

Informática Gráfica

Path Tracing

José Daniel Subías Sarrato (NIA: 759533)
Fernando Peña Bes (NIA: 756012)

2 de febrero de 2021

Índice

1. Introducción	1
2. Objetivos	2
3. Ecuación de Render	3
3.1. Calculo de la ecuación	3
3.2. Muestreo de los materiales	5
3.3. Pasos del algoritmo de Path Tracing	6
4. Convergencia	6
4.1. Caminos por píxel	7
5. Materiales	8
5.1. Lambertiano Difuso	9
5.2. Especular perfecto	10
5.3. Plástico	11
5.4. Dieléctricos	12
6. Fuentes de luz	13
6.1. Next Event Estimation	14
7. Iluminación Global	15
7.1. Sombras	15
7.2. Color Bleeding	16
7.3. Cáusticas	17
8. Tone Mapping	17
8.1. Clamping	18
8.2. Equalization	18
8.3. Equalize and Clamping	19
8.4. Gamma curve	20
8.5. Clamp and gamma curve	20
9. Opcionales	21
9.1. Paralelización	21
9.2. Bounding Volumen Hierarchies	23
9.3. BRDF de Phong	25
9.4. Tone Mapper de Reinhard 05	26
9.5. Primitivas adicionales	27
9.5.1. Triángulos	27
9.5.2. Cuadriláteros	27
9.5.3. Mallas de triángulos	28
9.6. Texturas	30
10. Profundidad de campo	32

11. Conclusiones	34
12. Control de esfuerzos	35

1. Introducción

Dentro del gran conjunto de disciplinas que abarca la informática, la informática gráfica desempeña la tarea de utilizar los ordenadores para generar imágenes. Esta rama de las ciencias de la computación se basa la aplicación de conceptos matemáticos como: geometría, estadística y álgebra así como de las leyes físicas. El principal objetivo intentar resolver el problema de: dado un modelo que intenta representar la realidad ¿De que color pinto los píxeles de una imagen para representar dicho modelo de la forma mas real posible? Es aquí donde surgen diversos algoritmos que intentan, resolver este problema de la manera mas eficiente posible. A lo largo de la asignatura de *Informática Gráfica*, se han descrito los algoritmos mas utilizados en el entorno empresarial.

En este trabajo, se pretende implementar el algoritmo de *Path Tracer*. Este algoritmo recibe como entrada una lista de geometrías en \mathbb{R}^3 y un cámara representada como un sistema de coordenadas contenido en \mathbb{R}^3 . Como salida el algoritmo genera una imagen en \mathbb{R}^2 que representa fielmente la geometría recibida vista desde el punto de enfoque de la cámara. La implementación a realizar de este algoritmo, parte de cero en su totalidad y utilizando solamente funciones de librería de la STL (*Standard Template Library*) [1] de C++. Como punto de partida se cuenta con las transparencias de la asignatura proporcionadas por el profesorado de la misma.

En esta memoria de proyecto se recoge una explicación detallada de la metodología seguida por el equipo de trabajo, para realizar un correcto desarrollo del proyecto. A su vez contiene expoliaciones precisas de los principios matemáticos y las decisiones tomadas para su correcta implementación. Puesto que las salidas del algoritmo son resultado gráficos, se han llevado a la práctica numerosos experimentos para mostrar el potencial de la solución final.

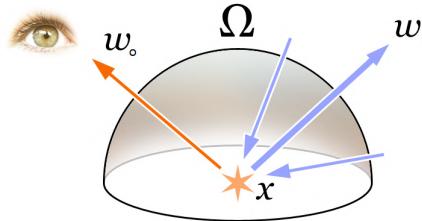
2. Objetivos

Los objetivos de este proyecto de la asignatura Informática Gráfica comprenden varios aspectos, entre los que destacan:

- Aprender los concepto básicos del campo de la *Informática Gráfica*, para el renderizado de imágenes dada un determinada geometría y un punto de vista.
- Comprender los principios matemáticos básicos sobre los que se desarrolla el campo de la *Informática Gráfica* y aprender a trabajar con distinto sistemas de coordenadas aplicando cambios de base y operaciones vectoriales.
- Entender a la perfección *La Ecuación de Render*, que define la interacción de la luz con los objetos en el renderizado de imágenes.
- Desarrollar desde cero una librería que englobe las operaciones y estructuras de datos, básicos para poder realizar operaciones geométricas en \mathbb{R}^3 .
- Conocer el funcionamiento del modelo cámara *Pinhole* y el trazado de rayos desde el espacio la cámara a la escena.
- Implementar una librería de geometrías básicas como esferas, planos infinitos, cuadrados, triángulos, mallas de triángulos...
- Adquirir la capacidad de diseñar imágenes con cierto grado de dificultad, constituidas por un determinado número de geometrías situadas en el espacio.
- Entender el comportamiento de la luz en la naturaleza y su interacción con los distintos materiales. Intentando lograr realizar una implementación precisa en bases a sus propiedades y conseguir dar efectos de profundidad, color blurring y iluminación global a las escenas diseñadas.
- Comprender los concepto sobre los que se basa la representación, de los colores en el espacio *RGB*.
- Desarrollar una implementación para lograr iluminación en las escenas mediante luces de área y puntuales, aplicando técnicas de muestreo como *Next Event Stimulation*.
- Estudiar las diversas *BRDFs* de los materiales lambertianos difusos, especulares perfecto, dielectricos...; Así como los métodos computacionales para reflejar en la medida de lo posible, estas características de manera real. Para posteriormente asignarlos a geometrías espaciales.
- Implementar operadores de *Tone Mapping* básicos para evitar tener que seleccionar la intensidad de las fuentes de luz de manera manual.
- Ver la aplicación de algoritmos probabilistas como la integración por *Monte Carlo* o *Ruleta Rusa* en *Informática Gráfica*. Con el objetivo de poder computar integrales no analíticas y seleccionar entre diferentes eventos, respectivamente.
- Incluir de manera opcional métodos de aceleración como paralelización y *BVH* (*Bounding Volume Hierarchies*), geometrías complejas, texturas, técnicas avanzadas de *Tone Mapping*, nuevos modelos de cámara, *BRDFs* complejas...

3. Ecuación de Render

La Ecuación de Render es la representación matemática de la interacción de la luz con los materiales. Esto la convierte en la base del renderizado de imágenes, y en el núcleo del algoritmo de Path Tracing. Dicha ecuación es la que se expone a continuación:



$$L_0(x, w_0) = \int_{\Omega} L_i(x, w_i) f_r(x, w_i, w_o) |n \cdot w_i| dw_i \quad (1)$$

La ecuación esta compuesta con tres términos, que definen la radiancia que llega a la cámara desde el punto x en la dirección w_o . El primer término $L_i(x, w_i)$ determina, la radiancia incidente recibida desde una dirección w_i . Esta puede ser emitida tanto por luces de área como lucen puntuales en el espacio. En apartados posteriores se realizará un análisis mas detallado de como se calcula la radiancia incidente, cuando un camino impacta con la superficie de un material. El segundo de los términos, $f_r(x, w_i, w_o)$, corresponde es la función de reflectancia del material sobre el que se estima la radiancia. Dicho término es la denominada Bidirectional Reflectance Distribution Function (BRDF) del material. Esta función modela como la luz es reflejada y distribuida a lo largo de la hemiesfera al impactar con una superficie opaca. Por último, es de gran importancia considerar el $|n \cdot w_i|$, dependiente del coseno que forman n (normal de la superficie) y w_i (rayo emitido desde x hasta la fuente de luz). Cuanto más perpendicular es la dirección de la luz incidente mayor es la energía recibida.

3.1. Calculo de la ecuación

Para lograr realizar una implementación eficiente del algoritmo de *Path Tracing*, es necesaria la resolución rápida de la integral de presente en la ecuación 1. En este apartado se va realizar una explicación teórica de la metodología seguida para lograr resolverla. Como se contará más adelante el algoritmo resultante queda parametrizado en función de una serie de parámetros. En apartados posteriores del documento se mostrarán pruebas variando dichos parámetros con el objetivo de mostrar los resultados de la implementación.

Puesto que el algoritmo de *Path Tracing* se basa en la creación de caminos desde la cámara, para determinar la radiancia de un píxel. Cada vez que uno de estos caminos emitidos desde el espacio local de la cámara al mundo choca con una superficie, hay que hacer el cálculo de la ecuación. El problema surge debido a que el cálculo de la integral no puede hacerse de manera analítica, lo implica que deben aplicarse técnicas numéricas que generen un resultado muy aproximado al real.

Los métodos de cuadratura como la aplicación de la Regla de Simpson o Integración por pasos, son caros en tiempo y no escalan bien cuando el número de dimensiones del espacio crece. Es por este último motivo que se ve la necesidad de utilizar un algoritmo más robusto y escalable, para la integración de la Ecuación de render. Dicho algoritmo no es otro que la integración mediante el método de Monte Carlo. Se trata de un algoritmo probabilista para el cálculo de integrales no analíticas, que escala bien al incrementar el número de muestras necesarias para integrar. En la siguiente ecuación puede verse la aproximación aplicada por Monte Carlo para integrar una función $f(x)$:

$$\int_a^b f(x)dx \approx \frac{b-a}{N} \sum_{i=1}^N f(x_i) \quad (2)$$

En la ecuación, N es el número de muestras aleatorias utilizadas y $x_i \in [a, b]$. Con esto ya se disponía de un procedimiento, para realizar el cómputo de la Ecuación de Render en la iteración de los rayos con las geometrías. No obstante, tal y como se explicó en las clases de teoría de la asignatura, el método de Monte Carlo es rápido pero tiene el inconveniente de que genera un ruido de alta frecuencia en el muestreo por la varianza que hay en las muestras. Es por esto que se optó por realizar un muestreo más avanzado, aplicando Muestreo por Importancia con el objetivo de disminuir esta varianza. Esto suponía buscar una función de probabilidad $p(x)$ que tuviera, en la medida de lo posible, una forma similar a la función original a integrar. Puede ser representada de la siguiente forma:

$$\int_a^b p(x)dx = 1 \quad (3)$$

donde $p(x) \geq 0 \quad \forall x \in [a, b]$, y $p(x) > 0$ si $f(x) > 0$.

La función $p(x)$ refleja de manera aproximada la distribución de densidad de la función $f(x)$ en el intervalo $[a, b]$. Lo que necesitamos es generar muestras aleatorias de acuerdo a la distribución $p(x)$. Para hacer esto, se aplica inversa de la función de distribución acumulada de $p(x)$ sobre números aleatorios uniformemente generados en el intervalo $[0, 1]$.

Si denotamos la inversa de la función de distribución acumulada como $c^{-1}(\xi_i)$ con $\xi_i \in [0, 1]$, podemos definir la siguiente aproximación de la integral original:

$$\int_a^b f(x)dx \approx \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{p(x_i)} \quad (4)$$

Finalmente, se pudo implementar el siguiente algoritmo para realizar el cálculo la Ecuación de Render.

