Universidad Zaragoza

Trabajo Fin de Grado

Simulación Adaptativa de Iluminación Global

Autor

Francisco Javier Fabre Herrando

Director

Adolfo Muñoz Orbañanos

Escuela de Ingeniería y Arquitectura
2016

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza

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. Francisco Javier Fabre Herrando,

con nº de DNI 73028687-F 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 en Ingeniería Informática, (Título del Trabajo) Simulación Adaptativa de Iluminación Global,

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

Zaragoza, 23 de junio de 2016

Fdo:

A la memoria de mi tía Ana.

Agradecimientos

Quiero dar las gracias, a Adolfo, por darme la posibilidad de realizar este trabajo. A él y a todas las personas del *Graphics and Imaging Lab* que desinteresadamente han compartido sus conocimientos conmigo. Por último, pero no por ello menos importante, a mi familia y amigos por apoyarme en los buenos y en los malos momentos. Gracias a todos.

Resumen

El proceso de simulación de imágenes por ordenador es conocido como renderizado. Este proceso requiere realizar los cálculos necesarios para simular las interacciones que la luz efectúa con los distintos objetos presentes en las escenas que se desean renderizar. Esto requiere la resolución de complejos algoritmos, que aproximan los modelos que definen las distintas propiedades de la materia.

Este proyecto trata de estudiar una implementación alternativa para el renderizado de imágenes por ordenador a la que actualmente se plantea de manera más general.

El proyecto trata una aproximación adaptativa para solucionar los problemas de cálculo integral ligados al renderizado de imágenes, puesto que de forma terminal las técnicas de renderizado solucionan algoritmos que poseen importantes componentes integrales.

La complejidad de los cálculos integrales necesarios para su resolución hace que una de las técnicas más usadas en la actualidad para resolver estos algoritmos, y por tanto realizar la simulación de imágenes por ordenador, sea el llamado Método de Montecarlo. Este método aproxima la solución utilizando números aleatorios de manera que se termine convergiendo a una aproximación adecuada.

La aproximación adaptativa que se propone, en contraposición al mencionado Método de Montecarlo, pretende analizar la complejidad de las distintas integrales que es necesario calcular a lo largo del proceso. De esta manera se pretende utilizar un número variable de muestras para la aproximación de los valores necesarios, que será proporcional a la complejidad de su cálculo. Frente al método aleatorio, esta aproximación debería utilizar un menor número de muestras en aquellos casos que no sea necesario un número elevado para determinar una solución lo suficientemente precisa.

Se ha seguido durante este trabajo la tradicional metodología de trabajo software, comenzando por una fase de aprendizaje teórico, necesario para la resolución del problema, seguido de una fase de análisis del mismo. A continuación se ha definido una serie de requisitos deseados para el trabajo. Por último se ha realizado el diseño del sistema y su implementación, así como una fase de estudio de los resultados de todo el trabajo realizado, comentando, además, las posibles aplicaciones futuras de este proyecto.

Índice

1. Introducción

En este documento se pretende recopilar la información relacionada con el **T**rabajo de **F**in de **G**rado (**TFG**) titulado *Simulación Adaptativa de Iluminación Global*.

El trabajo que se presenta, intenta explorar una alternativa al Método de Monte Carlo para el renderizado de imágenes por computador, utilizando una aproximación adaptativa. De esta manera, el proceso de obtención de muestras, que en el caso de Monte Carlo se realiza de manera aleatoria, se realice de manera adaptativa, tomando una mayor cantidad de muestras en aquellos lugares en los que los cálculos matemáticos las requieran para obtener una precisión adecuada. En contraposición, donde no sea necesaria una gran cantidad de muestras para obtenerla, se utilizará una menor cantidad.

1.1. Objetivos del proyecto

El objetivo de este trabajo es la realización de distintas aproximaciones de la ecuación de render con el objetivo de realizar imágenes sintéticas por ordenador, utilizando un método alternativo al conocido método de Monte Carlo .

En este trabajo se realizan distintas aproximaciones utilizando métodos adaptativos, en contraposición al método aleatorio ya mencionado, de manera que las integrales necesarias para los cálculos de la imagen sean aproximadas utilizando un número de muestras correlado al error numérico del cálculo de cada integral. Así, se adaptará el número de muestras para conseguir una precisión similar en toda la imagen.

Se desea, por tanto, comprobar la viabilidad de este método alternativo en distintos escenarios posibles, comparando los resultados obtenidos adaptativamente con los que los métodos ampliamente usados en la actualidad obtienen para distintas escenas.

1.2. Estructura de la memoria

La primera sección de este documento es esta propia introducción del proyecto, que trata los objetivos del mismo, así como la estructura de la propia memoria.

Posteriormente la sección 2 de esta memoria trata los fundamentos teóricos en los que se basa el trabajo realizado, explicados de manera general, ya que posteriormente se comenta su aplicación concreta en este trabajo.

Las tres siguientes secciones enfocan el trabajo realizado durante este proyecto.

La sección tercera trata el análisis del problema realizado. La cuarta sección explica la fase de diseño llevada a cabo, explicando cada una de las decisiones tomadas durante el mismo. La sección cinco, trata sobre la implementación del algoritmo, explicando como se han llevado los conceptos del diseño al programa final que resuelve el problema.

Como penúltima sección de esta memoria se presenta la validación y resultados del trabajo realizado.

En último lugar, dentro de la quinta sección de la memoria, se presentan las conclusiones, además de comentarios varios sobre el trabajo futuro que puede realizarse sobre este trabajo.

2. Conceptos Teóricos

Antes de analizar el problema de la simulación de iluminación global, es necesario entender los conceptos teóricos en los que se basa el mismo. En esta sección se explican los conceptos básicos en los que este trabajo se apoya de manera resumida. Una explicación más detallada puede encontrarse en los Anexos A y B

2.1. Path Integral

El problema que se plantea para la simulación de iluminación de Iluminación Global de manera adaptativa es, como ya se ha comentado durante la introducción, un problema de aproximación adaptativa de integrales.

Esto se debe a que, la formulación conocida como *Path Integral* permite cálcular el valor de cada uno de los pixeles de la imagen a simular según la ecuación:

$$I_j = \int_{\Omega} f_j(\bar{x}) d\mu(\bar{x}) \tag{2.1}$$

donde Ω es el conjunto de todos los *transport paths* de cualquier longitud.

El término f_j representa la llamada *Measurement Contribution Equation*, que designa la contribución de cada *path* (*path contribution*).

Por último, μ representa el *Path Space*, el cual puede expresarse como una secuencia de números del intervalo $[\mathbf{0} \ldots \mathbf{1}]$.

2.2. Técnicas de cuadratura

Sabiendo que es la ecuación de *Path Integral* la base para resolver nuestro problema de Iluminación Global, y puesto que esta es una ecuación integral, la resolución del problema consiste en resolver dicha ecuación. Para ello se utilizan las denominadas técnicas de cuadratura.

Los técnicas de cuadratura resuelven el problema de las ecuaciones integrales realizando aproximaciones suficientemente precisas mediante distintas evaluaciones del integrando, realizando combinaciones de estas evaluaciones para conseguir la aproximación. Existen múltitud de técnicas de cuadratura tal y como se expone en el Anexo B.1, pero las de más interés para este trabajo son concretamente el Método de Monte Carlo y las técnicas de cuadratura adaptativas.

El Método de Monte Carlo es de interés para este trabajo dado su amplio uso para la aproximación de integrales en multitud de ambitos, en este caso la simualción de iluminación global. La razón por la que el Método de Monte Carlo es tan utilizado esta relacionado con las caraterísticas del mismo:

- No crece en complejidad de manera exponencial respecto al número de dimensiones tratadas.
- Es un método no sesgado[1]
- Permite un control sencillo sobre el número de muestras que utiliza.

Por otro lado, las técnicas adaptativas son la base de cálculo integral para este trabajo, dado que se pretende realizar los cálculos integrales necesarios haciendo uso de estas, en contraposición al uso generalizado del Método de Monte Carlo.

Estas y otras técnicas de cuadratura existentes se encuentran explicadas con más detalle en el Anexo B.1.

[1] No introduce sesgo, es decir, error debido a la forma en la que las muestras son cogidas.

3. Análisis del problema

3.1. Estado del arte

El objetivo de este trabajo es la realización de una aproximación adaptativa para el problema de simulación de iluminación global.

Existen ya trabajos realizados sobre esquemas adaptativos para la resolución de este problema [HJW+08] [ZJL+15], que igualmente plantean esquemas de caracter adaptativo para soluciona el problema de iluminación global.

A diferencia la mayoría de estos, el planteamiento de este trabajo es realizar las aproximaciones necesarias para la resolución del problema utilizando técnicas de cuadratura cuyo muestreo no sea aleatorio, tales como las reglas de cuadratura de Simpson o del Rectángulo, con el objetivo de minimizar el ruido que técnicas de este tipo aportan a la imagen resultante, tal y como ocurre en los métodos de Monte Carlo.

