Universidad Zaragoza

Trabajo Fin de Grado

Renderizado adaptativo mediante técnicas de cuadratura

Adaptative rendering using quadrature techniques

Autor

Miguel Crespo Castaño

Director

Adolfo Muñoz Orbañanos

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Miguel Crespo Castaño,

con nº de DNI 73027024T en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado de Ingeniería Informática, (Título del Trabajo) Renderizado adaptativo mediante técnicas de cuadratura

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 29 de Junio de 2018

Fdo: Miguel Crespo Castaño

Agradecimientos

Este proyecto se ha desarrollado en el *Graphics and Imaging Lab* de la Universidad de Zaragoza. Quiero agradecer a todos sus miembros, y especialmente a Ibón, por haberme hecho sentir como si estuviera en casa y por toda la ayuda prestada durante los meses que he podido estar con ellos.

Me gustaría darle las gracias especialmente a Adolfo Muñoz, mi director, por su experiencia, apoyo y ayuda durante la realización de este trabajo.

Finalmente me gustaría dar las gracias a mi familia, y especialmente a mis padres, por el apoyo que me han brindado durante toda mi vida, ya que sin ellos, no habría llegado a ser quien soy hoy en día.

Resumen

La generación de imágenes fotorrealistas es un campo muy complejo, el cual es tratado por la informática gráfica. La dificultad del problema radica en que para generarlas es necesario la simulación físicamente correcta de todas las posibles interacciones de la luz con el medio.

Las técnicas más utilizadas son las basadas en métodos estocásticos, pero debido a su comportamiento aleatorio generan ruido en la imagen final. Dicho ruido es visible principalmente en aquellas zonas perceptualmente suaves y para eliminarlo son necesarios procesos de cálculo que pueden llegar a ser muy largos.

Se presenta en este trabajo un método distinto basado en técnicas de cuadratura cuya convergencia es mejor que la de los métodos estocásticos en zonas con iluminación suave. Dicho método funciona de una manera determinista utilizando un esquema adaptativo, el cual le permite distribuir el trabajo hacia aquellas zonas de mayor dificultad (dada una estimación del error de la escena) reduciendo el tiempo de cálculo y/o incrementando la precisión. Para ello se utiliza una estructura de datos optimizada para este problema que minimiza el tiempo de cómputo.

Finalmente se estudiará la utilización de un método híbrido entre la aproximación estocástica y el método propuesto que busca aprovechar las ventajas de cada uno y combinarlas.

Todo ello será integrado sobre la arquitectura del software de renderizado **Mitsuba** y validado frente a una técnica establecida como estándar llamada *Path Tracer* implementada en dicho software.

Índice

Capítulo 1

Introducción

1.1. Objetivo, alcance y contexto del proyecto

La simulación de iluminación global consiste en el cálculo físicamente correcto de todas las posibles interacciones de la luz con el medio.

Aunque nuevos métodos han sido desarrollados para su cálculo, lidiar con el ruido aleatorio de los métodos estocásticos sigue siendo un tema pendiente. Muchas propuestas han sido planteadas, las cuales principalmente se basan en la reducción de la varianza propia de la aleatoriedad, o en la realización de filtrados posteriores sobre el resultado final. En este trabajo se propone una nueva técnica determinista, la cual no genera el ruido resultado de las técnicas estocásticas.

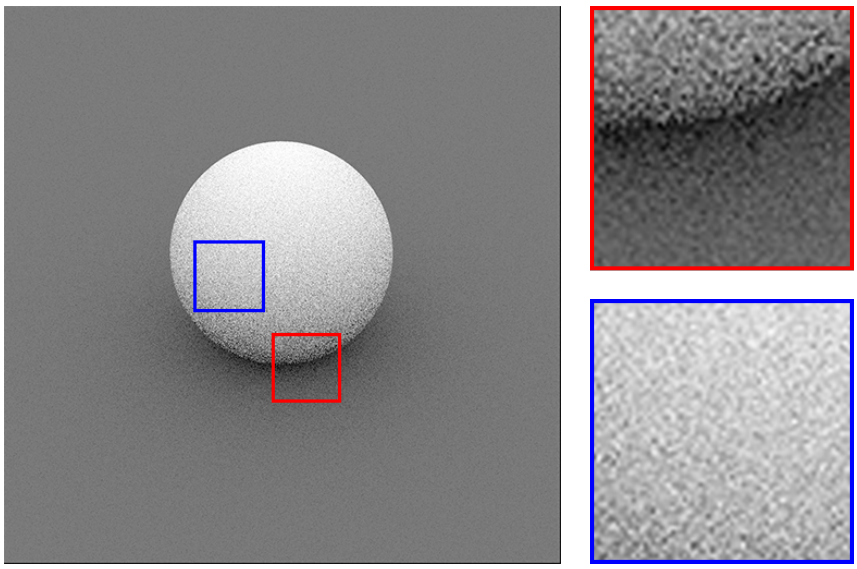


Figura 1.1: Ejemplo del ruido aleatorio generado por un método estocástico en una escena cuya iluminación es perceptualmente suave debido a su incapacidad de aproximar correctamente funciones con un componente suave.

El objetivo de este trabajo es obtener un método alternativo que converja mejor en zonas con iluminación donde las técnicas estocásticas encuentran un mayor problema

mediante la resolución analítica de la integral que modela el transporte de luz mediante el uso de técnicas de cuadratura.

El problema de las técnicas de cuadratura respecto a Monte Carlo es la memoria que necesitan debido a la maldición de la dimensionalidad, la cual expone que conforme aumentan las dimensiones de la integral a aproximar, el número de muestras necesarias crece exponencialmente. Ello muestra el principal problema de su utilización. Por tanto, si se quiere obtener una nueva técnica que rivalice, es necesaria una gestión de memoria correcta y eficiente, pero al mismo tiempo se necesitan que los cálculos sean lo más rápidos posibles, dado que la técnica con la se comparará se encuentra muy optimizada.

Finalmente, para su utilización se integrará en un software de renderizado utilizado frecuentemente en investigación, comparándolo con un método establecido como estándar y mostrando como se ha conseguido mejorar sus resultados.

El alcance de este proyecto incluye:

- Diseño de un método para aproximar la integral del transporte de luz mediante técnicas de cuadratura.
- Desarrollo de una estimación del error cometido en la aproximación con las técnicas de cuadratura y una estimación de las discontinuidades por dimensión.
- Creación de un esquema adaptativo que permita centrar el trabajo en aquellas zonas más complejas, reduciendo el tiempo de cálculo y/o incrementando la precisión.
- Desarrollo de un método híbrido estocástico y analítico.
- Integración del nuevo método en un software de renderizado utilizado en la investigación.
- Comparación de los resultados obtenidos con un método establecido como estándar.

1.2. Organización del proyecto

Primero se explicará el marco teórico necesario para comprender el trabajo en el Capítulo 2. En el Capítulo 3 se comentarán trabajos que tienen relación con este proyecto debido a que han influido en él, dado que han aportado conocimiento, o porque tratan el mismo tema desde otras perspectivas. Posteriormente , en el Capítulo 4 se explicará en profundidad la idea del nuevo método, consistiendo en la forma de calcular la integral n-dimensional que modela el transporte de luz, los diferentes

estimadores utilizados, la generación de la imagen, el esquema adaptativo y finalmente el método híbrido. En el Capítulo 5 se hablará del software donde se ha integrado y de su implementación en él, así como de los problemas y las decisiones que se han tomado durante esa etapa incluyendo la gestión de memoria y la estructura de datos utilizada. Finalmente, en el Capítulo 6 se mostrarán los resultados del nuevo método comparándolo con un estándar para validarlo y finalmente, en el Capítulo 7 se expondrán las conclusiones dados los resultados, el trabajo futuro y la planificación del proyecto.

1.3. Planificación

Durante los 9 meses aproximados de duración de este proyecto, el trabajo se ha divido en las tareas mostradas en la figura 1.2, la cual expresa la evolución temporal de este proyecto desde que se inició en Octubre de 2017 hasta que finalizó en Junio de 2018.

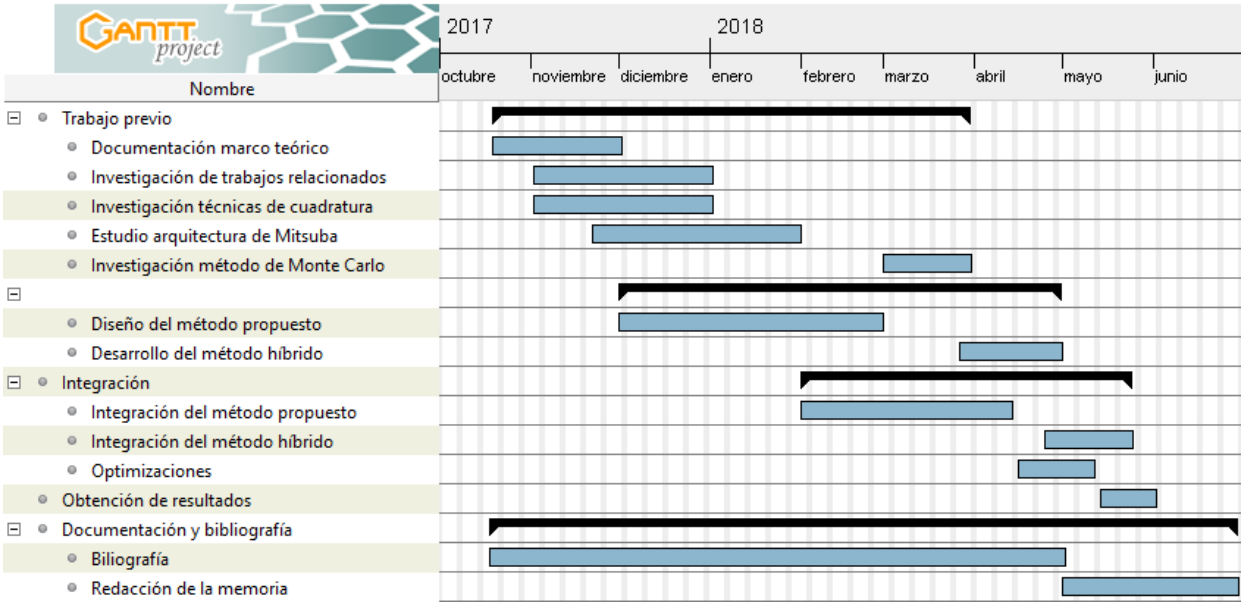


Figura 1.2: Diagrama temporal del proyecto

Capítulo 2

Marco teórico

En este capítulo se describen varios conceptos tanto teóricos como prácticos que se utilizan a lo largo de este trabajo. La intención de este capítulo es la de hacer de este documento un conjunto de información autocontenida. Sin embargo, explicar en detalle cada concepto sería demasiado extenso. Por ello, se explican los fundamentos de cada concepto que se consideran suficientes y necesarios para sustentar la teoría de este proyecto.

2.1. Transporte de luz

La simulación del transporte de luz funciona de manera análoga a la fotografía. Las imágenes se generan calculando la radiancia incidente en un sensor virtual proveniente de la escena. Para ello hay que calcular las posibles interacciones de la luz desde que es emitida por las fuente de luz hasta que llega al sensor.

2.1.1. Ecuación de Render

La fórmula que calcula la radiancia es la **Ecuación de Render** [Kaj86], la cual formula que la radiancia que llega a un punto desde una dirección es la radiancia emitida más la radiancia reflejada proveniente de todas las direcciones en la hemiesfera orientada con la normal de la superficie.

$$L_o(x, \omega_0) = L_e(x, \omega_0) + \int_\Omega f_r(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot n) d\omega_i \tag{2.1}$$

siendo:

- $\mathbf{L_o}(\mathbf{x}, \omega_\mathbf{0})$ es la radiancia emitida desde x hacia ω_0.
- $\mathbf{x}$ es la posición en el espacio.
- $\omega_\mathbf{o}$ es la dirección de la radiancia saliente.

