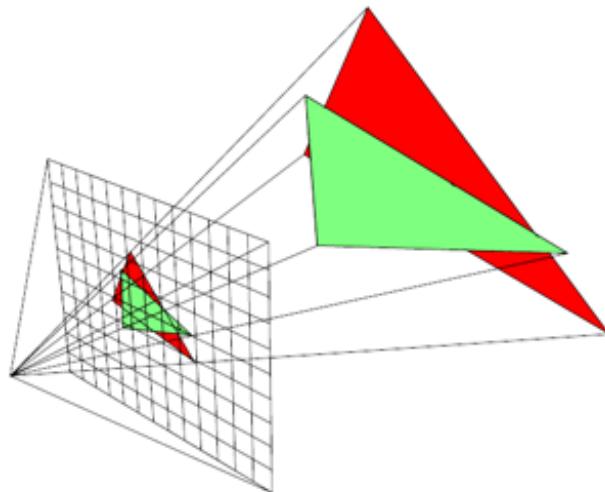




**Escuela de
Ingeniería y Arquitectura
Universidad** Zaragoza

INFORMÁTICA GRÁFICA

PathTracer



Alumnos: Puig Rubio, Manel Jordá 839304
Ye, Ming Tao 839757

11 de enero de 2025

Índice

1. Introducción	4
2. Ecuación de Render	4
3. Convergencia	6
3.1. Rayos por píxel	6
3.2. Materiales	7
3.3. Fuentes de luz	11
4. Efectos de iluminación global	13
4.1. Sombras duras	13
4.2. Sombras suaves	13
4.3. Color bleeding	15
4.4. Cáusticas	15
5. Extensiones	17
5.1. Paralelización con threads	17
5.2. Primitivas extra	18
5.3. Luces de área en Next Event Estimation	19
5.4. Texturas	20
5.5. ToneMapping de Reinhard	22
6. Carga de trabajo y metodología	23
7. Otros renders	24
7.1. Luz puntual	24
7.2. Luz de área	25
8. Bibliografía	26

1. Introducción

En este informe, se analizarán distintos aspectos del proyecto de *path tracing* elaborado a lo largo del cuatrimestre para la asignatura de Informática Gráfica.

2. Ecuación de Render

Para explicar cómo se ha estimado la **ecuación de render**, en primer lugar mostramos la propia ecuación original junto con una representación conceptual de la misma.

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

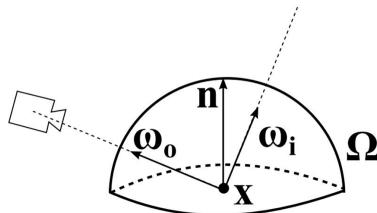


Figura 2.1: Representación visual de la ecuación de render. En la implementación, ω_o está en el sentido opuesto.

Esta ecuación representa la **luz saliente** L_o en un punto x y una dirección incidente ω_o determinados. Esto se obtiene sumando la emisión (L_e) del propio punto y la integral sobre todas las posibles direcciones incidentes ω_i del hemisferio Ω centrado alrededor de la normal n . En dicha integral, podemos diferenciar 3 términos principales:

- La contribución de la **luz incidente** L_i , que proviene de fuentes de luz como las luces puntuales o las luces de área.
- La función de reflectancia de la superficie f_r (o **BRDF**) que depende del material del objeto.
- El **coseno del ángulo incidente**, que atenúa la luz saliente en función del ángulo de incidencia respecto a la normal.

Para resolver la integral de la ecuación se ha utilizado el método de aproximación por **Monte Carlo**¹, con un muestreo de los rayos uniforme sobre el coseno (muestreo por importancia). En las **superficies difusas**, si desarrollamos $f_r(x, \omega_i, \omega_o)$ y $|n \cdot \omega_i|$ y dividimos por la probabilidad de obtener la dirección por muestreo por importancia, queda al final la siguiente ecuación:

$$L_o(x, \omega_o) \approx \sum_{i=1}^N \frac{2\pi L_i(x, \omega_i) k_d \sin \theta_i \cos \theta_i}{2\pi \sin \theta_i \cos \theta_i}$$

Si además anulamos los factores iguales del numerador y denominador, nos quedará de la siguiente manera:

¹Más información en: https://www.pbr-book.org/4ed/Monte_Carlo_Integration#

$$L_o(x, \omega_o) \approx \sum_{i=1}^N L_i(x, \omega_i) k_d$$

En superficies **especulares** y **refractantes**², el término $f_r(x, \omega_i, \omega_o)$ corresponde a los coeficientes k_e (reflexión especular) o k_r (refracción), que son deterministas y no incluyen pérdidas. En este caso, la probabilidad de obtener la dirección específica del rayo es 1, lo que simplifica la ecuación de forma similar a la anterior, sustituyendo el coeficiente difuso k_d por k_e o k_r , según corresponda.

Para decidir el evento asociado al rayo (reflexión, refracción o absorción), se utiliza el algoritmo de **Ruleta Rusa**.

En nuestra implementación, la luz incidente L_i en un punto sobre una superficie difusa se calcula utilizando la función **NextEventEstimation**. Esta función approxima el término $L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) |n \cdot \omega_i|$ mediante la siguiente fórmula:

- L_i se calcula con $\frac{p_i}{|c_i - x|^2}$.
- f_r equivale al coeficiente difuso k_d del objeto (en materiales difusos sería k_d/π , pero como se ha mencionado anteriormente, el π es anulado al dividir por la probabilidad de la dirección del rayo).
- El coseno del ángulo incidente se calcula con $|n \cdot \frac{c_i - x}{|c_i - x|}|$.

(c_i es el punto desde el que proviene la luz, y p_i es la radiación que emite la luz)

²Para su implementación nos basamos en la siguiente referencia: https://www.pbr-book.org/4ed/Reflection_Models/Specular_Reflection_and_Transmission

3. Convergencia

El estudio de la convergencia del Path Tracer se fundamenta en el uso de algoritmos como *Monte Carlo* y *Ruleta Rusa*. Estos algoritmos nos permiten parametrizar y controlar la calidad del render generado, en función de varios parámetros clave: el número de rayos por píxel (N), el número de rebotes de cada rayo (R), el número de objetos o primitivas en la escena (G) y el número de píxeles de la imagen (P). A partir de estos parámetros, la complejidad temporal del algoritmo se puede expresar como:

$$\mathcal{O}(N \times R \times G \times P)$$

Entre estos factores, el número de rebotes (R) merece una consideración especial, ya que su valor exacto puede resultar difícil de estimar de forma realista durante el análisis previo. Aunque el número máximo de rebotes puede ser limitado explícitamente en la implementación, la naturaleza no determinista del algoritmo *Ruleta Rusa* introduce una componente probabilística que influye en esta variable.

En particular, el número efectivo de rebotes depende de la probabilidad asociada a las interacciones entre los rayos y los diferentes objetos y materiales presentes en la escena. Esto significa que, para un análisis más detallado, sería necesario considerar las propiedades específicas de los materiales y la distribución de las geometrías en la escena.

3.1. Rayos por píxel

Se estudia la misma escena de una caja de Cornell con una luz puntual y con dos esferas: una de plástico (difuso + especular) y otra de cristal (refractante + especular), con distintos números de rayos por píxel en la figura 3.1.

Como puede observarse, hasta los **128 rayos por píxel** (figura 3.1d) existe un ruido bastante evidente en la escena, tanto en las paredes como en las esferas y sus sombras.

A partir de los **512 rayos por píxel**, el ruido comienza a volverse prácticamente imperceptible, especialmente en superficies con colores difusos, como las paredes. Estas superficies, al tener un color uniforme y estar menos afectadas por efectos complejos de iluminación global, convergen más rápidamente. Por otro lado, en las esferas el ruido sigue siendo más evidente. Esto se debe a que estas primitivas pueden incluir combinaciones de componentes de su BSDF, como una componente difusa (k_d) junto con una especular (k_s) o refractante (k_t). Cuando se mezclan distintos tipos de interacción de luz en el mismo objeto, se necesitan más rayos por píxel (RPP) para que la integración sea suave y realista.

Finalmente, con **2048 rayos por píxel** las imperfecciones en la imagen se vuelven prácticamente indetectables. Los distintos materiales en las esferas se han renderizado con suavidad y las sombras, los efectos de iluminación global, el *color bleeding* y la cáustica de la esfera de cristal están libres de ruido. Por lo tanto, podríamos considerar que, a partir de los 2048 rayos por píxel y con una resolución de 512x512 píxeles, la escena converge totalmente.