Igualmente se pretende que la aproximación sea parametrizable. Se desea que número de muestras utilizadas para el cálculo total pueda controlarse, tal y como puede hacerse en el caso de utilizar el Método de Monte Carlo.

3.2. Requisitos

Se a continuación los requisitos planteados para este proyecto y sobre los que se apoyará el resto del trabajo:

Requisitos funcionales

- Implementación de múltiples aproximaciones al problema del cálculo de la integral. Contando como mínimo las dos planteadas durante la fase de análisis.
- Implementación de un método adaptativo que permita un control de la muestras utilizadas en la aproximación similar al utilizado en el Método de Monte Carlo .
- Parametrización mediante linea de comandos de los distintos valores usados por el algoritmo.
- Posibilidad de definir distintas escenas sobre las que ejecutar el algoritmo de renderizado de manera que se pueda validar el algoritmo.

Requisitos no funcionales

- Minimizar el coste de memoria RAM utilizado durante la ejecución del algoritmo.
- Reducción del ruido existente en implementaciones que utilizan el método de Monte Carlo .
- Estudiar la viabilidad del algoritmo en distintas escenas frente a distintos métodos de renderizado.
- Posibilidad de visionar la generación de la imagen en tiempo real.

3.3. Analisis general del problema

Se pretende conseguir un método alternativo al Método de Monte Carlo para la simulación de iluminación global.

Este Método de Monte Carlo presenta ciertas ventajas por las cuales es ampliamente usado:

- No presenta el problema conocido como ”maldición de la dimensionalidad”, por lo que los cálculos no escalan con el número de dimensiones exploradas.
- Permite una amplio y sencillo control sobre el número de muestras empleadas durante el proceso de simulación.

El método que se presenta, intenta igualmente tener algunas de las ventajas más deseables que posee el Método de Monte Carlo, mientras que trata de subsanar alguno de sus problemas:

- Permite igualmente un control sobre el número de muestras utilizado al permitir elegir el número de subdivisiones realizadas.
- No presenta el ruido aleatorio existente en Monte Carlo.
- Las muestras se generan de manera adaptativa, de manera que zonas que no requiera gran cantidad de muestras para ser computadas no las tendrán, al contrario que en el Método de Monte Carlo.

Conocidas las características del método presentado, es necesario analizar el problema que presenta el desarrollo del mismo.

A la hora de analizar el problema hay que entender que nos encontramos ante la integración de una función multidimensional.

El proyecto existente que se utiliza como base para este trabajo facilita una función que devuelve el valor de la *path integral* (Sección A.2) para unos valores

concretos de los parámetros de la misma.

El número de parámetros que se pueden utilizar en esta función es virtualmente infinito, como ya se ha explicado en la Sección A.2, así mismo en esta sección se explica que los parámetros son expresables como números del intervalo $[0 \ldots 1]$. Dichos parámetros corresponden a:

Parámetro 0 Coordenada **X** del pixel de la imagen.

Parámetro 1 Coordenada **Y** del pixel de la imagen.

Parámetros 2 a N Ángulo de rebote del rayo de luz en el punto de intersección.

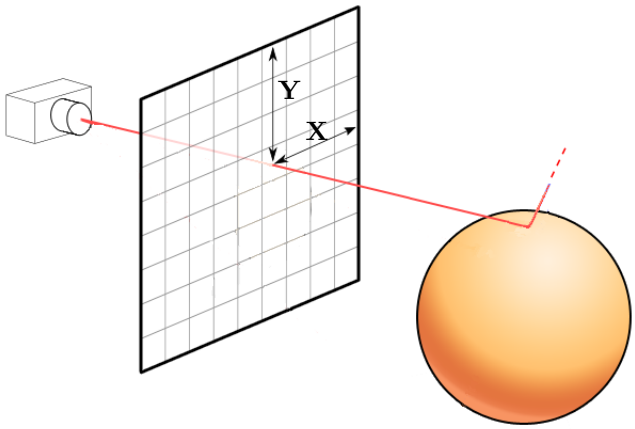


Figura 3.1: Los dos primeros parámetros representan las coordenadas del pixel de la imagen analizado, mientras que el resto de parámetros especifican el ángulo de los sucesivos rebotes que se producen.

Mediante esta función, se pretende realizar la aproximación necesaria para la computación de la iluminación global haciendo uso de una técnica adaptativa multidimensional.

Esta técnica presenta ciertas ventajas frente a métodos de integración alternativos que pueden ser utilizados para la resolución de este problema. Técnicas como el comentado método de Montecarlo poseen la desventaja de la existencia de ruido en sus aproximaciones (Figura 3.2), debido a la utilización de números aleatorios para la generación de sus muestras.

En contraposición, el método que se pretende realizar en este trabajo, no presenta ruido en la aproximaciones, e igualmente permite un control sobre el número de muestras utilizado durante las aproximaciones, tal y como el Método

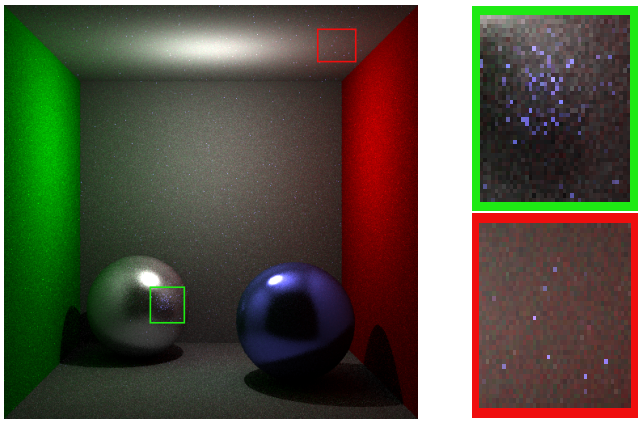


Figura 3.2: Ejemplo de un imagen renderizada utilizando el Método de Montecarlo. Se puede observar el ruido aleatorio generado en la imagen.

de Montecarlo permite.

Para abordar el problema de la resolución de esta integral multidimensional es necesario decidir como utilizar conjuntamente los conceptos de integración adaptativa e integración multidimensional vistos en la sección **??**.

La adaptación más sencilla sería realizar el esquema adaptativo planteado para integrales unidimensionales entendiendo que la integral multidimensional es realmente una sucesión de integrales unidimensionales, por las que se puede aproximar la integral.

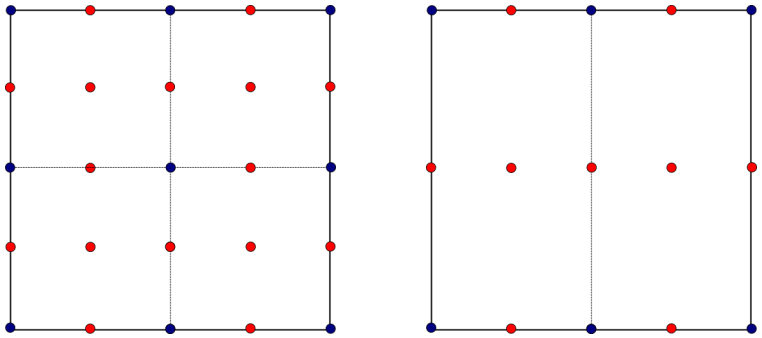
Sin embargo, esto haría que la comparativa que se realizara para estimar el error indicara unicamente la existencia de un error en el conjunto de la aproximación, lo que llevaría a tener que subdividir cada uno de los intervalos para lograr una mejor aproximación.

Un ejemplo de como evolucionaría la cantidad suponiendo que se parte desde una estructura 2D como en la Figura 4.1 sería lo que se puede observar en la Figura 3.3a.

Se plantea por tanto, calcular un error por dimensión, que permita encontrar cual de las dimensiones aporta un mayor error a la aproximación, de manera que la subdivisión se produzca únicamente en esta.

De esta manera se evita que se produzca posibles subdivisiones innecesarias que generarían una gran cantidad de cálculos que quizá no aportaran precisión.

Para realizar esto, se realizan los cálculos con ambas reglas de cuadratura aproximando la integral sobre cada una de las dimensiones haciendo uso del Teorema de Fubini (7.13). Así, se puede elegir la dimensión que mayor aporta y subdividir por ella (Figura 3.3b).



(a) División múltiple (b) División en 1 dimensión

Figura 3.3: Comparativa de como evoluciona el número de muestra cuando las subdivisiones se producen en todas las dimensiones o en solo la de mayor error (Trapecio: Muestras azules; Simpson: Muestras azules y rojas).

Igualmente, dado que se ha cálculado el error en cada una de las dimensiones se puede calcular sin coste añadido un error acumulado para toda la subdivisión (como la suma de los errores de cada dimensión), el cual se puede usar para ordenar cada una de las subdivisiones en la estructura.

Puesto que hay que hacer uso de dos técnicas de cuadratura distintas (para poder realizar una estimación del error), se elige utilizar para este trabajo las reglas de Simpson y de Trapecio.

Teniendo clara la técnica que se va a usar para realizar los cálculos adaptativos del problema, se plantea discutir 2 alternativas distintas sobre como abordar el planteamiento del problema como tal.

