

Práctica 4 - Path Tracing

Informática Gráfica

Curso 2021-2022



Universidad Zaragoza

TITULACIÓN: GRADO EN INGENIERÍA INFORMÁTICA

ASIGNATURA: INFORMÁTICA GRÁFICA (30234)

FECHA: 02 DE DICIEMBRE DE 2021

GRUPO: LUNES A - 18H A 20H - LAB 1.02

INTEGRANTES:	ECHAVARRI SOLA,	ÁLVARO	737400
	NUEZ MARTÍNEZ,	MARCOS	761319

ÍNDICE

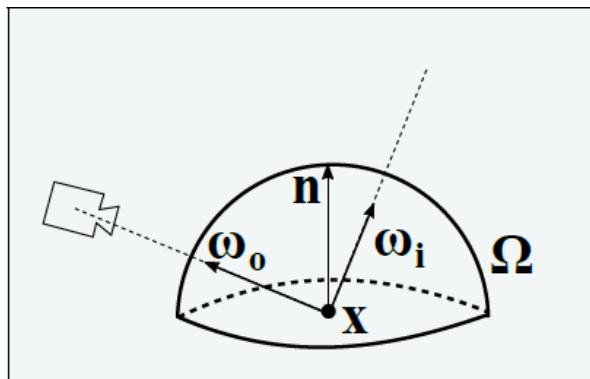
Diseño e Implementación	3
<u>1- Ecuación de Render</u>	3
1.1- Ecuación	3
1.2- Implementación	4
<u>2- Convergencia</u>	5
2.1- Convergencia respecto al tipo de luz utilizado	5
2.1.1- Luz de Área	5
2.1.2- Luz Puntual	7
2.1.3- Comparación	9
2.2- Convergencia de los materiales utilizados	10
<u>3- Iluminación global</u>	11
3.1- Especular	11
3.2- Refracción	12
3.2.1- Refracción perfecta	12
3.2.2- Cáusticas	12
3.3- Color bleeding	13
3.4- Soft shadows	14
3.5- Hard shadows	15
Opcionales	16
<u>1- Texturas</u>	16
<u>2- Paralelización</u>	18
<u>3- Progress Bar</u>	19
Resultados	20
Control de esfuerzos	22
<u>1- Planificación</u>	22
<u>2- Metodología</u>	22
<u>3- Tiempo dedicado</u>	23
Bibliografía	24

Diseño e Implementación

1- Ecuación de Render

1.1- Ecuación

En este apartado se va a explicar la ecuación de Render. A modo de introducción se va a proceder a contar en qué consiste dicha ecuación mediante el siguiente diagrama [1], el cual ilustra el fenómeno óptico producido:



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

Emisión Luz entrante BRDF Geometría

La ecuación de Render realiza una estimación de la radiancia emitida en el vector de dirección ω_o desde el punto X . Una vez obtenida se utiliza para realizar el cálculo del rayo lanzado desde la cámara hacia el punto X (pixel).

Como se puede apreciar en la ecuación de arriba, el rayo lanzado es el resultado del sumatorio de la luz de emisión y la luz reflejada:

- Luz de emisión: Es la luz emitida del objeto con el que el rayo lanzado está intersectando.
- Luz reflejada: Como su propio nombre indica es la luz reflejada, por el punto X con dirección ω_o , que llega directa e indirectamente por el resto de objetos que conforman la escena.

Como se puede observar, esta luz reflejada depende de 3 factores:

- Luz entrante: Es la propia iluminación de la escena
- BRDF (Bidirectional Reflectance Distribution Function): Define cómo la luz se refleja en los objetos de la escena.
- Geometría: Definida por el producto escalar de los vectores n y ω_i .

La integral en la que se utilizan los 3 factores mencionados no tiene una solución trivial en casos en los que la luz de emisión o la luz reflejada no se pueden aproximar mediante el uso de una función delta, lo que ocasiona que no tenga una solución analítica.

El que la integral no posea una solución analítica, para el caso mencionado, hace que haya que realizar una aproximación mediante el algoritmo de Monte Carlo; este algoritmo realiza una ruleta rusa la cual decide el evento que va a ocurrir, en función de los tres coeficientes (Lambertian diffuse, delta BRDF y delta BTDF) que tiene el material con el que intersecta el rayo lanzado, y con una probabilidad de que muera el rayo. Al ser una aproximación a medida que se van realizando mayor número de iteraciones se va aproximando mejor al resultado que tendría la integral

1.2- Implementación

Para realizar la estimación de la radiancia en un píxel se calcula la media de la radiancia obtenida al lanzar numRaysPerPixel rayos sobre ese mismo píxel: Estos rayos realizan un camino (path), el cual tiene como punto de partida el sitio donde esté situada la cámara, que viene marcado por el número de rebotes que realice dicho rayo hasta que muera.

La muerte de un rayo se puede dar por dos casos distintos:

- Muere de forma “natural” al alcanzar una fuente de luz.
- Muere de forma aleatoria debido a la ruleta rusa, que se utiliza en la implementación del algoritmo de Monte Carlo, la cual aplica le da una probabilidad pequeña a que ese rayo muera en cada rebote. Esto se realiza para evitar que un rayo rebote infinitamente ocasionando el colapso del programa. Para la generación aleatoria se ha utilizado una distribución uniforme que utiliza un generador mt19937 [\[2\]](#).

Por último, para obtener la estimación del color se utiliza una variable la cual almacena el color acumulado. Esta variable actualiza su valor cada vez que un rayo ha terminado su camino, de forma que al color acumulado se le suma el color que ha obtenido ese rayo durante su camino más el color de la luz directa dividido para el número de rebotes. Esto se realiza para obtener una única integral de Monte Carlo por pixel que media la radiancia de los caminos realizados por todos los rayos lanzados sobre él.

```
colorAcumulado = colorAcumulado + rayColor + colorLuzDirecta/numRebotes
```

2- Convergencia

2.1- Convergencia respecto al tipo de luz utilizado

2.1.1- Luz de Área

En cuanto al estudio de la convergencia con luz de Área sea ha renderizado una Cornell box con el techo como luz de área. Generando una iluminación general por toda la escena del mismo color que el techo (en este caso blanco), esto provoca ligeras sombras en las esferas pero sobre todo lo mas notable es el bleeding que los objetos de la escena provocan entre sí.