Algoritmo 1 Muestreo por importancia

```

1: function IMPORTANCESAMPLING(N)
2:    $\xi \leftarrow \text{UNIFORMNUMBERS}(N)$ 
3:    $x \leftarrow c^{-1}(\xi)$ 
4:   return MONTECARLO( $x$ )

```

El parámetro N corresponde con el número de muestra por píxel, para el cálculo de su radiancia final en la imagen. Con esto se disponía de una herramienta útil para hacer el coloreado de la imagen. Ahora lo único que faltaba por decidir era que función $p(x_i)$ utilizar para el muestreo, pero esto se explicará en el apartado 5 correspondiente a los materiales implementado. Pues depende en gran medida de la *BRDF* del material.

3.2. Muestreo de lo materiales

En *Path Tracing* hay que generar un nuevo rayo de salida, para lograr encontrar un objeto emisor de luz. Tal y como se vio durante las clases de teoría, generar un número determinado de rayos secundarios como sucede en el algoritmo de Ray Tracing es demasiado costoso. Por esta razón *Path Tracing* hace uso del algoritmo de Ruleta Rusa para determinar si seguir con el camino o terminar. Al igual que el algoritmo de *Monte Carlo* la *Ruleta Rusa*, se trata de un algoritmo probabilista.

Como se explicado con anterioridad el termino de la *BRDF*, depende exclusivamente de las propiedades del material. En la naturaleza existen materiales como el plástico con parte difusa y parte especular, los definido por la *BRDF* de *Phong* con también parte difusa y especular o los dieléctricos con propiedad especular y refractante. Visto esto surge el problema de disponer de varias funciones a ser integradas. Por ejemplo para materiales con parte difusa y parte especular, hay que decidir si se integra en función de la parte difusa o de la parte especular ya que la contribución de energía sera distinta para cada propiedad. Adicionalmente para cada tipo de propiedad hay que aplicar un muestreo distinto, como se explica en el apartado 5. Es aquí donde es necesaria la aplicación del algoritmo de *Ruleta Rusa* para hacer la mejor aproximación posible de la integral. Dicho algoritmo consistía en asignar una probabilidad p a cada propiedad del material de manera que:

$$\sum_{i=0}^{n-1} p_i < 1 \quad (5)$$

donde n representa el número de propiedades del material (normalmente igual a 2 o 3).

Como ejemplo, si se trata de un material como el plástico, que tiene parte difusa y parte especular la probabilidad de difuso correspondería con el máximo valor de la tupla *RGB* que lo define. Por otro lado, a la parte especular se le podría asignar cualquier valor, siempre que se cumpla la ecuación 4. Esto implica que debe dejarse un determinado rango de probabilidad, para representar la absorción del rayo por el material. Esto último es un modo evitar que existan caminos que reboten de forma indefinida, en busca de fuentes de luz. Con la probabilidades asignadas, podemos representarlas de la siguiente manera:



De modo que para seleccionar un evento, se genera un numero aleatorio $\xi_R \in [0, 1]$ y se elije el evento correspondiente al intervalo de probabilidad al que pertenece el número

aleatorio. Con esto se consiguen rayos que exploran todas las propiedades del material y se logra una apariencia real del objeto geométrico.

3.3. Pasos del algoritmo de Path Tracing

Una vez descritos en las secciones anteriores, los principales conceptos sobre los que se sustenta el algoritmo de *Path Tracing*. En este apartado se procede a hacer una explicación de los pasos a seguir para la generación de imágenes con *Path Tracing*. Asumiendo que se quiere renderizar una imagen de tamaño $n \times m$, la cual se representa con una matriz de píxeles. El algoritmo implementado sigue los siguientes pasos.

- Para cada píxel de la imagen se generan un total de N rayos (direcciones en el espacio con el mismo origen que la cámara).
- Dada la dirección del rayo se calcula cuál es la geométrica más cercana con la que intersecta.
- Conforme al algoritmo de *Ruleta Rusa*, se selecciona una estrategia de muestreo del material o absorción.
- En caso de que se seleccione absorción el algoritmo retorna una tupla *RGB* que representa el color negro. De lo contrario se calcula la perdida de energía por la absorción del material y la dirección del rayo de salida en función de la estrategia de muestreo. En caso de existir luces puntuales en la escena, se utiliza *Next Event Estimation* para muestreárlas.
- El rayo continua hasta que se produce un evento de absorción, no intersecta con ninguna geometría o choca con una luz de área. En este último caso, se multiplica el acumulado de perdida de energía del camino generado por la cantidad de energía emitida por el emisor y se retorna dicho valor.
- Una vez calculadas todas las N radiancias de un píxel, se hace la media de todas y se le asigna dicho valor *RGB* al píxel. Esto proceso es repetido para todos los $n \times m$ píxeles.

4. Convergencia

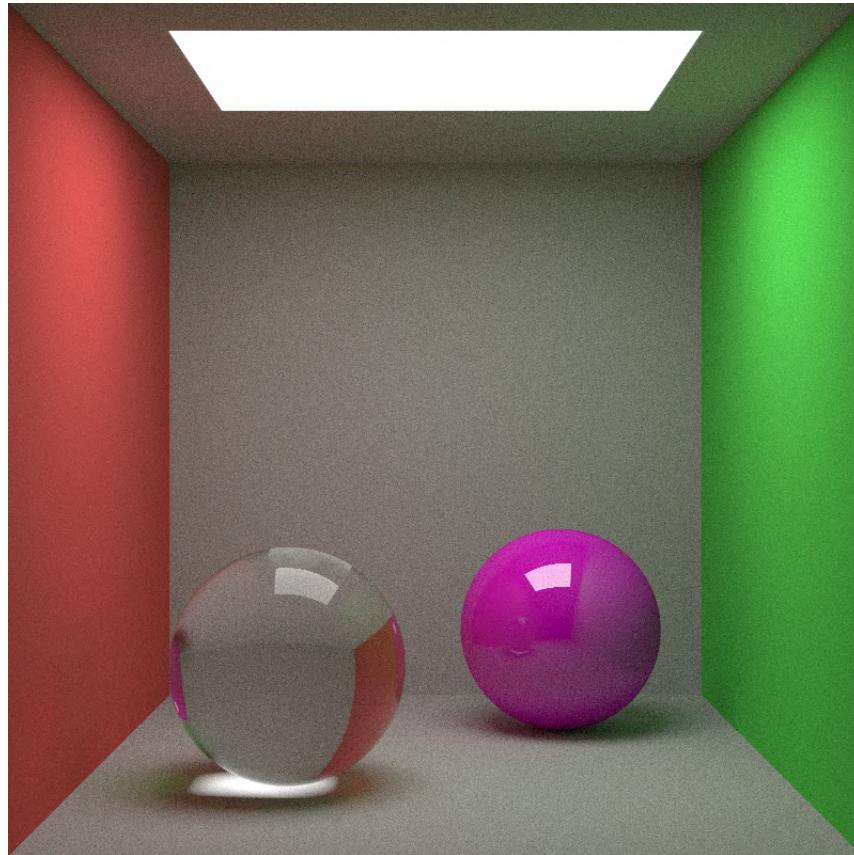
Aplicando los métodos descritos en el apartado anterior, el algoritmo de *Path Tracing* consigue un renderizado de imágenes en un aceptable. Otros algoritmos vistos en clase, como *Ray Tracing* tiene un coste demasiado elevado para ser utilizados en producción. Aplicando los algoritmos de *Monte Carlo* y *Ruleta Rusa*, *Path Tracing* consigue que la convergencia total del algoritmo, dependa solamente el número de *ppps* (*Paths Per Pixel*), el número píxeles y la cantidad de geometrías de la escena. De este modo puede determinarse una cota superior para el coste del algoritmo, con una complejidad asintótica temporal del siguiente orden:

$$O(N \times P \times G \times k), \quad (6)$$

donde N corresponde con el número de *ppps*, P con el número de píxeles de la imagen y G con el número de geometrías de la imagen. En lo que respecta al parámetro k debería representar el número de veces que rebota un rayo para generar un camino. No obstante

esta constante no puede ser calculada en una análisis a priori del algoritmo, sino que habría que realizar una cantidad considerables de experimentos. Esto se debe a que el número de rebotes que realiza un camino es determinado por la *Ruleta Rusa*, algoritmo que no es determinista. Es por esto que debe realizar se una estimación en función de las probabilidades de los materiales de la escena.

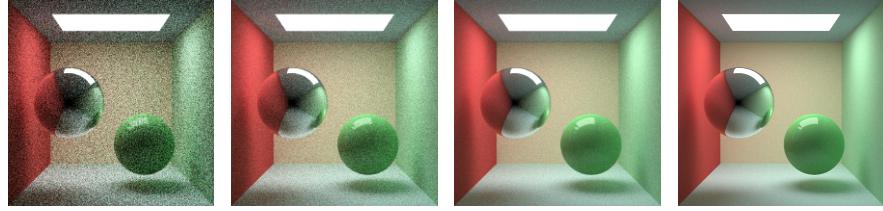
A continuación, puede verse un ejemplo de una imagen básica generada por el *Path Tracer* la cual está puesta como imagen por defecto.



En los siguientes apartados se va a proceder a mostrar ejemplos de imágenes renderizadas por el *Path Tracer*, comentando aspectos teóricos relacionando los materiales, *ppps* y la luz con el tiempo de convergencia.

4.1. Caminos por píxel

Como se ha comentado uno de los parámetros que afecta a la convergencia del algoritmo de *Path Tracing*, es el número de caminos lanzados por cada píxel (*paths per pixel, ppps*). Es por esto que a mayor número de *ppps* mayor tiempo de ejecución. No obstante esto reduce la presencia de ruido de *Monte Carlo* y el *aliasing* de la imagen. En las siguientes imágenes se muestra la misma escena renderizada con distinto número de *ppps*.



(a) Imagen generada con 16 paths per pixel.
(b) Imagen generada con 64 paths per pixel.
(c) Imagen generada con 256 paths per pixel.
(d) Imagen generada con 1024 paths per pixel.

Figura 1: Imágenes generadas variando el número de *ppps*.

Como se pude ver las imágenes anteriores la presencia de ruido es mucho mas alta en renders con menor número de *ppps*, no obstante al aumentar el número de *ppps* incrementa considerablemente el tiempo de cómputo del algoritmo. Esto puede verse reflejado en la siguiente gráfica donde en el eje *x* se varía el número de *ppps* y en el eje *y* se ve el tiempo de cómputo.