3.4. Subdivisión por pixel

En esta aproximación se ha obviado la integración de los dos primeros valores de la ecuación, de manera que para cada pixel existente en la imagen se ha optado por tomar un único valor de los párametros que están correlados con la situación del mismo, en lo que a sus coordenadas X e Y respecta (parámetros 0 y 1), intentando realizar la aproximaxión en base al resto de parámetros.

La causa por la cual esta aproximación decide realizar un muestreo en las dos primeras dimensiones, en lugar de realizar la aproximación de la integral con todas las dimensiones, es intentar reducir el problema que supone la ya mencionada maldición de la dimensionalidad.

Puesto que con la aproximación elegida para el cálculo multidimensional de la integral, el coste computacional crece exponencialmente en función a las di-

mensiones de estudio, sustituir el cálculo en las dos primeras dimensiones por una aproximación que usa un único valor para la función en sus dos primeros parámetros, reduce la cantidad de dimensiones que se integran. De esta manera un problema N-dimensional se reduce a un problema de N-2 dimensiones.

3.5. Subdivisión de la imagen

En esta aproximación si que se consideran las dos dimensiones de la imagen como dimensiones del problema, de manera que, al contrario que en la aproximación anterior, no se realizará una única muestra en estas, si no que el cálculo de la integral tendrá en cuenta el intervalo existente en ellas.

Como esta aproximación tiene en cuenta también las 2 dimensiones que hacen referencia a las coordenadas de los pixeles de la imagen, las subdivisiones pueden cubrir varios pixeles de la imagen al mismo tiempo. De esta manera se utilizaran menos muestras si es posible para realizar un cálculo que la otra aproximación habría realizado tomando muestras siempre en cada pixel por separado.

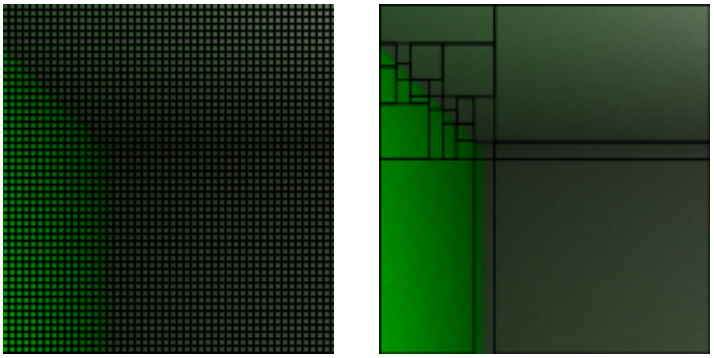


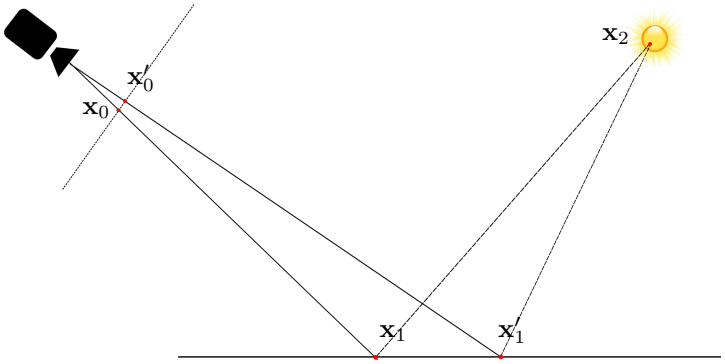
Figura 3.4: Ejemplo de como los pixeles de una de una región de una imagen (izquierda) podrían repartirse en distintas subdivisiones e tamaños distintos (derecha) si el problema se aproximara incluyendo las dimensiones de la imagen tal y como las lineas de la figura expresan.

Por tanto, es fácil llegar a la conclusión de que en esta aproximación se puede comenzar suponiendo una subdivisión única que cubra la totalidad de todos los intervalos existentes. Sabiendo esto, surge la duda de como asignar entonces los valores correspondientes a cada pixel de la imagen una vez el algoritmo ha finalizado.

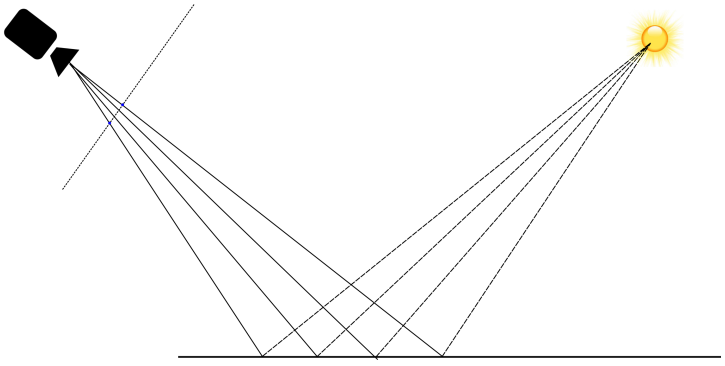
Además, este enfoque del problema permitiría que, puesto que en el momento que una de las subdivisiones conseguidas tenga nulo error, o al menos un error los

suficientemente bajo como para considerar que no se ha de subdividir, aproximar la integral de toda la subdivisión suponiendo que se ajusta a la aproximación que haría la regla de cuadratura usada.

Así sería posible utilizar las muestras que se han tomado para aproximar lo que podría ser una superficie de más de un pixel para aproximar el valor de cada pixel suponiendo que el comportamiento de los valores de la subidivisión se rige por el polinomio por el que se aproximó, permitiendo ahorrar una gran cantidad de muestras que en la otra aproximación al problema que se presenta en este trabajo si que son necesarias.



(a) Subdivisión por pixel



(b) Subdivisión de la imagen

Figura 3.5: Mientras que en la subdivisión por pixel se muestree cada uno de los mismo, en el caso de la subdivisión de la imagen, es posible que un único pixel de la imagen tenga que ser interpolado con las muestras resultantes de las múltiples divisiones que se hayan producido en las dimensiones que corresponden a la imagen.

4. Diseño

Se presenta en esta sección el diseño del algoritmo.

Aunque el diseño se encuentra ampliamente explicado en esta sección, está complementado con un diagrama de clases, cuyos nombres corresponden además con las clases finalmente implementadas que se explican a lo largo de la sección 5.

El diagrama de clases mencionado (en lenguaje **UML** [RJB04]), resultado de la fase de diseño se encuentra presente en el Anexo C de este documento.

4.1. Esquema general del algoritmo

El algoritmo general sigue el esquema planteado anteriormente, de manera que se ajusta al esquema adaptativo para integrales multidimensionales comentado anteriormente.

Se presenta a continuación la idea general del algoritmo expresada en un pseudocódigo fácilmente entendible:

```
Inicializar la cola de prioridad;
generar muestras iniciales;
iteraciones = 0;
mientras iteraciones <maxIters AND queue no vacia hacer
    division = pop(queue);
    // Subdivide dando 2 subdivisiones resultantes.
    subdivide(division, first, second);
    si error(first) >umbral entonces
        push(first);
    en otro caso
        Añadir aportación de first a la imagen;
    fin
    si error(second) >umbral entonces
        push(second);
    en otro caso
        Añadir aportación de second a la imagen;
    fin
fin
// Añadir aporaciones restantes a la imagen.
para cada division in queue hacer
    Añadir aportacion de division a la imagen;
fin
```

Algoritmo 1: Esquema general del algoritmo presentado en este trabajo.

En primer lugar se realiza un fase inicial de muestreo. Se realizan las subdivisiones iniciales sobre el problema y se almacenan en una estructura.

Posteriormente se realiza iterativamente un proceso en el cual se extrae de la estructura la subdivisión con mayor error estimado, para volver a subdividirla respecto, teniendo en cuenta que dimensión aporta mayor más error a la aproximación realizada, siendo esa misma la que elegirá para subdividir.

El uso de la estructura comentada se hace necesario puesto que se desea tener un almacenamiento de las distintas subdivisiones del problema que permita un acceso eficiente y rápido a aquellas con mayor error. Esta estructura se comenta con más detalle en una de las sucesivas secciones que se centra precisamente en la propia estructura.

Las nuevas subdivisiones resultantes, comentadas anteriormente, se introducen en la estructura, retirando de la misma estructura la subidivsión de la que se generaron, puesto que ahora está se encuentra aproximada como la suma de las resultantes del proceso comentado.

Adicionalmente puede establecerse un umbral de error a partir del cual se considere la aproximación como válida, de manera que si son generadas subdivisiones cuyo error es menor que este umbral, estas no sean introducidas en la estructura, y pasen directamente a formar parte de la aproximación definitiva.

Utilizar este umbral es interesante debido a que si únicamente se consideraran válidas aquellas con error estimado exactamente igual a 0, pocas subdivisiones serían elegidas antes de la finalización del algoritmo. Eligiendo un valor de umbral superior a 0, pero igualmente pequeño, se puede conseguir que aquellas subdivisiones con un error despreciable ya no vuelvan a ser subdivididas.