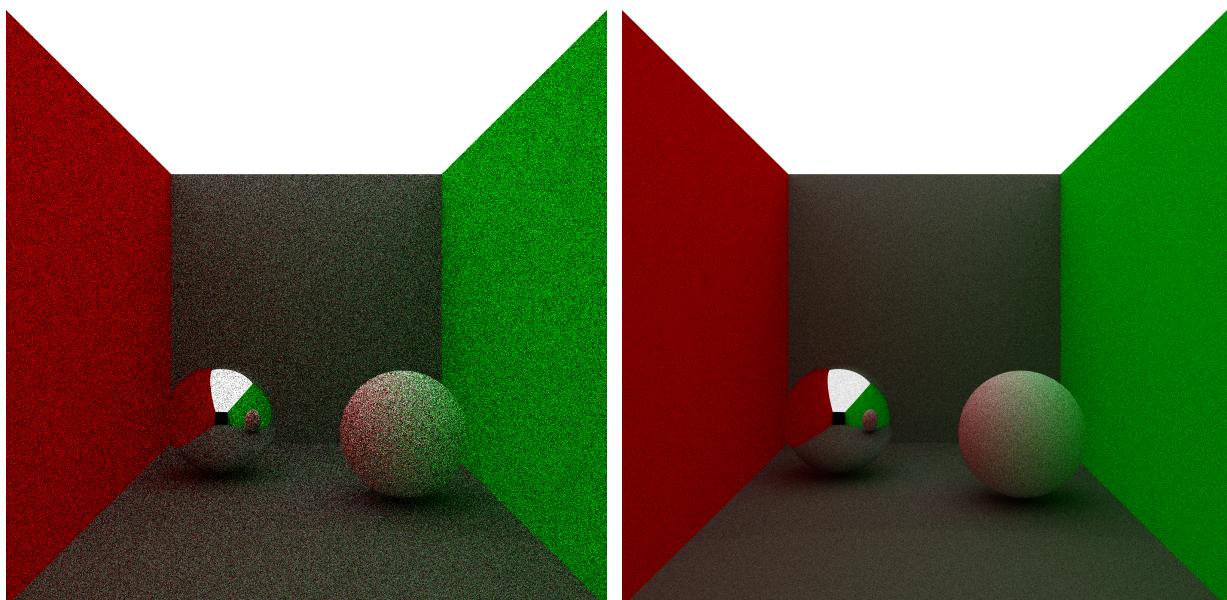


Figura 1 - Luz de Área y 1 Rayo

Figura 2 - Luz de Área y 10 Rayos

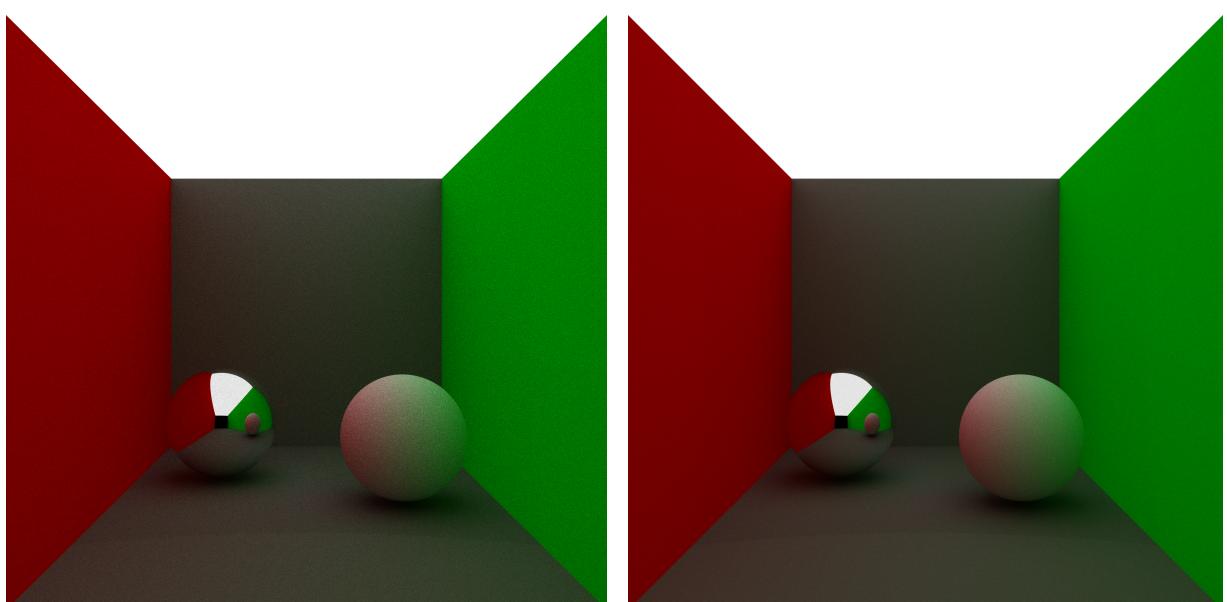


Figura 3 - Luz de Área y 50 Rayos

Figura 4 - Luz de Área y 150 Rayos

Resulta interesante observar el efecto espejo creado en la esfera especular y como se ve la escena desde otro punto de vista, como las sombras y luces de la escena también van apareciendo en el reflejo. Por último, cabe recalcar cómo el efecto de *bleeding* se hace más que evidente en la esfera opaca, su color blanco se ve manchado por las paredes rojas y verdes, siendo más fuerte esta última.

Además, se ha realizado una gráfica que refleja el coste en tiempo en función del número de rayos lanzados para la escena mencionada:

Convergencia Luz de Área

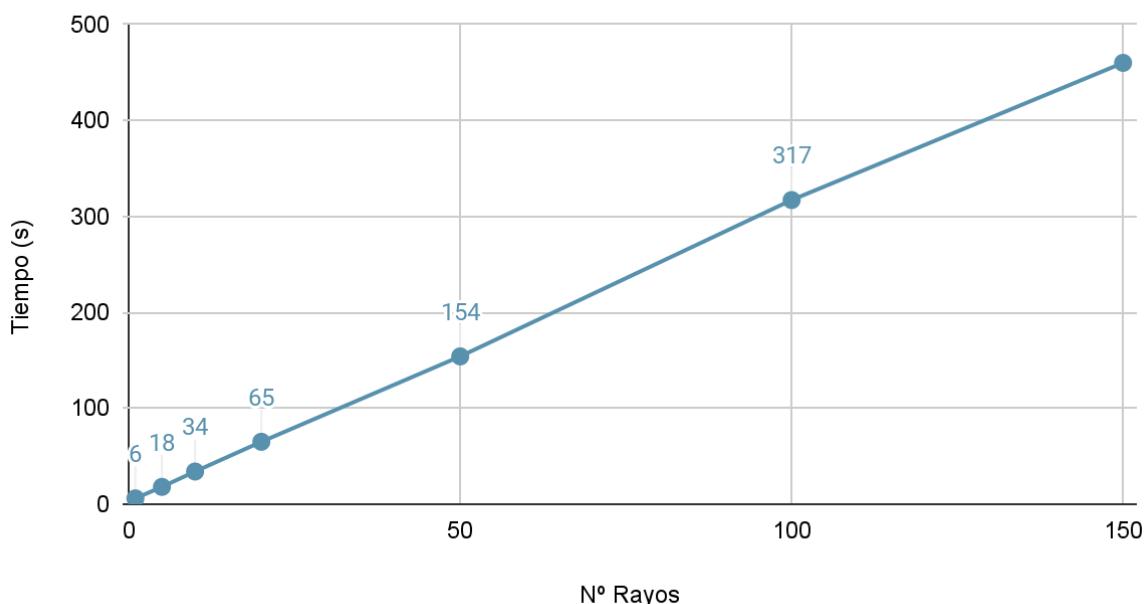


Figura 5- Gráfica Coste en tiempo/numRayos con Luz de Área

Se observa una convergencia rápida ya que a mayor número de rayos utilizados mayor es el coste en tiempo. El coste es aproximadamente de Orden(n) siendo n el número de rayos lanzados.

2.1.2- Luz Puntual

Este apartado es, en concepto, más simple que el anterior. En vez de haber una luz de área enorme y que los rayos devuelvan resultado si chocan con ella antes de morirse, se trazan rayos de luz desde la luz puntual a cada punto de rebote de los rayos y se comprueba si hay algún objeto por medio para decidir si hay sombra o no.

También se aplica un factor a la potencia de la luz que hace disminuir su influencia en las primitivas según la distancia y el factor de difusión del material.