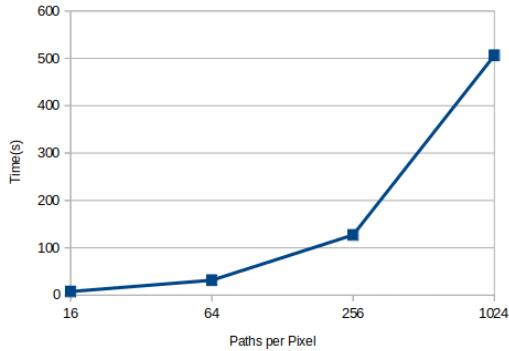


Figura 2: Comparación de tiempos variando el número de *ppps*

5. Materiales

Como se ha explicado en el apartado 3 del documento, la interacción de la luz con un determinado material viene dada por la *BRDF* del mismo. En la naturaleza existen una gran cantidad de *BRDFs* diferentes, que describen los diversos materiales. Para este proyecto se han implementados un conjunto reducido de *BRDFs* para dar apariencia a las geometrías. El los siguientes apartados se procede a hacer una explicación detallada, de las distintas *BRDFs* implementadas. En el análisis de tiempo del apartado anterior puede verse que un incremento de los *ppps* tiene un gran impacto en el coste de ejecución. Considerando los diferentes materiales implementados, esto no afectan al tiempo total de ejecución, sin embargo es de importancia destacar que el coste es mayor cuando la luz es generada por luces puntuales. Esto se debe a que el *Next Event Estimation* es computado en materiales difusos y el tiempo aumenta. En la siguiente figura puede verse

una comparación del tiempo de generación de las imágenes de la Figura 1 con luz de área y puntual.

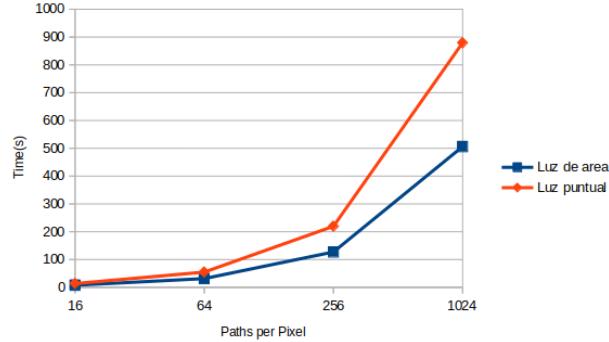


Figura 3: Comparación de tiempos variando el número de *ppps* con luz de area y puntual.

En las siguientes fotografías se muestran las escenas de la Figura 1 pero con iluminación por luz puntual. En la imágenes de las Figuras de la 5 a la 8 se puede ver como es mucho mas caro computar imágenes con luz puntual.

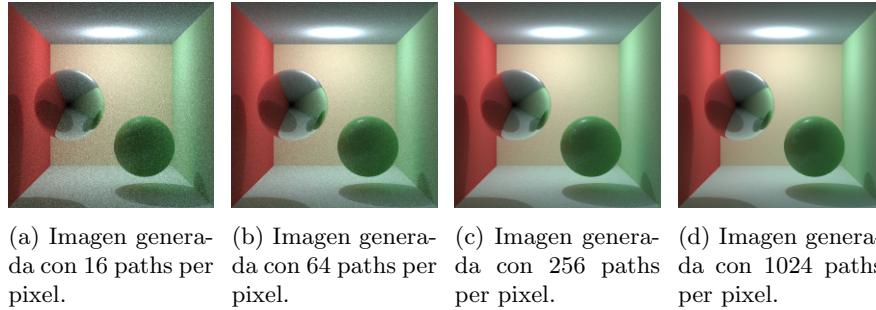


Figura 4: Imagenes generadas variando el número de paths per pixel con iluminación puntual.

5.1. Lambertiano Difuso

Los materiales perfectamente difusos son aquellos cuyo iluminado, depende únicamente del término del coseno de la *Ecuación de Render*. Dicha iluminación es totalmente independiente del punto de vista de observación, por lo el único elemento visual apreciable sobre estos son las sombras. La siguiente *BRDF*.

$$f_r(x, w_i, w_o) = \frac{k_d}{\pi} \quad (7)$$

Donde k_d es una constante que representa el color del material, en este caso puede ser un tupla de valores *RGB*. Por otro la es importante considerar la dirección del rayo de salida,

que genera el impacto del material. Dicho rayo porta el resto de la energía que no ha sido absorbida por el material. En materiales difusos la energía saliente se distribuye de forma uniforme, a lo largo de la semiesfera. Es por esto que hubo que aplicar una estrategia de muestreo, que representase fielmente esta distribución. Con el objetivo de que cuando la *Ruleta Rusa* seleccionase eventos difusos, dicha propiedad fuese explorada correctamente. Es por esta razón que decidió aplicarse un *Muestreo Uniforme del Coseno*. A continuación pueden verse algunos ejemplos de imágenes con materiales difusos, aplicando *Muestreo Uniforme del Coseno* con luz puntual y de área.

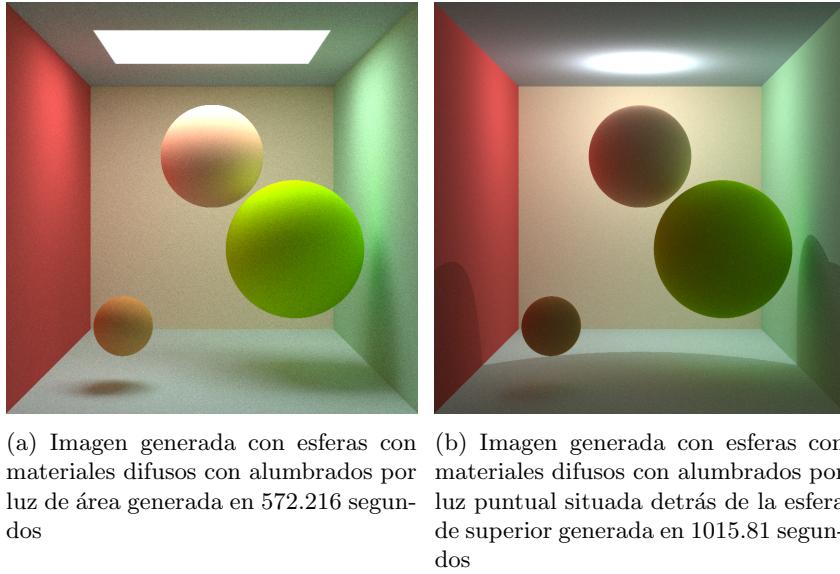


Figura 5: Imágenes con materiales difusos generadas con 1024 paths per pixel y distintos tipos de iluminación.

5.2. Especular perfecto

Aquellos materiales de la naturaleza que presentan un comportamiento, especular perfecto con la luz. Son los que no realizan ningún proceso de absorción de energía. Esto quiere decir que cuando un rayo de luz choca con estos, toda la energía que porta es conservada transportada por el rayo de salida reflejado. Es por esto que dicho fenómeno es representado por la siguiente *BRDF*.

$$f_r(x, w_i, w_o) = \frac{\delta_{w_r}(w_i)}{w_i \cdot n} \quad (8)$$

donde w_i representa el rayo incidente en el material, w_r el rayo reflejado y n , la normal de la superficie.

En lo que respecta a la dirección del rayo de salida, esta viene dada por la *Ley de Reflexión Total* la cual puede verse a continuación.

$$w_r = w_i - 2 \cdot n(w_i \cdot n) \quad (9)$$

donde w_i , w_r y n hacen referencia los mismo parámetros que los presentes en la ecuación 10.

Puesto que todo rayo que impacta en este tipo de materiales, es reflejado a partir de la ecuación 10. No es necesario aplicar ninguna estrategia de muestreo compleja, como ocurría en los materiales Lambertianos. A continuación pueden verse ejemplos de materiales perfectamente especulares.

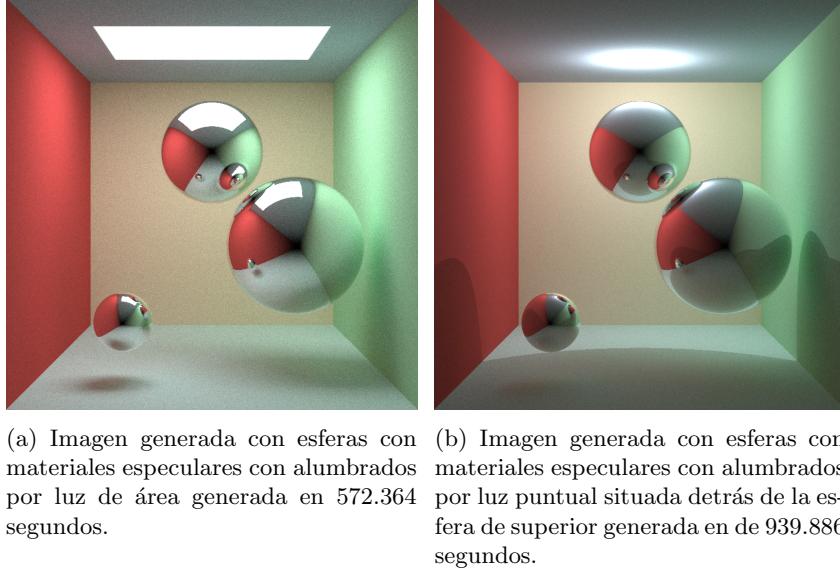


Figura 6: Imágenes con materiales especulares generadas con 1024 paths per pixel y distintos tipos de iluminación.

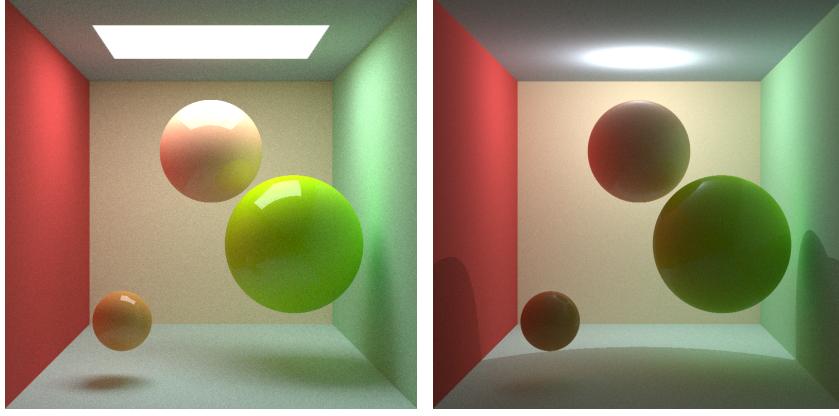
5.3. Plástico

En los dos apartados anteriores se han mostrado las *BRDFs* que simulan las propiedades de difusas y especulares de los materiales. Sin embargo en la naturaleza no todo es completamente difuso o especular. Existen materiales poseen varias propiedades, de manera que el aspecto que dan también depende del punto de vista. Un ejemplo de este tipo de materiales es el plástico, el cual posee una parte difusa y otra especular. A continuación puede verse la *BRDF*, que modela este tipo de comportamiento con la luz.

$$f_r(x, w_i, w_o) = \frac{k_d}{\pi} + \frac{\delta_{w_r}(w_i)}{w_i \cdot n} \quad (10)$$

Como se puede apreciar, se trata de una combinación lineal de las *BRDFs* anteriores. Esto permite diseñar geometrías mucho mas realistas que las anteriormente vistas. Puesto que en este tipo de materiales hay presentes dos propiedades, es a la hora de muestreárlas es el algoritmo de *Ruleta Rusa* el que decide que propiedad se muestrea en la intersección. Para esto se la asigna a la propiedad difusa $p_d = \max(kd)$, es decir el valor máximo de la tupla *RGB*. Por otro lado a la parte difusa se le debe asignar una probabilidad que cumpla

con la ecuación 5. Con esto puede seleccionarse por *Ruleta Rusa* la mejora estrategia de muestreo, de para la propiedad. En las siguientes imágenes puede apreciarse ejemplos de materiales plásticos.