- $\mathbf{L_e}(\mathbf{x}, \omega_\mathbf{0})$ es la radiancia emitida desde el punto x en la dirección ω_0.
- $\mathbf{\Omega}$ es la hemiesfera de radio unidad centrada alrededor de la normal de la superficie.
- $\omega_\mathbf{i}$ es la dirección negada de la radiancia incidente.
- $\mathbf{f_r}(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o})$ (*Bidireccional Reflectance Distribution Function*) es una función que define cómo se refleja la luz en una superficie dadas la dirección incidente ω_i y la de salida ω_0.
- $\mathbf{L_i}(\mathbf{x}, \omega_\mathbf{i})$ es la radiancia que llega al punto x dada una dirección ω_i.
- $\mathbf{n}$ es la normal de la superficie.
- $\omega_\mathbf{i} \cdot \mathbf{n}$ es la ley del coseno de Lamberg[1]. También puede escribirse como $cos(\theta_i)$

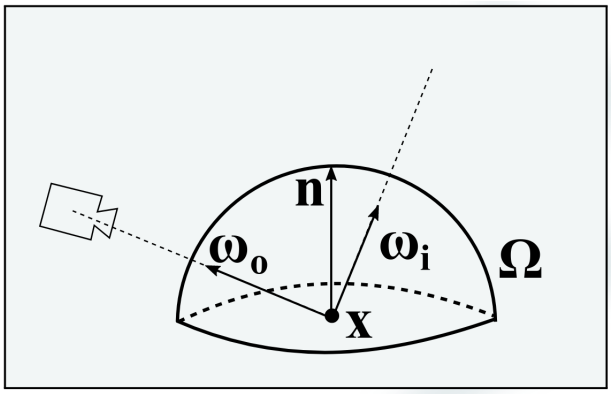


Figura 2.1: Fenómeno descrito en la ecuación de render: Radiancia que llega al sensor desde un punto x proveniente de la dirección w_i [a]

[a]Usada con permiso de Graphics and Imaging Lab

Se ha omitido por simplicidad la parametrización en la ecuación 2.1 que modela la longitud de onda y el instante de tiempo. Esta formulación del transporte de luz es recursiva, dado que la evaluación de L_i vuelve a ser la Ecuación de Render. Un ejemplo es la figura 2.2

2.1.2. Path Integral

Otra forma de formular el transporte de luz es mediante la **Path Integral** [Vea97], la cual modela que la radiancia incidente en el sensor es el total de las contribuciones

[1]La ley de Lamberg determina que la iluminación producida por una fuente de luz sobre una superficie es directamente proporcional a la intensidad de la fuente y al coseno del ángulo que forma la normal a la superficie con la dirección de la radiancia entrante.

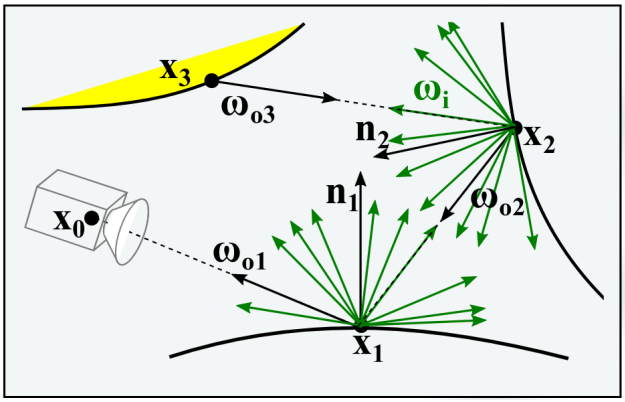


Figura 2.2: Ejemplo de la recursividad que se da en la Ecuación de Render para el cálculo del transporte de luz. Notesé como en los puntos x_1 y x_2 se necesita calcular la radiancia provenientes de todas las direcciones representadas con las flechas verdes [a]

[a] Usada con permiso de Graphics and Imaging Lab

de todos los posibles caminos que ha podido tener la luz, los cuales han contribuido al sensor.

$$I_j = \int_{\Omega} f_j(\overline{x})\, d\mu(\overline{x}) \tag{2.2}$$

donde

- $\mathbf{\Omega}$ es el conjunto de todos los posibles caminos de cualquier longitud que contribuyen y que van desde una fuente de luz hasta el sensor.
- $\mathbf{f_j}$ es la *Measurement Contribution Equation* la cual designa la contribución de cada camino.
- μ representa el espacio de caminos, el cual puede representarse como una secuencia de números en el intervalo [**0.**,**1**]

La contribución $f_j(\overline{x})$ de un camino lumínico $\overline{x}$ se calcula, dados unos vértices $x_0, x_1, ..x_n$ donde la luz ha tenido interacción con el medio, de la siguiente manera:

$$f_j(\overline{x}) = L_e(x_0)G(x_0 \leftrightarrow x_1)\left[\prod_{i}^{k-1} \rho(x_i)G(x_i \leftrightarrow x_{i+1})\right] W_e(x_k) \tag{2.3}$$

donde

- $\mathbf{L_e}$ es la radiancia emitida por la fuente de luz.
- $\mathbf{G(x_i \leftrightarrow x_{i+1})}$ es la atenuación entre dos vértices debido a la distancia entre ellos y al término geométrico.

– $\rho(\mathbf{x_i})$ es la **BSDF**, la cual modela como se refleja la luz en la superficie dada por el vértice.

– $\mathbf{W_e}(\mathbf{x_k})$ es la respuesta del sensor ante la radiancia incidente.

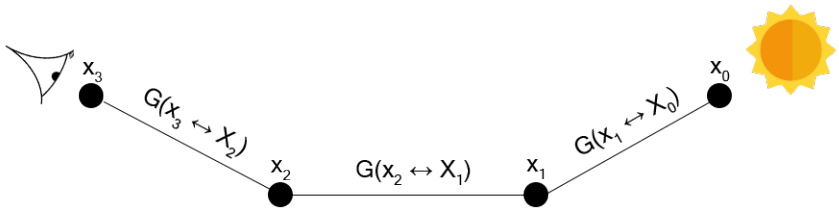


Figura 2.3: Ejemplo de como es la atenuación en un camino de cuatro vértices desde el sensor hasta una fuente lumínica.

Por tanto se puede eliminar la recursividad de la Ecuación de Render (2.1.1) debido a que ahora solo se calcula la radiancia en un camino. Por tanto, si se itera sobre el total de caminos, se puede seguir calculando la radiancia total de la escena.

Debido a su formulación, dependiendo de cuantas interacciones se modelen con la *Path Integral* (2.2), se irán incrementando las dimensiones de la integral.

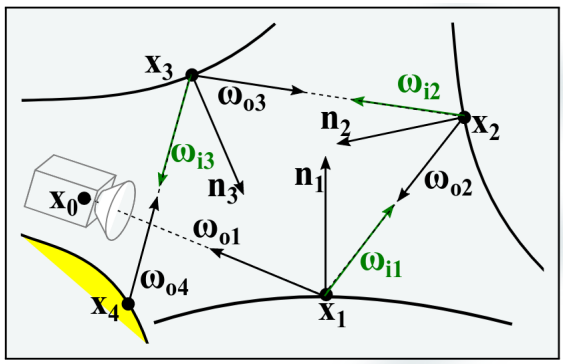


Figura 2.4: Ejemplo donde la recursividad se ha eliminado , debido a que ahora la contribución final solo es calculada para el camino definido por $x_0.x_1, x_2, x_4$[a]

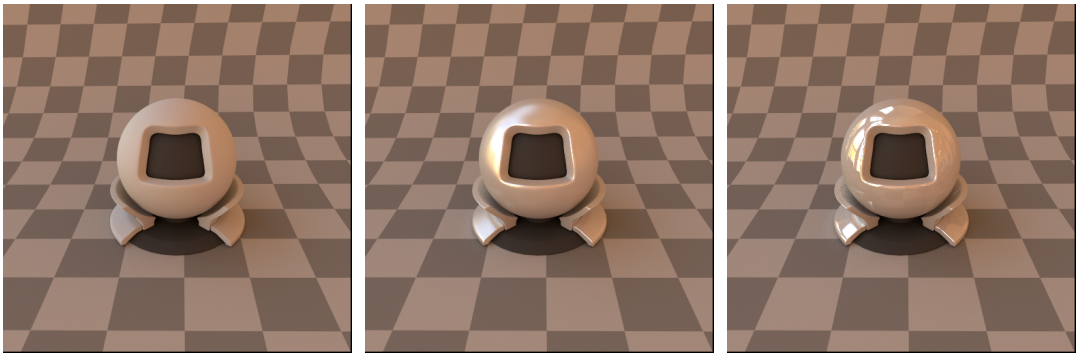
[a]Usada con permiso de Graphics and Imaging Lab

2.1.3. BSDF - *Bidirectional scattering distribution function*

La función bidireccional de distribución de la dispersión o **BSDF** (por sus siglas en ingles) es una función que define como la luz se refleja en una superficie opaca

dadas una dirección entrante y una dirección saliente, lo cual define la apariencia de la misma. Para que sea físicamente correcta debe cumplir el principio de conservación de la energía, es decir, la energía saliente debe de ser igual o menor que la energía incidente.

En función de como se distribuye la energía, se habla de BSDF difusas cuando la energía se distribuye uniformemente en todas las direcciones, o especulares cuando la energía se concentra alrededor de la dirección de salida. Un ejemplo del comportamiento de las diferentes BSDF es la figura 2.5.



(a) Difuso puro. (b) Brillante o *glossy*. (c) Especular perfecto

Figura 2.5: Ejemplo de los diferentes tipos de BSDF.

2.2. Interpolación polinómica

La interpolación mediante polinomios consiste en la obtención del polinomio de menor grado posible que pasa por un conjunto de puntos dados.

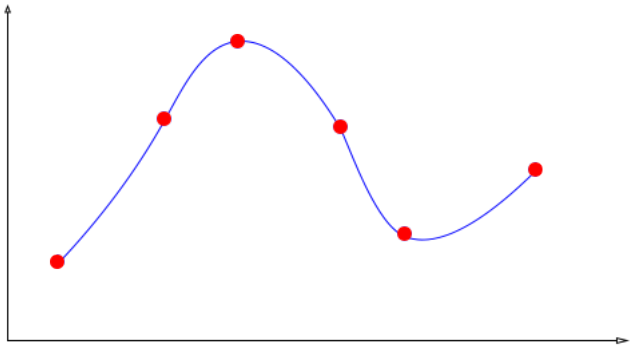


Figura 2.6: Ejemplo de la idea: dados los puntos rojos se obtiene un polinomio que pasa por ellos.

Dependiendo del grado de dicho polinomio, se necesitarán más o menos puntos para obtenerlo. Se utiliza principalmente cuando se tiene una función de la cual se conocen varios puntos y se necesita aproximar los valores que toma fuera de dichos puntos.

2.3. Integración numérica

La integración numérica consiste en una serie de algoritmos utilizados para calcular el valor numérico de una integral definida. Es también utilizado el término cuadratura numérica (o simplemente cuadratura) para referirse de igual manera a la integración numérica, y aunque habitualmente se utiliza en el caso de integrales de una única dimensión, también se utiliza para referirse en algunos casos a integrales de más dimensiones.

2.3.1. Técnicas de cuadratura

Cuando la integral que se pretende calcular no tiene expresión analítica se utilizan las técnicas de cuadratura, las cuales son métodos de integración numérica que pueden ser descritos generalmente como una combinación de evaluaciones del integrando para obtener una aproximación a la integral.

$$\int_a^b f(\mathbf{x})dx \quad \simeq \quad (b-a)\sum_{i=1}^{N} w_i f(\mathbf{x_i}) \tag{2.4}$$

donde N, w_i y x_i son parámetros dependientes de la técnica.

Hay una extensa familia de métodos que se basan en aproximar la función a integrar $f(x)$ por otra función $g(x)$ de la cual se conoce la integral exacta. La función que sustituye a la original se encuentra mediante interpolación polinómica de forma que en un cierto número de puntos tenga el mismo valor que la original.

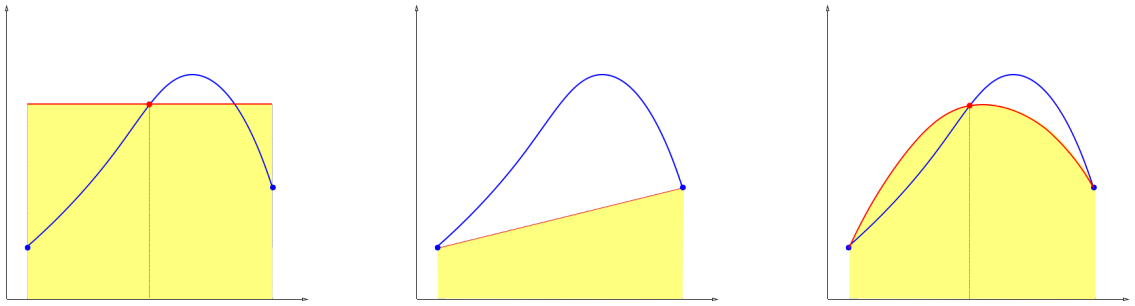
La interpolación con polinomios evaluados en puntos equidistantes en $[a, b]$ da las fórmulas de *Newton-Cotes*. Son ejemplos de ellas la regla del punto medio (polinomio interpolador de grado 0, utiliza un punto), regla del trapecio (polinomio interpolador de grado 1, utiliza dos puntos) o la regla de Simpson (polinomio interpolador de grado 2, utiliza tres puntos) Un ejemplo se encuentra en la figura 2.7.