4.2. Subdivisión por pixel

La aproximación por pixel se ha diseñado de tal manera que los cálculos que se realizan para aproximar las dos primeras dimensiones son obviados. Estas dimensiones serán unicamente muestreadas, de manera que para cada pixel de la imagen se considere válida una aproximación que toma como valor de estas dos dimensiones las coordenadas centrales de dicho pixel.

Así, se aproximará mediante el algoritmo el valor que ha de asignarse a este pixel en la imagen final.

Para la estimación del error sobre cada dimensión, bastaría con realizar repetidamente las integrales unidimensionales hasta que el problema quedara como una integral unidimensional sobre la dimensión, en la cual se desea realizar la estimación del error. Esto sería posible realizarlo para cualquier dimensión de la integral, gracias a la propiedad enunciada en el Teorema de Fubini, visible en la ecuación (7.13).

De esta manera el problema ya se adaptaría al esquema planteado para el cálculo adaptativo visto en la sección B.3, pudiendo interpretar como la nueva

función a integrar la obtenida mediante integración repetida sobre las las dimensiones que no son la de estudio. Así, solo restaría realizar la aproximación mediante dos reglas de cuadratura distintas, y realizar el cálculo del error como la comparativa del resultado de ambas, que sería la estimación del correspondiente a la dimensión que no se integró iterativamente.

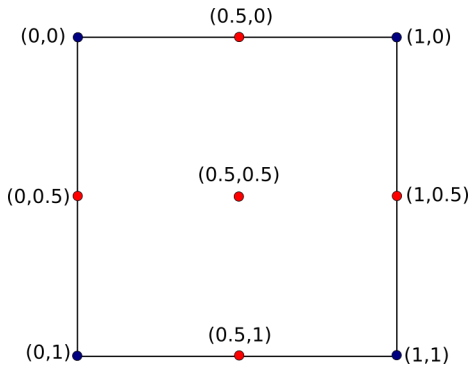


Figura 4.1: Ejemplo de muestras necesarias para las reglas de Simpson y Trapecio en 2 dimensiones (Trapecio: Muestras azules; Simpson: Muestras azules y rojas).

Si este proceso se repite, dejando fija cada una de las dimensiones, y se almacena el error obtenido en cada una, se puede saber que dimensión acumula un mayor error, de manera que se subdivida el intervalo correspondiente a dicha dimensión, y se aproxime la integral múltiple como la suma de las dos integrales mútiples obtenidas según los intervalos que se tuvieran tras la subdivisión.

Como se ha comentado previamente (Sección B.3) el cálculo de la estimación del error se realiza sin coste adicional, dado que para el cálculo adaptativo se hará uso de las reglas de cuadratura de Simpson (7.10) y Trapecio ((7.9)).

El número de muestras necesarias sería, por tanto, 3^N, siendo N el número de dimensiones de estudio. Un ejemplo de la posición de las muestras en 2 dimensiones puede observarse en la Figura 4.1.

4.3. Subdivisión de la imagen

Como se ha comentado, el planteamiento del problema también puede orientarse de manera que los cálculos no se realicen por cada pixel de la imagen, si no que se consideren las dimensiones X e Y de la imagen dentro del proceso que seguirá el algoritmo adaptativo para calcular la integral.

De esta manera, se llegará en la mayoria de casos a subdivisiones que incluyan varios pixeles debido a como se han subdivido los intevalos correspondientes a

las dos primeras dimensiones del problema. Esto permitirá, que si el error de esa subdivisión se considerá despreciable, incloso en esas 2 primeras dimensiones, sea posible apriximar toda esa superficie de pixeles cubierta por la subidivisión con una cantidad, menor de muestras que en la aproximación anterior al problema, en la que cada pixel es considerado de manera individual.

Para cacular los valores de los pixeles de la subidivisión habría que seguir el siguiente proceso.

Suponiendo que por integración reiteradada con las técnicas comentadas, se integra hasta que quede una función únicamente dependiente de los parámetros X e Y, dado que en el resto de dimensiones se ha aproximado la integral, podemos suponer que los valores de esta función, que hemos reducido a 2-dimensional se ajustan al polinomio en 2 dimensiones que nos marca la interpolación que se realiza por la Regla de Simpson.

Esta suposición marcaría que la ecuación de render 2-dimensional se ajusta al siguiente polinomio general:

$$f(x,y) = c_{22}x^2y^2+c_{21}x^2y+c_{20}x^2+c_{12}xy^2+c_{11}xy+c_{10}x+c_{02}y^2+c_{01}y+c_0 \quad (4.1)$$

De este polinomio es posible, así mismo, calcular su integral bidimensional de manera analítica:

$$\iint f(x,y)dxdy = \frac{x^3}{3}\Big(\frac{c_{22}y^3}{3}+\frac{c_{21}y^2}{2}+c_{20}y\Big)+\frac{x^2}{2}\Big(\frac{c_{12}y^3}{3}+\frac{c_{11}y^2}{2}+c_{10}y\Big) + x\Big(\frac{c_{02}y^3}{3}+\frac{c_{01}y^2}{2}+c_0y\Big) \quad (4.2)$$

De esta manera es posible hacer uso de esta ecuación para cálcular el valor que en cada subdivisión aporta por separada a cada uno de los píxeles que engloba, cálculando los coeficiente con las muestras que ya se han tomado para la Regla de Simpson sobre dos dimensiones (la de la imagen).

Una vez se tienen estos valores se resuelve el sistema que permite obtener los valores de los coeficientes, de manera que quede una ecuación que permite aproximar el valor para el area de cada uno de los pixeles que se encuentran dentro de la subdivisión.

De esta manera se realiza una aproximación para cada uno de los píxeles sin tener que cálcular muestras adicionales a las que ya se utilizarón para los cálculos necesarios referentes a la parte adaptativa ya comentada ampliamente en secciones previas.

4.4. Estructura para las subdivisiones

Una parte indispensable del diseño del algoritmo es el diseño de una estructura que almacene las subdivisiones que el algoritmo genera, y los datos de las

mismas.

Así mismo, para entender el espacio multidimensional que estas subdivisiones tratan, se propone abstraer el problema al caso 2-dimensional, de manera que la estructura que formarían los intervalos sería un cuadrado (Figura 4.1).

4.4.1. Datos a almacenar

La estructura de las subdivisiones almacena la siguiente serie de datos necesarios para el algoritmo:

- **Posición de los vertices**: Requiere 2 valores en coma flotante (dato tipo *float*) para cada una de las dimensiones que se tengan en cuenta ($2 * 4$ bytes por dimensión).
- **Evaluación del *Path Contribution***: El valor aportado por la subdivisión a la aproximación de la integral (Anexo A.2). Almacena un valor en RGB[2] donde cada canal se encuentra alamcenado en un *float* ($3 * 4$ bytes).
- **Aproximación del error acumulada**: Suma del error estimado (Anexo B.3) para cada una de las dimensiones. Utiliza un dato de tipo *double* para mayor precisión (8 bytes).
- **Dimensión de mayor error**: Almacena cual de las dimensiones analizadas aporta mayor error, para realizar subivisiones por la misma. Usa un dato de tipo *unsigned char* (1 byte).

De esta manera cada una de las dimensiones requiere un total de $21 + 8 * N$ bytes, donde N es el número de dimensiones analizadas en el problema.
Adicionalmente los diseños por pixel necesitan 8 bytes para almacenar las coordendas del pixel al que hacen referencia

Sería posible reducir este número, pues la representación de la subdivisión requeriría unicamente los datos de los vértices, dado que el resto pueden calcularse a partir de estos, pero el almacenamiento adicional de datos permite ahorrar cálculos repetidos.

4.5. Cola de prioridad

También es necesario pensar en que estructura se van a almacenar estas subdivisiones, puesto que sería interesante acceder rápidamente, y sin un coste computacional elevado, a aquella subdivisión que, entre las presentes en un momento dado del algoritmo, tenga el mayor error.

[2]Espacio de color que permite representar un color haciendo uso de los valores del color **rojo**, **verde** y **azul** que aditivamente forman el color que se desea representar.

La estructura más conocida que cumple con estos requerimientos es la **cola de prioridad**, y de sus implementaciones se elige la conocida como **montículo** por ser de las más eficientes [CLRS09, Section 6.5].

4.5.1. Montículo

Esta implementación de la cola de prioridad tiene unos costes acordes a las necesidades del problema, ya que las inserciones y borrados tienen un coste $O(\log n)$, además de un acceso con coste $O(1)$ al primer elemento de la estructura, que por definición es el mayor (o menor, según se indique) elemento respecto a una ordenación indicada.

Utilizando esta estructura, de manera que la ordenación coloque como primer elemento de la cola de prioridad la subdivisión con mayor error, tendríamos un acceso rápido al elemento que tendría mayor prioridad a la hora de subdividir, y podrían insertarse fácilmente con coste logarítmico las subdivisiones resultantes mientras que este elemento, ya subdividido, se retira de la estructura.

4.5.2. Estructura propia

A pesar de que el montículo presenta una cualidades prácticamente óptimas para los requerimientos del problema presenta una desventaja.