(a) Imagen generada con esferas con materiales plásticos con alumbrados por luz de área generada en 551.948 segundos.
(b) Imagen generada con esferas con materiales plásticos con alumbrados por luz puntual situada detrás de la esfera de superior generada en 996.912 segundos.

Figura 7: Imágenes con materiales plásticos generadas con 1024 *paths per pixel* y distintos tipos de iluminación.

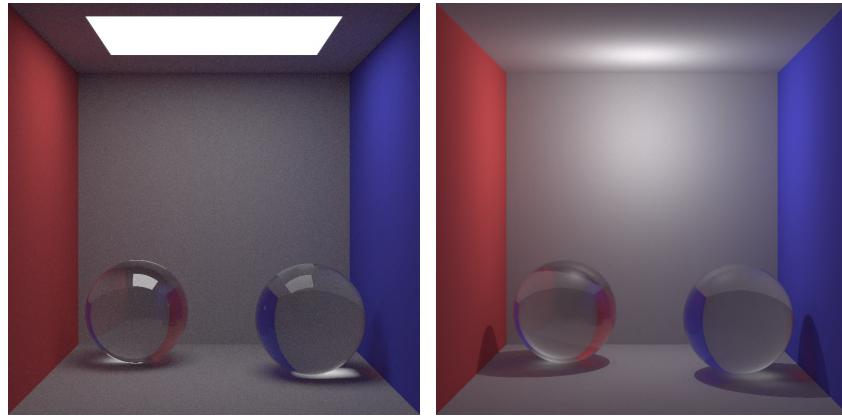
5.4. Dieléctricos

Los materiales dieléctricos son aquellos que presentan una baja conductividad eléctrica. Aquellos objetos de este tipo de materiales, no absorben energía de igual modo que ocurría con los materiales especulares. Esto solo poseen propiedades de reflexión y refracción para los rayos incidentes. Dichas propiedades vienen definidas por la siguiente *BRDF*.

$$f_r(x, w_i, w_o) = k_s(w_i) \frac{\delta_{w_r}(w_i)}{w_i \cdot n} + k_t(w_i) \frac{\delta_{w_t}(w_i)}{w_i \cdot n} \quad (11)$$

donde k_s corresponde con el coeficiente especular del material y k_t con el coeficiente de refracción y n la normal de la superficie.

Como puede apreciarse ambos coeficientes están parametrizados en función del rayo incidente w_i . El cálculo de dichos coeficientes ha sido implementado mediante las *Ecuaciones de Fresnel* [2]. Puesto que estos materiales cuentan con propiedades especulares y de refracción, estas son muestreadas mediante *Ruleta Rusa*, aplicando la ecuación 10 para la parte especular. Por otro lado, la propiedad de refracción es calculada mediante la *Ley de Snell* [3]. Es de importancia destacar que para el renderizado de este tipo de materiales hay que asignarle al entorno y a los propios materiales un índice de refracción. En las siguientes figuras puede verse un varios ejemplo, modificando dichos índices.



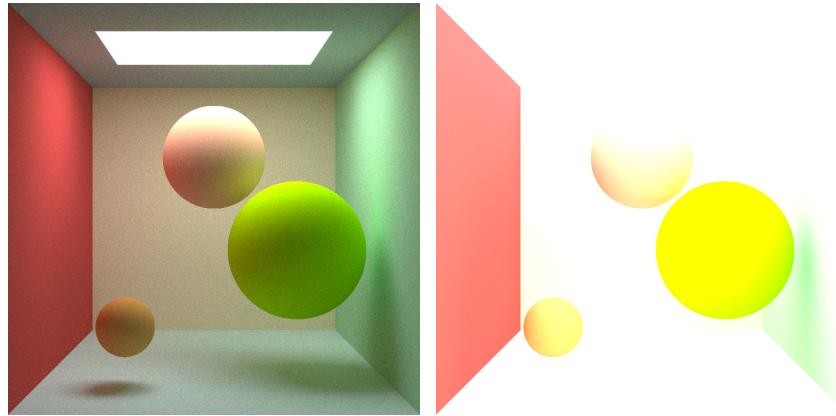
(a) Imagen generada con esferas con materiales dieléctricos con alumbrados por luz de área generada en 563.346 segundos.

(b) Imagen generada con esferas con materiales dieléctricos con alumbrados por luz puntual generada en 959.34 segundos.

Figura 8: Imágenes con materiales dielectricos generadas con 1024 *paths per pixel* y distintos tipos de iluminación. El índice de refracción del medio es 1,000293 (aire), el de la bola de la izquierda es 2,42 (diamante) y el de la izquierda es 1,5 (cristal).

6. Fuentes de luz

La luz que puede presentarse en las escenas renderizadas por el *Path Tracer*, puede ser emitida por luces puntuales o luces de área. Para la implementación de estas últimas no se ha desarrollado ninguna técnica avanzada. Pues estas pueden ser modeladas tanto como plano infinitos como planos finitos. Como puede apreciarse en las siguientes figuras donde aparece la misma escena renderizada con un plano finito y uno infinito, el aumento de la superficie de la luz implica una mayor presencia de luz en la imagen. También puede notarse como el tiempo es muy superior al renderizar imágenes iluminadas por cuadrados que con planos infinitos, pues en este último caso es probable que el rayo impacte con la superficie emisora.



(a) Imagen generada con una iluminación por plano finito en un tiempo de 572.216 segundos

(b) Imagen generada con una iluminación por plano en el techo infinito en un tiempo de 365.698 segundos.

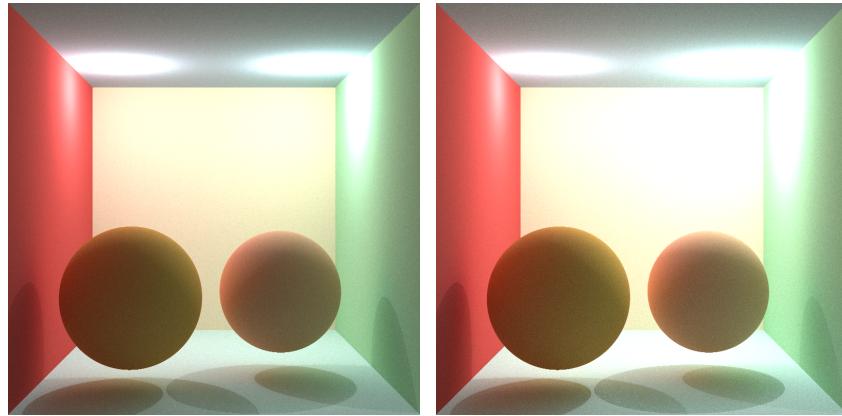
Figura 9: Imágenes generadas con la presencia de varias luces puntuales en las escenas.

Puesto que la probabilidad de que los caminos de *Path Tracing*, choquen con una directiva geométrica que emite luz es alta. No se ha considerado necesaria utilizar técnica complejas como simularlas como un conjunto grande de luces puntuales. Simplemente los caminos rebotan por la escena hasta alcanzar una fuente de luz o ser absorbidos por un material. En caso de impactar con un emisor, se multiplica la energía de emisión por la del coeficiente de pérdida del camino y se retorna la radiancia resultante. En la imágenes de la Figura 5 a la 8 se muestran diversas imágenes con iluminación puntual y de área.

6.1. Next Event Estimation

Como se a explicado en la sección anterior, las luces de área tienen una alta probabilidad de ser encontradas por los caminos trazados. Por desgracia no ocurre lo mismo para las luces puntuales, pues la probabilidad de que un camino impacte con un punto en el espacio es 0. Esto obliga a aplicar otras técnicas mas avanzadas para tener en cuenta la continuación de luces puntuales en materiales que absorben energía. Para esto se ha decidido implementar un muestreo mediante *Next Event Estimation*. Esta técnica de muestreo consiste en trazar rayos de sombra hacia luces puntuales, cuando un rayo choca con un material que no es delta (tiene propiedad difusa) para saber si contribuyen a la radiancia.

En las siguientes imágenes pueden apreciarse ejemplos de escenas iluminadas por varias luces puntuales.



(a) Imagen generada dos luces puntuales de igual potencia.

(b) Imagen generada con dos luces puntuales donde la luz de la derecha tiene mas potencia que la luz de la izquierda.

Figura 10: Imágenes generadas con la presencia de varias luces puntuales en las escenas.

Los tiempos de renderizado las escenas que tienen luces puntuales son superiores en comparación con las que tienen luces de área. Esto se debe a que realizar el cálculo de la iluminación con luces puntuales es mas caro debido al sobre coste de hacer *Next Event Estimation* sobre materiales difusos, pero también tienen menos ruido, ya que el muestreo de la luz tiene menos varianza.

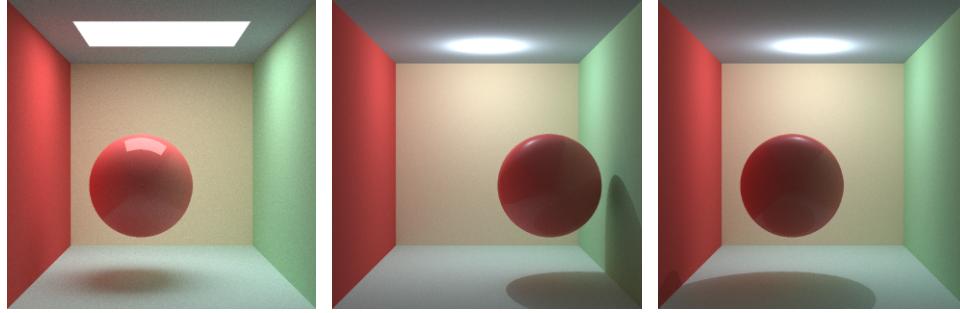
7. Iluminación Global

Una correcta implementación de la propiedades de la luz, es vital en el renderizado de imágenes para conseguir que estas sean realistas. El algoritmo de *Path Tracing* entregado, permite el renderizado de imágenes, con iluminación global. En otras palabras permite la aparición de efectos realistas en la escenas, como por ejemplo: sombras, *Color Bleeding* o cásticas. Es por esto que en este apartado se va a proceder a mostrar de forma detallada dichos efectos, causado por los elementos explicados en el apartado anterior.