2.3.2. Técnicas de cuadratura anidadas

Las técnicas de cuadratura anidadas aplican simultáneamente una regla de grado mayor y otra con grado menor para dar una estimación de la solución y del error.

$$\hat{F^H} = \sum_{i=1}^{n} w_i^H f\left(x_i^H\right) \qquad \hat{F^L} = \sum_{i=1}^{m} w_i^L f\left(x_i^L\right) \tag{2.5}$$

siendo $\hat{F^H}$ y $\hat{F^L}$ el resultado de la regla de mayor y menor grado respectivamente $(n > m)$. Los correspondientes pesos $\left\{w_i^H\right\}$ y $\left\{w_i^L\right\}$ así como los puntos utilizados



(a) Regla del punto medio (b) Regla del trapecio (c) Regla de Simpson

Figura 2.7: Differentes ejemplos del resultado de aplicar diferentes reglas de cuadratura sobre la misma función en el mismo intervalo.

$\{x_i^H\}$ y $\{x_i^L\}$ son específicos de cada regla.

2.3.3. Teorema de Fubini

Las técnicas de cuadratura suelen definirse en una sola dimensión. Por tanto si se quiere resolver una integral multidimensional mediante técnicas de cuadratura, uno puede expresar la integral de la siguiente manera (dado el teorema de Fubini):

$$\int\int_R f(x,y)dydx = \int_a^b \left(\int_c^d f(x,y)dy\right) dx = \int_c^d \left(\int_a^b f(x,y)dx\right) dy \tag{2.6}$$

Dicho teorema explica que si la función es continua en la región rectangular a integrar, es posible calcular una integral doble utilizando integración iterativa y que el orden en el cual se haga no modifica el resultado.

Esto aplicado, por ejemplo, con la regla de Simpson en dos dimensiones sería lo siguiente:

$$\begin{aligned}\int_0^1 \left(\int_0^1 f(x,y)dx\right) dy \;&\approx\; \frac{1-0}{6}\int_0^1 f(x,0) + 4f(x,0{,}5) + f(x,1)\; dx \\ &= \frac{1-0}{6}\left[\frac{1-0}{6}\Big(f(0,0) + 4f(0{,}5,0) + f(1,0)\Big)\right. \\ &\quad + \frac{1-0}{6}\Big(f(0,0{,}5) + 4f(0{,}5,0{,}5) + f(1,0{,}5)\Big) \\ &\quad \left. + \frac{1-0}{6}\Big(f(0,1) + 4f(0{,}5,1) + f(1,1)\Big)\right]\end{aligned} \tag{2.7}$$

donde se ha integrado primero sobre la variable y y luego sobre la variable x.

2.3.4. Método de Monte Carlo

Otra técnica de integración numérica es el Método de Monte Carlo, la cual está basada en números aleatorios. Mientras que otras técnicas evalúan la integral en un número fijo de puntos predefinidos, Monte Carlo elige pseudo-aleatoriamente los puntos a evaluar (llamados **muestras**). Dado que se tiene control sobre el número de puntos que se van a utilizar, esto lo hace especialmente útil en el cálculo de integrales con un alto número de dimensiones debido a que no sufre de la **maldición de la dimensionalidad**.

La **maldición de la dimensionalidad** consiste en que cuando se aumenta la dimensionalidad del espacio, aumenta exponencialmente el volumen del espacio, por lo cual aumenta exponencialmente el número de puntos necesarios para calcularlo.

En el caso de Monte Carlo, como no necesita de un número fijo de puntos para calcular cada dimensión, no le afecta. En el caso de las técnicas de cuadratura, como se necesitan de un número fijo de puntos por dimensión, son afectadas gravemente por este efecto.

La fórmula del método de Monte Carlo es la siguiente:

$$I \simeq Q_n \equiv V * \frac{1}{N} \sum_{i=1}^{N} f(\overline{x}_i) \tag{2.8}$$

siendo V el volumen del dominio a integrar y N el número de puntos que se utilizan.

Su convergencia está asegurada dada la ley de grandes números, la cual expone que con un número infinito de muestras, el resultado de Monte Carlo es el valor de la integral.

$$\lim_{x \to \text{ínf}} Q_n = I \tag{2.9}$$

Capítulo 3

Trabajo relacionado

La generación de imágenes sin ruido es un campo de investigación todavía en constante desarrollo debido a la complejidad del problema. En este capitulo, se van a resumir aquellos trabajos relacionados con este proyecto que tratan de solucionarlo.

Si se utilizan técnicas estocásticas, una forma de evitar que se genere ruido es mediante la toma de muestras que maximice la importancia de cada una. Para ello, Hachisuka et al. [Hac+08] proponen que no se tomen las muestras teniendo en cuenta los píxeles, como se haría normalmente, sino que se distribuyan dependiendo del espacio n-dimensional de la integral. Dicha aproximación permite obtener muestras libres de ruido. Su algoritmo se basa en dos fases: en la primera fase distribuyen las muestras en el espacio centrándose en aquellas zonas con un contraste mayor, es decir, concentran adaptativamente las muestras en aquellas zonas que se consideran más complicadas. En una segunda fase, integran las muestras obtenidas previamente en todas las dimensiones menos en el espacio de imagen.

Otra forma de abordar la generación de imágenes sin ruido es mediante la sustitución de los métodos estocásticos por técnicas analíticas. En su trabajo, Muñoz [Mn14] analiza la utilidad de utilizar técnicas de cuadratura en el cálculo de la interacción de la luz en medios participativos debido a que son capaces de adaptarse matemáticamente a la función subyacente. Para ello analiza la regla de cuadratura de Simpson anidada con la regla del trapecio, comprobando su convergencia y estabilidad ante singularidades.

Una aproximación con ideas similares es el trabajo de Fabre [FHMO16]. En él, presenta un método basado en las mismas reglas de cuadratura anidadas que Muñoz pero aplicadas sobre la *Path Integral* para la generación de imágenes libres de ruido. Este trabajo se basa en dichas ideas.

Capítulo 4

Diseño del nuevo método

En este capítulo se va a hablar de las características del nuevo método, así como de las decisiones tomadas. Primero se expondrá la manera de aproximar la *Path Integral* y lo que conlleva hacerlo de esta forma. A continuación se comentará el estimador de error que se utiliza durante la fase adaptativa y la forma de elegir la mejor dimensión a subdividir para disminuir el error. Posteriormente se comentará la idea del algoritmo adaptativo y se expondrá la forma de obtener, dada la aproximación realizada, la radiancia de cada píxel en la imagen final. Finalmente se introducirá el método híbrido con Monte Carlo.

4.1. Aproximación de la ecuación de Render

Como se comentó en la sección 2.1.1, la ecuación de render (2.1) tiene un comportamiento recursivo, el cual es indeseado debido a la cantidad de cálculos que serían necesarios para su evaluación. Otra aproximación, también comentada en la sección 2.1.2 es la *Path Integral* (2.2), la cual sustituye la recursividad por un componente iterativo. El problema con dicha formulación es el número de dimensiones de la integral que necesitan calcularse, dados los rebotes de la luz que se simulen. Dicho problema a los métodos basados en Monte Carlo no les afecta, pero si se utilizan técnicas de cuadratura, es un factor determinante.

Cada técnica de cuadratura viene determinada por el número de puntos necesarios para aproximar la función. Se han elegido dos reglas anidadas para este trabajo: la regla de Boole (cinco puntos) y la regla de Simpson (tres puntos). El utilizar una regla de cinco puntos permite una mayor precisión en la aproximación más luego la reutilización de dichos puntos para la subdivisión selectiva de la dimensión de mayor error (sección 4.2). Por ello se la consideró como la elección idónea. En el caso de la regla de Simpson, se ha elegido para ser utilizada en la estimación del error debido a que tiene menor grado que la regla de Boole y no se necesitan de nuevos puntos para su cálculo (se

reutilizan tres de los cinco puntos utilizados en la regla de Boole).

Por tanto, se tiene que la *Path Integral* (2.2) puede representarse como una integral n-dimensional cuyo dominio de integración va de 0 a 1.

$$I_j = \int_\Omega f_j(\overline{x})\, d\mu(\overline{x}) \equiv \int_0^1 .. \int_0^1 f_j(x_1, x_2, ..., x_n)\, dx_1 ... dx_n \tag{4.1}$$

cuyas dimensiones pueden representarse de la siguiente manera:

- **Dimensión 0** sería el eje x en el espacio de pantalla.
- **Dimensión 1** sería el eje y en el espacio de pantalla.
- **Dimensiones 2 y 3** sería el primer rebote indirecto de la luz.
- **Dimensiones 4 y 5** sería el segundo rebote indirecto de la luz.
- **Dimensiones 6..n** serían los siguientes rebotes indirectos.

Dada la ecuación (4.1), se puede aproximar en una dimensión utilizando por ejemplo la regla de Simpson, dando:

$$\int_0^1 f_j(x)\, dx \quad \approx \quad \frac{h}{3}\left(f(0) + 4f(0{,}5) + f(1)\right) \tag{4.2}$$

siendo h la distancia entre los puntos equidistantes utilizados en la regla de cuadratura.

La aproximación realizada en la ecuación (4.2) solo sirve en el caso de que la integral sea de una dimensión, dado que la técnica de cuadratura es unidimensional. Si se quiere extrapolar al caso multidimensional, dado el Teorema de Fubini descrito en la sección 2.3.3, hay que integrar recursivamente en cada dimensión. Un ejemplo sobre como se calcularía el caso 2-dimensional es la ecuación (2.7).

Con ello se consigue una aproximación correcta de la *Path Integral* siempre y cuando la función a aproximar tenga un comportamiento suave. Debido a que puede no darse el caso, para incrementar la precisión de la aproximación se puede subdividir el intervalo a integrar en un número **n** de subintervalos.

4.2. Estimación del error total y por dimensión

Para que el esquema adaptativo funcione correctamente se necesita de una estimación del error cometido.

Muchas aproximaciones se han dado para obtener una estimación del error más precisa. Dada la categoría de estimadores que son lineales, se ha concluido que los

distintos tipos que cumplen con dicha característica son equivalentes entre sí [Gon10] [Lau85].

Por tanto, en este trabajo se ha optado por la utilización de la diferencia absoluta entre la regla de Boole y la regla de Simpson como estimación del error. Este estimador entra dentro de la categoría de estimadores lineales como diferencia absoluta entre reglas de distinto grado [Gon10].

$$\xi \approx |Q_{n1}[a,b] - Q_{n2}[a,b]| \tag{4.3}$$

Como el resultado de la integral es un vector, la estimación final del error contando con cada componente es la siguiente:

$$\xi \approx \|Q_{n1}[a,b] - Q_{n2}[a,b]\|_1 \tag{4.4}$$

Aparte de dicha estimación, dado que se está trabajando sobre un espacio en píxeles, se puede incrementar la fiabilidad de la estimación añadiéndole sesgo hacia aquellas divisiones que ocupan un gran espacio en la imagen. Por tanto, la estimación final del error es la siguiente:

$$\xi \approx \|Q_{n1}[a,b] - Q_{n2}[a,b]\|_1 * \left(1 + \frac{nPixels}{totalPixels}\right) \tag{4.5}$$

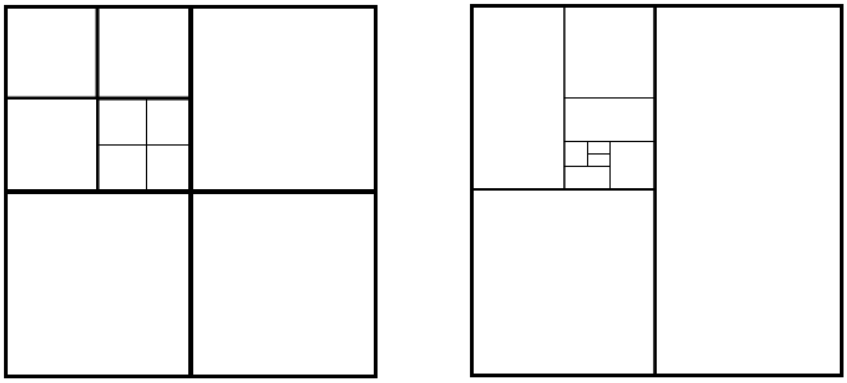
siendo $nPixels$ el número de píxeles influenciados por la división y $totalPixels$ el total de los píxeles de la imagen.

Habiendo obtenido una estimación del error, si dicha división tiene un error mayor que el umbral establecido, se procedería a subdividir el intervalo. Si se subdividen todas las dimensiones, esto conlleva que por cada operación de subdivisión se generarían $\mathbf{d^N}$ nuevas subdivisiones (siendo d el número de divisiones generadas por dimensión). Un ejemplo sería la figura 4.1.

Esto es inviable, conforme se incrementan las dimensiones a calcular, tanto por el coste prohibitivo en memoria como en cálculo. Una aproximación más correcta sería producir subdivisiones únicamente en una dimensión, puesto que cada una tiene una dificultad diferente a la hora de aproximarla.

Una forma de obtener una estimación de la dificultad para aproximar una dimensión es mediante un operador de cuartas diferencias [BEG91]. La idea es obtener una aproximación de cuanto ha variado la integral en esa dimensión, dados unos puntos a evaluar centrados en la dimensión que se pretende estimar.

Para ello se define el siguiente operador de cuartas diferencias, el cual utiliza 5 puntos:



(a) Ejemplo de subdivisión en todas las dimensiones.