Es cierto que los costes que esta estructura presenta son ciertamente bajos, pero, puesto que el crecimiento del número de datos en este problema es elevado (al analizar un dato generamos dos nuevos), y al analizar los datos estos son extraídos de la estructura y en la mayoría de los casos provocan que dos nuevos datos sean generados, nos encontramos con que realmente la operación de menor coste de la cola de prioridad es superada por las 3 operaciones de coste $O(\log n)$ que produce en el peor de los casos la obtención de uno de los datos de la estructura, que se ve reducida a una operación de igual coste en el caso mejor.

Además, esta estructura al utilizar únicamente memoria RAM, y crecer el número de datos de la forma ya comentada, el coste de memoria puede crecer hasta el punto de llegar a ser un problema crítico.

Esto ocurre sobre todo en el planteamiento que utiliza un montículo global y realiza tratamiento por pixel, donde el número máximo de subdivisiones presentes en la estructura depende de la siguiente ecuación:

$$subdivisiones = (altura * anchura * (subdivisionesIniciales + 1)) + iteraciones \tag{4.3}$$

Se observa, por tanto, que sobre todo para imágenes grandes este planteamiento aumenta en gran medida el número de subdivisiones presentes en la estructura, por lo que se pretende diseñar una estructura para tratar casos en

los que esto ocurra.

Por ello, se ha planteado el diseño de una estructura que pueda ser utilizada en el caso de que los factores comentados anteriormente suponga un problema. Esta estructura se ha diseñado de manera que, aunque su eficiencia en acceso a datos puede llegar a no ser tan elevada como la que se consigue mediante la utilización de una cola de prioridad, permita relajar el uso excesivo de RAM.

Puesto que no es estrictamente necesario subdividir aquella región que tenga el mayor error, sino subdividir prioritariamente las de mayor error, podemos utilizar una estructura que no devuelva estrictamente la región con el mayor error entre todas, sino una de las que mayor error posea.

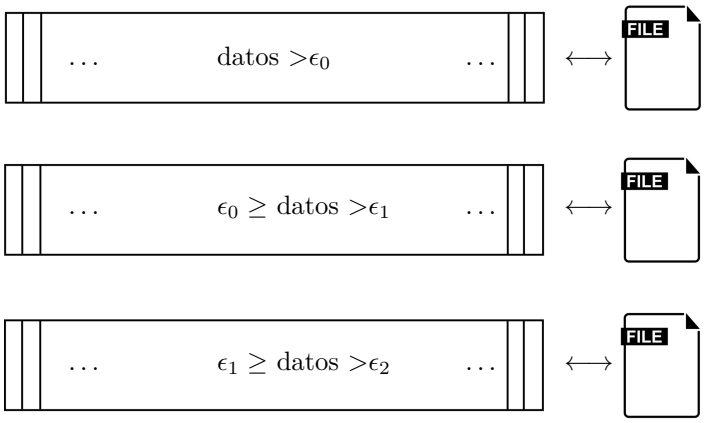


Figura 4.2: Los datos de la nueva estructura son divididos entre distintos vectores según el resultado de unas comparaciones. La estructura utiliza ficheros como almacenamiento secundario para cada vector.

Basándose en esta idea, la estructura diseñada implementa distintos grupos de subdivisiones con error similar, de manera que se acceda por prioridad a estos grupos, devolviendo en primer lugar subdivisiones que pertenezcan al grupo con mayor error, y si este está vacío, se proceda a devolver datos del siguiente grupo de datos. Adicionalmente los datos introducidos a la estructura se introducirán en el grupo al que correspondan según su error.

De esta manera el acceso a los datos es de orden $O(1)$ frente a la inserción del montículo con coste $O(n \log n)$.

Para solucionar el problema de almacenamiento presente igualmente en el montículo, la estructura realiza un volcado a disco físico cuando el número de datos supera un límite específico.

Esto sin embargo puede hacer que el coste de los accesos a los datos puede crecer, dado que al introducir un dato en la estructura se podría provocar un

volcado a disco de parte de los datos, aumentando el coste de la operación. Igualmente si la extracción de un dato de uno de los grupos comentados produjera que este quedara vacio, sería necesario analizar si se han volcado datos a disco, y si así ha sido, cargarlos desde el mismo, lo que de igual manera que al introducir un dato, aumentaría el coste de la operación.

Por tanto, esta segunda estructura que se plantea, se presenta como alternativa a la cola prioridad, principalmente, para aquellos casos en los que una cola prioridad falle en cuanto a motivos de almacenamiento se refiere, aunque si la cantidad de datos generados es pequeña, y la estructura no necesita de volcados a disco las inserciones y extracciones tendrán un coste más eficiente que en la cola de prioridad.

5. Implementación

5.1. Tecnologías y herramientas utilizadas

Para la implementación del algoritmo se ha utilizado como lenguaje de programación **C++**, concrétamente se ha realizado la implementación en la versión C++14 (se puede saber más sobre la extensión de esta versión sobre el lenguaje C++ en [Str13]), utilizando además las conocidas librerias para este lenguaje denominadas *Boost* [boo].

El código de la implementación ha sido compilado haciendo uso del compilador Clang [cla], en el sistema operativo Fedora 23, aunque el proyecto es compilable en cualquier otro entorno Unix.

También se ha utilizado la herramienta CMake [cma] para la construcción del proyecto, así como su configuración.

La implementación de este proyecto utiliza a un software ya desarrollado como base, haciendo uso de las funciones ya implementadas para realizar el muestreo de datos, así como la generación de imágenes con los datos que el algoritmo que se implementa en este trabajo aproxima. La autoría de este software base pertenece a Adolfo Muñoz Orbañanos[3], director de este Trabajo de Fin de Grado.

5.2. Implementación de las subdivisiones

En el apartado de diseño, se han comentado las estructuras necesarias para el funcionamiento del algoritmo. En este caso, la implementación de dichas estructuras ha sido realizada mediante clases C++, de manera que se utilicen instanciaciones de las mismas en caso de ser necesarias.

Respecto a las estructuras ya comentadas para almacenar la información que corresponde a las subdivisiones, se han implementado dos clases, ***PointDivision*** y ***PointDivisionImage***.

La primera de estas clases implementa la estructura necesaria para las subdivisiones que se plantean en la primera aproximación del problema. Almacena los datos necesarios, y posee funciones para consultarlos, así como distintas funciones que realizan los cálculos del error sobre cada dimensión, y la aproximación del valor correspondiente para la función en el intervalo que cubren, utilizando valores fijos para las dos primeras dimensiones (X e Y), que pueden ser consultados igualmente a efectos de conocer el pixel al que hacen referencia.

[3] http://giga.cps.unizar.es/~amunoz/index.php/es/

La segunda clase, implementa una estructura similar a la primera, con la diferencia de que en esta se tienen en cuenta los valores de las dos primeras dimensiones a la hora de realizar cálculos, puesto que en estas dimensiones se consideran también intervalos, y no valores fijos como en el caso anterior.

5.3. Colas de prioridad

Igualmente, las estructuras en las que se almacenan los objetos resultantes de instanciar estas clases a lo largo del algoritmo se encuentran implementadas mediante clases.

En el caso de la cola de prioridad, esta ya se encuentra implementada dentro de la librería *Standard Template Library (STL)*, por lo que se ha hecho uso de esta implementación.

Respecto a la otra estructura comentada en la sección de diseño, se ha implementado una clase C++, haciendo uso de unas funciones igualmente implementadas en las clases correspondientes a las subdivisiones. Estas funciones han sido implementadas para poder realizar un guardado de los objetos en ficheros, así como la para construir estos objetos a partir de datos previamente guardados en un fichero.

De esta manera, la clase implementada realiza un guardado en distintos grupos de vectores, que en caso de llegar a un limite de datos, son volcados a ficheros en disco, e igualmente si quedan vacíos se llenan con los datos de estos ficheros. Estos vectores pueden variarse en número y límite de datos, y puede elegirse a partir de que valores de error es elegido uno u otro vector para el almacenamiento de los datos.

Esta clase se ha implementado de manera que presente unas funciones similares a las que posee una cola la cola de prioridad, simplemente a efectos de similitud entre ambas estructuras (funciones *push*, *pop*). De esta manera, una vez se haya construido un objeto de esta clase, el funcionamiento externo del mismo será equivalente al de una cola de prioridad, variando únicamente el funcionamiento interno.

5.4. Implementación de los engines

Respecto a la implementación del algoritmo en si mismo, las distintas variantes del mismo se han implementado igualmente en clases,de manera que los ejecutables del proyecto hagan uso de las mismas. Estas clases se han denominado ***engines***

Estas clases (a partir de a se construyen con referencias a otras ya existentes en el código base utilizado para el proyecto, y como se ha comentado son las que permiten realizar un muestreo de los distintos valores de la función *path integral* necesarios para el desarrollo del algoritmo implementado.

Estas clases utilizadas son la clase ***RecursiveRayTracer*** y las distintas clases que esta misma utiliza, tanto para su definición como para su funcionamiento.