7.1. Sombras

La generación de sombras en las imágenes, resulta sencillo en el algoritmo de *Path Tracing*, al ser generadas por la oclusión de la luz entre geometrías. Puesto que se dispone de distintos tipos de fuentes de luz, es decir puntuales y de área, pueden aparecer distintos tipos de sombras en las escenas. Podemos clasificar las sombras en *Hard Shadows* y *Soft Shadows*, en función de lo definidas que estén. La primeras surgen cuando la iluminación es producida por luz puntual, y se realiza el cálculo de *Next Event Estimation*. Esto produce que las regiones de los objetos que son iluminados por la luz directa, sea mucho menos oscuros que los opacados por otras geometrías. Por otro lado si la eliminación de la escena corre a cargo de luces de área, las sombras tienden a no estar tan bien definidas.

Esto no se debe solamente a la falta del computo de *Next Event Estimation* sino que al ser superficies mas grandes de luz, es mas difícil que sea opacada de forma tan fácil como con luces puntuales. En la siguientes imágenes pueden apreciarse diversos ejemplos de sombras con luces de área finitas e infinitas y puntuales.



(a) Escena renderizada con luz de área.

(b) Escena renderizada con luz puntual y sombra entre suelo y pared.

(c) Escena renderizada con luz puntual y sombra entre suelo y pared.

Figura 11: Imágenes generadas con luz puntual y luz de área.

Con el objetivo de que el lector pueda ver ejemplo de imágenes, con varias luces puntuales. En las siguientes figuras pueden verse ejemplos de escenas con varias luces puntuales.

7.2. Color Bleeding

Este fenómeno de la naturaleza se produce cuando un objeto es coloreado por luz recibida desde otro objeto cercano. Esto suele ocurrir cuando hay presentes materiales con propiedades difusas en la escena. A continuación se muestra un conjunto de imágenes en las que se puede apreciar la presencia de este fenómeno.

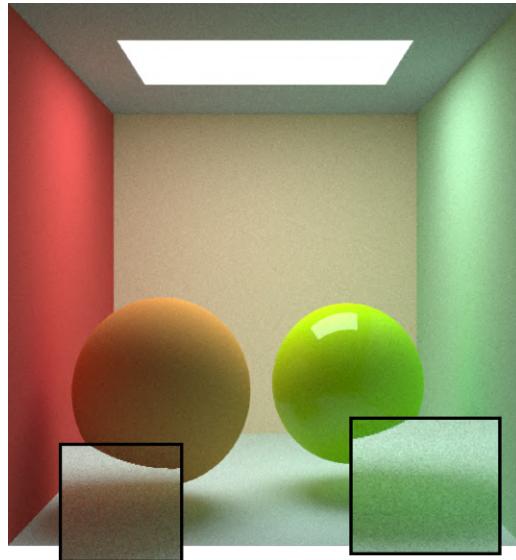


Figura 12: Esferas difusa y plastica que producen color bleeding sobre el suelo.

7.3. Cáusticas

De igual modo que el *Color Bleeding*, este tipo de fenómenos en las escenas son producidos por la interacción de la luz sobre los materiales. En este caso se produce cuando la luz atraviesa materiales dieléctricos y este impacta con una fuente de luz. Por desgracia como se ha explicado en el apartado 6.1, es matemáticamente imposible que un rayo choque con una luz puntual. Esta una de las principales limitaciones del algoritmo de *Path Tracing*, por lo que para poder hacer un mejor procesado de este fenómeno sería necesario añadir ciertas extensiones al algoritmo o utilizar otros vistos en clase como *Photon Mapping*. Lo mas parecido a cáusticas que puede lograrse en *Path Tracing*, son los efectos producidos por luces de área. Simular cáusticas más complejas es muy complicado con este algoritmo.

8. Tone Mapping

Todas las imágenes mostradas en apartados previos del documento, han sido generadas en *Alto Rango Dinámico* más comúnmente conocido como *HDR* (*High Dynamic Range*) por sus siglas en inglés con 0-10000000 como rango de luminancia. Dado que el tipo de formato utilizado para guardar las imágenes en disco era .ppm, este solo podía trabajar en *LDR* (*Low Dynamic Range*) con 0-255 como rango de luminancia. Esto suponía un problema a la hora de diseñar imágenes, pues implicaba tener que perder tiempo en calcular la energía de las fuentes de luz para conseguir buena iluminación. Por esta razón se decidió implementar filtros de *Tone Mapping*, para transformar imágenes en *HDR* a *LDR*. Puesto que la luminancia de los píxeles de la imagen ha sido representada mediante tuplas *RGB*, se trabaja directamente sobre estos tres canales, y no se hace ninguna transformación a otro espacio de color con canal de luminancia. A continuación se muestran todos los operadores de *Tone Mapping* implementados. Donde L_i representa la luminancia del pixel,

L'_i la luminancia del píxel tras aplicar el *Tone Mapper* y L_{max} el mayor valor de la imagen.

8.1. Clamping

Este filtro pone a 255 cualquier valor que supera dicho valor, de modo que elimina en su totalidad picos altos de energía.

$$L'_i = \min(L_i, 255) \quad (12)$$

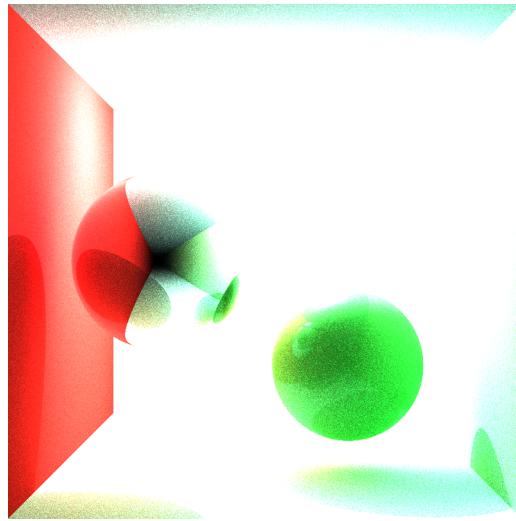


Figura 13: Imagen habiendo aplicada el operador de tone mapping clamp.

8.2. Equalization

La ecualización de la imagen consiste en hacer una normalización de los valores por el máximo.

$$L'_i = \frac{L_i}{L_{max}} \quad (13)$$



Figura 14: Imagen habiendo aplicado el operador de tone mapping equalization.

8.3. Equalize and Clamping

Es *Tone Mapper* no es otra cosa que aplicar primero un clamp y posteriormente una ecualización, siendo pues una combinación de los dos anteriores.

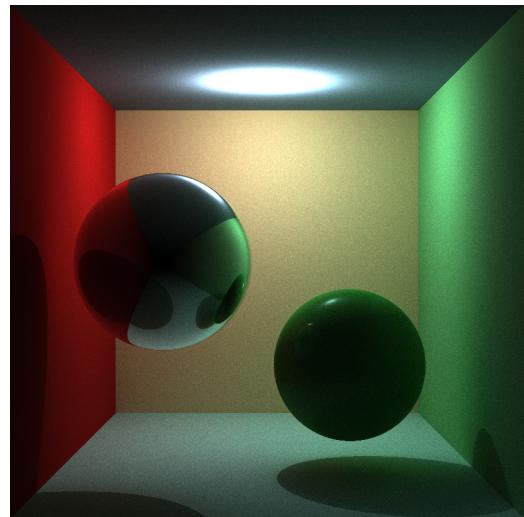


Figura 15: Imagen habiendo aplicada el operador de tone mapping clamp y equalization con un clampleo al 0.78 del valor de entrada máximo.

8.4. Gamma curve

Consiste en aplicar una curva gama a todos los píxeles de la imagen.

$$L'_i = L_i^{\frac{1}{\gamma}} \quad (14)$$

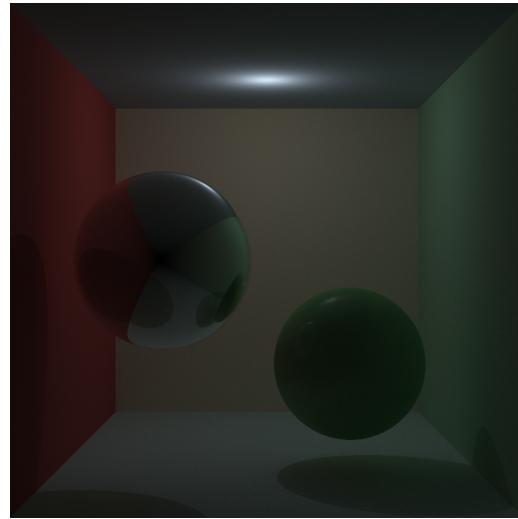


Figura 16: Imagen habiendo aplicada el operador de tone mapping gamma curve con $\gamma = 2,2$.

8.5. Clamp and gamma curve

Finalmente este último se trata de una combinación del *Tone Mapper* anterior y el especificado en el sección 7.1. Principalmente consiste en aplicar primero una operación de clamp y posteriormente una curva gamma.

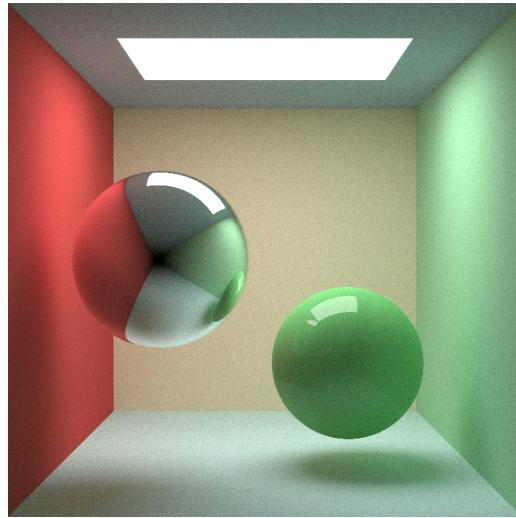


Figura 17: Imagen habiendo aplicado el operador de tone mapping gamma curve con $\gamma = 2,2$ con un clampleado al 0.86 del valor de entrada máximo.

9. Opcionales

De manera optativa se decidió hacer la implementación de extensiones al algoritmo descrito en los apartados previos. Dichas mejoras abarcan desde la inclusión de la reducción el tiempo de renderizado, como paralelización o *BVHs* (*Bounding Volumen Hierarchies*), hasta nuevos materiales, texturas y filtros de *Tone Mapping*. En este apartado se va a realizar un explicación detallada de cada una.