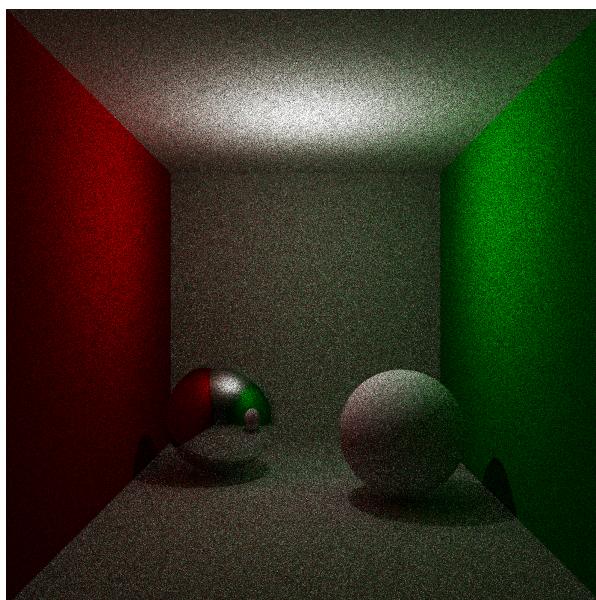


Figura 6 - Luz Puntual y 1 Rayo

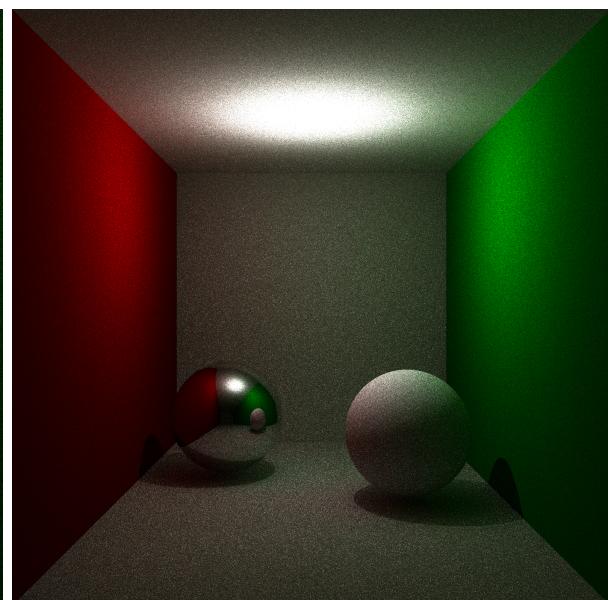


Figura 7 - Luz Puntual y 10 Rayos

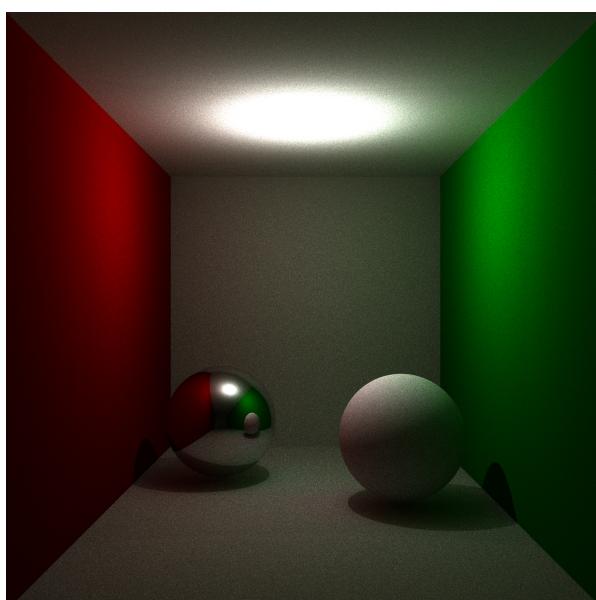


Figura 8 - Luz Puntual y 50 Rayos

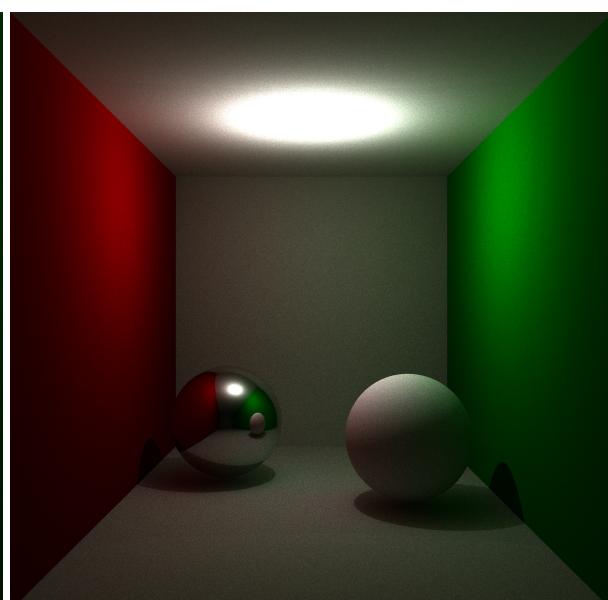


Figura 9 - Luz Puntual y 100 Rayos

Todo este proceso genera las imágenes de más arriba, aunque son mas oscuras que las luces de área y, como ya se explorará en el siguiente apartado, su coste de tiempo es mucho mayor, este segundo método ofrece un renderizado mucho mas real, sobre todo con el tratamiento de luces y sombras.

Por último, se ha realizado una gráfica que refleja el coste en tiempo en función del número de rayos lanzados para la escena mencionada:

Convergencia Luz Puntual

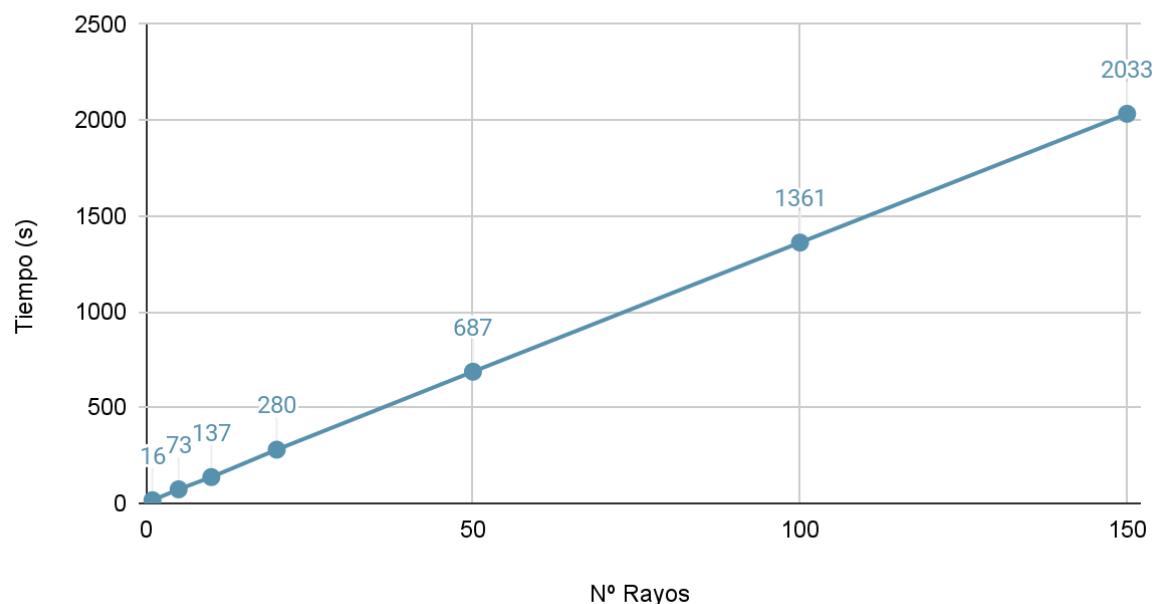


Figura 10 - Gráfica Coste en tiempo/numRayos con Luz Puntual

No hay mucho que comentar sobre esta gráfica, su comportamiento lineal es idéntico al de la gráfica de [Luz de Área](#) solo que con una pendiente mucho mas acentuada, lo que refleja un crecimiento en el coste mucho mayor según se aumenta el número de rayos.

2.1.3- Comparación

El número de rayos lanzados por pixel es un parámetro muy ligado a la nitidez de la imagen resultante. Esto es debido a que cuantas mas iteraciones de Monte Carlo son aplicadas en un mismo píxel más se acerca la aproximación a la imagen real. Esto es acentuado por el método de iluminación elegido ya que hace variar el número de rebotes lo que impacta directamente en su coste. Eso es debido a que el coste total es:

$$\text{altura de la imagen} * \text{anchura de la imagen} * \text{nºrayos} * \text{nºrebotes}$$

En el caso de la Luz Puntual los rayos rebotan hasta que la ruleta rusa (Monte Carlo) decida que se mueren, no parara ni cuando encuentre una fuente de luz (ya que no hay en la escena) ni cuando deje de interactuar con algo (es una escena con 4 paredes techo y suelo). El resultado es un incremento enorme en el número de rebotes lo que afecta directamente al número de iteraciones.