Las clases correspondientes para estos *engines* implementan por tanto los distintos diseños del algoritmos, de manera que las clases ***EngineQuadrature-Global*** y ***EngineQuadraturePerpixel*** implementan ambas el algoritmo realizando cálculos para cada pixel (Sección 4.2), diferenciandose en que la primera utiliza una única cola de prioridad para todo el proceso, mientras que la segunda utiliza una cola de prioridad en cada uno de los pixeles del proceso.

Así mismo existe un tercer engine, ***EngineQudratureImage***, que implementa el algoritmo realizando subdivisiones que incluyen las dimensiones de la imagen, X e Y (Sección 4.3).

Igualemente existe un *engine* adicional por cada uno de los comentado que utiliza como cola de prioridad la estructura comentado en la Sección 4.5.2, que tiene el nombre del *engine* comentado anteriormente seguido por la palabra *File* (***EngineQuadratureImageFile***, ***EngineQuadratureGlobalFile***, ...).

5.5. Ejecutables del proyecto

Respecto a los ejecutables que se comentaba con anterioridad, existe un ejecutable para cada implementación del algoritmo (es decir, cada *engine* implementado), que utiliza la clase correspondiente a la implementación del algoritmo que se desea ejecutar en el mismo.

De esta manera, el ejecutable genera dos objetos de la clase que le corresponde, para generar por separado la aproximación de la componente especular de la imagen y la aproximación de la componente difusa, utilizando ambas para generar la imagen final.

Todos estos ejecutables poseen opciones para cambiar los valores de las distintas variables de los algoritmos que ejecutan, mediante la introducción de valores distintos a los predefinidos haciendo uso de la linea de comandos durante su invocación, así como opciones para elegir la escena sobre la que se ejecutará el algoritmo, y el formato en el que se guardará la imagen resultado de la ejecución.

6. Validación y Resultados

Durante esta sección se van a mostrar los resultados de la técnica implementada durante este proyecto.

Como se ha comentado, el objetivo de este trabajo es realizar una técnica de simulación de iluminación global de caracter adaptativo, alternativa a las técnicas existentes para ello, por lo que igualmente durante esta sección se realizarán comparativas de otras técnicas con los resultados obtenidos mediante los algoritmos implementados en este trabajo.

6.1. Validación

En esta sección se pretende verificar que el algoritmo que se presenta funciona correctamente. Para ello se realiza una comparativa de los resultados que se obtienen utilizando este para realizar la simulación de iluminación global de una escena frente a los resultados que se obtienen para dicha escena haciendo uso de técnicas de *path tracing*[4] tradicional. Concretamente se va a realizar la comparativa utilizando el diseño del algoritmo que realiza subdivisiones adaptativas en cada uno de los pixeles, como ya se ha presentado en las secciones 3.4 y 4.2.

La Figura 6.1, muestra como el algoritmo presentado converge a una solución equivalente a la obtenida mediante *path trace* en un tiempo sustancialmente menor. A pesar de ello se puede observar un sesgo en los resultados del algoritmo, debido a que el muestreo del algoritmo siempre se realiza bajo un patrón.

Los parámetros que aparecen en esta figura, así como algunos que aparecen en sucesivas figuras corresponden a:

- ***paths***: Número de caminos explorados mediante el algoritmo de *path trace*. Un número elevado de caminos supondrá un mejor resultado, pero a su vez hará que aumente el tiempo de computo.
- **div**: Número de divisiones iniciales que realiza el algoritmo adaptativo.
- **iter**: Número de iteraciones realizadas por el algoritmo adaptativo.
- **thres**: Valor umbral o de corte con el cual se discrimina que datos no se almacenan en la estructura (si su error es menor que el valor de este umbral.

Para mayor compresión del significado de estos parámetros puede observarse su relevancia en el algoritmo, revisando el pseudocódigo que expresa el funcionamiento general del mismo en (Algoritmo 1).

[4] Método de simulación de iluminación global basado en la integración por el Método de Monte Carlo, desarrollado a partir de las ideas expuestas en [Kaj86] y en [Vea98]. Es por tanto un método no adaptativo al contrario que el presentado a lo largo de este trabajo.

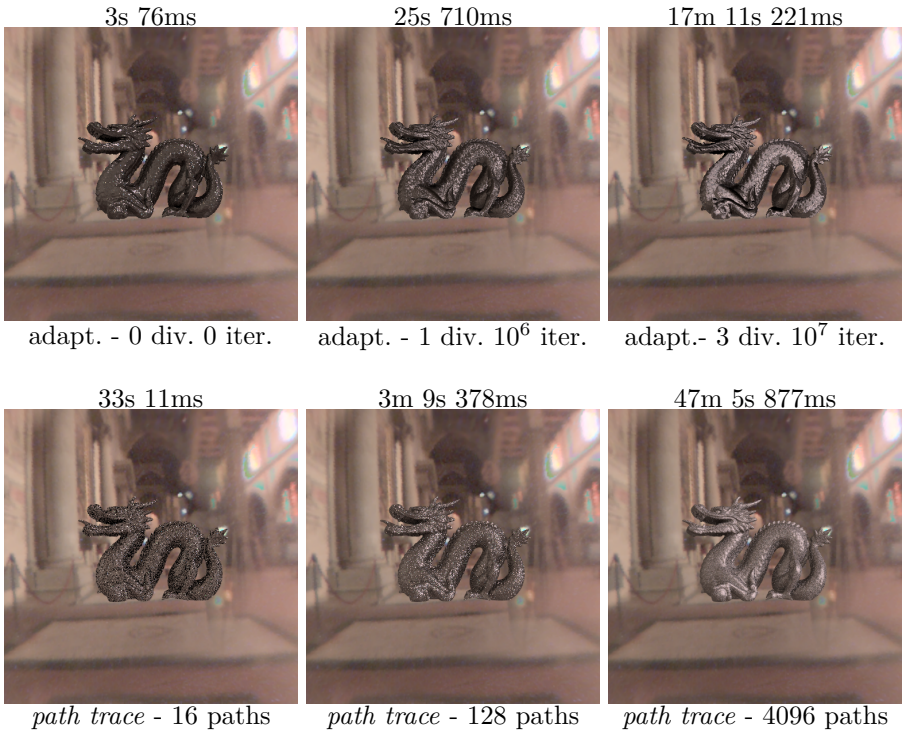


Figura 6.1: El algoritmo adaptativo que se presenta (fila superior) hacia a la solución óptima para la escena que se presenta. Este algoritmo presenta resultados similares (se obtiene cierto ruido estructurado) a los obtenidos por el algoritmo de *path trace* (fila inferior) en tiempos de ejecución similares

6.2. Subdivisión de la imagen

Como se ha expuesto en la fase de análisis en la sección 3.5 y durante el diseño en la sección 4.3, se ha diseñado igualmente el algoritmo para poder realizar divisiones, no solo en cada uno de los pixeles, si no subdividiendo todo el espacio de la imagen.

Este algoritmo de subdivisión presenta reseñables resultados para tareas tales como el *antialiasing*[5]. Este concepto se puede observar en la Figura 6.2 donde se muestra una comparativa entre los distintos resultados de métodos para una simulación de la iluminación directa en una escena concreta.

Así mismo, este algoritmo puede utilizarse para realizar reducir el ruido causado cuando se hace uso del Método de Monte Carlo. El concepto del ruido existente en el Método de Monte Carlo se encuentra más explicado en el Anexo B.2.

[5]El proceso de *antialiasing* pretende minimizar el efecto causado por el muestreo de señales de manera digital, también llamado *aliasing*.