9.1. Paralelización

Hacer una ejecución secuencial del algoritmo de *Path Tracing*, puede suponer un alto coste temporal. No obstante puesto que el cálculo de la radiancia de un píxel, es independiente del resto de píxeles de la imagen. Resulta posible hacer una ejecución paralela del algoritmo, repartiendo la carga de trabajo entre múltiples hilos. Viendo esto se optó por implementar una ejecución paralela del algoritmo. Para esto fue necesario el uso de la librería de hilos de la *STL* y una implementación de una cola concurrente de elementos. Para esto último se decidió utilizar la cola implementada en una de las prácticas de la asignatura de *Programación de Sistemas Concurrente y Distribuidos*. Dicho esto, el problema puede verse como un problema de productores y consumidores. Los consumidores parten la imagen a renderizar en N trozos y la van colocando en la cola, mientras que los consumidores van cogiendo dichas regiones y realizan el renderizado. La razón por la que se optó por hacer la particiones en rectángulos fue por para reducir los fallos de cache a bajo nivel. En la siguiente imagen puede verse de forma gráfica el proceso de trabajo.

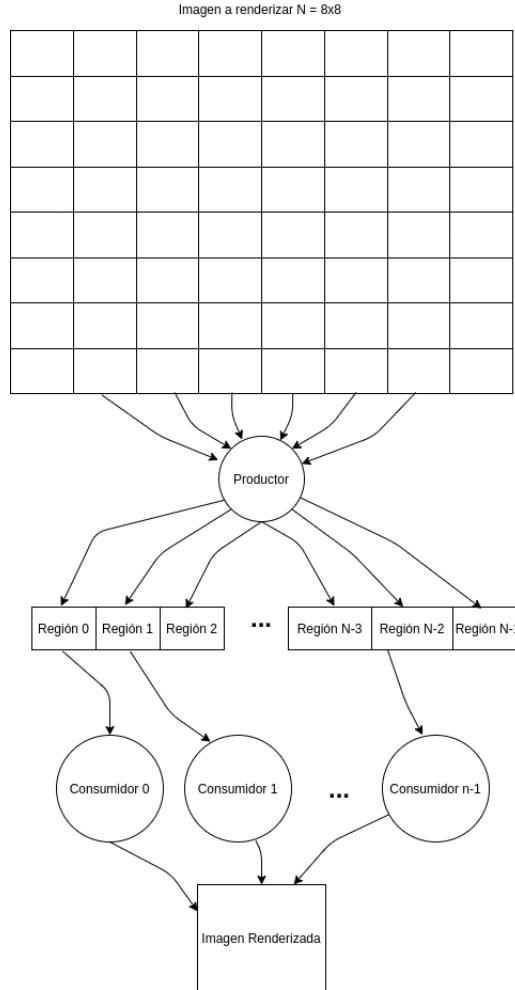


Figura 18: Imagen que refleja el proceso de paralelización de la imagen

Como puede verse reflejado en la imagen solamente se utiliza un único productor. Esto se debe a que la carga de trabajo que este recibe, es muy inferior a la de los consumidores al solo tener que dividir la imagen. Por este motivo a nivel de implementación se optó por no gastar un hilo del hardware para el productor, sino que tal tarea correría a cargo del programa principal. Esto implica que se disponía de tantos hilos, como núcleos tenía la máquina de ejecución. Por último es de importancia destacar que hubo que añadir a la cola concurrente un contador de tareas realizadas, para evitar un *dead lock* al final de la ejecución. A continuación, se muestra una pequeña tabla en la que se compara la ejecución secuencial, con la paralela mostrada en la Figura 2 variando en número de *pps*.

ppps	Tiempo con 8 hilos	Tiempo Secuencial
16	7.7323	19.2509
64	31.5305	72.3457
256	127.15	282.219
1024	506.235	1104.08

Cuadro 1: Comparación de tiempos entre ejecución paralela y secuencial

9.2. Bounding Volumen Hierarchies

Una Bounding Volume Hierarchy (BVH) es una estructura de aceleración que es de gran utilidad para disminuir el tiempo en el que se tarda en encontrar la primera figura con la que intersecta un rayo.

Todas las primitivas geométricas tienen asociada una bounding box que las contiene. La idea es agrupar en una estructura con forma de árbol todas las figuras de la escena de forma que en los nodos sean bounding boxes que envuelvan todas las primitivas descendientes de ese nodo, y en los nodos hoja se encuentre una lista con las primitivas que envuelven.

En la Figura 19 se encuentra un ejemplo de una BVH, usando rectángulos como bounding boxes.

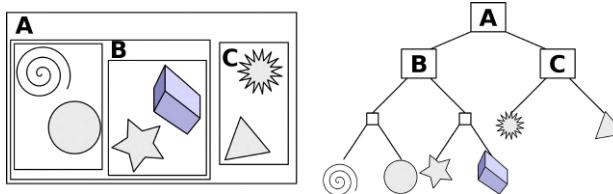


Figura 19: Ejemplo de BVH. https://en.wikipedia.org/wiki/Bounding_volume_hierarchy

A la hora de elegir el tipo de bounding box a usar, se busca que tenga una forma muy sencilla, para que su almacenamiento el cálculo de la intersección con ellas rápido, y por otro lado se busca que puedan ajustarse lo más posible a las figuras geométricas. Se ha decidido usar Axis-Aligned Bounding Boxes, una de las más usadas para construir BVH. Como están alineadas con los ejes, se pueden definir a partir de dos puntos p_{min} y p_{max} , siempre cumpliendo el invariante $p_{min}[x] \leq p_{max}[y]$.

El algoritmo de subdivisión para construir la jerarquía es bastante sencillo, no se ha implementado ninguna heurística avanzada. Este algoritmo consiste en:

- Calcular la AABB que envuelve a un conjunto de figuras.
- Escoger la dimensión más grande de la AABB.

- Subdividir en conjunto de las figuras por la mediana a lo largo de la dimensión escogida, comprobando el valor de uno de los dos puntos de las AABB de cada figura en esa dimensión.
- Se crean dos AABB, una que contenga todas las figuras que hayan quedado a la izquierda de la media, y otro con las que han quedado a la derecha. El objeto que se encuentra en la mediana, y por el cual se ha realizado la subdivisión, se añade en las dos AABB.

Para construir la BVH se ha seguido una estrategia *top-down*, se comienza la AABB raíz que envuelve a todas las primitivas de la escena, y se va subdividiendo recursivamente hasta que se llega a tener AABB que envuelven un número determinado de elementos. En este número hay un compromiso, si es pequeño, las BVH serán muy profundas, pero si es grande habrá que hacer más trabajo en el nodo hoja para comprobar si se intersecta con alguna de las figuras que envuelve. Se ha decidido fijarlo en 10, ya que es un número que experimentalmente daba buenos resultados tanto para escenas sencillas como complicadas.

Una vez construida la BVH con las primitivas de las escenas, hace falta que definir la estrategia para encontrar la primera primitiva con la que intersecta un rayo. Se ha utilizado la siguiente:

- Primero se intenta intersectar con la raíz.
- Si se intersecta con la raíz, se intenta intersectar con los hijos, si no el rayo no intersecta con ninguna figura y se puede terminar el recorrido.
- Si el rayo intersecta con los dos, se exploran las dos ramas, aunque la intersección con alguna AABB produzca un valor t aunque sea negativo.
- Si sólo se intersecta con un hijo, se sigue por esa rama.
- Una vez que se ha llegado a un nodo hoja, se intenta intersectar con todas las primitivas que contiene y se devuelve la más cercana (ahora si se descartan las que producen valores t menores que 0).
- Una vez que se ha explorado el árbol recursivamente, podando los nodos como se ha explicado, se devuelve la figura que ha producido el t mayor que 0 más pequeño.

Usar BVH consigue reducir la complejidad para encontrar la primera figura con la que intersecta un rayo de lineal a logarítmico, en el número de primitivas de la escena, ya que no hace falta explorar los nodos hijos si no se intersecta con los nodos padres.

Aunque existen estrategias más especializadas y con mejor rendimiento que la utilizada, la mejora obtenida es enormemente significativa, y ha permitido renderizar imágenes como la siguiente en un tiempo razonable.



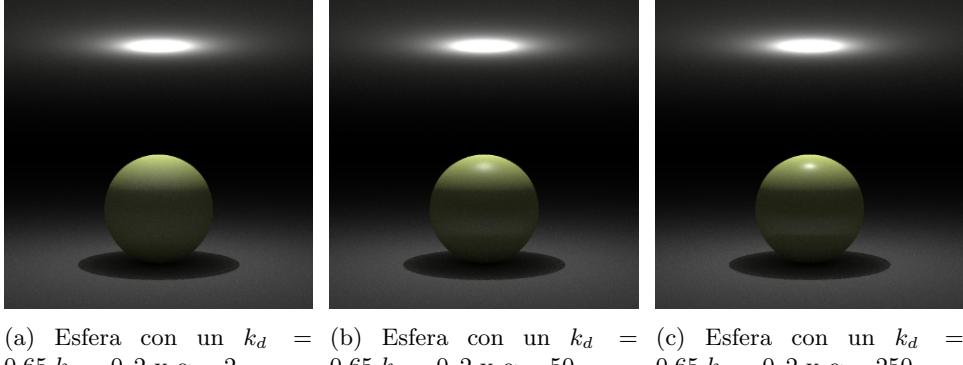
Figura 20: Escena con tres conejos renderizada con 1240 *ppp* en 89 minutos. 208363 primitivas en total.

9.3. BRDF de Phong

Como se ha explicado en el apartado 5.2 el modelado de materiales se hace en base a la *BRDF* de los mismos. En dicho apartado se han presentado ejemplos de materiales difusos, especulares perfectos, dieléctricos y plásticos. Sin embargo existe un amplio número de *BRDFs*, es por esto que de manera opcional el equipo decidió implementar de el modelo de *Phong*. Dicho modelo permite representar materiales denominados como *Glossy*. A continuación se muestra la *BRDF* que permite definir esto materiales.

$$f_r(x, w_i, w_o) = \frac{k_d}{\pi} + k_s \frac{\alpha + 2}{2 \cdot \pi} |w_o \cdot w_r|^\alpha \quad (15)$$

En lo que respecta a este tipo de materiales, para la parte difusa se le aplica un *Muestreo por el Coseno* cuando se selecciona por *Ruleta Rusa*. En lo que respecta a la parte espeacular, esta no es modelada con la ley de *Reflexión total* vista en la ecuación 10. Es por esto que requiere un tipo de muestreo específico en base al rayo reflejado, pues es en dicha dirección por donde los materiales reflejan la mayor parte de la energía. En lo que respecta al parámetro α , este sirve para indicar el brillo del material (*Shininess*). A continuación, pueden verse ciertos ejemplos ciertos ejemplos de materiales variando el parámetro α .



(a) Esfera con un $k_d = 0,65, k_s = 0,2$ y $\alpha = 2$ (b) Esfera con un $k_d = 0,65, k_s = 0,2$ y $\alpha = 50$ (c) Esfera con un $k_d = 0,65, k_s = 0,2$ y $\alpha = 250$

Figura 21: Esfera con material modelado con el modelo de phong.