(b) Ejemplo de subdivisiones solo en la dimensión más problemática

Figura 4.1: Tras 3 iteraciones subdiviendo en todas las dimensiones se generan 10 subdivisiones, mientras que para generar las mismas subdivisiones dividiendo en solo una dimensión se necesitan de 9 iteraciones. El control que se tiene mediante el segundo método es mucho mayor que con el primero.

$$D_i\mathbf{f} = \left\| \mathbf{f}(\mathbf{u}(\alpha_1)_i) + \mathbf{f}(\mathbf{u}(-\alpha_1)_i) - 2\mathbf{f}(\mathbf{u}) - \frac{\alpha_1^2}{\alpha_2^2}(\mathbf{f}(\mathbf{u}(\alpha_2)_i) + \mathbf{f}(\mathbf{u}(-\alpha_2)_i) - 2\mathbf{f}(\mathbf{u})) \right\|_1 \quad (4.6)$$

siendo α_1 y α_2 constantes positivas, $\mathbf{u}$ el vector cuyos componentes son las coordenadas centrales de cada dimensión y $\mathbf{u}(\alpha)_i$ es el vector resultante de $\mathbf{u}$ tras sumarle a su i-ésimo componente $\frac{\alpha v_i}{2}$, siendo v_i la longitud de la dimensión.

En este trabajo se han utilizado $\alpha_1 = 2$ y $\alpha_2 = 0{,}5$, lo cual permite reutilizar las 5 muestras utilizadas para calcular Boole durante la estimación del error. Por tanto no hay coste extra de evaluar este operador.

Al igual que sucedía con la estimación del error, aprovechándose de la información del espacio de imagen, se puede introducir sesgo hacia aquellas dimensiones que ocupen un mayor número de píxeles. Para ello se define la siguiente heurística:

$$H(i) = \begin{cases} 2 + \frac{pixelsX}{totalPixelsX}, & \text{si } i = 0 \\ 2 + \frac{pixelsY}{totalPixelsY}, & \text{si } i = 1 \\ 1, & \text{en el resto de dimensiones} \end{cases} \quad (4.7)$$

siendo i la dimensión sobre la cual se aplica la heurística, $pixelsX$ y $pixelsY$ el número de píxeles afectados por la subdivisión en el eje horizontal y vertical respectivamente de la imagen y $totalPixelsX$ $totalPixelsY$ el total de píxeles en el eje horizontal y vertical respectivamente de la imagen.

Por tanto, la estimación final de la complejidad de una dimensión viene dado por:

$$C_i = D_i \mathbf{f} * H(i) \tag{4.8}$$

La dimensión elegida para subdividir será aquella que tenga el valor máximo de aplicar esta estimación. En el caso que dos dimensiones tengan el mismo valor de estimación, se elegirá aquella de mayor longitud.

4.3. Algoritmo adaptativo

Debido a la posible existencia de oscilamiento en la función a aproximar, aplicar la regla de cuadratura sobre todo el intervalo es inviable. Como se comentó en la sección 4.1, la subdivisión en intervalos menores permite una mejor aproximación de la función.

Para realizar dichas subdivisiones de una manera óptima, se plantea que se realicen adaptativamente dado el error que haya podido cometer la regla de cuadratura en dicho subintervalo.

La idea del algoritmo es que partiendo de una subdivisión inicial, esta se vaya subdividiendo tanto como sea necesario para aproximar correctamente la función. En el momento que el error cometido sea aceptable o si se han llegado a las iteraciones máximas, se deja de subdividir y se almacena su resultado [BEG91] [GM80] [FHMO16]. El pseudo-código se encuentra presentado en el Algoritmo 1.

Para ello, se necesita una gestión de la memoria excelente y de una estructura de datos que permita obtener la división con mayor error en todo momento, con un funcionamiento lo más cercano al óptimo debido al volumen elevado de subdivisiones que se tendrán que manipular. De ello se hablará en las secciones 5.3 y 5.4 respectivamente.

El método permite una alta parametrización, permitiendo su adaptación a distintas configuraciones si es necesario. La lista de parámetros que acepta es la siguiente:

- **Iteraciones máximas**. Definen el número de operaciones de subdivisión que se realizarán.
- **Umbral de error**. Define el error mínimo para considerar una subdivisión pendiente de subdividir.
- **Subdivisiones iniciales en espacio de imagen**. Permite definir subdivisiones iniciales establecidas en las primeras dos dimensiones.
- **Subdivisiones iniciales en iluminación**. Permite establecer subdivisiones iniciales en las dimensiones a partir de la segunda.

Algoritmo 1: Pseudocódigo del algoritmo adaptativo

```
1  inicio
2      Generar subdivisiones iniciales dados parámetros de entrada
3      Introducirlas en la estructura de datos
4      iteraciones := 0
5      mientras heap no vacío y iteraciones <= ITERACIONES MÁXIMAS
        hacer
6          Obtener división con mayor error de la estructura de datos
7          Generar sus subdivisiones
8          para cada subdivisión generada hacer
9              si su error es menor que un UMBRAL entonces
10                 Obtener la radiancia por píxel
11             fin
12             en otro caso
13                 Introducirla en la estructura de datos
14             fin
15             iteraciones + = 1
16         fin
17     fin
18     mientras queden divisiones por procesar hacer
19         Obtener una división
20         Obtener la radiancia por píxel
21     fin
22 fin
```

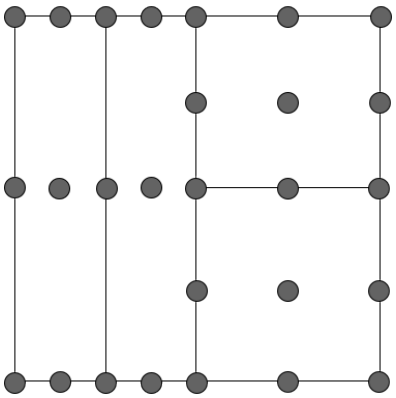
El número de iteraciones máximas define el total de muestras que se calcularán. Por tanto, este algoritmo se relaciona con los basados en Monte Carlo en que las iteraciones máximas modelan lo mismo que las muestras por píxel utilizadas por Monte Carlo. En ambos casos, permiten controlar el tiempo consumido. Un ejemplo es la figura 4.2.

Este algoritmo basa su eficacia en la calidad de la estimación del error. Cuanto más se acerque dicha estimación al error real, más certeras serán las subdivisiones y menos cálculos serán necesarios para aproximar de una mejor manera la integral.

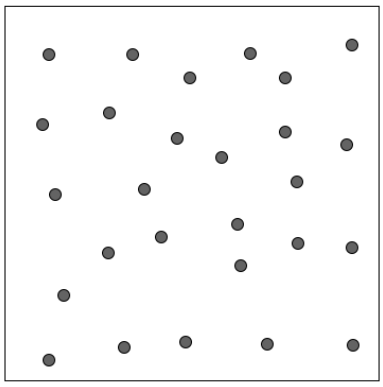
4.4. Interpolación en el espacio de imagen

Dado que las divisiones que se realizan pueden englobar varios píxeles de la imagen final, se necesita obtener la contribución de la radiancia de la división para cada uno.

Puesto que las muestras que se han obtenido son equidistantes entre sí y el dominio es rectangular, una forma de obtener la contribución por píxel es mediante el uso de un polinomio interpolador sobre las dimensiones 0 y 1 de la integral, es decir, en las dimensiones del espacio de imagen. Para ello, previamente se han tenido que integrar el resto de dimensiones de iluminación aplicando integración iterativa.



(a) Ejemplo de muestras calculadas tras 3 iteraciones del método basado en cuadratura.



(b) Ejemplo de muestras calculadas por un método basado en Monte Carlo.

Figura 4.2: Relación entre el parámetro de iteraciones máximas del algoritmo adaptativo con el de muestras por píxel de Monte Carlo. Ambos parámetros controlan el total de muestras a procesarse y por ende, el tiempo de cálculo. En ambos casos se han calculado 27 muestras (Sólo se almacenan 3 puntos por dimensión en el método de cuadratura por simplicidad).

Dado que se tienen 5 puntos ya calculados por dimensión, la mejor forma de aprovechar su información es mediante un polinomio interpolador de grado 4 (el cual utiliza todos esos puntos):

$$f(x) \approx c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \tag{4.9}$$

donde c_i son los coeficientes que definen al polinomio de grado 4.

Generalizándolo al caso bidimensional, el correspondiente polinomio de grado 4 sería:

$$\begin{aligned} f(x,y) \approx\ & c_{44}x^4y^4 + c_{43}x^4y^3 + c_{42}x^4y^3 + c_{42}x^4y^2 + c_{41}x^4y + c_{40}x^4 + \\ & c_{34}x^3y^4 + c_{33}x^3y^3 + c_{32}x^3y^3 + c_{32}x^3y^2 + c_{31}x^3y + c_{30}x^3 + \\ & c_{24}x^2y^4 + c_{23}x^2y^3 + c_{22}x^2y^3 + c_{22}x^2y^2 + c_{21}x^2y + c_{20}x^2 + \\ & c_{14}xy^4 + c_{13}xy^3 + c_{12}xy^3 + c_{12}xy^2 + c_{11}xy + c_{10}x + \\ & c_{04}y^4 + c_{03}y^3 + c_{02}y^3 + c_{02}y^2 + c_{01}y + c_{00} \end{aligned} \tag{4.10}$$

Dado el polinomio definido en la ecuación (4.10), se puede calcular su integral analítica, siendo:

$$
\begin{aligned}
I(x,y) = & y^5\left(\frac{c_{44}x^5}{25} + \frac{c_{34}x^4}{20} + \frac{c_{24}x^3}{15} + \frac{c_{14}x^2}{10} + \frac{c_{04}x}{5}\right) + \\
& y^4\left(\frac{c_{43}x^5}{20} + \frac{c_{33}x^4}{16} + \frac{c_{23}x^3}{12} + \frac{c_{13}x^2}{8} + \frac{c_{03}x}{4}\right) + \\
& y^3\left(\frac{c_{42}x^5}{15} + \frac{c_{32}x^4}{12} + \frac{c_{22}x^3}{9} + \frac{c_{12}x^2}{6} + \frac{c_{02}x}{3}\right) + \\
& y^2\left(\frac{c_{41}x^5}{10} + \frac{c_{31}x^4}{8} + \frac{c_{21}x^3}{6} + \frac{c_{11}x^2}{4} + \frac{c_{01}x}{2}\right) + \\
& y\left(\frac{c_{40}x^5}{5} + \frac{c_{30}x^4}{4} + \frac{c_{20}x^3}{3} + \frac{c_{10}x^2}{2} + c_{00}x\right)
\end{aligned} \tag{4.11}
$$

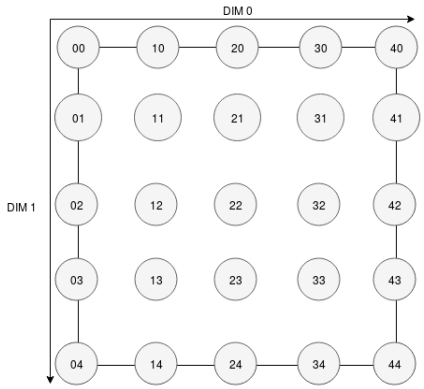


Figura 4.3: Ejemplo de las muestras tomadas en el caso bidimensional con sus coordenadas

Para el cálculo de los coeficientes del polinomio, sería necesario resolver un sistema de 25 ecuaciones con 25 incógnitas. La figura 4.3 muestra las muestras que tiene cada división, con sus subíndices.

Teniendo calculados los coeficientes que definen el polinomio interpolador de grado 4 que pasa por los 25 puntos, solo faltaría ir recorriendo cada píxel influenciado por la división, obteniendo su área dentro de la división y calculando mediante su integral analítica (4.11) la radiancia resultante.

Para ello hay que tener en cuanta las distintas configuraciones que puede tener una división sobre los píxeles. Un ejemplo de ello es la figura 4.4.

El caso más sencillo es cuando una división se alinea perfectamente con un único píxel (caso **A**). En ese caso, toda la radiancia de la división contribuye únicamente a dicho píxel. Otra configuración posible es que la división sea menor que el tamaño de un píxel (caso **B**), pero solo siga afectando a uno. En ese caso, la radiancia total recibida en el píxel será resultado de la contribución de dos o más divisiones. El resto

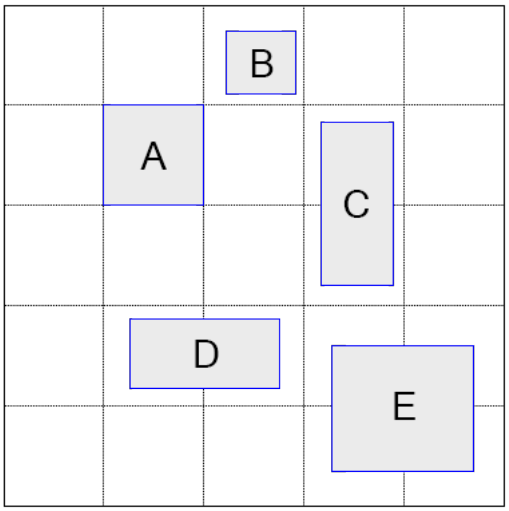


Figura 4.4: Ejemplo con las distintas configuraciones que puede tener una división sobre los píxeles de la imagen.

de casos son cuando una división engloba dos o más píxeles (tanto en el eje horizontal (caso **D**) como vertical (caso **C**) o ambos (caso **E**)). En estos casos, habrá que iterar sobre los píxeles afectados y dependiendo de su área dentro de la división, calcular cuanta radiancia contribuye.