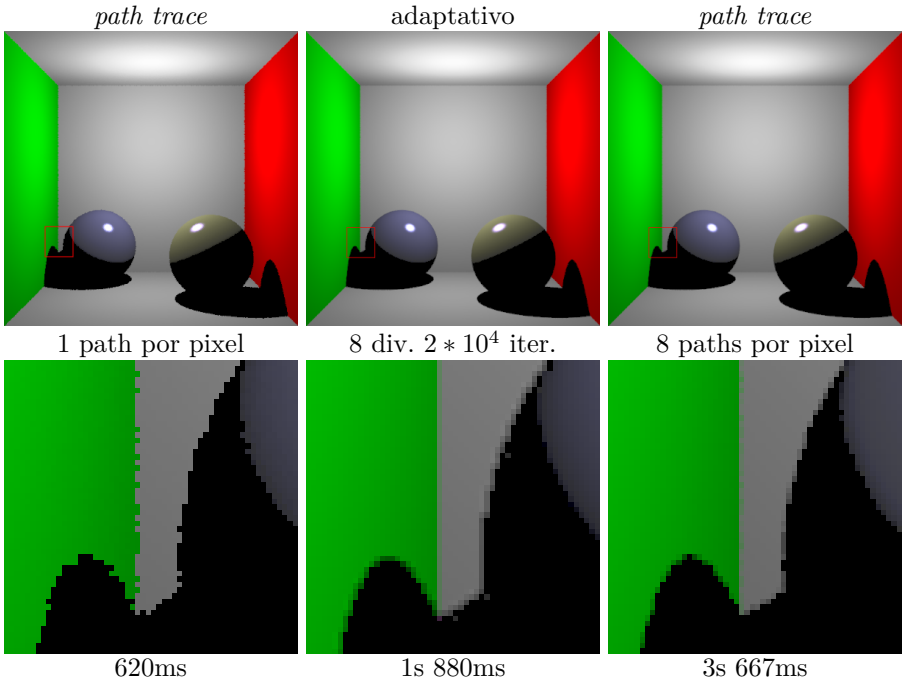


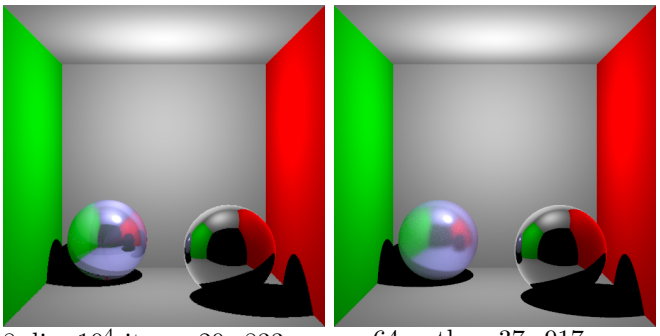
Figura 6.2: El algoritmo de subdivisión de la imagen (imagen central) mejora el primer caso presentado que solo hace uso de una muestra en cada pixel, mientras que presenta una calidad equiparable a la conseguida que si se utilizan 8 muestras por pixel, presentando además un tiempo de ejecución menor.

Se presenta en la Figura 6.3 una comparativa entre la misma escena realizada haciendo uso del algoritmo de subdivisión de la imagen y del Método de Monte Carlo de manera simple para poder comparar como el número de muestras usadas afecta a ambos.

6.3. Estructura de almacenamiento

Como ya se ha explicado en el apartado de diseño en la sección 4.5, todas las versiones del algoritmo hacen uso de una cola de prioridad para el almacenamiento de los datos, así como la selección del orden de subdivisión. Esta cola de prioridad idealmente se plantea como un mónticulo por las propiedades del mismo, sin embargo, en casos en los que la cantidad de datos crece altamente, la implementación de la cola de prioridad como un montículo deja de ser una opción, por lo que como se propone, se plantea el uso de una estructura menos restrictiva, pero con menores costes tanto de memoria como de ejecución de sus operaciones.

Como se puede observar en los resultados de la Figura 6.4, la estructura de



8 div. 10^4 iter. - 20s 833ms 64 paths - 37s 917ms

Figura 6.3: El algoritmo de subdivisión de la imagen (izquierda) no muestra el ruido aleatorio que se produce al utilizar el Método de Monte Carlo (derecha). Así mismo consigue generar un resultado comparable al Método de Monte Carlo en un tiempo menor.

almacenamiento diseñada como alternativa al montículo presenta mejores coste de memoria, mientras que mantiene unos resultados similares a los obtenidos haciendo uso del montículo.

0 div. $7{,}5 * 10^6$ iter 0.04 thres. 0 div. $9{,}5 * 10^6$ iter 0.04 thres.

Figura 6.4: Los resultados obtenidos haciendo uso del montículo (izquierda) son comparables a los obtenidos mediante la estructura alternativa (derecha), quitando algunas zonas subdividas de manera equivocadamente prioritaria que otras, mientras que el coste de memoria es altamente inferior.

Sobre el coste de memoria empleado por cada uno de los algoritmos, se puede especificar un máximo pero no el coste exacto, debido al valor de umbral presente, así como el proceso interno de la estructura alternativa al montículo. De esta manera los coste de memoria física, no disco, máximos serían de aproximadamente 300 MB para el caso del montículo, frente a aproximadamente

31MB en el caso de la estructura alternativa ya que se encuentra fijado por la propia estructura.

7. Conclusiones y trabajo futuro

7.1. Conclusiones obtenidas

Tanto durante el proceso de planteamiento, diseño e implementación de este trabajo, como durante la fase de análisis de los resultados que se han obtenido, se ha llegado a múltiples conclusiones.

En primer lugar, el algoritmo que este trabajo propone presenta una importante mejora para casos de mapas de entorno y para escenas que presentan zonas "suaves", que métodos aleatorios como el Método de Monte Carlo simulan con ruido debido a su aleatoriedad. Para dichas zonas, el método adaptativo que este trabajo propone consigue reducir el ruido así como la cantidad de muestras empleadas para la obtención de resultados.

En lo que a conclusiones más concretas respecta, se ha observado que, como era esperable, el método propuesto con subdivisiones en todas las dimensiones de la imagen, encuentra problemas en zonas de las imágenes que presenten discontinuidades, donde necesita realizar una gran cantidad de subdivisiones para conseguir un resultado, al contrario que en las zonas suaves de las mismas. Este hecho se tratará más profundamente en el apartado de Trabajo Futuro.

Por último, comentar las conclusiones extraidas a partir del proceso de diseño e implementación de la aproximación del error de las subdivisiones. Durante este proceso se ha sopesado el uso de distintos métodos para su cálculo, que favoreciera en gran medida que las subdivisiones se realizarán correctamente. Para ello se han realizado distintas pruebas, utilizando distintas fórmulas para cálcular la distancia entre los valores que se obtienen con la aproximación por la Regla de Simpson y la Regla de Trapecio, así como el uso de distintos espacios de color, para poder cálcular la aproximación del error como se comenta en el Anexo B.3. Este proceso ha permitido determinar que el uso de espacios de color como HSV[6] o HSL[7] para la comparativa necesaria en las aproximaciones mencionadas, supone un beneficio gracias a la representación que este tipo de espacios de color ofrecen, en el que cada canal aporta más significado para distinguir diferencias entre colores.

[6] *Hue, Saturation, Value* o en español Matiz, Saturación y Valor
[7] *Hue, Saturation, Lightness* o en español Matiz, Saturación y Luminosidad

7.2. Trabajo futuro

Como se ha comentado, la aplicación del algoritmo para todas las subdivisiones requiere de gran cantidad de subdivisiones en zonas de altos contrastes. Se propone como trabajo futuro el diseño de un algoritmo híbrido entre el Método de Monte Carlo y el proceso de subdivisión de la imagen que se realiza en todas las dimensiones. Dado que este proceso ya obtiene espacio multidimensionales con errores estimados similares, sería de interes observar el comportamiento de una aplicación del Método de Monte Carlo en función de estos espacios obtenidos.

Así mismo, el diseño e implementación, tanto de otra aproximación del error (distintos espacios de color a los utilizados), sobre todo para el caso de subdivisión de todas las dimensiones, como de un distinto proceso de interpolación sería interesante como campo de estudio.

Por último, es importante mencionar que el algoritmo presentado no se encuentra implementado de manera altamente eficiente, por lo que una posible vía de trabajo futuro sería la mejora de eficiencia del mismo, así como de las estructuras de almacenamiento, siendo viable la implementación de nuevas con distinta gestión de los datos almacenados que la que presentan las actuales.

Referencias

[boo] Boost. `http://www.boost.org/`. 1.59.0.

[cla] Clang. `http://clang.llvm.org/`. 3.7.1.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[cma] Cmake. `https://cmake.org/`. 3.4.1.

[HJW+08] Toshiya Hachisuka, Wojciech Jarosz, Richard Peter Weistroffer, Kevin Dale, Greg Humphreys, Matthias Zwicker, and Henrik Wann Jensen. Multidimensional adaptive sampling and reconstruction for ray tracing. In *ACM Transactions on Graphics (TOG)*, volume 27, page 33. ACM, 2008.

[Kaj86] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[RJB04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004.

[Str13] Bjarne Stroustrup. *A Tour of C++*. 2013.

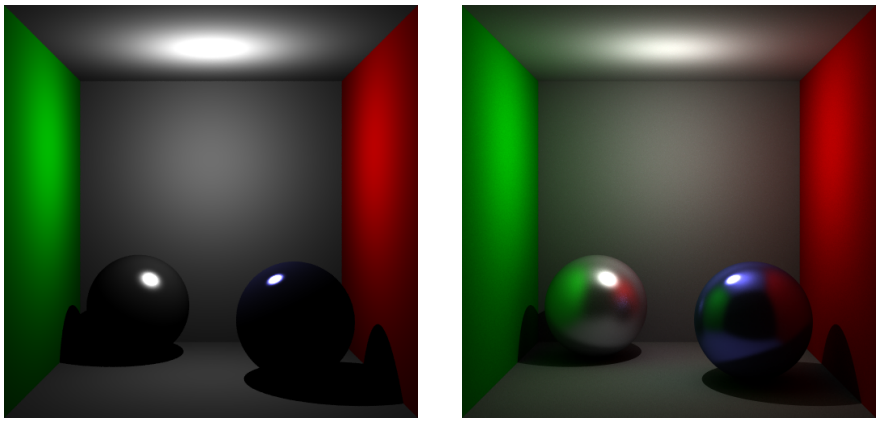
[Vea98] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[ZJL+15] Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. Recent advances in adaptive sampling and reconstruction for monte carlo rendering. *Computer Graphics Forum*, 34(2):667–681, 2015.

