lo que depende de encontrar la luz a la hora de calcular el color del píxel de la imagen. Se puede comprobar para las figuras 4 y 5 con el mismo número de caminos por píxel se observa que la escena con la luz puntual tiene menos ruido que la que tiene la luz de área.

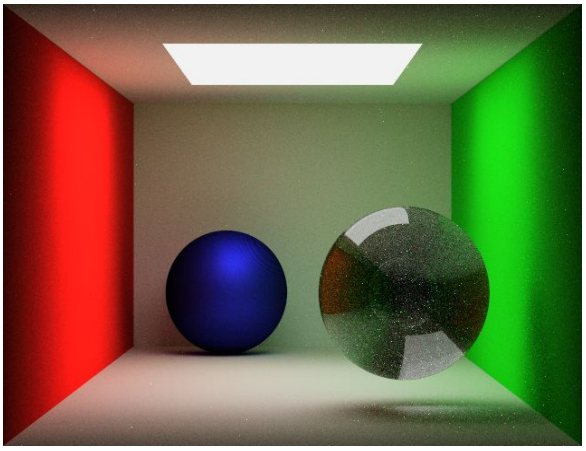


Figura 4: *Luz de área. 5K ppp. Tiempo render 24 horas aproximadamente.*

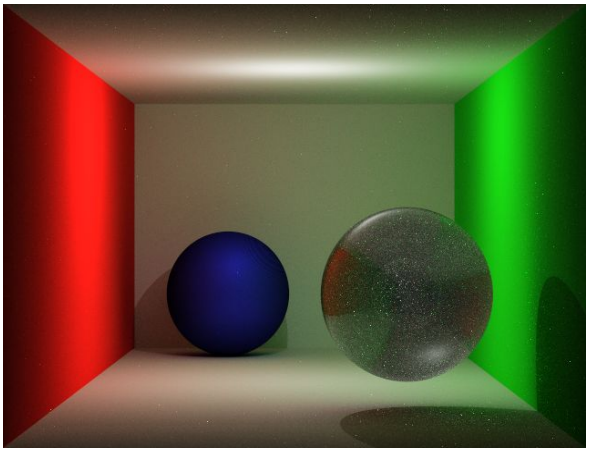


Figura 5: *Luz puntual. 5k ppp. Tiempo render 17 horas aproximadamente.*

Efectos de la iluminación global

Los efectos más sencillos de obtener con *Path tracing* son color bleeding (explicado anteriormente) y soft shadows. Éste último sólo se puede conseguir con luces de área (con las luces puntuales únicamente podemos obtener hard shadows que son menos realistas). Las soft shadows son fáciles de obtener por la propia naturaleza del algoritmo, ya que en un punto sombreado se lanzan múltiples caminos (caminos por píxel) y no todos salen rebotados en la misma dirección, habrá algunos que saldrán rebotados hacia la fuente de luz de forma que la sombra será cada vez más clara conforme nos acercamos a los bordes, produciendo un

efecto más realista. Se puede observar con claridad la diferencia entre soft shadows y hard shadows en las figuras 6 y 7, estas figuras son zoom de las figuras 4 y 5.

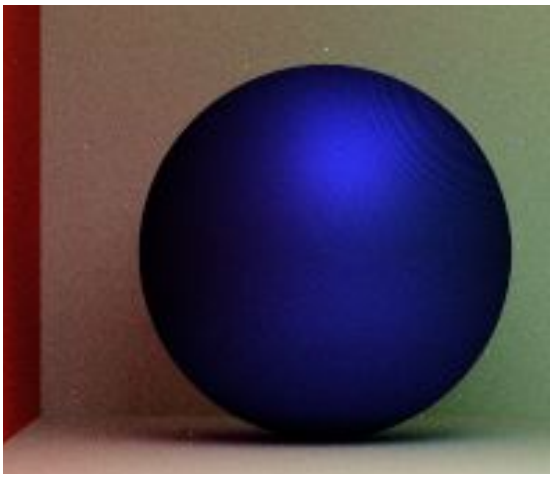


Figura 6: *Luz de área. 5K ppp. Tiempo render 24 horas aproximadamente.*

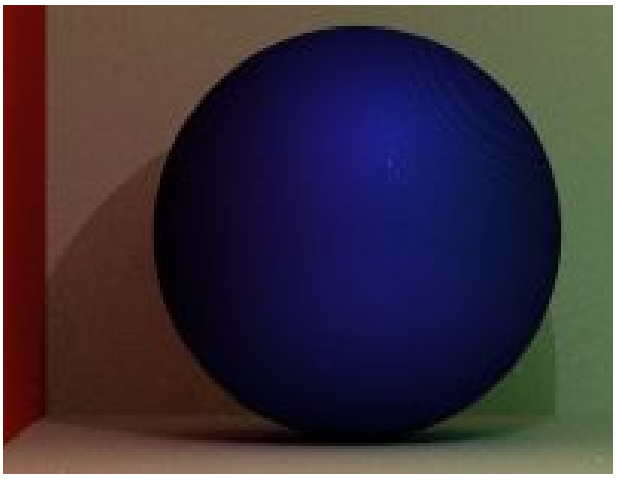


Figura 7: *Luz puntual. 5k ppp. Tiempo render 17 horas aproximadamente.*

Hay ciertos requerimientos para poder obtener los efectos de soft shadows, hard shadows, color bleeding y cáusticas. Para los dos primeros ya se ha mostrado arriba (depende del tipo de luz si área o puntual), para color bleeding únicamente es necesario que el material del objeto tenga una componente difusa para poder emitir en el color correspondiente y para las cáusticas es necesario que haya una fuente de luz de área y que un objeto de la escena refracte luz (por ejemplo una esfera de cristal), además el objeto debe estar cerca de otro difuso como puede ser una pared para que se aprecie el efecto de las cáusticas.