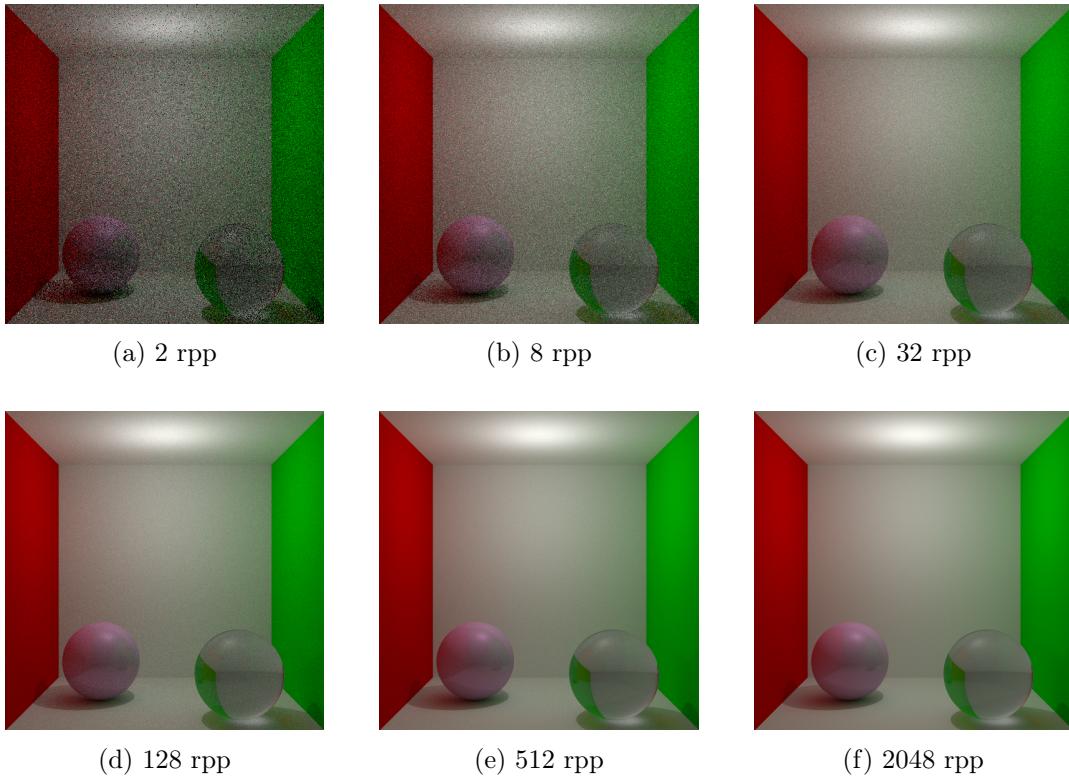


Figura 3.1: Convergencia de rayos por píxel

3.2. Materiales

Para analizar la convergencia de los materiales, se han llevado a cabo pruebas utilizando los tres tipos de fuentes de luz implementadas. La convergencia de las propias fuentes de luz será abordada posteriormente.

Particularmente, tal y como está implementado el Path Tracer, se favorece la convergencia de materiales difusos por encima de los especulares y refractantes. El motivo es que no se calcula *Next Event Estimation* cuando un rayo choca contra un material componente especular o refractante, por lo que es fácil que dicho rayo acabe sin contribuir nada a menos que choque contra una superficie difusa (esto se notaría principalmente con pocos rebotes y rpp). Por ello, vamos a emplear una escena con una geometría sencilla y establecer el máximo de rebotes por rayo en 5. Además, se ha empleado un coeficiente de absorción del 10% para todos los materiales para tener una igualdad de condiciones y se ha empleado en todos el mismo tonemapper: ecualización + gamma.

3.2.1. Materiales iluminados con luz puntual

Los materiales convergen aproximadamente a la misma velocidad. Destaca ligeramente el material especular, ya que calcular la dirección especular de un rayo es menos costoso computacionalmente que calcular el NEE en el caso del material difuso o la dirección refractante en el caso del material refractante. Particularmente, si nos fijamos en el reflejo de la esfera especular (figura 3.3), el oscurecimiento progresivo de la escena a medida que nos vamos alejando está muy bien logrado y es un indicio de buena convergencia.

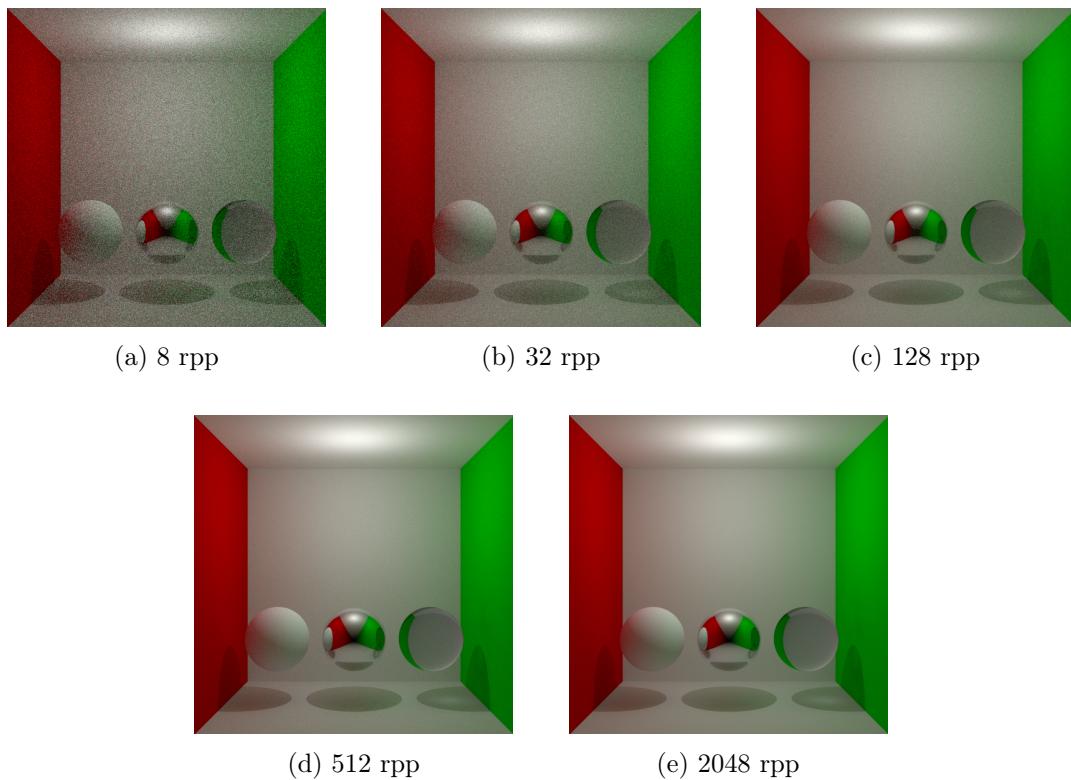


Figura 3.2: Convergencia de materiales con luz puntual

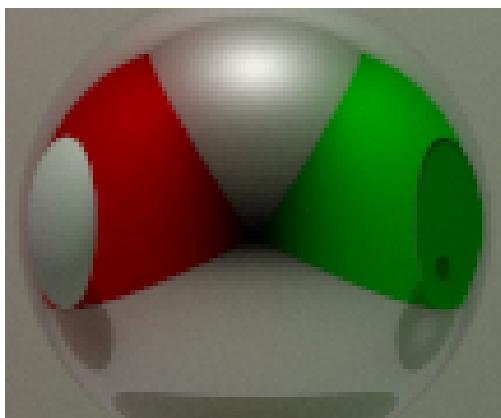


Figura 3.3: Convergencia de material especular con luz puntual y 2048rpp (zoom)

3.2.2. Luz de área sin NEE

Los tres materiales convergen prácticamente a la misma velocidad, por la misma razón que en la luz puntual. Destaca de nuevo el material especular, consiguiendo una convergencia ligeramente mejor que los otros dos materiales (ya que, por ejemplo, aún podemos ver algo de ruido en los materiales difuso y refractante de la figura 3.4e, el cual el material especular consigue "disimular" debido al reflejo en tamaño reducido). Por otra parte, debido al número de rebotes puesto y que la luz de área sea infinita, conseguimos que el reflejo de la esfera especular (figura 3.5) tenga muy buena convergencia incluso para distancias lejanas.