4.5. Reducción del grado del polinomio interpolador

La idea de aproximar el valor de los píxeles con un polinomio de grado 4 proviene de que se tienen los puntos necesarios ya calculados y así se aprovecha toda la información posible. La idea intuitiva es que a mayor grado, mejor se puede adaptar el polinomio a la función subyacente. El problema es que los resultados obtenidos contienen artefactos visuales, los cuales son resultado del oscilamiento del polinomio interpolador y dicho problema se conoce como el *fenómeno de Runge*.

El **fenómeno de Runge** es un problema de oscilamiento en los extremos de un intervalo que ocurre cuando se utiliza interpolación polinómica con polinomios de grado alto con puntos equidistantes entre sí. Un ejemplo es la figura 4.5. En los puntos utilizados en la interpolación, el error de los polinomios azul y verde frente a la función roja es cero. Pero entre los puntos de interpolación y especialmente cerca de los extremos del intervalo, el error entre la función roja y los polinomios se vuelve peor conforme aumenta el grado de estos.

Por tanto, la solución es disminuir el grado del polinomio a la hora de interpolar, pero siguiendo usando toda la información posible. Por ello, se optó por eliminar el uso

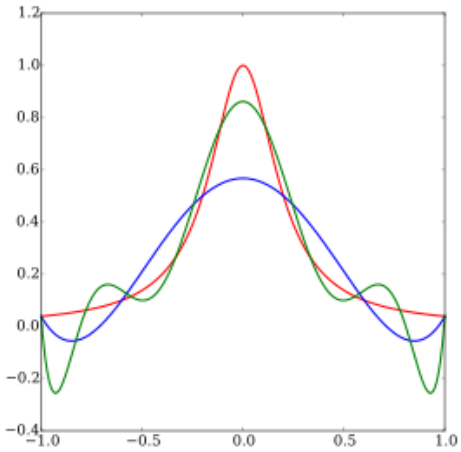


Figura 4.5: La curva roja es la función de Runge. La curva azul es un polinomio interpolador de grado 5. La curva verde es un polinomio interpolador de grado 9.
Fuente: Wikipedia

únicamente de un polinomio de grado alto por el uso de varios de menor grado. Un ejemplo sería la figura 4.6.

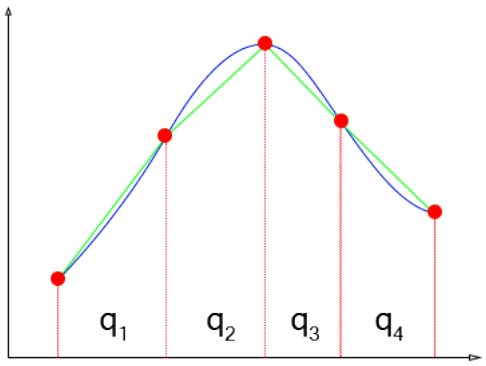


Figura 4.6: La curva azul es el polinomio de grado 4 resultado de aproximar los 5 puntos. La curva verde es el resultado de aproximar los 5 puntos mediante cuatro polinomios de grado 1 definidos en los intervalos q_i.

Aunque se probó con una combinación de polinomios interpoladores de grado 2, aún se seguían produciendo artefactos en los bordes debido a la curvatura que tienen dichos polinomios. El mejor resultado ha sido mediante la utilización de polinomios de grado 1.

Dado el intervalo $[a_x, b_x]$ y $[a_y, b_y]$ que va desde un punto muestreado hasta su adyacente, la función en esa zona se puede aproximar con un polinomio bidimensional de grado 1 como:

$$f(x, y) \approx c_{00} + c_{01}x + c_{10}y + c_{11}xy \tag{4.12}$$

Dada la ecuación (4.12), su integral analítica sería:

$$I(x, y) = \frac{c_{11}x^2y^2}{4} + \frac{c_01x^2y}{2} + \frac{c_{10}xy^2}{2} + c_{00}xy \tag{4.13}$$

Entonces, el espacio de imagen de la división se puede dividir en 16 cuadrantes, en el que cada uno tiene definido un polinomio interpolador bidimensional de grado 1. Un ejemplo sería la figura 4.7. En cada cuadrante, solo se utilizarán los puntos de sus esquinas para obtener el polinomio correspondiente. Con ello, se sigue aprovechando toda la información sin que se generen artefactos visuales.

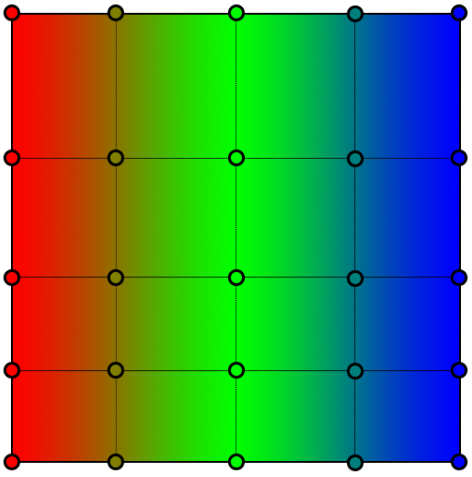


Figura 4.7: Ejemplo de la subdivisión del espacio en 16 cuadrantes, los cuales engloban todo el espacio inicial. En cada cuadrante está definido un polinomio de grado 1. Los círculos representan las muestras utilizadas para generar el polinomio de ese cuadrante.

Por tanto, para calcular los coeficientes de las distintas interpolaciones bilineales, se necesita resolver el siguiente sistema lineal 16 veces:

$$\begin{pmatrix} 1 & x_0 & y_0 & x_0y_0 \\ 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_2 & x_2y_2 \\ 1 & x_3 & y_3 & x_3y_3 \end{pmatrix} \begin{bmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{bmatrix} = \begin{bmatrix} Q_0 \\ Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} \tag{4.14}$$

siendo x_i e y_i las coordenadas de la esquina iésima de un cuadrante, c_{ij} el coeficiente correspondiente del polinomio y Q_{ij} la muestra tomada en la esquina iésima correspondiente.

La radiancia total de la subdivisión se calcularía como el sumatorio de la radiancia de cada uno de los cuadrantes aproximados con un polinomio:

$$I_{div} \approx \sum_{q \in \Omega} I'_q \tag{4.15}$$

siendo Ω el conjunto de polinomios interpoladores de la división e I'_q la integral análitica de dicho polinomio.

4.6. Método híbrido con Monte Carlo

El problema con los métodos estocásticos es la varianza[1] inherente a su aleatoriedad. Una forma de reducirla es aprovechándose de la información que se pudiese conocer de la función a integrar.

Una técnica que utiliza dicha idea se llama variables de control [Vea97]. La idea es obtener una función $g(x)$ la cual se pueda integrar analíticamente, y sea similar a la original que se pretende integrar, y calcular la integral de la diferencia con la original $f(x)$. Un ejemplo es la figura 4.8.

Entonces la integral puede reescribirse de la siguiente manera:

$$I \approx \int_\Omega g(x)d\omega(x) + \int_\Omega f(x) - g(x)d\omega(x) \tag{4.16}$$

donde $f(x)$ es la función original y $g(x)$ es la función que aproxima a la original.

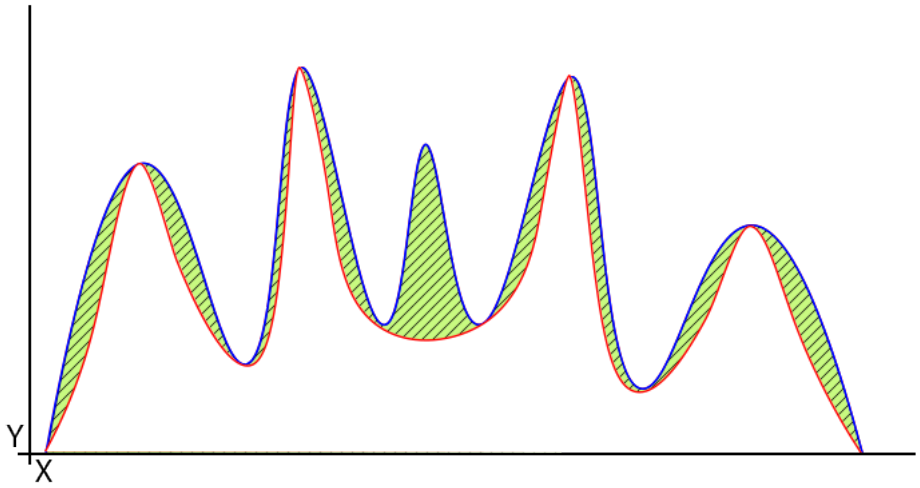


Figura 4.8: La curva azul es la función $f(x)$. La curva roja es la función aproximada $g(x)$. La diferencia entre el área de las dos funciones es la zona verde.

La primera integral por tanto se puede calcular analíticamente, mientras que para calcular la segunda se aplica el método de Monte Carlo, dando:

$$F \approx \int_\Omega g(x)d\omega(x) + \frac{1}{N}\sum_{i=1}^{N} f(x_i) - g(x_i) \tag{4.17}$$

siendo N el número de muestras que se utilizan para el cálculo con Monte Carlo.

Utilizando este método, se tiene que la nueva estimación utilizando la diferencia entre las dos funciones debería tener una menor varianza que únicamente calculando mediante Monte Carlo la función en sí:

[1] La varianza es la medida de como son de diferentes una serie de valores

$$V\left[f(X_i) - g(X_i)\right] \leq V\left[f(X_i)\right] \tag{4.18}$$

Aplicado junto al método propuesto, dada la estimación del error de la escena, aquellas zonas que todavía tengan un error superior al umbral establecido (si se han dado las subdivisiones necesarias) serán zonas donde la función a integrar tenga un componente de alta frecuencia difícil de aproximar por el método. (Un ejemplo es el pico central de la figura 4.8, el cual no es tenido en cuenta por la aproximación.)

Por ello se propone este método híbrido. En esas zonas se tendrá ya calculada una función $g(x)$ bastante próxima a la original. Por lo cual, lo único que quedará será calcular el componente de alta frecuencia, lo cual se realiza con la diferencia entre la función original y la aproximación.

Por otro lado, esta técnica permitiría aprovechar que Monte Carlo no le afecta la maldición de la dimensionalidad para poder calcular conjuntamente un número mayor de dimensiones de las que pueden calcular las técnicas de cuadratura sin degradar en exceso el rendimiento. En el capítulos 6 se presentan dos resultados aplicando esta idea (figuras 6.4 y 6.5).

Capítulo 5

Integración en Mitsuba

En este capítulo se va a hablar de como se implementó el método. Para ello primero se dará una pequeña introducción a Mitsuba. A continuación se hablará sobre la implementación de las divisiones, la gestión de memoria y la estructura de datos. A continuación, se comentará la integración del método mediante la interfaz para *plugins* de Mitsuba. Finalmente se hablará sobre los problemas de precisión numérica que se han tenido durante la integración.

5.1. Introducción a Mitsuba

Mitsuba [Jak10] es un software de renderizado basado en físicas orientado a la investigación. Está escrito en C++ y hace un especial énfasis en las técnicas experimentales, por ello su arquitectura es extremadamente modular, consistiendo en librerías centrales y plugins con las distintas técnicas. Finalmente comentar que Mitsuba dispone de una estructura de multi-procesamiento tanto en local como en distribuido.

Dichas características lo hacen la elección idónea sobre la cual implementar el nuevo método propuesto en este trabajo, puesto que permitirá tanto comparar el método con otra técnica como tener una plataforma sobre la cual empezar a trabajar.

Finalmente comentar que este integrador hace uso de una versión modificada del integrador *PathTracer* existente en Mitsuba.

5.2. Implementación de las subdivisiones

Como se comentó en la sección 4.3, la implementación de las divisiones es una parte crucial. Se necesita tanto de una estructura mínima en memoria (debido a que se tendrán que almacenar un gran número de ellas) como que sus operaciones sean eficientes en tiempo. Se va a proceder a hablar sobre las decisiones que se tomaron

durante la fase de implementación, de las cuales, una decisión importante ha sido la precisión numérica necesaria. Ello ha obligado a utilizar el tipo de dato *double*.