En la siguiente gráfica se puede observar una comparación entre el método de Luz de Área y Luz Puntual:

Convergencia Luz Puntual / Área

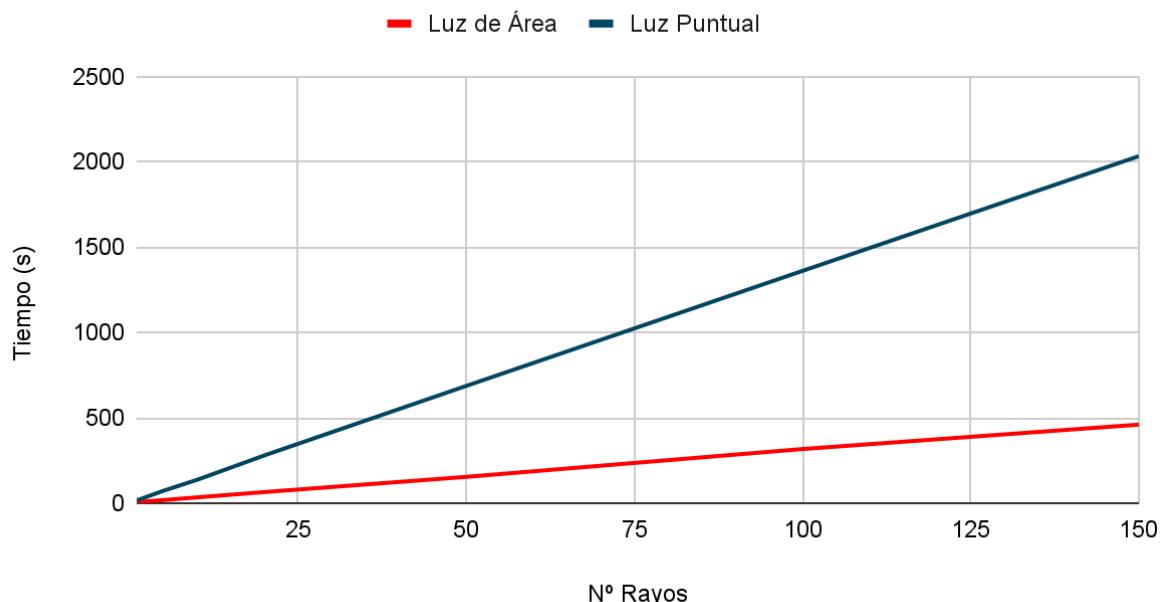


Figura 11 - Gráfica comparativa del Coste en tiempo/numRayos entre Luz de Área y Puntual

Aunque las imágenes resultantes de luz Puntual puedan parecer mucho mas reales y por tanto la razón lógica a tomar, el coste en tiempo juega un gran factor en contra de este método. La gráfica de arriba muestra una gran

diferencia en sus costes en tiempo, la Luz de Área crece a un ritmo de 3.65 segundos por rayo lanzado ($T = 3,652380 \times \text{nºrayos}$) mientras que la Luz Puntual sigue un crecimiento de 14.27 segundos por rayo lanzado ($T = 14,275 \times \text{nºrayos}$). Esta gran diferencia juega un papel crucial a la hora de una implementación real ya que según el tiempo del que se dispone para el renderizado se gastará mas o menos en el hardware y en los recursos para hacer funcionar ese hardware (personal y energía).

2.2- Convergencia de los materiales utilizados

Los materiales de la escena están definidos, en parte, por unos coeficientes que especifican cómo se comportan ante de los diferentes eventos: difusión, especular y refracción. Esto se aplica sobre todo a la ruleta rusa (se utiliza además como factor extra al calcular como los colores se extienden por la escena). Esta ruleta rusa decide si cada rayo muere o no, pero además, según los coeficientes del material en cuestión decide si va a ocurrir difusión, especular o refracción.

Esto afecta directamente a la convergencia de la escena, no es lo mismo un material que siempre hace difusión a un material especular. El caso mas extremo sería por ejemplo dos esferas especulares, los rayos no dejarían de rebotar de una esfera a la otra hasta que la ruleta rusa decidiera que mueren, causando un efecto espejo “infinito” en la imagen.

Resulta interesante experimentar con los valores de los coeficientes y ver los resultados, se puede llegar a una escena mas detallada, mas realista modificando los materiales sin aumentar en exceso el coste en tiempo.

3- Iluminación global

3.1- Especular

Un material especular es lo mismo que un espejo, es por eso que refleja los rayos que inciden sobre él con una dirección especular. La dirección especular es una entre todas las direcciones de la esfera que es de tipo delta.

Para obtener este efecto se ha añadido a la escena una esfera de material especular perfecto ($kD: 0$, $BRDF: 1$, $BRTF:0$), otra esfera de material difuso perfecto la cual se puede ver reflejada en la imagen, luz puntual establecida en el centro del techo y planos difusos de diferentes colores:

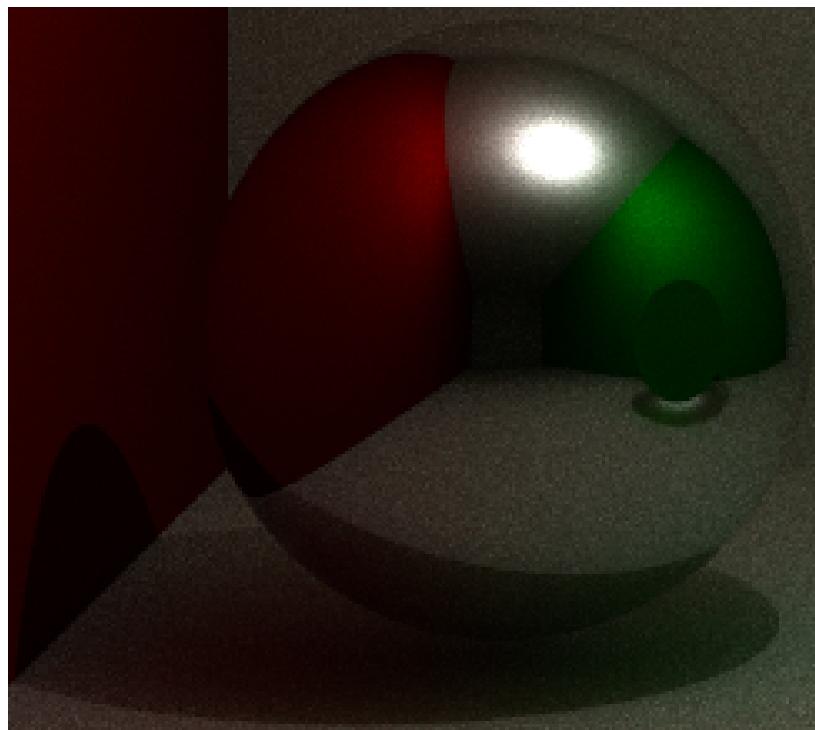


Figura 12 - Especular perfecta

3.2- Refracción

3.2.1- Refracción perfecta

La propiedad de refracción es el cambio de dirección que experimenta un rayo al pasar de un medio material a otro. Sólo se produce si el rayo incide oblicuamente sobre la superficie de separación de los dos medios y si estos tienen índices de refracción distintos. La refracción se origina en el cambio de velocidad de propagación del rayo, cuando pasa de un medio a otro.

El comportamiento de los materiales dotados con esta propiedad queda definido por el valor de la función delta que se les aplica.