9.4. Tone Mapper de Reinhard 05

En el apartado 7 del documento, se explica el conjunto de *Tone Mappers* implementados para realizar la conversión desde *HDR* a *LDR*. Entre dichos filtro, aplicar una curva gamma da buenos resultados. Sin embargo este campo es muy estudiado en la *Informática Gráfica*, y es por esto que surgen nuevas implementaciones que generan resultado de mejor calidad. Por estos motivos se decidió hacer una implementación del *Tone Mapper de Reinhard 05* [4] donde el cálculo de la radiancia resultante viene expresada por las siguientes ecuaciones.

$$\bar{L}_w = \frac{1}{N} \exp \left(\sum_i \log(\delta + L_i) \right) \quad (16)$$

$$L'_i = \frac{a}{\bar{L}_w} L_i \quad (17)$$

$$L''_i = \frac{L'_i \left(1 + \frac{L'_i}{L_{max}^2} \right)}{1 + L'_w} \quad (18)$$

Donde la radiancia resultante corresponde con L''_i y nuevamente L_{max}^2 corresponde con el máximo valor de la imagen. Desgraciadamente no se ha logrado hacer una implementación satisfactoria de este operador debido a fallos de desbordamiento con los *floats*. En las siguientes figuras se muestran ejemplos fallidos obtenidos sobre la misma imagen presentada en el apartado 6.

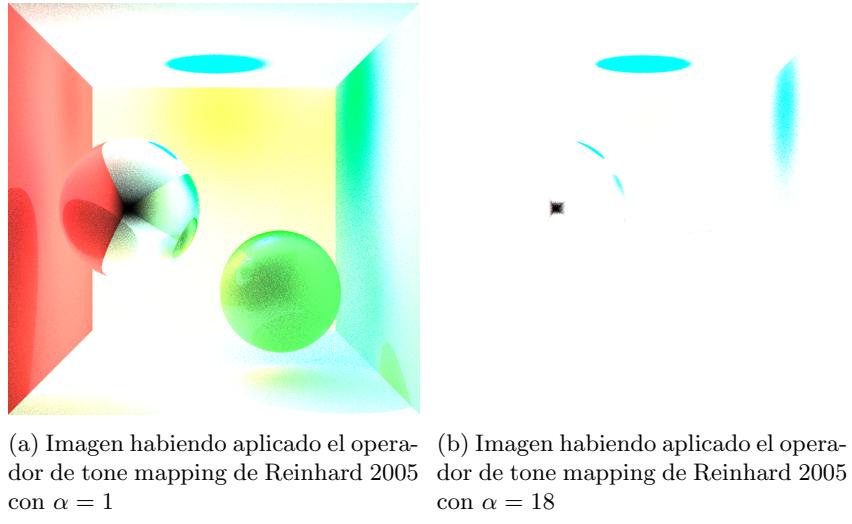


Figura 22: Imágenes tonemapeadas con el operador de Reinhard 05.

9.5. Primitivas adicionales

Con el objetivo de poder crear escenas mas complejas, se decidió añadir triángulos y cuadriláteros. Gracias a estas primitivas se pudieron implementar de luces de área y incluir objetos complejos como por ejemplo mallas de triángulos o luces de área.

9.5.1. Triángulos

La intersección rayo-triángulo se ha implementado mediante el algoritmo Möller-Trumbore [5]. Este es un algoritmo muy rápido, ya que no necesita calcular la ecuación del plano en el que se encuentra el triángulo.

Para cada triángulo, además de sus tres vértices (v_1 , v_2 y v_3), es posible almacenar la normal asociada a cada vértice (n_1 , n_2 y n_3). En este triángulos se consideran de tipo *smooth* y cuando se calcula la normal en un punto del triángulo se interpola a partir de las normales de los vértices. Esto permite renderizar mallas de triángulos de forma suave. Para obtener la normal interpolada en un punto, primero se calculan las coordenadas baricéntricas (α, β, γ) de ese punto dentro del triángulo, la normal resultante es:

$$\frac{\alpha n_1 + \beta n_2 + \gamma n_3}{\|\alpha n_1 + \beta n_2 + \gamma n_3\|} \quad (19)$$

En el caso de que el triángulo no sea *smooth*, la normal es igual en todos los puntos:

$$\frac{(v_2 - v_1) \times (v_3 - v_1)}{\|(v_2 - v_1) \times (v_3 - v_1)\|} \quad (20)$$

9.5.2. Cuadriláteros

Los cuadriláteros se representan con dos triángulos coplanares. Para asegurar de que ambos triángulos son coplanares, los cuadriláteros se construyen indicando únicamente la

esquina superior derecha e izquierda y la esquina inferior izquierda. La esquina inferior derecha se calcula a partir de las demás.

9.5.3. Mallas de triángulos

Las mallas de triángulos se implementan como listas de triángulos, de forma que un rayo intersecta con la malla si intersecta con alguno de sus triángulos.

Las mallas se cargan desde ficheros PLY. Estos ficheros incluyen una descripción de la malla a partir de sus vértices y sus caras. Se soporta que cada vértice contenga una normal asociada, como se ha explicado antes, para poder renderizar mallas de triángulos de forma suave.

Durante la implementación del renderizador, se introdujo la posibilidad de visualizar el mapa de normales de la escena. Esto ha sido de gran utilizar al depurar las mallas de triángulos y al comprobar si la interpolación de las normales de los vértices era correcta.

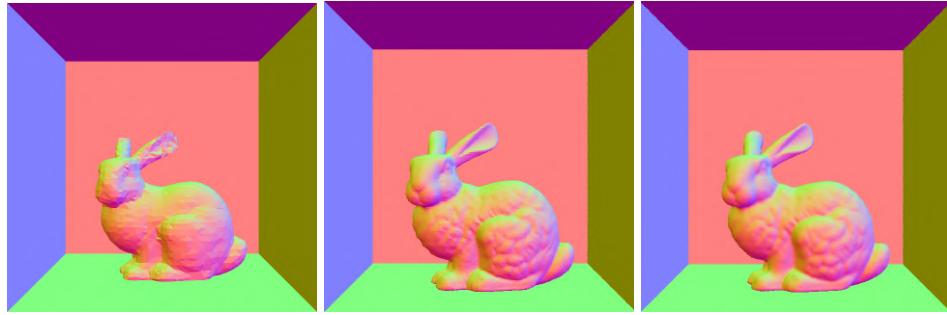


Figura 23: Mapas de normales de una escena con un conejo con diferentes número de triángulos.

En la Figura 24c se muestra uno de los errores que aparecieron mientras se implementaba el suavizado de mallas de triángulos. Debido a errores en el cálculo de las coordenadas baricéntricas de un punto, aparecían bultos en la superficie de la malla debido que las normales interpoladas eran incorrectas. La técnica de *bump mapping* alcanza un efecto similar, aunque de forma intencionada, modificando las normales de la superficie de una figura según un *bump map* para darle un aspecto rugoso sin modificar su geometría.

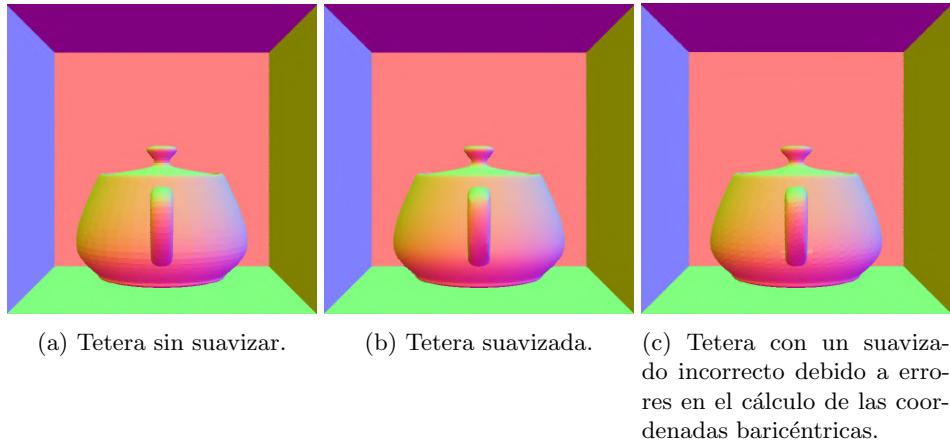


Figura 24: Mapas de normales de una escena con una tetera. La tetera tiene 6320 triángulos en todas las imágenes.

En Internet es complicado encontrar modelos PLY [6] que incluyan las normales de los vértices, así que la solución fue descargar modelos que no las tuvieran y calcularlas en un programa de procesado de mallas como Meshlab. En Meshlab esto se puede hacer cargando una malla, yendo a File→Export Mesh... y seleccionando la opción Normal de la sección Vert antes de guardarla.

Las mallas de triángulos permiten representar formas muy complejas como la que se ve en la siguiente imagen.

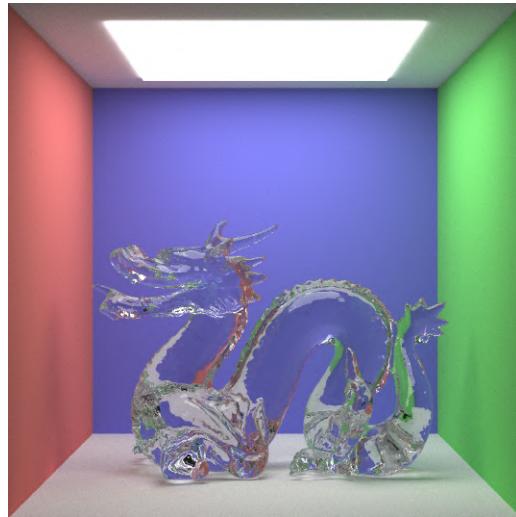


Figura 25: Stanford dragon dieléctrico (índice de refracción 1,52) renderizado con 1024 *ppp* en 122 minutos. El dragón tiene 47794 triángulos.

9.6. Texturas

La inclusión de nuevos tipos de materiales adicionales como podría ser el modelo de *Phong*, al proyecto permite crear geometrías con aspectos más reales. No obstante es posible crear apariencias mucho más parecidos a la realidad, únicamente pegando texturas sobre la geometría. Por esta razón se consideró apropiada la inclusión de texturas al trabajo. Para logralo, se pensó desde un principio hacer una parametrización u, v de la textura, siguiendo el siguiente esquema.

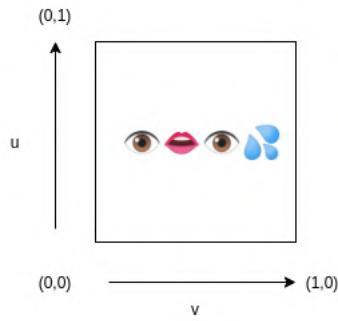


Figura 26: Parametrización de la imagen de la textura.

Una vez hecho esto, solamente sería necesario calcular la coordenada u y la coordenada v en la textura cuando un rayo intersecta con la geometría. Esto dependía del tipo de geometría pues no era lo mismo hacer una parametrización en una esfera que en un plano. En el caso de la esferas el cálculo correspondía con las siguientes ecuaciones.

$$u = 0,5 - \arcsin(d_y)/\pi \quad (21)$$

$$v = 0,5 + \arctan 2(d_z, d_x)/2\pi \quad (22)$$

donde d corresponde con el vector que va desde el centro de la esfera al punto de intersección.