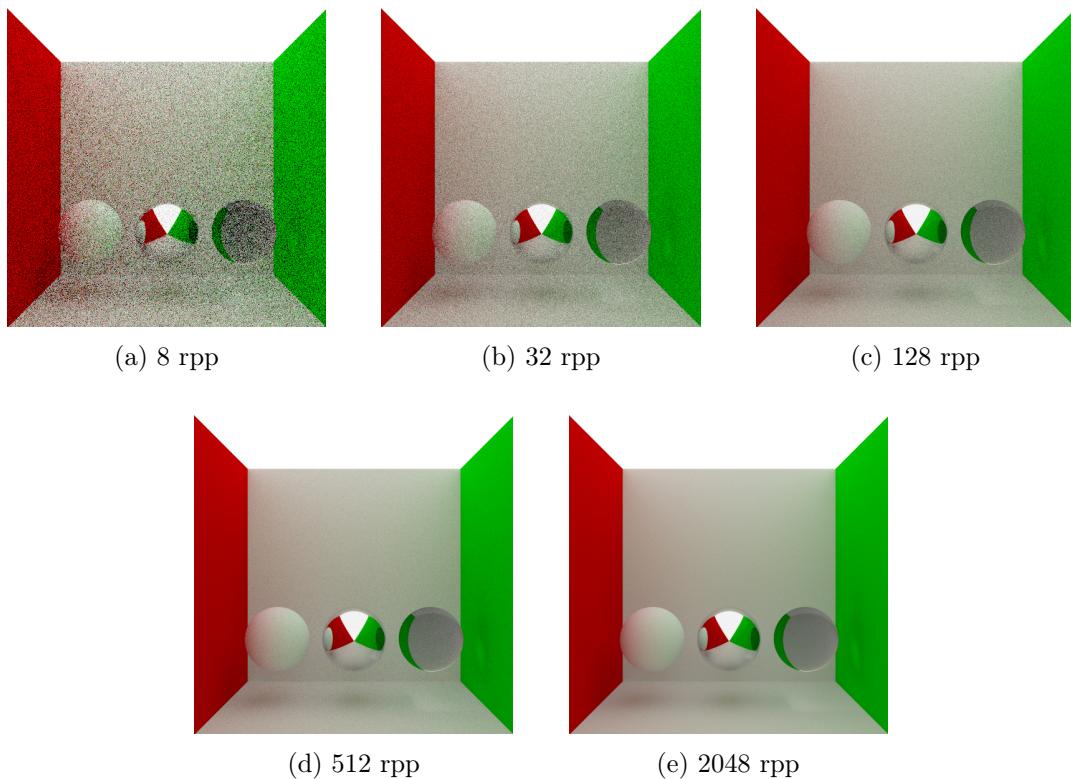


Figura 3.4: Convergencia de materiales con luz de área sin NEE

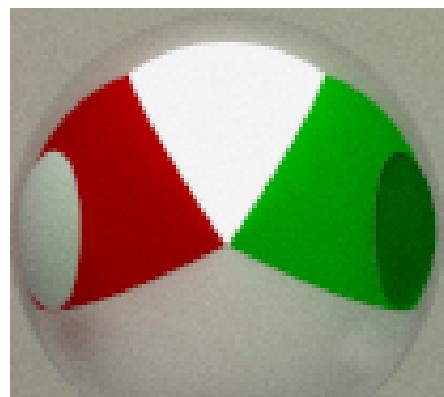


Figura 3.5: Convergencia de material especular con luz de área infinita sin NEE y 2048rpp (zoom)

3.2.3. Luz de área con NEE

En cuanto a la velocidad de convergencia, destacan notablemente los materiales especular y refractante, siendo el material difuso el más lento de todos. Esto se debe a que, al intentar iluminar cada punto difuso del material, se tienen que ir muestreando puntos aleatorios en la luz de área hasta que alguno ilumine a dicho punto. Sin embargo, para los otros dos materiales, al no afectar el NEE, estos cálculos no son necesarios. Para el caso particular de este tipo de luz, es claramente notable que no converge tan bien el material especular ya que si nos fijamos en la figura 3.7, veremos que el a medida que la distancia aumenta, el ruido también lo hace. Concluimos que si tenemos materiales especulares y los rayos deben viajar una gran distancia para interseccar, el muestreo de puntos de luz de la luz de área con NEE es menos recomendable que la luz puntual en tema de convergencia.

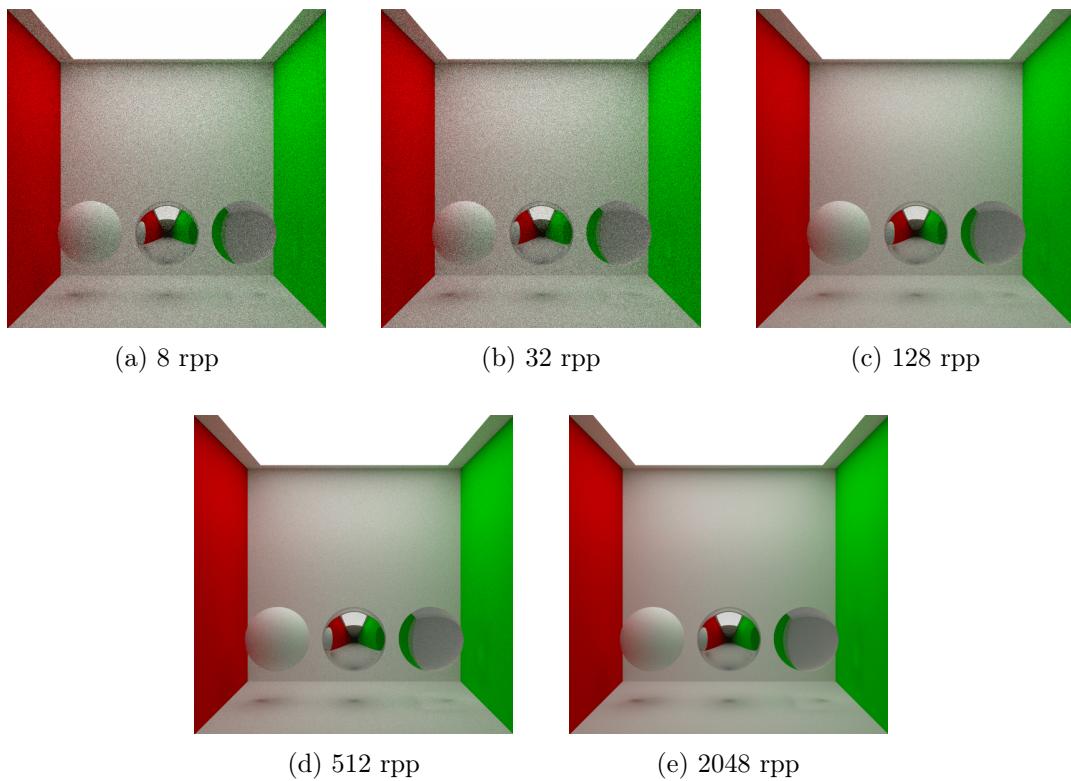


Figura 3.6: Convergencia de materiales con luz de área con NEE

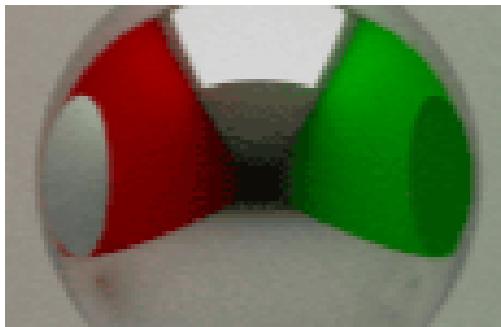


Figura 3.7: Convergencia de material especular con luz de área NEE y 2048rpp (zoom)

3.3. Fuentes de luz

Para este apartado, debido a que las figuras del apartado anterior también permiten comparar la convergencia de las fuentes de luz, se hará referencia a estas.

3.3.1. Luz puntual

Converge relativamente rápido, ya que utiliza solo 1 cálculo de Next Event Estimation (por cada luz puntual) en cada punto difuso. Observando la figura 3.2e, que no tiene apenas ruido en ninguna zona, podemos confirmar que la luz puntual es la que, en general, mejor converge con menos rayos por pixel. Además, tal y como hemos comentado sobre la figura 3.3, se comporta bien para distancias lejanas.

3.3.2. Luz de área sin NEE

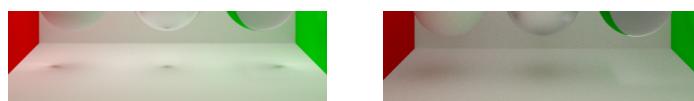
Es la que converge más rápido, pues la contribución de un rayo depende solamente de que choque contra una luz de área (nueva condición terminal, devuelve la potencia de la luz), evitándonos los cálculos del Next Event Estimation. Observando la figura 3.4e, analizamos que converge bastante bien, aunque algo peor que la luz puntual, pues en esta figura sí que se puede apreciar algo más de ruido con los mismos rayos por píxel (en comparación con la figura 3.2e). Sin embargo, el caso de la luz de área infinita favorece mucho la convergencia de zonas a una distancia lejana (figura 3.5). Esto no ocurriría si hubiésemos limitado el tamaño de la luz de área, ya que entonces sería cada vez más complicado que los rayos choquen contra una luz de área (para contribuir) a medida que se alejan de la fuente de luz.

3.3.3. Luz de área con NEE

Es la que converge más lento, ya que para tenerlas en cuenta para el NEE, es necesario muestrear un punto aleatorio del objeto que hace de luz de área. Además, ese punto que muestreamos tiene que iluminar el punto que nos interesa. Es por ello, que debemos reintentarlo un cierto número de veces, lo cual resulta costoso para escenas con geometrías complicadas. Además, posteriormente hace una estimación de la ecuación de render, cálculo que hace todavía más lenta la convergencia.