El efecto más difícil de conseguir es el de las cáusticas, debido a que es muy difícil reconstruir el camino de una cáustica (o imposible en el caso de las luces puntuales), por ello para que se consiga una solución realista es necesario un gran número de caminos por píxel. Con otras técnicas como *Photon Mapping* el conseguir este efecto se simplifica a cambio de un sesgo añadido.

Ejemplos de cáusticas conseguidas se pueden ver en las figuras 8 y 9. Sin embargo en la figura 10 se observa que como la escena tiene una fuente de luz puntual no aparece el mismo efecto que cuando hay una luz de área, es decir no hay ninguna cáustica.

Figura 8: *Zoom a la esfera de cristal de la figura 3.*

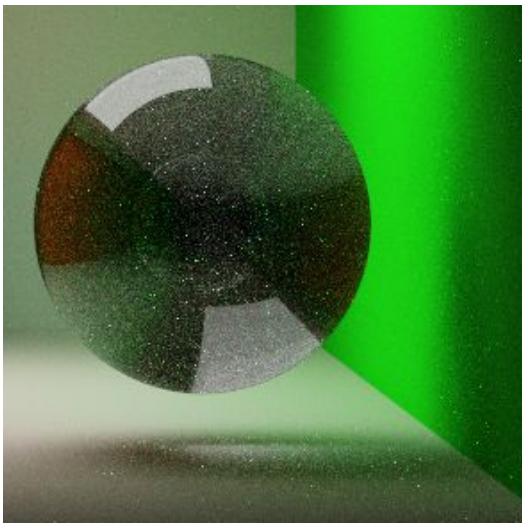


Figura 9: *Zoom a la esfera dieléctrica de la de la figura 4.*

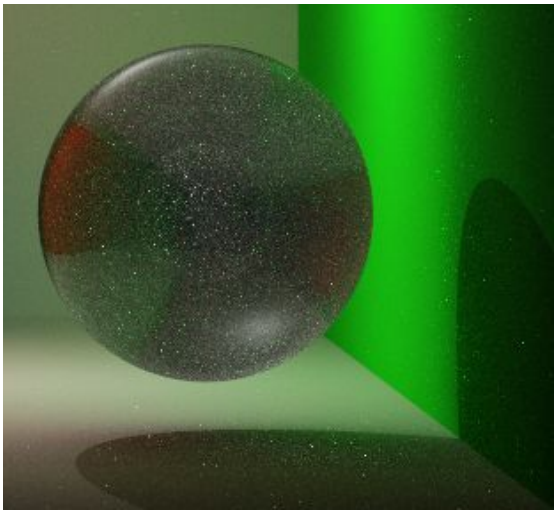


Figura 10: *Zoom a la esfera dieléctrica de la figura 5.*

Extensiones implementadas

Además de las mencionadas anteriormente (texturas y paralelización). Se han implementado diversas figuras geométricas como conos, cilindros, cubos, así como triángulos y discos (para estos dos no hay disponible ningún render) [6]. Se pueden apreciar en la figura 2. Estas figuras aportan mayor riqueza visual a las imágenes generadas debido a que hay más geometrías representables pero no supone una mejora al algoritmo.

A la hora de convertir las imágenes se ha hecho uso del Tone Mapper implementado, se ha desarrollado el operador de Reinhard global [8] para obtener mejores resultados a la hora de convertir las imágenes a LDR.

Bibliografía

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

[2] - *Graphics and Imaging Lab. Monte Carlo.*
https://moodle.unizar.es/add/pluginfile.php/2303601/course/section/356530/07-montecarlo-handout.pdf

[3] - *Wikipedia (10 January 2020, at 13:55). Snell's Law. https://en.wikipedia.org/wiki/Snell%27s_law*

[4] - *Scratchapixel 2.0. Introduction to Shading (Reflection, Refraction and Fresnel).*
https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

[5] - *Wikipedia (2 February 2020, at 08:15). Fresnel equations.*
https://en.wikipedia.org/wiki/Fresnel_equations

[6] - *Wolfram MathWorld: The Web's Most Extensive Mathematics Resource.*
http://mathworld.wolfram.com/

[7] - *Graphics and Imaging Lab. Imaging.*
https://moodle.unizar.es/add/pluginfile.php/2303601/course/section/356530/04-imaging-handout.pdf

[8] - *Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. ACM Trans. Graph., 21(3):267–276, July 2002.*
http://www.cmap.polytechnique.fr/~peyre/cours/x2005signal/hdr_photographic.pdf