Para obtener este efecto se ha añadido a la escena una esfera de material refracción perfecta (KD: 0, BRDF: 0, BRTF: 1), otra esfera de material difuso perfecto, luz puntual establecida en el centro del techo y planos difusos de diferentes colores que son los que aparecen sobre la esfera de la imagen de abajo.

3.2.2- Cáusticas

Las cáusticas son las envolventes de los rayos de luz refractados por nuestras esferas de material refractado, o la proyección de esa envolvente de rayos en otra superficie. Estas cáusticas se pueden apreciar en la siguiente imagen:

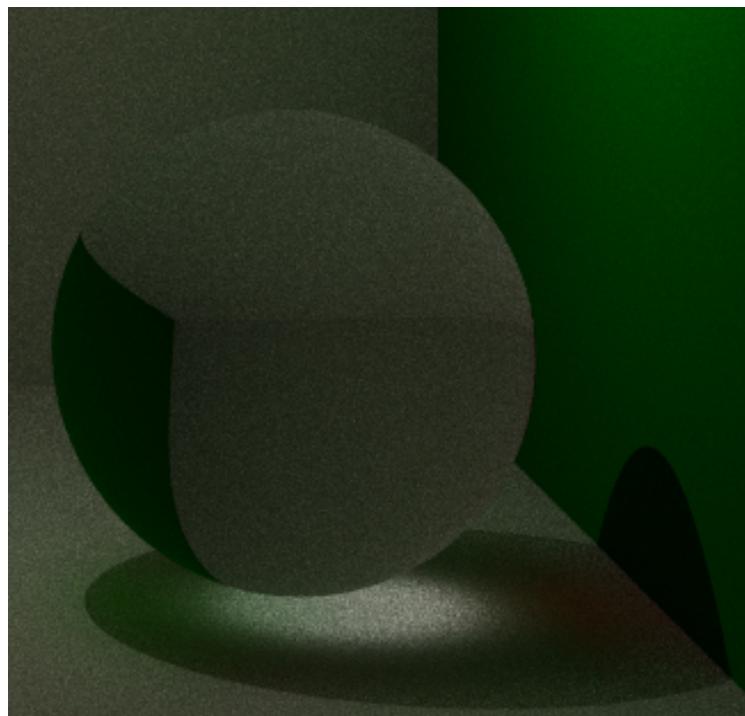


Figura 13 - Refracción perfecta & Cáusticas

3.3- Color bleeding

El efecto *bleeding* presente en el color de la escena intenta simular la influencia que tiene el color de cada material de la escena en otros objetos de la misma. Si son materiales difusos, o semi-difusos, los objetos cercanos a estos materiales deberían verse manchados por su color. Una pared roja junto a una esfera blanca reflejaría el rojo como una mancha difusa en la esfera blanca, por poner un ejemplo de *bleeding*.

Este efecto se ha conseguido realizando una media de los colores de los materiales por los que el rayo que renderiza la escena rebota, cuando el rayo muere o encuentra luz de área se devuelve como resultado esta media junto a la iluminación puntual si la hubiera.

En la siguiente imagen se puede observar una esfera blanca de difusión perfecta (kD: 1, BRDF: 0, BRTF:0) cerca de una pared verde y roja.

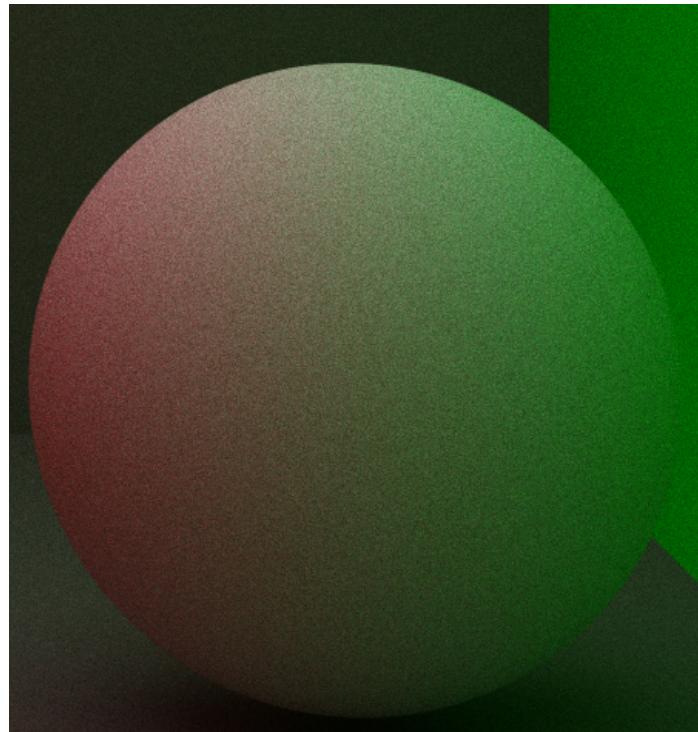


Figura 14 - Color bleeding

La esfera de arriba muestra claramente un efecto *bleeding* en ambas mitades. La parte izquierda se ve más afectada por la pared roja al otro lado de la escena, mientras que la parte derecha está, en comparación, más afectada por la pared verde que tiene justo al lado.

La influencia de la pared verde es claramente mayor que la pared roja, esto es debido a que es mucho mas probable (ángulo de 180º respecto a la normal aproximadamente) que un rayo rebote en la pared verde que en la roja.

3.4- Soft shadows

El proceso de iluminación con luz de Área tiene un efecto secundario provocado por el comportamiento general de los rayos. Los rayos que mas veces rebotan tienen mas probabilidad de morir al no encontrar la luz de Área. Si por ejemplo un rayo rebota en la base de una esfera, por lo general rebotara mas veces que si rebotara en la parte mas alta ya que tendra el suelo o paredes mas lejos. Esto a la larga acaba generando unas sombras suaves en zonas como las bases de las esferas o las esquinas de las paredes, donde los rayos se ven obligados a rebotar mas veces.

Es un resultado indirecto de renderizar la escena mediante rebotes de rayos lanzados desde la cámara y depende completamente del tamaño y posición de los objetos pero sobre todo de la luz o luces de Área.

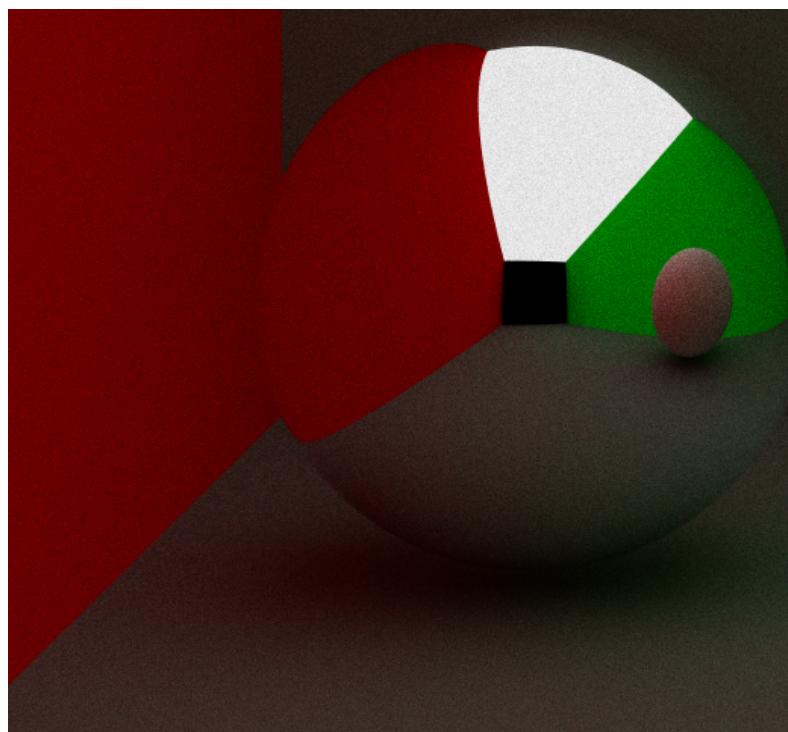


Figura 15 - Soft shadows

3.5- Hard shadows

Las *hard shadows* o sombras fuertes son un efecto similar a las *soft shadows* creadas mediante luz de Área pero algo más acentuadas y dependientes de la posición de la luz Puntual. Esto resulta en unas sombras más oscuras y definidas, mucho más afectadas por la iluminación y materiales de la escena.