En la figura 3.6, destacan algunos elementos que difieren bastante de las otras fuentes de luz. Por ejemplo:

- Las sombras de las esferas proyectadas en el suelo, pues difieren mucho con las sombras vistas en las figuras 3.2 y 3.4: Al utilizar NEE en distintos puntos alrededor de las esferas, se iluminan más zonas del suelo que en los ejemplos anteriores, provocando ese punto de sombra.



(a) Con NEE

(b) Sin NEE

Figura 3.8: Comparación de las sombras de luces de área

- En la pared verde derecha se puede intuir una cáustica de la esfera de cristal parecida a las de los demás apartados, puesto que esta depende de que varios caminos iluminados se concentren al pasar por la superficie de la esfera (es decir, que la luz provenga de varios sitios distintos

no influye tanto en esta cáustica). Sin embargo, en las sombras de la pared y el suelo y en la cáustica del suelo, justo por la razón mencionada en el punto anterior, son prácticamente imperceptibles.

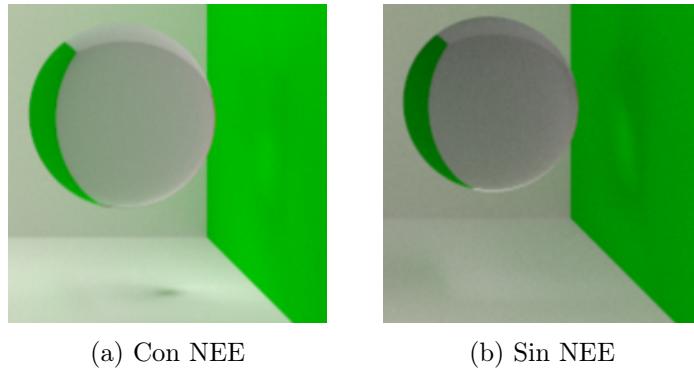


Figura 3.9: Comparación de las cáusticas de luces de área

3.3.4. Conclusión

Las **luces de área sin NEE** son las que convergen más rápido, debido a la ausencia de cálculos de NEE. Dependen de la facilidad de que un rayo choque con ellas (tamaño de la luz, escenas con geometría que no entorpezca rebotes, número de rebotes). Para los casos en los que tiene una superficie pequeña, las escenas convergen notablemente peor, debido a la baja probabilidad de impacto de un rayo con la luz.

Estas van seguidas de las **luces puntuales**, que convergen significativamente más rápido justo por el motivo contrario. Obtenemos resultados decentes en prácticamente todos los casos.

Por último, las **luces de área con NEE** tienen una convergencia muy superior a su versión sin NEE cuando la luz es pequeña (figura 5.2d vs figura 4.2e). Son las que más lento convergen, pues dependen de la suerte de muestrear el punto de luz correcto (si existe). Sin embargo, cabe recalcar que si se reduce la cantidad de puntos difusos 'difíciles' de iluminar por la luz de área, tiene una convergencia casi igual a la luz puntual³.

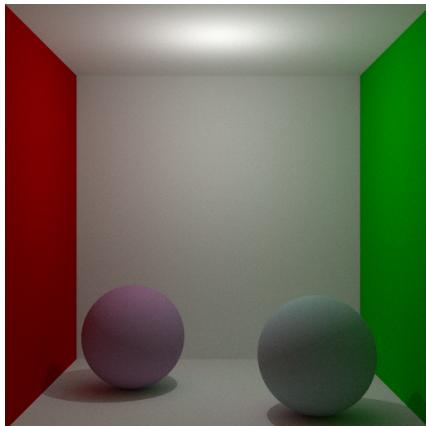
³Esto es porque se muestran menos puntos de luz hasta hallar uno que ilumine los puntos difusos, y hace uso de una fórmula algo parecida a las luces puntuales que explicamos en el apartado de extensiones. Un ejemplo sería una caja de Cornell vacía, sin esferas difusas, pues estas tienen puntos en la parte inferior cercana al suelo que son menos probables de alcanzar.

4. Efectos de iluminación global

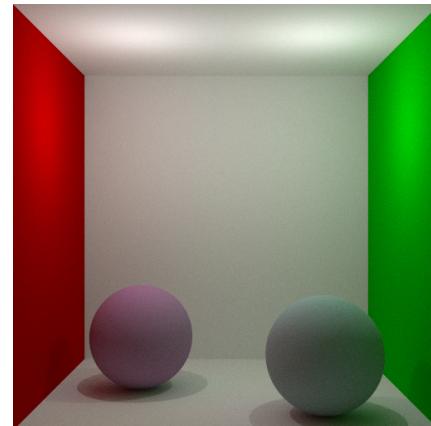
4.1. Sombras duras

Se logran con **luces puntuales**, y se obtienen muy fácilmente con *path tracing*. Para conseguirlas, necesitamos un objeto **difuso**, una luz puntual situada a un lado de ese objeto, y otro objeto difuso (como un plano) situado al otro lado, para que se pueda proyectar la sombra correctamente (cuanto más cerca estén estos objetos, más definida queda la sombra).

En la figura 4.1a se ha utilizado 1 sola luz puntual, que crea una sombra redonda oscura y **bien definida** en sus bordes en el suelo. También podemos diferenciar fácilmente en la figura 4.1e qué parte de las esferas está iluminada por la luz directa. En la figura 4.1b, al haber utilizado dos luces distintas, se observan 2 sombras muy duras en cada esfera, cuya intersección forma una **sombra aún más oscura y definida**. En este caso ya no es tan evidente diferenciar la parte de la esfera iluminada por luz directa (figura 4.1f).



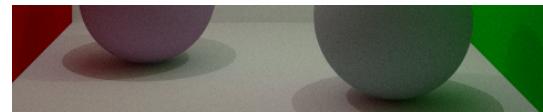
(a) Sombras duras con 1 luz puntual



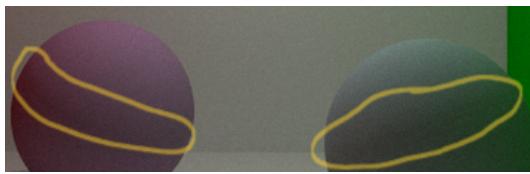
(b) Sombras duras con 2 luces puntuales



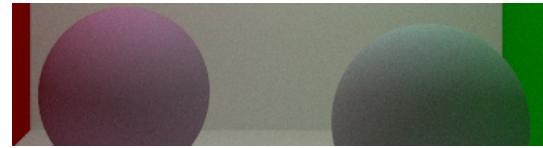
(c) Sombras duras con 1 luz puntual (zoom)



(d) Sombras duras con 2 luces puntuales (zoom)



(e) Sombras en esferas con 1 luz puntual (zoom)



(f) Sombras en esferas con 2 luces puntuales (zoom)

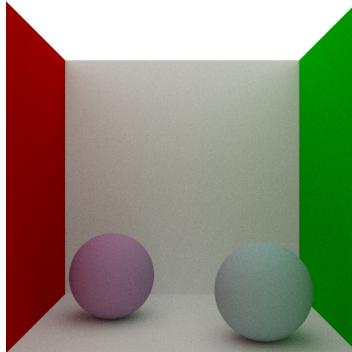
Figura 4.1: Sombras duras. Paredes no blancas tienen $kd=0,6$ para reducir *color bleeding*

4.2. Sombras suaves

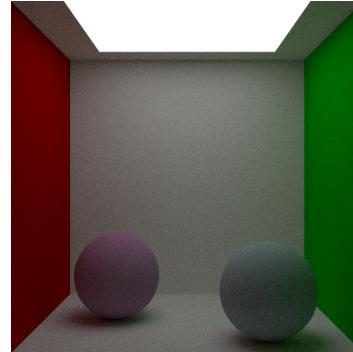
Se logran con **luces de área** y se obtienen muy fácilmente con *path tracing*. Para conseguirlas, necesitamos un **objeto difuso**, una luz de área situada a un lado de ese objeto, y otro objeto difuso

(como un plano) situado al otro lado, para que se pueda proyectar la sombra correctamente (cuanto más cerca estén estos objetos, más definida queda la sombra).

En la figura 4.2, se puede observar que cuanto más grande es la luz de área, **más difusa es la sombra**. En la figura 4.2e, las sombras son algo suaves en los bordes pero se sigue distinguiendo una forma redonda bastante definida. Sin embargo, en la figura 4.2a, la sombra es suave en casi todo su área, concentrándose algo más en el centro (que está más cerca de la esfera) y suavizándose mucho en cuanto se aleja un poco, teniendo unos bordes tan suaves que hacen difícil la distinción entre las zonas con y sin sombra.