5.2.1. Memoria

Cada división necesita almacenar un número mínimo de atributos. Siendo **N** el número de dimensiones y **L** las muestras necesarias por dimensión, la lista de atributos a almacenar es la siguiente:

- **Coordenadas iniciales**. Se almacenan mediante un array de *double* de N elementos.
- **Extensión por dimensión**. Se almacenan mediante un array de *double* de N elementos.
- **Muestras obtenidas**. Se almacenan en un array n-dimensional. Se necesitarán almacenar $\mathbf{L^N}$ elementos.
- **Error cometido en la subdivisión**. Se almacena mediante un *double*.
- **Dimensión con mayor error**. Se almacena mediante un *entero*.

De lo anterior, el almacenamiento de las muestras es el principal factor de utilización de memoria.

5.2.2. Operaciones

Dada la naturaleza del problema, todas las funciones que operan sobre el array n-dimensional son de carácter recursivo. Esto conlleva una sobrecarga si se implementan de esa manera, la cual es perfectamente evitable. Por ello se evitó la utilización de recursividad implementando funciones iterativas.

Por otro lado, toda subdivisión (excluyendo la inicial) comparte un número de muestras con su padre (división de la que procede). Un ejemplo en dos dimensiones es la figura 5.1. Por tanto, se puede reducir el número de operaciones de muestreo reutilizando dichas muestras.

Otro factor importante es la obtención de los coeficientes de los polinomios interpoladores, los cuales se obtienen resolviendo un sistema lineal. Para ello se ha elegido la librería de álgebra lineal *Eigen* [GJ+10]. De todos los *solvers* que la librería tiene, se ha optado por **FullPivHouseholderQR** debido a su precisión y estabilidad numérica.

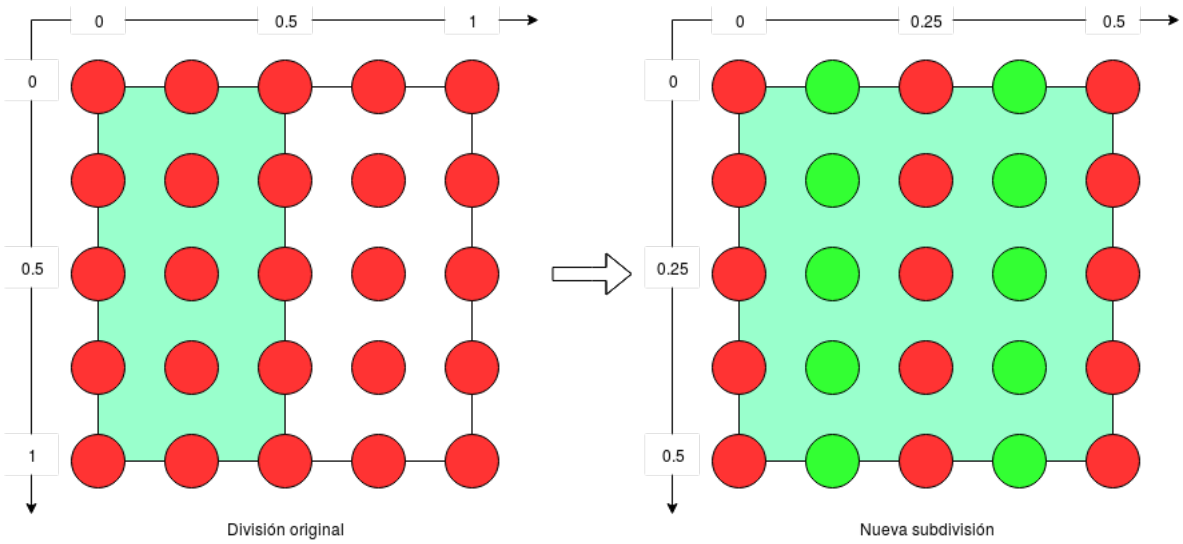


Figura 5.1: Esquema de la distribución de las muestras tras una subdivisión

La figura representa las muestras que se comparten entre dos divisiones. Dado el padre (izquierda) y una subdivisión hijo (derecha, representada dentro del padre mediante el color aguamarina), los círculos rojos representan las muestras que el padre tiene ya calculadas y los círculos verdes, tras la división en el eje horizontal, las nuevas muestras que se tendrán que calcular en la subdivisión hijo.

Cada vez que una división tenía que calcular la radiancia por píxel, inicialmente debía obtener los coeficientes de los polinomios interpoladores correspondientes resolviendo para ello los sistemas de ecuaciones necesarios (4.14). Aún utilizando un *solver* cuyo coste en tiempo fuera mínimo, el coste total temporal de dichas operaciones era notable. Una solución es cambiar la resolución de los sistemas a dos fases:

- En la primera fase se precalcula la descomposición de la primera matriz en la ecuación (4.14). Esto reduce efectivamente el cálculo más complejo que se tenía a una única vez. La idea es que si se calculan las posiciones normalizadas entre 0 y 1, esa matriz no cambia entre divisiones dado que todos los puntos son equidistantes entre sí.

- En la segunda fase, dada la descomposición y las muestras correspondientes de la división, se calculan los coeficientes necesarios.

5.3. Gestión de memoria

Debido a la naturaleza del algoritmo, se construyen y se destruyen un número elevado de divisiones. Ello conlleva que un esquema en memoria dinámica sin control sea ineficiente y pueda conllevar a su fragmentación. Para solucionarlo, se ha utilizado el patrón de diseño *Object Pool* para dar solución a este problema.

5.3.1. Introducción al patrón de diseño

El patrón *Object Pool* define una zona de memoria reservada donde se guardan objetos similares. Cuando es necesario, se extrae un puntero a un segmento del tamaño necesario para el tipo de objeto y se inicializa ahí. Cuando dicho objeto se destruye, se marca esa zona como libre y queda pendiente de que sea solicitada. En caso de que la *pool* se llene, se aloja una nueva zona de memoria y se utiliza.

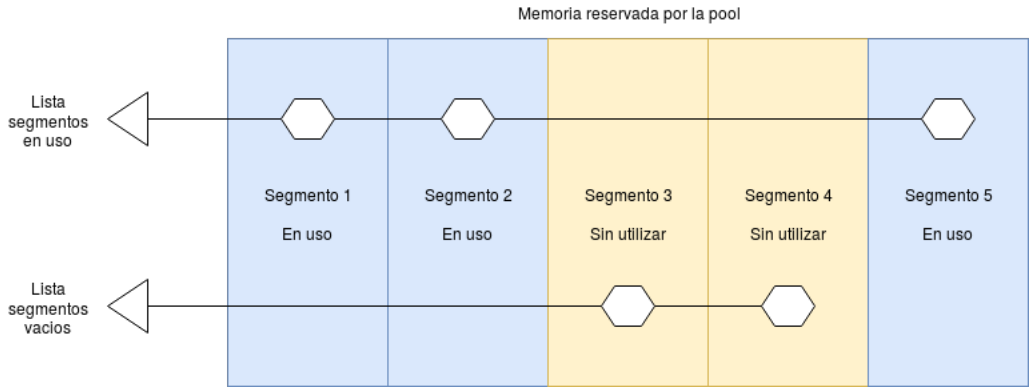


Figura 5.2: Diagrama que representa una posible implementación de una *object pool* mediante listas

Al tener la memoria reservada de antemano, no hay costes extras de ir reservando a cada creación ni tampoco se produce fragmentación dado que esa zona de memoria solo será liberada al destruirse la *pool*. Con ello se reducen las llamadas al sistema operativo, así como evitar la fragmentación de reservar y liberar memoria dinámica.

5.3.2. Implementación

Se ha optado por utilizar la implementación de la librería **Boost** [Boob], dado que cumple todos los requisitos para este proyecto. Por otro lado, mediante la interfaz que provee, se puede definir el número de elementos iniciales que la *pool* debe reservar. Como el método define un número de iteraciones máximas a ejecutar, y cada división genera siempre el mismo número de subdivisiones, se puede obtener una cota máxima de las divisiones que se tendrán que almacenar. Con ello se puede realizar una reserva inicial de la memoria que luego reutilizará el algoritmo durante toda su ejecución.

El utilizar este patrón implica que hay que llevar la cuenta del número de referencias activas de cada división para no liberar un recurso que todavía sigue en uso. La solución ha sido la utilización de *shared_ptr* con un destructor configurado para marcar el recurso como libre en la *object pool* correspondiente.

Por otro lado, cada thread generara su propia *pool* para evitar problemas de accesos concurrentes.

5.4. Estructura de datos

Como se habló en la sección 4.3, el método adaptativo necesita de una estructura de datos que le permita obtener la subdivisión de mayor error de todas las almacenadas.

Una primera aproximación fue utilizar una cola de prioridad. Una cola de prioridad es una estructura de datos basada en un *heap*[1] que permite en tiempo $O(1)$ obtener el elemento que maximiza un criterio de comparación. De normal, estas colas de prioridad tienen coste $O(log\ n)$ tanto para insertar un nuevo elemento como para extraer el elemento máximo.

El problema con el algoritmo adaptativo es que aunque se destruyen bastantes divisiones, se crean todavía más. Básicamente cada vez que se obtiene el elemento máximo, luego se tendrá que eliminar dicha división aparte de tener que insertar dos o más nuevas divisiones. Por tanto, el coste $O(log\ n)$ toma mucha más importancia que el coste constante de obtener el elemento máximo debido al ritmo en el cual crece n.

Otra opción, conociendo el problema de acumular todas las divisiones en una única cola, es separar las divisiones dado su error en distintos buffers con tamaño máximo [FHMO16]. Con ello, ya no se tiene una única cola de prioridad sino varios buffers separados por rangos de error los cuales ya no se ordenan. Como en cada buffer solo entran divisiones con error parecido, si se van extrayendo las divisiones desde aquellos buffers con el rango de error mayor, se simula la ordenación de las colas con prioridad.

Esta opción aborda el problema, pero no lo soluciona del todo. Las limitaciones más directas son que ya no se elige siempre la subdivisión con mayor error y que hay que elegir de antemano el número de buffers así como el rango de errores que comprenden.

Se presenta una estructura similar que aborda estos dos últimos problemas y los soluciona.

La nueva estructura de datos se basa en el **Fibonacci heap** implementado en **Boost** [Booa], el cual es una estructura de datos para operaciones en colas de prioridad que consiste en una colección de árboles ordenados como montículos, cuya particularidad es que al contrario que otros tipos de *heaps*, insertar un nuevo elemento tiene coste $O(1)$. Para conseguirlo, muchas de las operaciones del *Fibonnaci Heap* no son ejecutadas al momento sino cuando son necesarias. Un ejemplo es al insertar un nuevo elemento: el elemento se introduciría en la estructura pero no sería hasta necesitar el elemento con mayor error que se ordenaría el *heap*.

Aún así, por cada vez que se tenga que extraer el elemento con mayor error, el coste de la operación podría ser prohibitivo dependiendo del número de elementos

[1] Un *heap* es una estructura de datos de tipo árbol con información perteneciente a un conjunto ordenado. Un ejemplo es un montículo de máximos, en el cual, cada nodo padre tiene un valor mayor que el de cualquiera de sus hijos.

almacenados. Para ello se adopta también la idea de utilizar varios buffers, pero siendo cada buffer un *Fibonacci Heap*. Con ello se divide la carga de almacenamiento mientras que se mantiene el acceso al elemento mayor en todo momento.

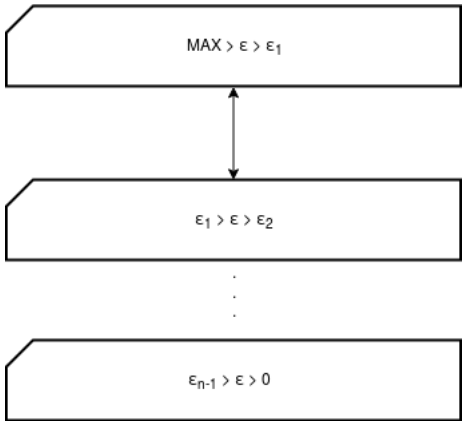


Figura 5.3: Diagrama de la estructura de datos

Para permitir que la nueva estructura se adapte a cualquier tipo de escena eficientemente, se empieza con un número inicial de buffers, los cuales se van incrementando conforme son necesarios. Posteriormente, se utiliza una heurística para determinar cuando se necesitan más buffers en un intervalo de error. Dicha heurística solo se aplica a aquellos buffers que tienen más elementos que un mínimo establecido (se evita la generación de buffers al principio de ejecución cuando hay pocos elementos almacenados).

La heurística utilizada es la siguiente:

$$H(i) = c(i) >= 0{,}4n \tag{5.1}$$

siendo $c(i)$ el número actual de elementos en el buffer y n el total de elementos guardados en todos los buffers que controla la estructura de datos.

Si la heurística determina que hay que generar nuevos buffers, se obtiene el rango de errores entre el buffer que ha dado positivo a la heurística y su buffer anterior, generando dos nuevos buffers entre ellos dos (Figura 5.4).

Con ello se consigue que no existan acumulaciones excesivas de elementos en ningún buffer y por tanto que no exista degradación de rendimiento.