Este efecto se ha conseguido trazando rayos de luz desde cada rebote de los rayos principales (los lanzados desde la cámara) hasta la posición de la luz Puntual.

Si se encuentra algún objeto en el camino del rayo se determina que en ese píxel hay sombra, el rayo principal sigue rebotando aumentando el valor de luz en ese píxel. Por último, se divide por el número de rebotes que ha realizado el rayo para dar como resultado una media de la iluminación detectada en ese punto.

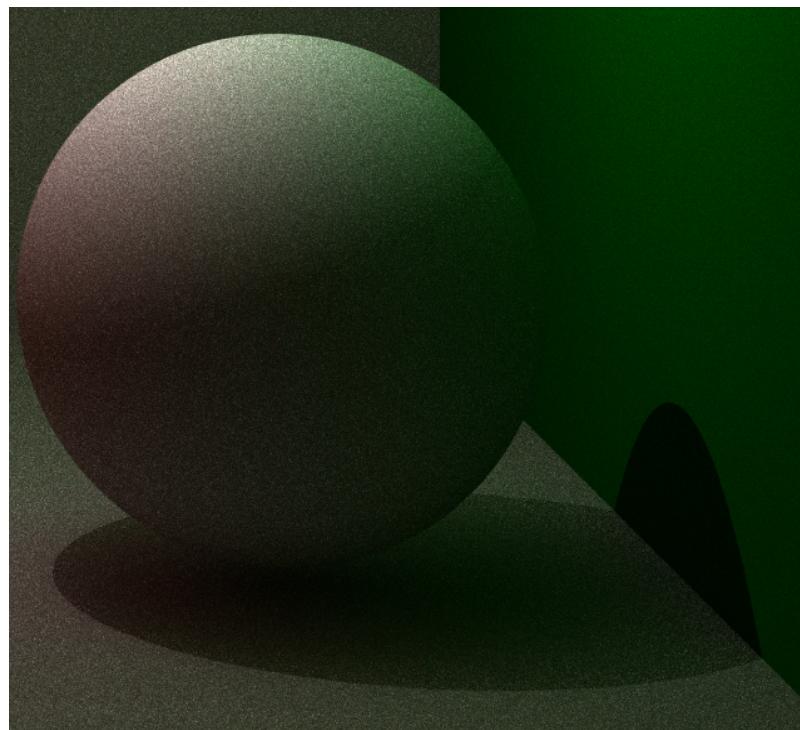


Figura 16 - Hard Shadows

En la imagen de arriba podemos observar una esfera blanca difusa ($kD: 0$, $BRDF: 1$, $BRTF: 0$) junto a una pared verde, en la escena hay una luz Puntual en el techo que genera una *hard shadow* en la base de la esfera. La sombra sigue la figura de la esfera y se ve proyectada en la pared verde, en la sombra del suelo se pueden observar otros tonos rojos y verdes creados por los rebotes del rayo principal. Este mismo proceso que genera las sombras también genera un efecto *bleeding* algo menos notorio que con luz de Área.

Opcionales

1- Texturas

Se ha implementado un sistema de texturas al proyecto, cualquier primitiva en la escena puede tener cualquier imagen, en formato ppm, como su color en vez de un color plano de tipo tupla RGB. Por parámetro se especifica que texturas se van a utilizar, se introduce el nombre del archivo .ppm que además debe estar incluido en el fichero textures, dentro de lib en la raíz del proyecto.

Obviamente si se introduce una imagen demasiado pequeña apenas se distinguirá y si se introduce una demasiado grande no se mostrará entera. Respetando estos límites mencionados, se recalculan los ejes de los planos y se encuadra la textura en la escena.

Como la escena se encuentra justo en el $x = 0.0$ y $y = 0.0$, las coordenadas de los impactos de los rayos pueden ser negativos en sus coordenadas y por tanto las texturas pueden ver sus ejes invertidos, efecto que no queremos conseguir. Además si la imagen es suficientemente grande, mas o menos la mitad del espacio disponible, esta se renderiza mal ya que la imagen empieza en el $(0.0\ 0.0)$ en vez de en la esquina de ese plano. Utilizando unas esquinas backgroundLeft y frontRight (definidas en $(-20,-20,-100)$ y $(20,20,0.0)$ respectivamente) se consigue encuadrar la imagen correctamente dentro de la propia escena.

Por último, en cuanto a la lectura y procesado de los archivos ppm se ha reutilizado el código de la práctica 2 en la cual se trataban estos archivos. Las mismas funciones han acabado siendo muy útiles sin necesidad de modificarlas.

Las siguientes texturas están ya incluidas en la carpeta /lib/textures en formato ppm.



Figura 17 - xokas.ppm

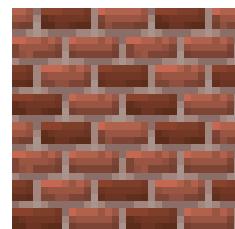


Figura 18 - bricks.ppm

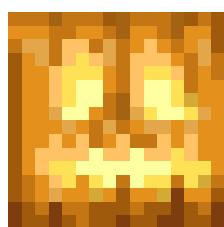


Figura 19 - jack_o_lantern.ppm



Figura 20 - pepe.ppm

El ejecutable está configurado para que todos los elementos de la escena puedan tener texturas, según el número de texturas introducidas por línea de comandos, se relacionarán texturas con unas primitivas u otras. Por ejemplo si ejecutas `./practica4 2000 2000 100 xokas bricks bricks bricks pepe`, el fondo tendrá la textura xokas.ppm, el suelo y las paredes tendrá la textura bricks.ppm y el techo la textura pepe.ppm. En cambio si solo introduces la primera textura (xokas) solo el fondo tendrá textura.