(a) Sombras suaves con luz de área infinita



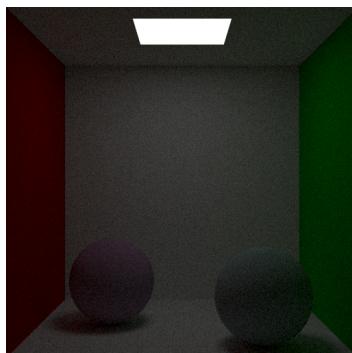
(b) Sombras suaves con luz de área grande



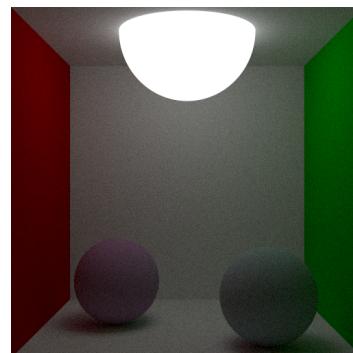
(c) Sombras suaves con luz de área infinita (zoom)



(d) Sombras suaves con luz de área grande (zoom)



(e) Sombras suaves con luz de área pequeña



(f) Sombras suaves con luz de área esférica



(g) Sombras suaves con luz de área pequeña (zoom)



(h) Sombras suaves con luz de área esférica (zoom)

Figura 4.2: Sombras suaves

4.3. Color bleeding

Se logran con **objetos difusos** y se obtienen muy fácilmente con *path tracing*. Para conseguirlas, necesitamos dos objetos difusos con coeficientes muy altos de difusión (en el caso de la figura 4.3, todos los objetos tienen un coeficiente de difusión = 0,9) y, idealmente, que estén cerca entre sí. Veremos que los objetos con colores muy saturados (como las paredes laterales) casi no reciben *color bleeding*, sin embargo, emiten mucho color en los objetos más blancos (como la esfera izquierda o el suelo) y algo menos en los objetos más oscuros (como la esfera dercha).

Cuando iluminamos con luz puntual (figura 4.3a), el color rojo de la pared izquierda sangra mucho en la esfera blanca que tiene al lado y en la zona del suelo que proyecta su sombra. El color verde de la pared derecha sangra bastante en la esfera azul y se nota todavía más en la sombra que proyecta dicha esfera. También se aprecia algo de sangrado azul en la derecha de la esfera izquierda.

Cuando iluminamos con luz de área (figura 4.3b), los colores se proyectan de forma parecida al caso de luz puntual, pero el sangrado es menos llamativo. Esto es debido a la diferencia de contribución de los rayos (NEE de luz puntual vs choque de rayo contra luz de área sin NEE).

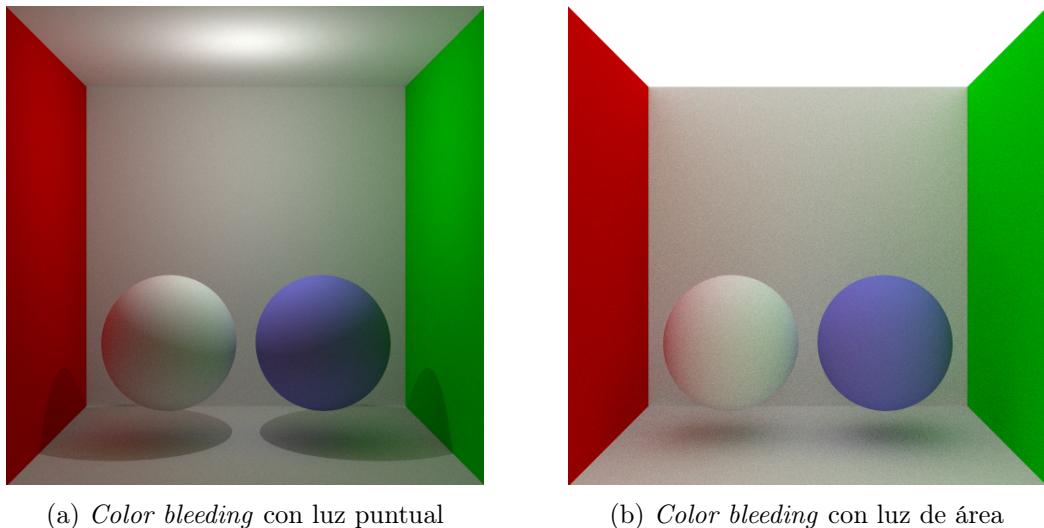
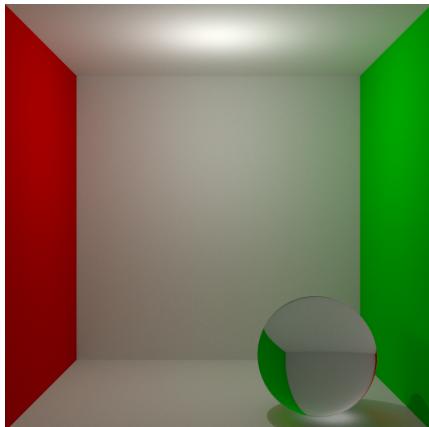


Figura 4.3: Sombras duras

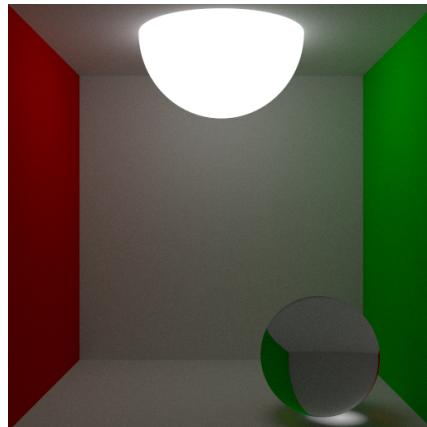
4.4. Cáusticas

Con luces puntuales es difícil que ocurra este fenómeno, pues la probabilidad de que un rayo (tras pasar por un material delta) encuentre una fuente de luz puntual es nula (la cáustica de la figura 4.4c es muy tenue). Por ello, lo más cercano que se puede obtener en *path tracing* es empleando luces de área (sin tenerlas en cuenta en NEE), como luces esféricas (figura 4.4d), cuadradas (figura 4.4g) o infinitas (figura 4.4h). El problema de estos casos es que la convergencia es muy lenta, y se necesitan muchos rayos para conseguir una cáustica realista (pues a estas anteriores les faltan detalles que solo se pueden conseguir con *photon mapping*).

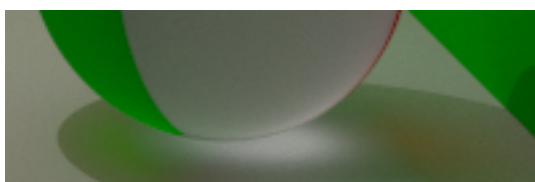
Otro aspecto a comentar es que estas cáusticas no son del todo realistas, pues solo ocurren porque reflejan los puntos del techo que aparecen más iluminados al estar muy cerca de la luz. Se puede apreciar en las imágenes con zoom de la figura 4.4 que las 'cáusticas' son solo un reflejo de la iluminación que hay encima de la esfera.



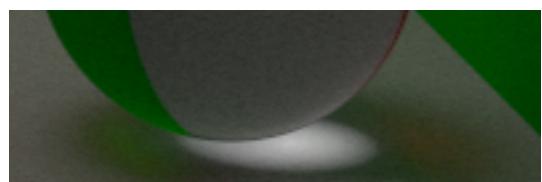
(a) Cáusticas con luz puntual



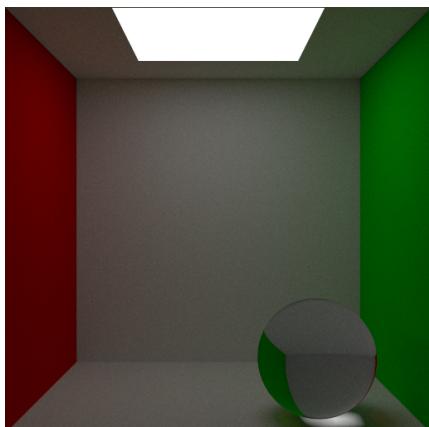
(b) Cáusticas con luz de área esférica



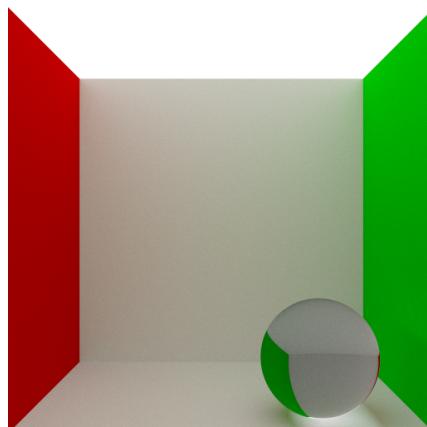
(c) Cáusticas con luz puntual (zoom)



(d) Cáusticas con luz de área esférica (zoom)



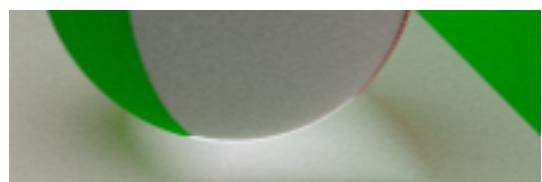
(e) Cáusticas con luz de área cuadrada



(f) Cáusticas con luz de área infinita



(g) Cáusticas con luz cuadrada (zoom)



(h) Cáusticas con luz de área infinita (zoom)

Figura 4.4: Cáusticas

5. Extensiones

5.1. Paralelización con threads

Con el objetivo de agilizar la renderización de las escenas, decidimos realizar una paralelización del cálculo de los píxeles de la imagen basada en sus filas. Esto se puede dado que los píxeles de la imagen no tienen ninguna relación entre sí y nuestro código se basaba en dos bucles `for` que recorrían el alto y ancho de la imagen para ejecutar la función que calculaba la radiancia saliente del pixel correspondiente. Visto esto, optamos por emplear *threads* de la librería estándar, puesto que ya estábamos familiarizados de asignaturas anteriores y para poder obtener output del progreso, tuvimos que emplear una variable global atómica que emplearían los threads para hacer prints por pantalla de forma ordenada y coherente.