Finalmente, tras llegar a un mínimo de elementos almacenados dentro de la estructura, aquellos buffers con cero elementos son eliminados debido a que ya no serán utilizados, puesto que mientras el algoritmo avanza, el error de las nuevas divisiones será cada vez más pequeño y los buffers con rangos de error superiores nunca más serán utilizados.

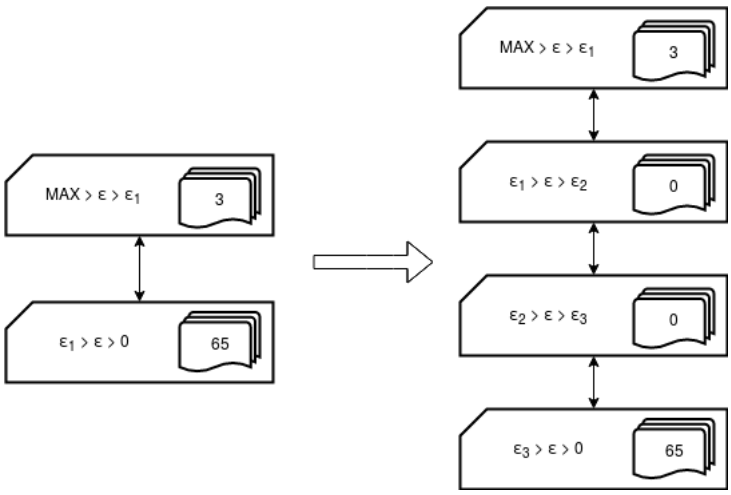


Figura 5.4: Ejemplo de como se generan los buffers adaptativamente. A la izquierda la estructura de datos antes de la subdivisión (el segundo buffer ha dado positivo en la heurística). A la derecha, la estructura de buffers resultante de adaptarse.

5.5. Arquitectura del plugin

La arquitectura del proyecto viene marcada por la interfaz que provee **Mitsuba**.

Aquellos plugins que se encargan de renderizar escenas son llamados **integradores**. Mitsuba proporciona dos tipos de interfaces para ellos: una específica y otra general. Debido a que el nuevo método no entra dentro de las especificaciones de la interfaz específica, se ha tenido que implementar utilizando la general.

Si se quiere implementar la interfaz genérica, el plugin debe encargarse tanto de su inicialización, como de la generación de tareas, su procesado y posterior recolección. Todo ello apoyándose sobre la arquitectura de paralelización de Mitsuba. La arquitectura del plugin se puede resumir con el diagrama de la figura 5.5

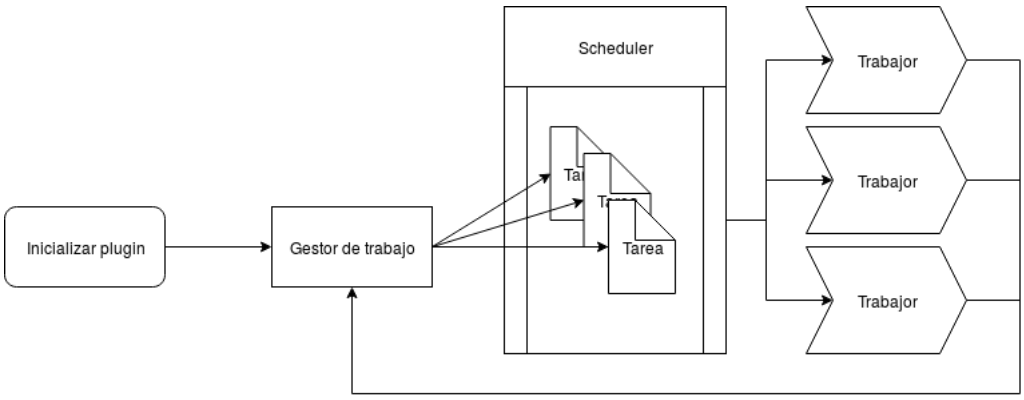


Figura 5.5: Esquema con la arquitectura del plugin implementado

Inicialmente, el plugin configura todo lo necesario para su funcionamiento. En esta fase se obtienen los parámetros de configuración, se generan tantas subdivisiones en

espacio de pantalla como se soliciten y se reserva memoria para las *pools* de divisiones y las estructuras de datos que luego serán utilizadas por los trabajadores para el procesamiento de las tareas. Al implementarlo así, se reduce al mínimo las tareas de gestión de memoria dinámica, acelerando el proceso individual de cada trabajador sin que influya en la paralelización.

Posteriormente, el plugin genera tantas tareas como subdividiones iniciales se hubiesen creado, las cuales son independientes entre sí. Cada tarea será procesada en paralelo, acelerando su procesamiento. Una vez se ha generado una tarea, se registra en el *scheduler* para que sea procesada por un trabajador que se encuentre sin trabajo o hubiese acabado su anterior tarea.

Una vez un trabajador coge una tarea, se aplica el método propuesto en el trabajo. Durante la fase de implementación se optó por hacer al método agnóstico del sistema sobre el cual funciona. Por ello se definen dos zonas de colisión:

Obtención de las muestras . Se ha optado por suministrarle al método propuesto un *callback* el cual se encarga de devolverle la muestra correspondiente dados unos parámetros de entrada. Por tanto la gestión para generar una muestra queda confinada a una función dentro del plugin..

Radiancia por pixel . El plugin es el encargado, dada una subdivisión, de calcular los píxeles influidos por ella y calcular la radiancia aportada a cada uno.

Tras acabar de procesar una tarea, el gestor del plugin recibe el resultado del trabajo el cual será una zona de la imagen final ya calculada.

Para finalizar la integración del plugin dentro de Mitsuba, se ha procedido a integrar los parámetros de configuración dentro de la interfaz, así como poder modificarlos desde los ficheros que describen escenas. Una captura de la interfaz se encuentra en la figura 5.6.

5.6. Precisión numérica

El principal problema durante la integración ha sido la falta de precisión en las operaciones matemáticas. Ello ha obligado a trabajar con el tipo *double* en todo momento, con el coste que conlleva y teniendo que usarlo tanto en el nuevo método como dentro de Mitsuba en aquellas partes colindantes entre los dos. Aún incrementando la precisión de las variables, se seguían produciendo errores numéricos. Debido a ello se ha trabajado con las integrales normalizadas al intervalo [0.,1] (lo cual también ha

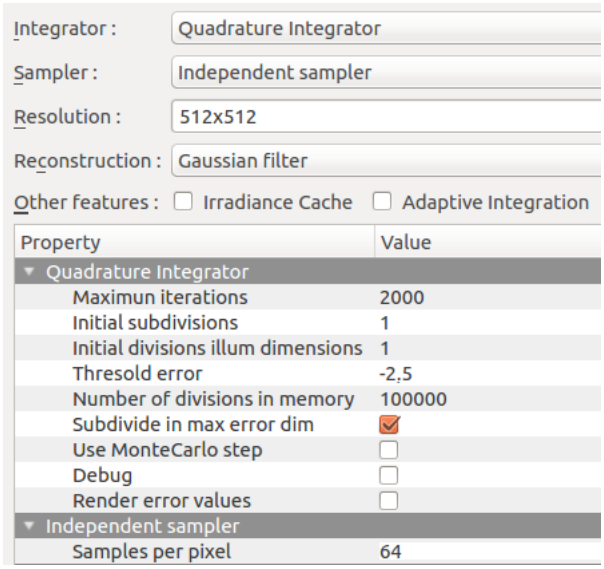


Figura 5.6: Imagen de los parámetros del nuevo método integrados dentro de la interfaz de Mitsuba.

ayudado a disminuir el coste de ejecución permitiendo el precalculo de los polinomios interpoladores como se comentó en la sección 5.2.2).

Finalmente, a la hora del cálculo de la dimensión de mayor error, se ha tenido que tener en cuenta errores cercanos a cero en el operador de cuartas diferencias para evitar el funcionamiento erróneo de la estimación.

Capítulo 6

Resultados

Se presentan a continuación los resultados relevantes obtenidos, comparándolos con el método establecido como estándar y altamente optimizado llamado *Path Tracer*.

Como una pequeña introducción, *Path tracer* es un método basada en Monte Carlo, el cual calcula iterativamente la radiancia de la escena mediante la formulación de la *Path Integral* (2.2). Comparándola con el método desarrollado, se tiene que *Path Tracer* no hace casi uso de memoria dado que los cálculos iterativos son independientes entre sí, mientras que la técnica propuesta hace un uso intensivo de ella al tener que que almacenar todas las subdivisiones necesarias para el cálculo correcto de la iluminación.

Como se comentó en la sección 4.3, la relación que tienen ambas técnicas para poder compararlas son el número de iteraciones máximas y las muestras por píxel (**mpp**), debido a que ambos parámetros son los responsables finales del tiempo de cálculo de cada método.

Ambos métodos se encuentran integrados en Mitsuba y todas las imágenes han sido renderizadas usándolo.

Las figuras 6.1 y 6.2 muestran una comparativa del ruido generado por cada método. La imagen izquierda corresponde a utilizar *Path Tracer* y la central corresponde al método propuesto (ambas con un tiempo de cálculo similar). La imagen derecha corresponde a aplicar *Path Tracer* con un número elevado de muestras por pixel. Debido a su capacidad para converger, el método propuesto es capaz de obtener resultados en bastante menor tiempo que *Path Tracer*.

Aunque este método está específicamente pensado para escenas con sombras suaves, la figura 6.2 muestra como el método propuesto es agnóstico al material en la escena, es decir, no necesita realizar ninguna presunción adicional para su cálculo. Un ejemplo son los materiales especulares en los que *Path Tracer* no tiene ningún problema y, como se puede ver en la misma figura, el nuevo método converge hacia la misma solución e incluso mejora el resultado de *Path Tracer* en tiempo similar. Otro ejemplo con un

grado de especularidad mayor sería la figura 6.3.

En las figuras 6.4 y 6.5 se muestran los resultados de aplicar el método híbrido para el cálculo de más dimensiones de la *Path Integral*, específicamente, para calcular las interacciones de la luz con el cristal. En ambos casos se puede apreciar como se ha producido sesgo debido a las subdivisiones que hubiese realizado previamente la fase adaptativa del método. Aparte, al tener que aplicar una fase final con Monte Carlo por cada píxel influenciado por la división, el tiempo de cálculo se incrementa notablemente frente a la versión propuesta determinista.

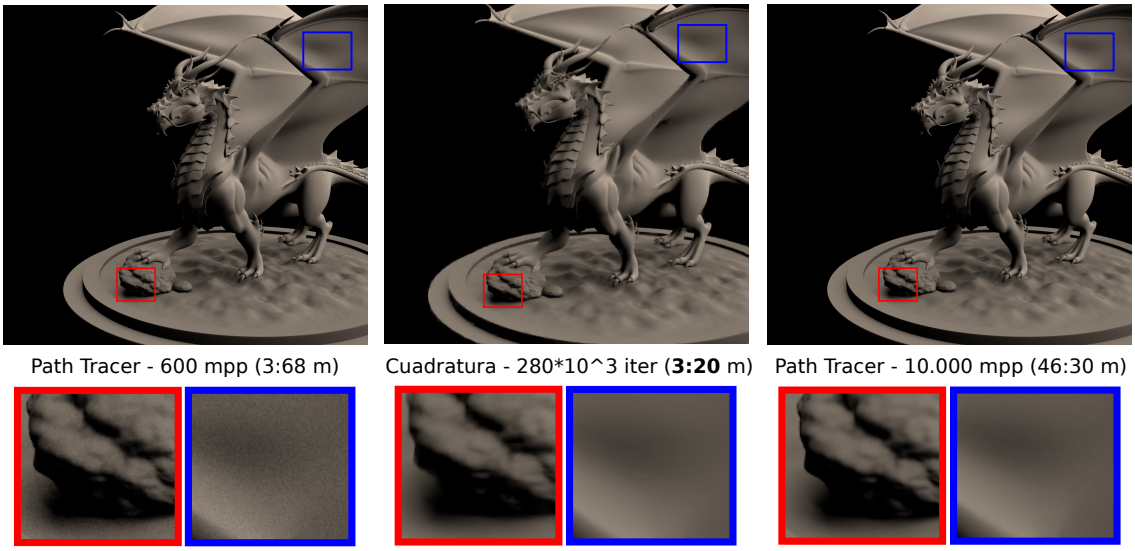


Figura 6.1: Figura de un dragón con material difuso iluminada por una luz de área que simula el sol.