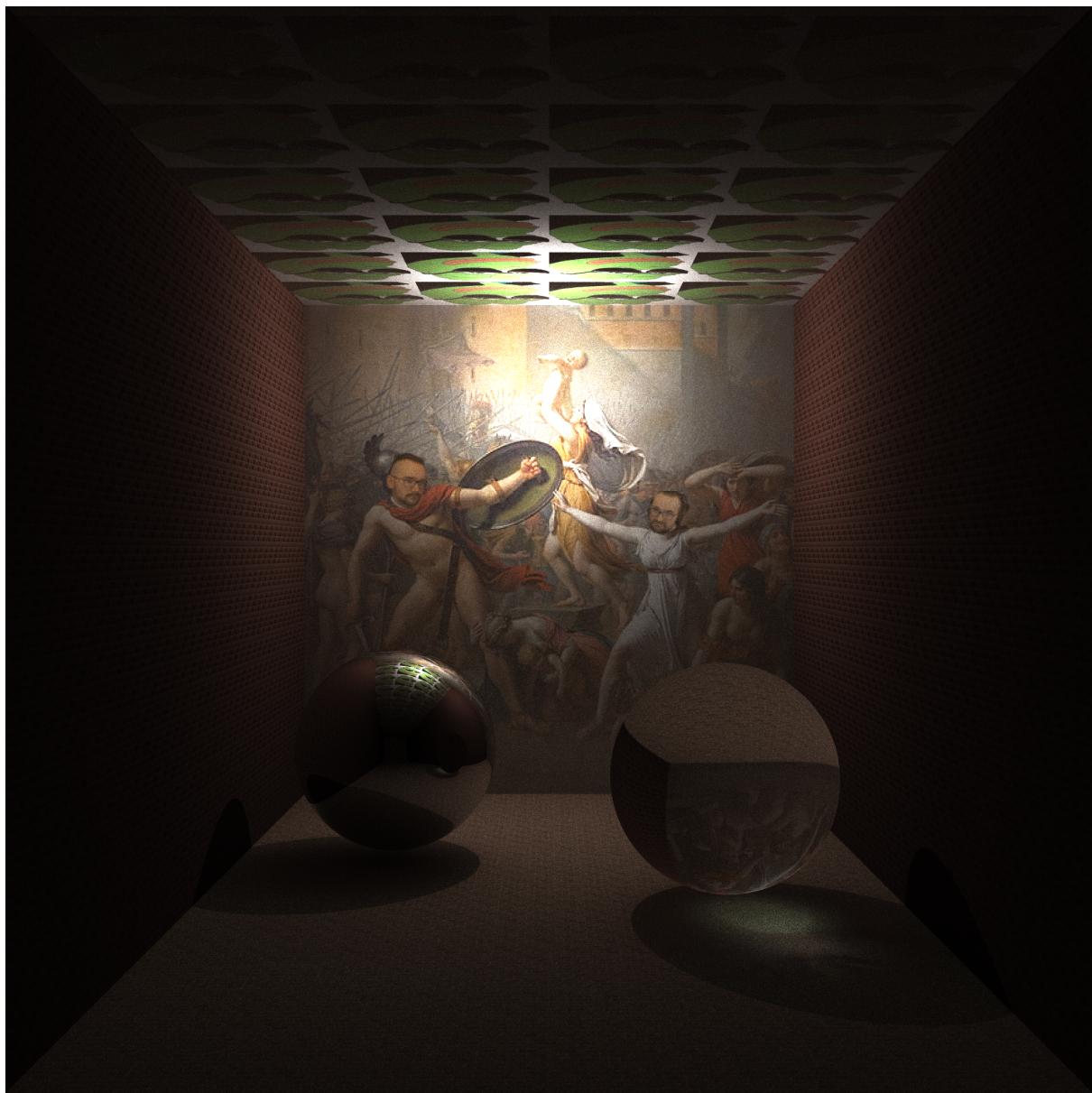


Figura 21 - Escena con varias texturas y luz Puntual

2- Paralelización

Este optional consiste principalmente en el lanzamiento de threads para mejorar el coste en tiempo de la generación de imágenes.

Para realizar este optional se ha utilizado la librería estándar de C++ llamada “thread”. Para aprender cómo utilizarla nos hemos apoyado en una página web [\[3\]](#) que contiene un tutorial.

La implementación ha sido la siguiente: La estrategia utilizada ha sido la de lanzar tantos procesos como el ordenador del que lo está ejecutando permita, lo cual ha sido posible gracias a la función “hardware_concurrency()”. En función al número de procesos que se hayan lanzado se realiza un reparto de los píxeles para que cada uno trate píxeles diferentes de la imagen, para ello se pasa a la función “shootingRaysAux()” dos parámetros para indicar por qué pixel empieza y en cual acaba cada thread.

Una vez lanzados los threads, el hilo principal quiere esperar a que un hilo finalice correctamente. Entonces, usamos `join()`. Si el hilo principal inicial no espera a que termine el nuevo hilo, este continuaría hasta el final del `main()` y finalizaría el programa, posiblemente antes de que el nuevo hilo tuviera la oportunidad de ejecutarse.

3- Progress Bar

Para la realización de este opcional se ha utilizado una librería externa llamada Progress Bar [\[4\]](#), la cual ha sido obtenida de Github.

Somos conscientes de que no se pueden utilizar librerías externas, pero para poder implementar esta librería tuvimos que realizar una serie de cambios tanto en la propia librería como en el programa principal, puesto que al lanzar millones de rayos el uso de una progress bar ralentizaba la generación de la imagen. Además, independientemente de que no se haya realizado plenamente la progress bar creemos que merece la pena utilizarla por el feedback que devuelve al usuario ya que permite ver cuánto le queda para finalizar y el tiempo que lleva ejecutándose.

Para evitar la ralentización del programa mediante su uso se modificó la librería añadiéndole un mutex para garantizar la exclusión mutua en el acceso a la variable que lleva la cuenta del progreso del programa y para desplegar visualmente la barra. De esta forma nos aseguramos que la edición de cada proceso en la ProgressBar se realiza de forma exclusiva.

Además, se encontró otro problema que era el de que al actualizar tantas veces el valor la propia barra se volvía loca mientras se mostraba en la terminal. Para ello, se decidió que no se mostrara en la terminal siempre que terminara un rayo sino que se actualizara cada 100 rayos lanzados, de esta forma el número de veces que se llamaba a la función `display()` era $100 * \text{numThreads}$ veces menor favoreciendo al correcto funcionamiento de la barra.

Por último, la siguiente imagen refleja como se ve la progress bar utilizada:



Figura 22 - Progress Bar

Resultados

Analizando los resultados obtenidos, tanto en luz de Área como luz Puntual, se han obtenido unas imágenes muy realistas con gran nitidez y sin fallos evidentes en el render. La iluminación de luz de Área genera una imagen con mucha luz pero unas sombras poco definidas y reflejos poco marcados, mientras que la luz Puntual crea una escena considerablemente más oscura pero con grandes detalles en sombras y luces. Cabe destacar el efecto especular y de refracción que muestran unas cáusticas muy interesantes y unos reflejos con buen detalle.

Sin embargo, por mucho que nos pese no hemos podido eliminar todos los fallos de la escena, sigue habiendo un degradado fuerte en algunos de los planos. Lo que parece ser una línea que separa justo el plano en dos de forma vertical (horizontal en el caso de techo y suelo), que crea una zona más clara que la otra. Suponemos que es algún tipo de fallo de precisión/cálculo del rebote de los rayos en algunos casos en los que la normal y el rayo incidente cumplen ciertas condiciones, aún suponiendo esto, nos ha sido imposible detectar esas condiciones y nos hemos conformado con el resultado obtenido en ambos campos.