La implementación se basa en una repartición equitativa entre los threads que tenemos elegidos (por default, se elige el máximo posible). En caso de que `num_filas % num_threads != 0`, simplemente se reparten las filas restantes entre los threads.

La idea de esta estrategia surgió a raíz del apartado de "Extensiones" del guión de esta misma memoria publicado por los profesores ([10]), y fue implementado de manera intuitiva.

Respecto a los resultados obtenidos, la conclusión general es que la renderización va el doble de rápido cada 4 threads, lo cual es una gran mejora con los procesadores de hoy en día. En la tabla 5.1 se analizan los tiempos de renderizado (de una escena de ejemplo de caja de Cornell muy simple) con distinto número de threads.

Sin threads	4 threads	8 threads
27s	15s	7s

Tabla 5.1: Análisis de coste de tiempo según la utilización de threads

5.2. Primitivas extra

Con el objetivo que aportar riqueza a las escenas, optamos por implementar lo siguiente:

Triángulo⁴: está definido por tres puntos en el espacio y para calcular la intersección se ha empleado el algoritmo de Möller–Trumbore.

Mesh de triángulos: con el objetivo de importar objetos 3D como el conejo de Stanford. Se ha creado un módulo para gestionar archivos .ply y una clase Mesh que almacena los vértices y triángulos obtenidos de dicho .ply. Esta clase Mesh sirve principalmente para optimizar los cálculos de intersección a través de su atributo `esferaMinima`. Esta esfera engloba toda la malla de triángulos, y se utiliza como pre-condición en la intersección de un rayo con la malla de triángulos: primero se calcula si el rayo interseca con la esfera, y si no lo hace, no es necesario iterar por todos los triángulos (ya que no intersecará con ninguno). Se observa en la figura 5.1a.

Cuboide: está definido por un array de 6 planos que deben definirse previamente (asignación de propiedades, texturas, etc.). Se observa en la figura 5.1b.

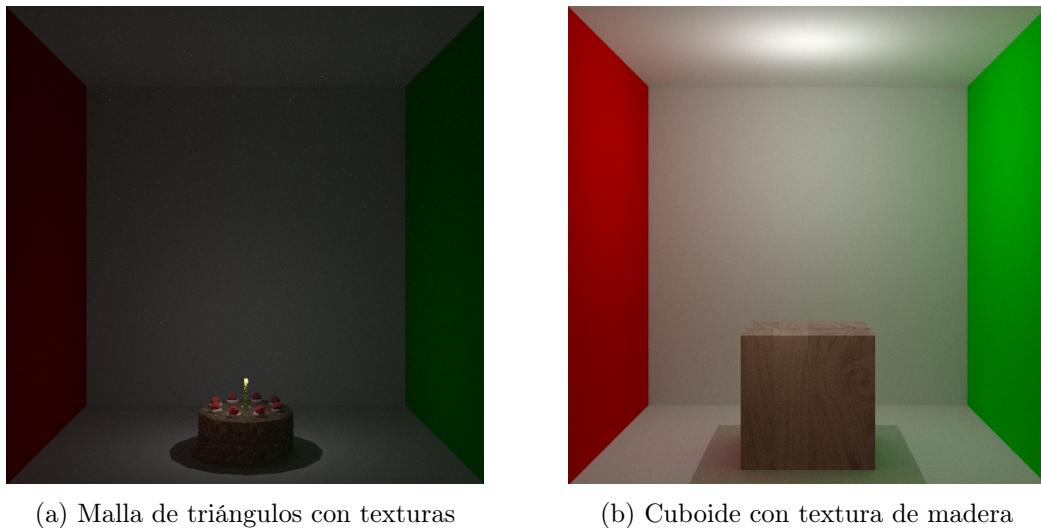


Figura 5.1: Otras geometrías

⁴Fuentes empleadas: <https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection.html> y https://en.wikipedia.org/wiki/M%C3%B6ller-E2%80%93Trumbore_intersection_algorithm

5.3. Luces de área en Next Event Estimation

Con el objetivo de probar otro tipo de luz, se ha implementado la luz de área con NEE. Su implementación en Next Event Estimation es ligeramente similar a la de las luces puntuales, pero en este caso se ha empleado la siguiente ecuación⁵ para su estimación:

$$\frac{1}{N} \sum_{i=1}^N \left[\frac{L_e(x_l) \cdot \cos(\omega_i, n_{x_l}) \cdot \cos(-\omega_i, n_x)}{|x - x_l|^2 \cdot p(x_l)} \right]$$

Si x_l son los puntos a muestrear en la luz de área, la contribución final de un único punto x_l a un punto x en la escena se define a partir de los siguientes términos:

- $L_e(x_l)$: Es la intensidad de la luz de área en el punto x_l . Si la fuente de luz emite la misma intensidad independientemente de la posición en su superficie, L_e será un valor constante definido en la escena.

- $\omega_i = \frac{x - x_l}{|x - x_l|}$: Es el vector de dirección normalizado que apunta desde x_l (punto muestreado en la fuente de luz) hacia x (punto iluminado en la escena).

- n_{x_l} : Es la normal de la superficie de la fuente de luz en el punto x_l .

- $\cos(\omega_i, n_{x_l})$: Es el coseno del ángulo entre la normal n_{x_l} y el vector ω_i , que representa la incidencia de la luz en el punto x_l .

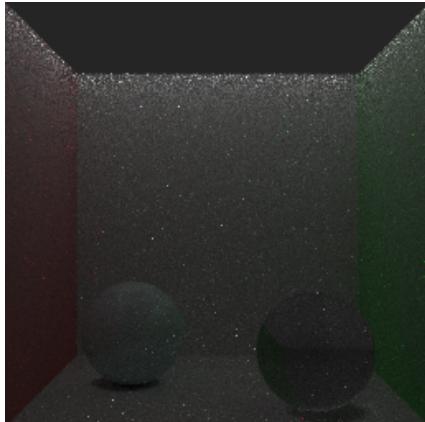
- $p(x_l)$: Es la probabilidad de haber muestreado el punto x_l en la luz de área. Para una fuente de luz uniforme, $p(x_l) = \frac{1}{\text{Área(luz)}}$.

En implementación, simplemente destaca el método de cada primitiva para muestrear un punto x_l que ilumine el punto x (con un límite de intentos máximo). Para el caso particular de los planos, se puede elegir cómo de grande es la parte luminosa de un plano y dónde se sitúa (hay que definir límites). Evidentemente, no se permite que un plano con luz de área NEE sea infinitamente grande porque la probabilidad de muestrear ese punto sería 0.

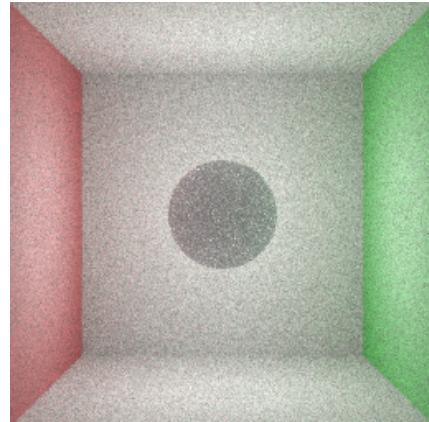
Algunas dificultades encontradas han sido la necesidad de añadir nuevos márgenes de error a las intersecciones rayo-primitiva y a las funciones de pertenencia de un punto a una primitiva para muestrear puntos de luz (figura 5.2a). Por ejemplo, las esferas son las que más problemas han dado puesto que cuanto más pequeña es, más margen de error hay que añadir para calcular si un punto pertenece a la esfera (figura 5.2b).

También se ha tenido que limitar la distancia al cuadrado mínima permitida ($|x - x_l|^2$) para evitar valores explosivos. Con el objetivo de compensar las zonas de la escena afectadas por esta decisión, la contribución de un rayo cuando choca contra una luz de área NEE es el power de dicha fuente de luz pero solo para los rayos muy cercanos a la fuente de luz. De lo contrario ocurre lo de la figura 5.2c (igualmente esto no llega a arreglar el problema a la perfección). Sin embargo, para los casos en los que la luz de área es más pequeña funciona perfecto (figura 5.2d). En conclusión, la luz de área con NEE es una propuesta intermedia entre luces puntuales y luces de área.