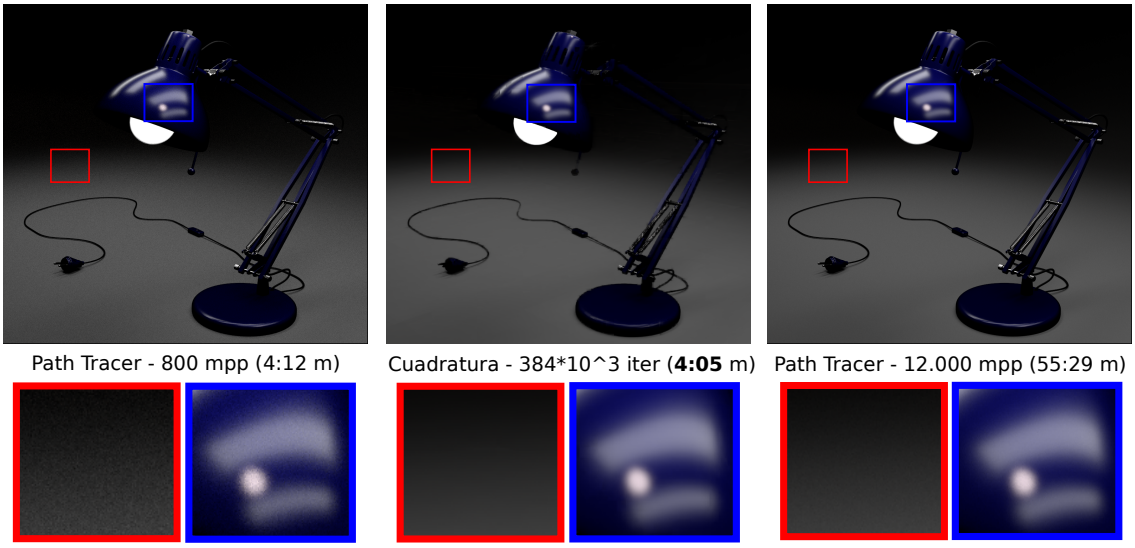
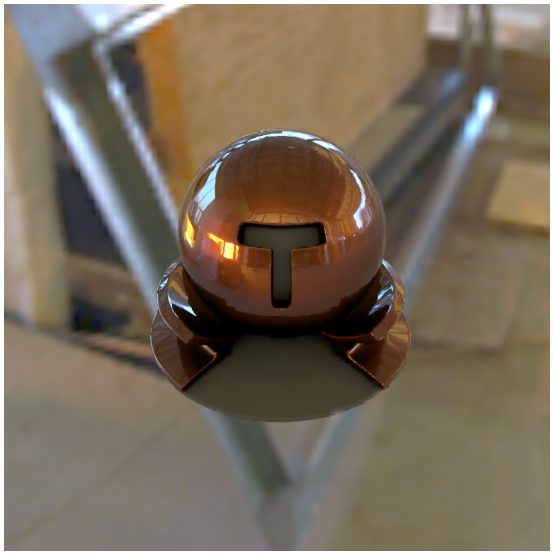


Figura 6.2: Escena cuya iluminación proviene principalmente de la bombilla situada en la lámpara con un material *glossy*.



(a) Cuadratura - $256 * 10^3$ iter (1:78 m)

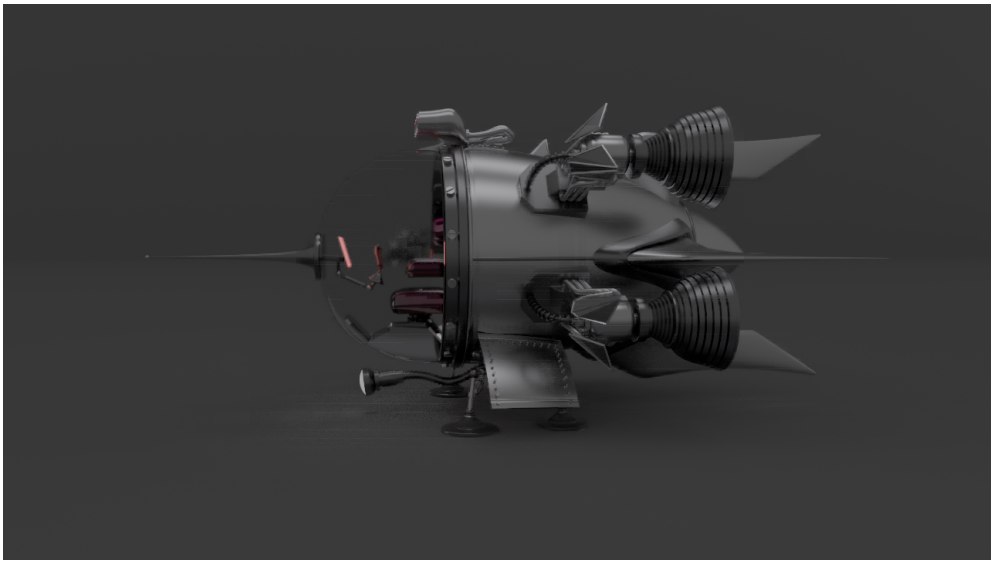
(b) Path Tracer - 800 mpp (2:33 m)

Figura 6.3: Escena con un modelo con material especular iluminado por un mapa de entorno.

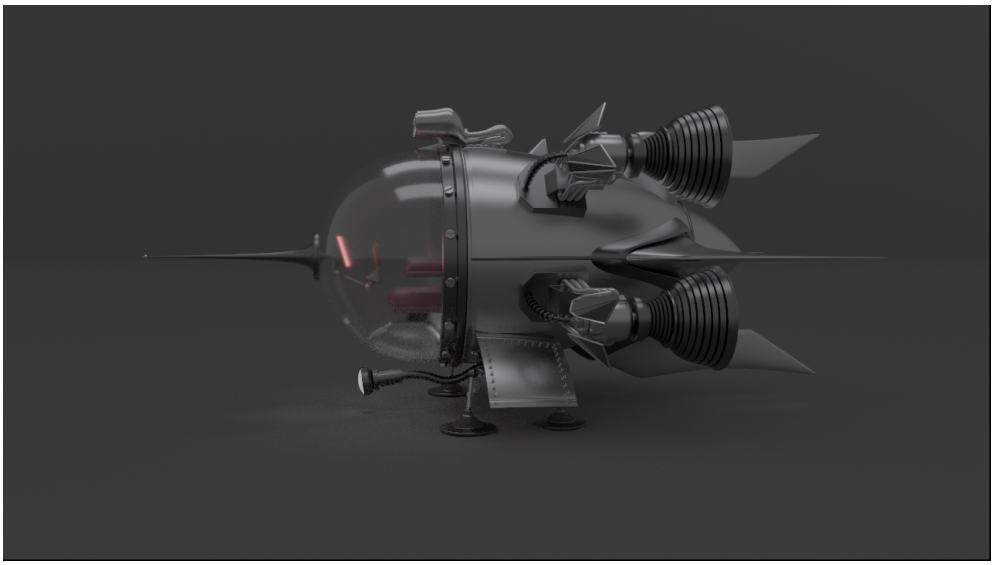
(a) Híbrido - $192 * 10^2$ iter - 8 mpp (7:88 m)

(b) Path Tracer - 1200 mpp (8:81 m)

Figura 6.4: Escena con un modelo de una cafetera altamente especular. Se han calculado hasta 65 rebotes de la luz con el medio.



(a) Híbrido - $200 * 10^3$ iter - 8 mpp (12:41 m)



(b) Path Tracer - 2000 mpp (9:03 m)

Figura 6.5: Escena con un modelo de una nave espacial con cristal. La iluminación se ha calculado teniendo en cuenta hasta 65 posibles rebotes de luz en el medio.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Discusión y trabajo futuro

En este trabajo se ha estudiado el uso de las técnicas de cuadratura como alternativa a los métodos estándares estocásticos para la generación de imágenes físicamente correctas. Como se ha mostrado en el capítulo 6 (gracias a su integración en el software de renderizado Mitsuba), el método propuesto no sólo es capaz de generar imágenes sin ruido, sino que también requiere de un menor tiempo de cómputo que alternativas como *Path Tracer*, convirtiéndolo en una opción interesante frente a métodos estándares.

Para conseguir estos resultados, han sido necesarios unos procesos de refinamiento sobre la implementación debido a las necesidades que se dan de evaluar analíticamente la integral mediante reglas de cuadratura. Principalmente, la inclusión de la estructura de datos ha permitido acelerar el proceso de cálculo en gran medida al ser el principal cuello de botella durante el procesamiento.

Aún así quedan fronteras abiertas y muchas oportunidades de trabajo futuro. Por ejemplo, uno de los problemas de este método es que encuentra dificultades en aquellas zonas con detalles debido al componente de alta frecuencia, el cual es difícil de aproximar mediante técnicas de cuadratura. Para solucionarlo, un posible trabajo futuro sería la mejora del método híbrido con Monte Carlo. También se mostró en el capítulo 6, que la utilización del método híbrido para calcular un número elevado de dimensiones de la integral es viable aunque se introduce sesgo en el resultado. Otra posible vía de trabajo futuro sería reducir el tiempo que cuesta calcular Monte Carlo durante la fase híbrida mediante la reutilización de los distintos puntos elegidos pseudo-aleatoriamente entre las distintas divisiones.

Finalmente, cuando se presentan discontinuidades que no se encuentran alineadas con los ejes del espacio de imagen, para aproximarlas correctamente el método realiza un número elevado de divisiones hasta encajar con la discontinuidad. Un trabajo futuro sería la sustitución del espacio que representa cada division por un n-simplex, lo cual

permitiría dotar de una mayor flexibilidad al método.

7.2. Conclusiones personales

Este trabajo ha servido como una forma de demostrar los conocimientos adquiridos durante la carrera, así como la capacidad adquirida para resolver problemas complejos. También ha sido mi puerta de entrada al mundo de la investigación, en el que he descubierto un entorno desafiante y lleno de posibilidades. En resumen este laborioso trabajo, el cual a veces me ha puesto a prueba, es para mí el satisfactorio punto final de estos cuatro años de carrera.

Bibliografía

[BEG91] Jarle Berntsen, Terje O. Espelid y Alan Genz. "An Adaptive Algorithm for the Approximate Calculation of Multiple Integrals". En: *ACM Trans. Math. Softw.* 17.4 (1991), págs. 437-451. ISSN: 0098-3500. DOI: 10.1145/210232.210233. URL: http://doi.acm.org/10.1145/210232.210233.

[Booa] Boost. *Fibonacci heap documentation*. URL: http://www.boost.org/doc/libs/1_49_0/doc/html/boost/heap/fibonacci_heap.html.

[Boob] Boost. *Object pool documentation*. URL: http://www.boost.org/doc/libs/1_42_0/libs/pool/doc/interfaces/object_pool.html.

[FHMO16] Francisco Javier Fabre Herrando y Adolfo Muñoz Orbañanos. "Simulación Adaptativa de Iluminación Global". En: (2016).

[GJ+10] Gaël Guennebaud, Benoît Jacob y col. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[GM80] A.C. Genz y A.A. Malik. "Remarks on algorithm 006: An adaptive algorithm for numerical integration over an N-dimensional rectangular region". En: *Journal of Computational and Applied Mathematics* 6.4 (1980), págs. 295 -302. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0771-050X(80)90039-X. URL: http://www.sciencedirect.com/science/article/pii/0771050X8090039X.

[Gon10] Pedro Gonnet. *A Review of Error Estimation in Adaptive Quadrature*. 2010.

[Hac+08] Toshiya Hachisuka y col. "Multidimensional adaptive sampling and reconstruction for ray tracing". En: *ACM Transactions on Graphics (TOG)*. Vol. 27. 3. ACM. 2008, pág. 33.

[Jak10] Wenzel Jakob. *Mitsuba website*. 2010. URL: http://www.mitsuba-renderer.org.

[Kaj86] James T. Kajiya. "The rendering equation". En: *SIGGRAPH Comput.Graph.* (1986).

[Lau85] Dirk P. Laurie. "Practical error estimation in numerical integration". En: *Journal of Computational and Applied Mathematics* 12-13 (1985), págs. 425 -431. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(85)90036-6. URL: http://www.sciencedirect.com/science/article/pii/0377042785900366.

[LWZ12] Xiao-Dan Liu, Jia-Ze Wu y Chang-Wen Zheng. "KD-tree based parallel adaptive rendering". En: *The visual computer* 28.6-8 (2012), págs. 613-623.

[Mn14] Adolfo Muñoz. "Higher Order Ray Marching". En: *Computer Graphics Forum* 33.8 (2014), págs. 167-176. ISSN: 1467-8659. DOI: 10.1111/cgf.12424. URL: http://dx.doi.org/10.1111/cgf.12424.

[Pat73] T. N. L. Patterson. "Algorithm 468: Algorithm for Automatic Numerical Integration over a Finite Interval [D1]". En: *Commun. ACM* 16.11 (1973), págs. 694-699. ISSN: 0001-0782. DOI: 10.1145/355611.362543. URL: http://doi.acm.org/10.1145/355611.362543.

[Vea97] Eric Veach. *Robust monte carlo methods for light transport simulation*. 1610. Stanford University PhD thesis, 1997.

[Zwi+15] Matthias Zwicker y col. "Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering". En: *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)* 34.2 (2015), 667–681. DOI: 10.1111/cgf.12592.

Lista de Figuras