Para el caso de planos infinitos, puesto que el rango de la textura es finito, fue necesario crear un método para lograr hacer un patrón que pudiera repetirse a lo largo del plano. Es por esto que se decidió crear un sistema de coordenadas para el plano donde el origen correspondería al punto $(0,0)$ que aparece en la imagen 26. En dicho sistema de coordenadas se representó la textura de la siguiente manera.

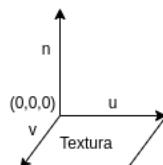


Figura 27: Sistema de coordenadas creado sobre un plano para texturas.

Donde n corresponde con la normal del plano y los vectores u y v son dos vectores pertenecientes al plano. En las siguientes imágenes pueden verse ejemplos de texturas pegadas sobre planos infinitos.



Figura 28: Imagen con planos junto con representación gráfica del esquema de la Figura 27.

Finalmente la última de las geometrías sobre las que se aplicó el mapeo de texturas sobre cuadrados, siendo esta la implementación mas sencilla de todas las realizadas. Puesto que un cuadrado representaba un plano finito se podía realizar el cálculo de la coordenada u y v tomando como base el siguiente esquema.

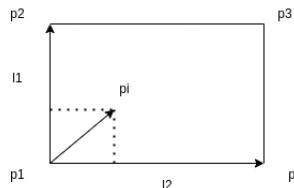


Figura 29: Esquema seguido para hacer la parametrización u v en cuadrados.

En dicho esquema los puntos $p1, p2, p3$ y $p4$ representan las esquinas del cuadrado en el espacio, pi el punto de intersección del rayo y $l1$ y $l2$ los vectores del lado derecho y la base del cuadrado. Dicho esto las coordenadas u y v pueden calcularse mediante la proyección del vector entre $p1$ y pi . Si denotamos tal vector como w pueden aplicarse los siguientes cálculos para determinar las coordenadas u y v .

$$u = \frac{|w| \cdot \cos \theta_1}{|l1|} \quad (23)$$

$$v = \frac{|w| \cdot \cos \theta_2}{|l2|} \quad (24)$$

donde $\cos \theta_1$ corresponde con el coseno del ángulo que forma w y $l1$ y $\cos \theta_2$ es el coseno del ángulo que forman w y $l2$.

Dicho esto, en la siguientes imágenes se pueden ver ejemplos de texturas colocadas sobre cuadrados y esferas.

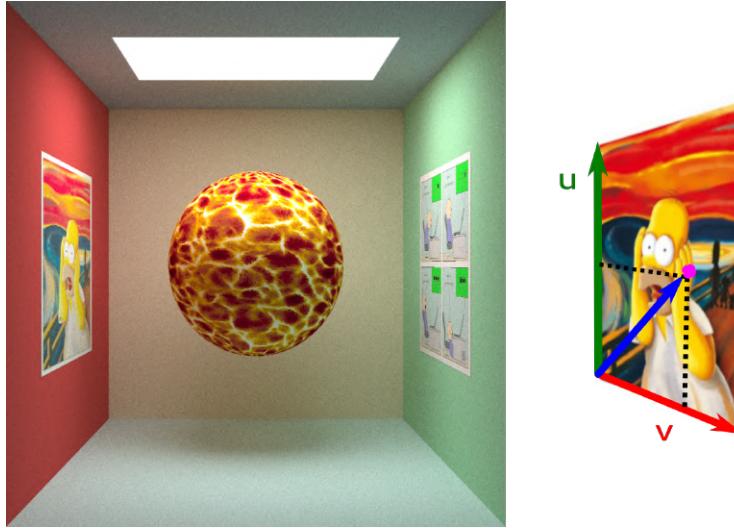


Figura 30: Imagen con texturas sobre esfera y cuadrados junto con representación gráfica del esquema de la Figura 29

10. Profundidad de campo

Hasta ahora, la cámara se ha modelado como una cámara *pinhole*, en la que los rayos tienen siempre el mismo origen. Una aproximación más realista son las cámaras *thin lens*. En este tipo de cámaras el origen de los rayos es variable, puede ser cualquier punto que se encuentre dentro de un área, cuyo tamaño lo define la apertura. Se llaman *thin lens* porque simulan lentes cuyo grosor es despreciable.

Como el algoritmo de renderizado con el que se está trabajando está basando en Monte Carlo, cada vez que se lanza un rayo se puede elegir su origen de forma aleatoria en ese área, y al lanzar varios rayos para el mismo píxel y hacer la media de la radiancia calculada cada vez, se podrá aproximar el efecto de profundidad de campo. Cuanto mayor sea la superficie sobre la que se eligen puntos aleatorios (la apertura), menor es la profundidad de campo, y por tanto menor es la región alrededor del plano de enfoque en la que los objetos están enfocados.

En la implementación, la apertura tiene forma circular y se especifica mediante su radio. Para elegir un punto en la apertura hace falta muestrear de forma uniforme un círculo. Esto se hacía en un principio mediante *rejection sampling*, pero esta técnica es poco eficiente ya que un gran número de puntos eran rechazados cada vez que se generaba un rayo. Para solucionarlo, se pasó a realizar el muestreo usando la función de distribución de probabilidad, de forma similar al método utilizado para muestrear puntos en la hemisferia.

En las siguientes imágenes se puede ver el efecto conseguido con este tipo de cámara.

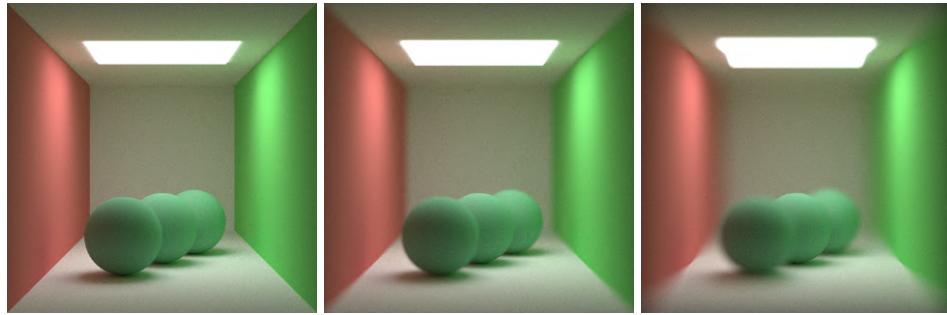


Figura 31: Misma escena con diferentes tamaños de la apertura de la cámara. La primera imagen se ha renderizado con apertura 0. La cámara de la tercera imagen tiene una apertura mayor que la de la segunda.

11. Conclusiones

- Se han comprendido a la perfección los conceptos los elementos básicos de la *Informática Gráfica* para resolver el problema de representación de modelos 3D en 2D.
- Se ha asimilado la metodología de trabajo con varios sistemas de coordenadas en 3D, aprendiendo a trabajar realizando cambios de base en espacio vectoriales en \mathbb{R}^3 .
- Se ha entendido por completo el modelado de la interacción de la luz con los materiales, mediante la ecuación de render.
- Se ha desarrollado una librería de geometrías básica y eficiente para la realización de operaciones geométricas, constituyendo así la base del algoritmo de *Path Tracing*.
- Se ha implementado el modelo de cámara *Pinhole* así como un sistema de trazado de rayos, para conseguir representar elementos en espacios \mathbb{R}^3 en \mathbb{R}^2 .
- Se han desarrollado primitivas geométricas básicas como esferas y planos, para poder generar escenas renderizadas básicas como una *Cornell Box*. De manera opcional se ha integrado mallas de triángulos para lograr tener objetos más complejos.
- Como se ha visto a lo largo del informe, se ha logrado con creces diseñar imágenes complejas teniendo en cuenta la iluminación de la escena y la posición y materiales de los elementos.
- Se han entendido las diversas propiedades de la luz en la naturaleza para poder tenerlas en cuenta en la generación artificial de gráficos por ordenador.
- El equipo a logrado comprender a la perfección el funcionamiento del espacio de colores *RGB*, para la creación de imágenes en formato *LDR* y *HDR*.
- Se ha hecho una correcta implementación de luces puntuales y de área para lograr una correcta iluminación de las escenas.
- Se conseguido comprender a la perfección la interacción de la luz con los materiales mediante las diversas *BRDFs* estudiadas.
- Se ha logrado realizar una implementación correcta de los operadores de *Tone Mapping* básicos para poder realizar una correcta conversión, de *HDR* a *LDR*.
- La necesidad de implementar algoritmos probabilistas como *Ruleta Rusa* y *Monte-carlo* ha permitido comprender comprender el funcionamiento de ambos algoritmos y la necesidad de los mismos para la optimización del algoritmo de *Path Tracing* .
- Por parte del equipo de trabajo se considera que se ha realizado la implementación de un considerable número de apartados opcionales, que dan al proyecto la posibilidad de generar resultados bastante completos.

12. Control de esfuerzos

José Daniel Subías Sarrato

Horas invertidas: 115

Tareas realizadas:

- Implementación de la librería de geometrías. 3h
- Implementación del ray tracing. 4h
- Implementación de la paralelización. 7h
- Implementación de texturas. 10h
- Implementación de tone mapping básico. 5h
- Implementación del tone mapping de Reinhard. 05 10h
- Implementación de Path Tracer básico. 12h
- Implementación de materiales básicos. 15h
- Implementación de la BRDF de Phong. 13h
- Redacción de la memoria. 24h
- Diseño de imágenes para la memoria. 12h

Fernando Peña Bes

Horas invertidas: 118

Tareas realizadas:

- Implementación de la librería de geometrías. 3h
- Implementación del ray tracing. 4h
- Implementación de tone mapping básico. 5h
- Implementación de Path Tracer básico. 12h
- Luces puntuales y *Next Event Estimation*. 10h
- Materiales dieléctricos con Fresnel. 15h
- Triángulos y mallas de triángulos. 20h
- Suavizado de mallas de triángulos. 5h
- Bounding Volume Hierarchies. 25h
- Cámaras *thin lens*. 2h
- Diseño de imágenes para la memoria. 12h
- Redacción del la memoria. 5h

Referencias

- [1] Link la documentacion de la STL de C++. https://www.cppreference.com/Cpp_STL_ReferenceManual.pdf.
- [2] Ecuaciones de Fresnel. https://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf.
- [3] Ley de Snell. https://en.wikipedia.org/wiki/Snell%27s_law.
- [4] Paper del Tone Mapper de Reinhard 05 http://www.cmap.polytechnique.fr/~peyre/cours/x2005signal/hdr_photographic.pdf
- [5] Algoritmo de Möller Trumbore para la intersección rayo-triángulo. https://en.wikipedia.org/wiki/MllerTrumbore_intersection_algorithm
- [6] Formato PLY. <https://people.sc.fsu.edu/~jburkardt/data/ply/ply.html>