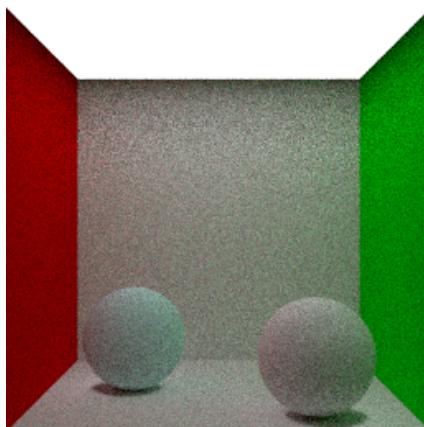
⁵Créditos al Profesor Julio Marco por la ayuda proporcionada



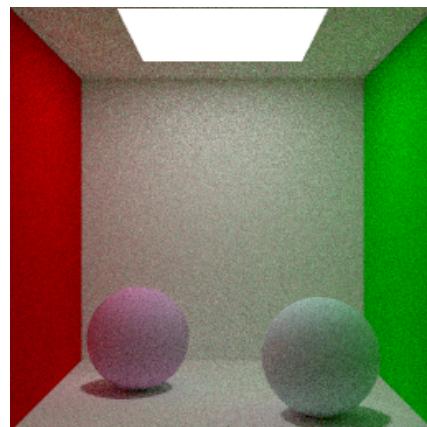
(a) Plano como luz. Sin limitar distancia mínima y sin margen de error pertenencia adecuado



(b) Esfera como luz. Sin margen de error pertenencia personalizado



(c) Plano como luz. Márgenes de error correctos, límite de distancia al cuadrado, choque contra límite de distancia, return power para puntos cercanos devuelve 0.



(d) Plano como luz ($1/u^2$). Márgenes correctos, return power para puntos cercanos.

Figura 5.2: Renders progresivos por los fallos, dificultades y aciertos

5.4. Texturas

Para esta extensión, se han tenido que implementar una clase **Textura** que es un atributo de cada primitiva y convierte ppm's tipo P6 a una matriz de valores RGB. También se han añadido métodos en la clase padre **Primitiva** para devolver la radiancia en el punto (con textura) que queremos. Sin embargo, son las primitivas hijas (plano, esfera, triángulo), las encargadas de pasar de un punto 3D de la primitiva al correspondiente punto 2D de la textura y obtener su valor.

Además, tenemos unos comportamientos particulares en cada primitiva:

- **Planos:** las texturas pueden tener efecto *tiling* que podemos controlar escalando la textura o añadiéndole un offset para moverla por el plano. Se puede observar una textura en un plano escalada en varios tamaños en la figura 5.3.
- **Esferas:** las texturas están implementadas para que cubran toda la esfera. Es especialmente útil para, por ejemplo, simular un globo terráqueo o cualquier otro objeto esférico.

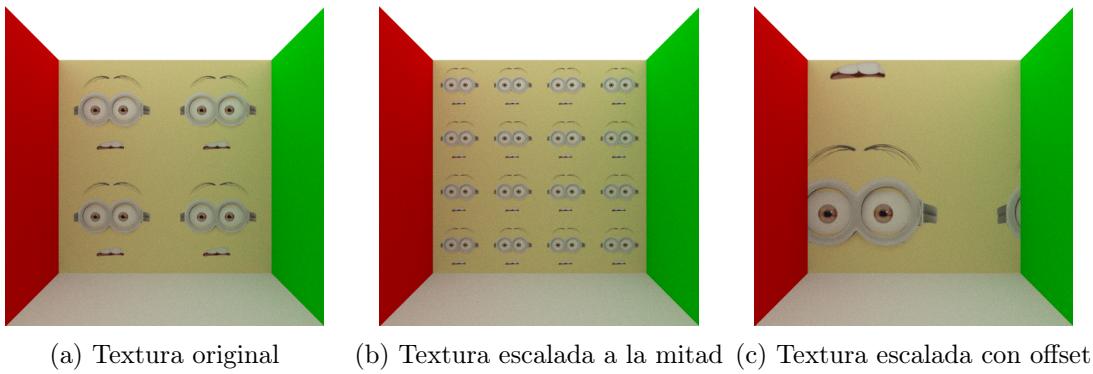


Figura 5.3: Texturas en plano

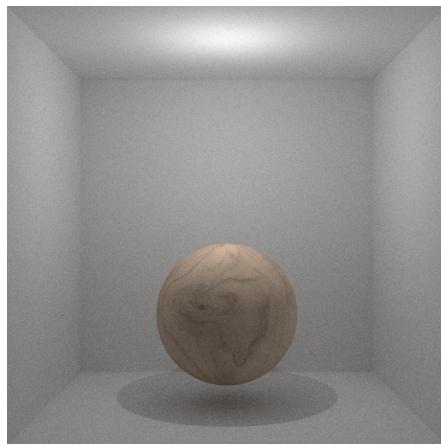


Figura 5.4: Esfera con textura

- **Triángulos:** como esta era una primitiva extra y nuestro objetivo con ella era emplearla para modelado de objetos 3D, hemos implementado los métodos de mapeado y cargado de texturas de forma similar al formato .obj. De esta forma, a cada vértice del triángulo en nuestra escena le asignamos su correspondiente vértice de textura. Se puede apreciar en la figura 5.1a antes vista.

La idea de implementar estas texturas surgió a raíz del apartado de "Extensiones" del guión de esta misma memoria publicado por los profesores ([10]).

5.5. ToneMapping de Reinhard

El algoritmo de Reinhard aplicado a *tone mapping* es un algoritmo ampliamente utilizado para convertir imágenes en alto rango dinámico (HDR) en imágenes de rango dinámico limitado (LDR) de manera natural y agradable para el ojo humano. Este mapeo imita algunas características de cómo los humanos perciben el brillo y los detalles en las escenas del mundo real.

Se ha implementado otro algoritmo de *tone mapping* igual que los obligatorios, con la siguiente ecuación:

$$v_o = v_i \cdot \frac{1 + \frac{v_i}{lmax^2}}{1 + v_i}$$

siendo v_o el valor de salida, v_i el valor de entrada y $lmax$ el valor máximo RGB que se encuentra en la imagen.

Se ha optado por equalizar la imagen antes de aplicarle Reinhard, con un límite de $lmax/1,5$, ya que sino se veía muy oscura.

En la figura 5.5 se muestra la misma imagen sin *tone mapping*, con la ecuación de Gamma+Clamp (con $\text{gamma} = 2.2$) y con la ecuación de Reinhard. Como se puede apreciar, la de Reinhard está demasiado saturada. En general se ve mejor la versión de Gamma+Clamp, por eso ha sido la utilizada durante el resto de figuras.

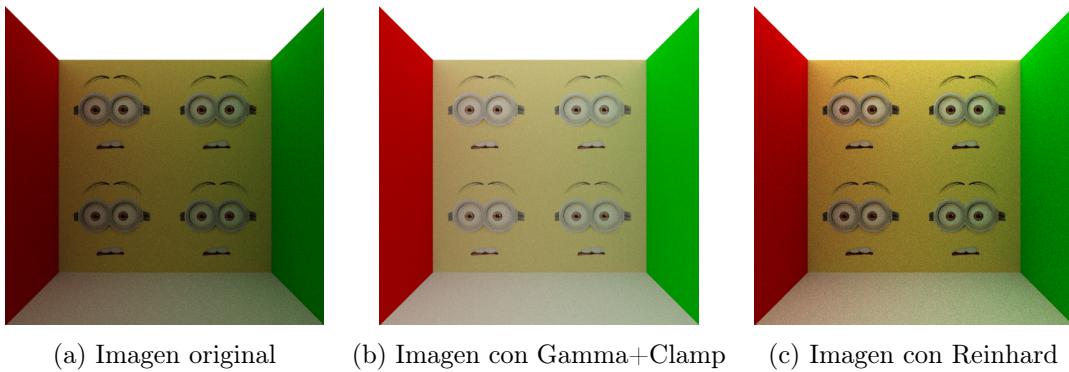


Figura 5.5: Comparativa de *tone mappers*

La ecuación utilizada y la información necesaria para comprender cómo funciona el algoritmo ha sido extraída de esta página⁶. La inspiración ha sido obtenida gracias al repositorio de referencia mencionado en la bibliografía ([7]).

⁶ToneMapping de Reinhard: <https://64.github.io/tonemapping/#>

6. Carga de trabajo y metodología

Se utilizó un repositorio en GitHub para gestionar el código de manera compartida y registrar el progreso del proyecto. Los commits se emplearon como un mecanismo para documentar las tareas, ya fueran pendientes o completadas. Aunque el trabajo fue mayoritariamente individual, con tareas asignadas a cada miembro, también se realizaron sesiones conjuntas en momentos clave, especialmente cuando era necesario modificar código crítico o abordar decisiones importantes.

Además, se elaboró una lista de prioridades que seguimos estrictamente para las extensiones implementadas. Esto nos permitió mantener el enfoque en los aspectos más relevantes del proyecto y evitar desviarnos hacia objetivos secundarios.