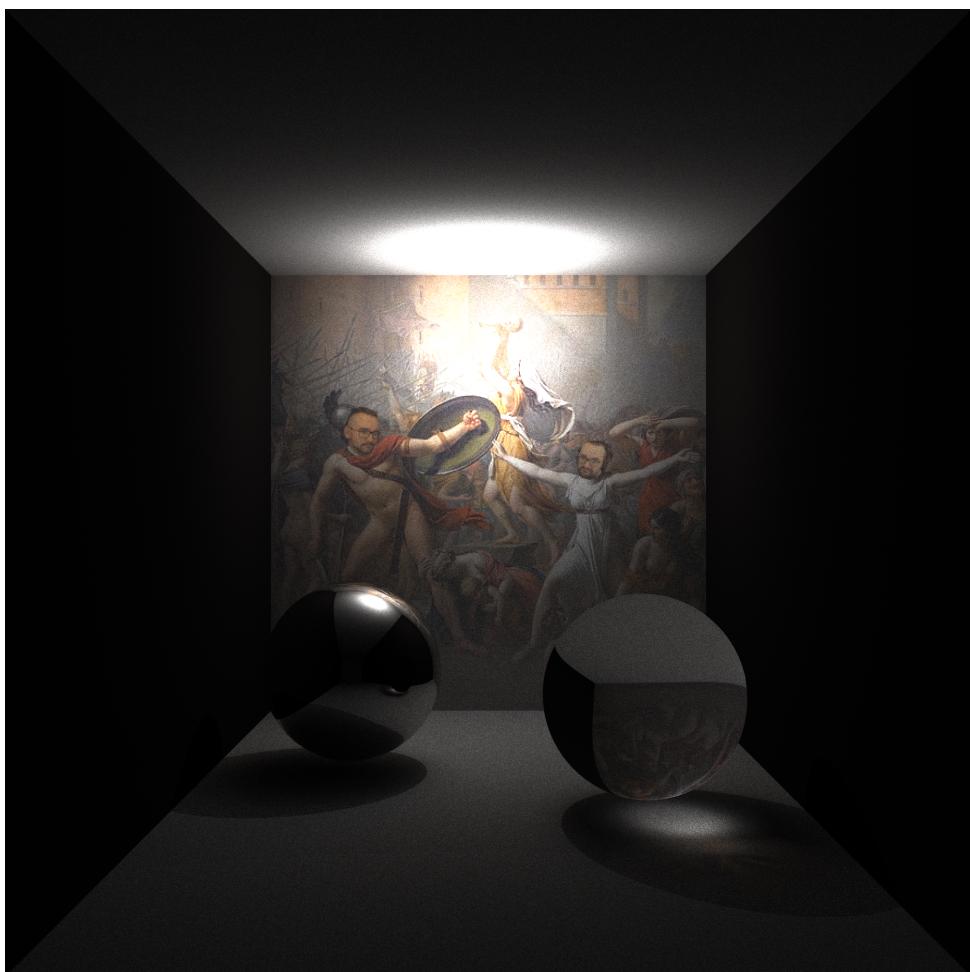


Figura 23 - Escena con textura en el fondo, luces puntuales y especular-refracción

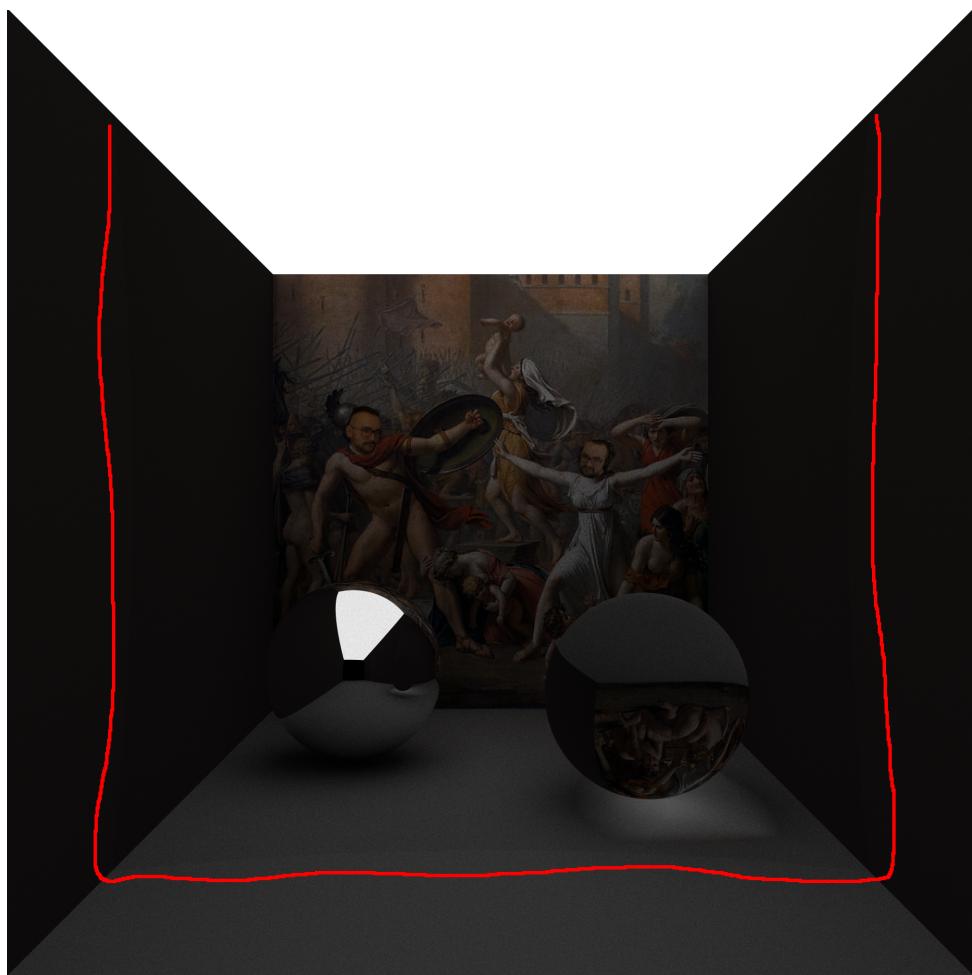


Figura 24 - Escena con textura en el fondo, luz de área y especular-refracción

En esta segunda imagen se puede observar el fallo mencionado anteriormente.
(remarcado en rojo)

Control de esfuerzos

1- Planificación

El comienzo de esta práctica tuvo lugar el 8 de Noviembre a las 18:00 y se recomendaba que se tuviera hecha para el 29 de Noviembre, es por ello que desde el comienzo se decidió dedicarle todo el tiempo posible para llegar a ese deadline de forma que se pudiera dejar la memoria terminada en torno a la primera semana de Diciembre.

Para alcanzar el objetivo se intentó tener en semana y media la parte de luces en el área. Sin embargo, debido a ciertos problemas con las normales y con la utilización función intersección de las esferas se demoró 1 semana más.

Tras alcanzar el primer gran objetivo que era realizar la luz de área. Se intentó realizar las luces puntuales en 1 semana con el fin de tener la parte práctica terminada para el día 29, lo cual se consiguió terminando así la práctica el 29 a mitad de tarde.

Además, se estableció que la memoria debía quedar hecha antes del viernes 3 de Diciembre para así poder dedicar el puente a ponerse al día con otras asignaturas y comenzar la práctica 5 de esta asignatura.

Por último, se solicitó una tutoría al profesor Adolfo para así corroborar que los resultados obtenidos de la práctica 4 eran correctos y con ello dar la práctica como finalizada.

2- Metodología

Para la realización de la práctica se decidió realizar los distintos apartados conjuntamente, de forma que así los dos integrantes del grupo nos enteraríamos por igual de las decisiones de implementación tomadas y compartiríamos conocimientos.

Para poder trabajar sobre el proyecto en conjunto, se utilizó una extensión de Visual Studio Code llamada “Live Share” la cual permite a un usuario crear una sesión online en la que comparte su proyecto con permisos de lectura y escritura, permitiendo así al otro miembro escribir, editar y depurar el código en tiempo real.

Además, se creó un proyecto en GitHub con el objetivo de llevar un mantenimiento de las diferentes versiones del proyecto y para que el usuario que no hubiera creado la sesión de Live Share pudiera tener el código del proyecto actualizado.

Esta forma de trabajar fue posible debido a que ambos tenemos unos horarios compatibles por lo que podíamos quedar a horas que nos beneficiaran a ambos.

3- Tiempo dedicado

En este conjunto de tablas se puede apreciar el número de horas dedicadas por cada integrante del grupo para los diferentes hitos del proyecto. Como se ha mencionado en la metodología aplicada toda la práctica ha sido realizada conjuntamente, por lo que el número de horas han sido las mismas para cada integrante.

Integrante	Marcos Nuez Martínez			
Tareas	Implementación	Opcionales	Depuración	Memoria
Horas	30	10	80	15

Integrante	Álvaro Echavarri Sola			
Tareas	Implementación	Opcionales	Depuración	Memoria
Horas	30	10	80	15

Como se puede ver, el número de horas dedicadas a la realización de esta práctica han sido un total de 135 horas.

Bibliografía

- [1] *Diagrama de la ecuación de render.* URL: <https://moodle.unizar.es/add/pluginfile.php/4805064/course/section/480981/09-pathtracing-handout.pdf>
- [2] *Generador de números “aleatorios”.* URL: <https://stackoverflow.com/questions/21237905/how-do-i-generate-thread-safe-uniform-random-numbers>
- [3] Thread Tutorial. URL: https://www.bogotobogo.com/cplusplus/C11/1_C11_creating_thread.php
- [4] Progress Bar. URL: <https://github.com/prakhar1989/progress-cpp>