Módulo	Manel	Ming Tao
Geometrías base	2h	6h
ToneMapping	5h	2h
Interacción rayo-escena	13h	15h
Path Tracer	28h	30h
Materiales	2,5h	1,5h
Luces de área NEE	0h	24h
Paralelización	0,5h	1h
Reinhard	1h	0h
Geometrías adicionales	12h	7h
Texturas	11h	8h
Memoria	7h	10,5h
Total	82h	105h

Tabla 6.1: Análisis de carga de trabajo

7. Otros renders

7.1. Luz puntual

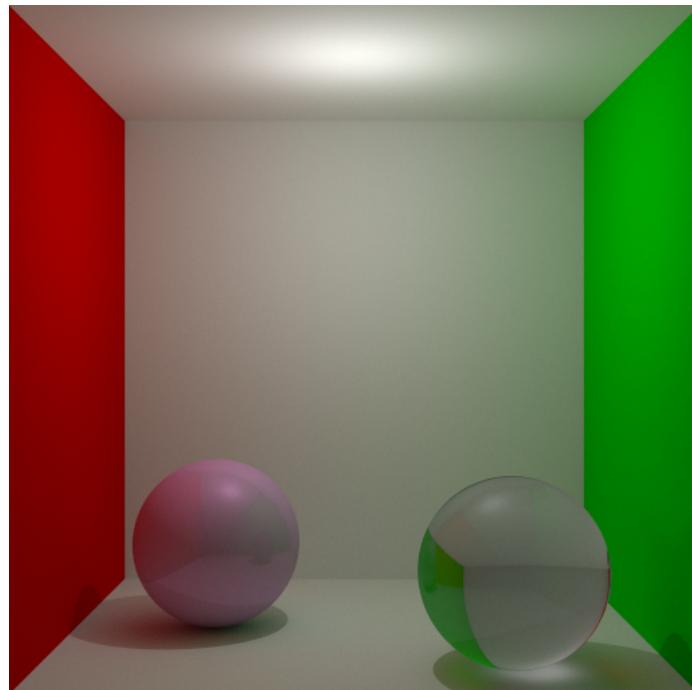


Figura 7.1: Luz puntual con esferas de plástico y cristal

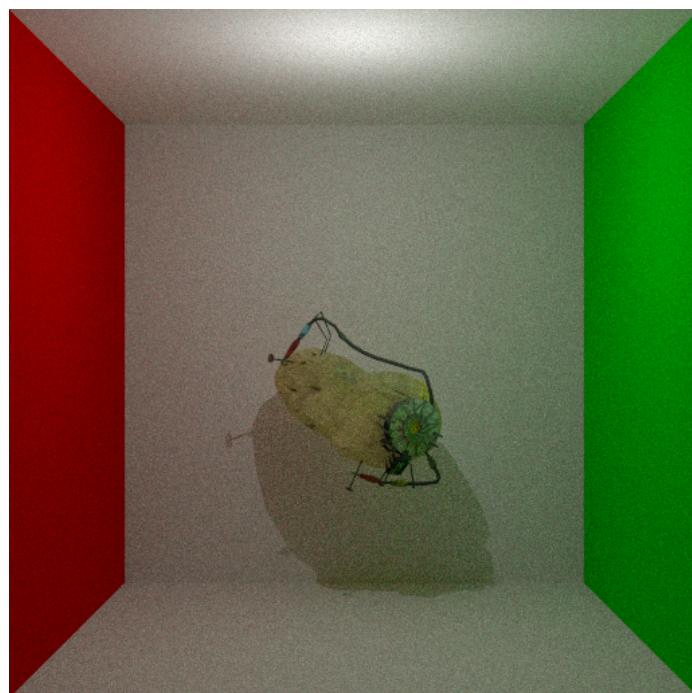


Figura 7.2: Luz puntual con modelo de una patata



Figura 7.3: Luz puntual con modelo de una tarta

7.2. Luz de área

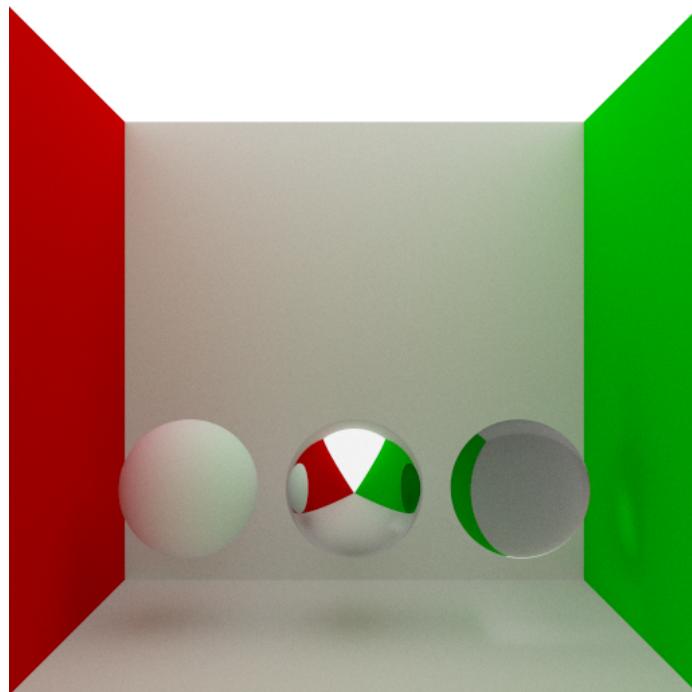


Figura 7.4: Luz de área infinita con esferas difusa, especular y refractante

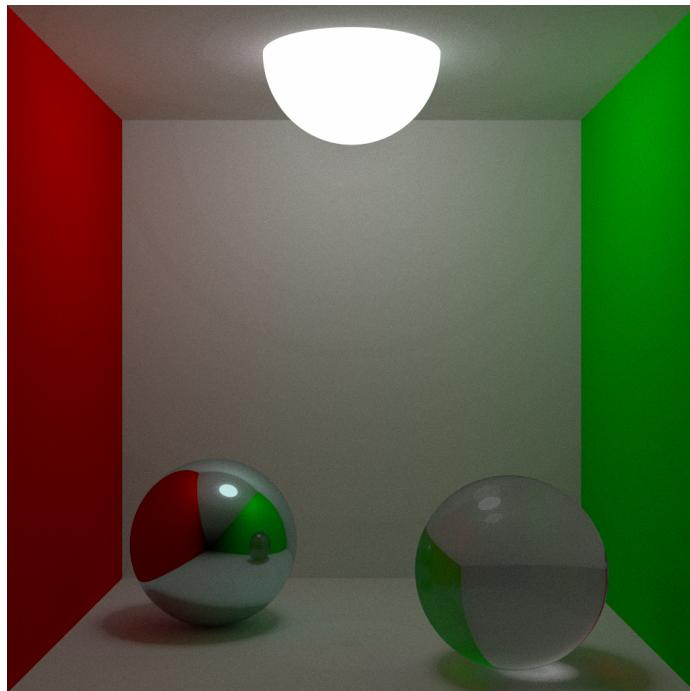


Figura 7.5: Luz de área esférica con esferas especular (tintada de azul) y cristal

8. Bibliografía

- [1] *Algoritmo de Möller–Trumbore para intersección rayo-triángulo.* URL: https://en.wikipedia.org/wiki/M%C3%B3ller%20-%E2%80%93Trumbore_intersection_algorithm. (accessed: 09.01.2025).
- [2] *Herramienta IA de apoyo y consulta.* URL: <https://chatgpt.com/>. (accessed: 09.01.2025).
- [3] *Modelo 3D de patata (portal).* URL: <https://sketchfab.com/3d-models/potato-glados-from-portal-2-a81175115d544237a19070c582882f02>. (accessed: 09.01.2025).
- [4] *Modelo 3D de tarta (portal).* URL: https://www.models-resource.com/pc_computer/portal/model/15917/. (accessed: 09.01.2025).
- [5] *Physically Based Rendering: From Theory To Implementation.* URL: <https://www.pbr-book.org/4ed/contents>. (accessed: 03.01.2025).
- [6] *Renderizando un triángulo.* URL: <https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection.html>. (accessed: 09.01.2025).
- [7] *Repositorio de referencia de José Daniel Subías.* URL: <https://github.com/dsubias/IG-mini-PT>. (accessed: 09.01.2025).
- [8] *Repositorio de referencia de Julia Guerrero Viu.* URL: <https://github.com/juliagviu/ComputerGraphics>. (accessed: 09.01.2025).
- [9] *Transparencias de teoría de Informática Gráfica.* URL: https://moodle.unizar.es/pluginfile.php/12027972/mod_folder/content/0/02-geometry.pdf. (accessed: 09.01.2025).

- [10] *Transparencias y guión de las prácticas de laboratorio de Informática Gráfica.* URL: https://moodle.unizar.es/add/pluginfile.php/12027997/mod_label/intro/04-pathtracing-es.pdf?time=1729241807345. (accessed: 09.01.2025).