Anexos

A. Simulación de Iluminación Global

La simulación de iluminación global trata de modelar escenas tridimensionales utilizando, además de la iluminación que proviene directamente de las luces existentes en las escenas que se pretenden simular, los rayos de luz que provienen de esas fuentes de luz, pero que han sido reflejados en las diversas superficies de la escena, formando lo que se conoce como iluminación indirecta.



(a) Iluminación local (b) Iluminación global

Figura 7.1: Comparativa de la misma escena teniendo en cuenta únicamente iluminación directa, y computando iluminación global.

Los algoritmos que se usan para la simulación de gráficos 3D por computadora que consideran una simulación de iluminación global consiguen, por tanto, una simulación más completa que aquellos que tienen en cuenta únicamente la luz directa en sus cálculos.

Aún así, el computo de esta iluminación indirecta supone un gran coste computacional si lo comparamos con el coste que lleva simular únicamente escenas con iluminación directa.

Una clara comparación entre una escena sin iluminación global y otra con iluminación global se puede observar en la Figura 7.1.

A.1. La Ecuación de Render

Este proceso de simulación de iluminación se realiza utilizando lo que conocemos como la Ecuación de Render [Kaj86].

La ecuación de render es una ecuación integral que define la radiancia que parte desde un punto hacia un dirección como la suma de la radiancia que dicho

punto emite, además de la radiancia reflejada, bajo una aproximación geométrica (Figura 7.2).

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t)(\omega_i \cdot \mathbf{n}) d\omega_i \tag{7.1}$$

donde:

- $\mathbf{x}$ es la localización en el espacio.
- ω_o es la dirección de la luz que parte del punto $\mathbf{x}$.
- λ es una longitud de onda de la luz.
- t es el tiempo.
- $\mathbf{n}$ es la normal a la superficie en el punto $\mathbf{x}$.
- ω_i es la dirección de la luz que llega al punto $\mathbf{x}$.
- $L_o(\mathbf{x}, \omega_o, \lambda, t)$ es la Radiancia Espectral total de longitud de onda λ que se emite en la dirección ω_o en el tiempo t desde una posición $\mathbf{x}$.
- $L_e(\mathbf{x}, \omega_o, \lambda, t)$ es la radiancia emitida.
- Ω es la semiesfera centrada alrededor de $\mathbf{n}$ que contiene todos los valores posibles de ω_i.
- $\int_\Omega \cdots d\omega_i$ es la integral sobre Ω.
- $f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t)$ es la **BRDF** [8], la proporción de luz reflejada desde ω_i hacia ω_o en la posición $\mathbf{x}$, tiempo t, y en la longitud de onda λ.
- $L_i(\mathbf{x}, \omega_i, \lambda, t)$ es la Radiancia Espectral de longitud de onda λ que llega a $\mathbf{x}$ desde la dirección ω_i en el tiempo t.
- $\omega \cdot \mathbf{n}$ es el factor de debilitamiento del la radiancia entrante debido al ángulo de incidencia. Suele escribirse como $\cos\theta_i$.

Un ejemplo de su uso en un caso de iluminación local (como el presentado en la Figura 7.1a), con distintas fuentes de luz, la luz incidente se computaría como un sumatorio de la luz que aportarían las distintas k fuentes de luz presentes en la escena.

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \sum_{i=0}^{k} f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t)(\omega_i \cdot \mathbf{n}) \tag{7.2}$$

[8] Bidirectional Reflectance Distribution Function, o en español, Función de Distribución Bidireccional de la Reflectancia

En los casos de iluminación global, los cálculos se realizarían utilizando de manera recursiva la propia Ecuación de Render para obtener la luz indirecta que llegaría hasta un punto.

Cuando se utiliza la Ecuación de Render para la simulación de imágenes por ordenador, inicialmente los puntos de estudio serían los píxeles de la imagen, ya que sabiendo que luz emiten en dirección a la hipotética cámara de la escena, es posible asignar un color a cada pixel de la imagen que se obtendrá como resultado del proceso de simulación de iluminación global.

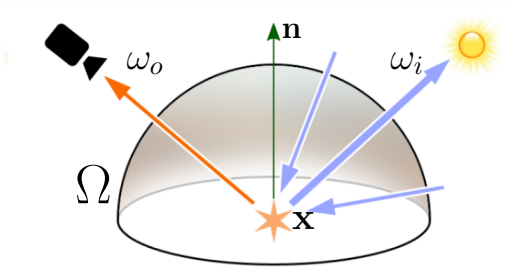


Figura 7.2: Fenómeno descrito por la ecuación de render. La ecuación de render define la luz emitida desde un punto **x** en una dirección dada.

Se sobrentiende entonces la importancia de esta ecuación para cualquier técnica de renderizado, y la importancia de encontrar aproximaciones lo más cercanas al autentico valor devuelto por la misma dentro del campo de la simulación de imágenes por ordenador.

Aun así, la ecuación de render no contempla fenómenos que, aunque complejos en términos físicos, son habituales en escenas cotidianas. Efectos como la fluorescencia, la fosforescencia, la interferencia de la luz o el fenómeno de scatering no se contemplan al usar la ecuación de render para simular escenas por computador. Esto se debe a que la ecuación de render simplifica el fenómeno de interacción de la luz al nivel de la óptica geométrica, mientras que los fenómenos comentados pertenecen a los campos de la óptica de ondas, la óptica electromagnética o la óptica cuántica.

A.2. Path Integral

Para utilizar la Ecuación de Render en el proceso de simulación de Iluminación Global es necesario, como ya se ha comentado, hacer un uso recursivo de la misma.

Esto puede conllevar, en caso de querer simular con exactitud el comportamiento de la luz en un escena, un computo elevado de cálculos recursivos, costosos en términos de cálculo para un computador.

Sin embargo, existe un formulación distinta para el fenómeno de propagación

de la luz, la llamada *Path Integral* [Vea98, Section 8].

La *Path Integral* presenta una formulación en forma de una única integral (Ecuación 7.3) frente a la recursividad que presenta la ecuación de render para el problema:

$$I_j = \int_{\Omega} f_j(\bar{x}) d\mu(\bar{x}) \tag{7.3}$$

donde Ω es el conjunto de todos los *transport paths* de cualquier longitud.

El término f_j representa la llamada *Measurement Contribution Equation*, que designa la contribución de cada *path* (*path contribution*).

Por último, μ representa el *Path Space*, el cual puede expresarse como una secuencia de números del intervalo $\mathbf{[0 \ldots 1]}$. Este hecho es importante, dado que el algoritmo presentado hace uso de este hecho para definir los intervalos de integración.

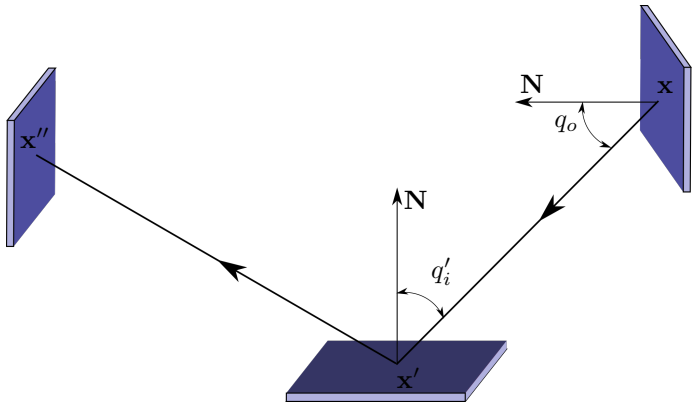


Figura 7.3: Geometría para el la ecuación de transporte de la luz en la *three-point form*

La *Measurement Contribution Equation*, comentada anteriormente, puede definirse mediante la siguiente expresión:

$$f_j(\bar{x}) = L_e(\mathbf{x}_0 \leftarrow \mathbf{x}_1) \prod_{i=1}^{k-1} f_s(\mathbf{x}_{i-1} \leftarrow \mathbf{x}_i \leftarrow \mathbf{x}_{i+1}) G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \cdot W_e^{(j)}(\mathbf{x}_{k-1} \leftarrow \mathbf{x}_k) \tag{7.4}$$

Esta ecuación se puede obtener extendiendo recursivamente la Ecuación de Render, partiendo desde la *Measurement Equation* que presenta [Vea98, Section 3] y aplicando un cambio de variable hasta que se obtenga la siguiente ecuación:

$$
\begin{aligned}
I_j = & \sum_{k=1}^{\infty} \int_{\mathcal{M}}^{k+1} L_e(\mathbf{x}_0 \leftarrow \mathbf{x}_1) \prod_{i=1}^{k-1} f_s(\mathbf{x}_{i-1} \leftarrow \mathbf{x}_i \leftarrow \mathbf{x}_{i+1}) G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \\
& \cdot W_e^{(j)}(\mathbf{x}_{k-1} \leftarrow \mathbf{x}_k) dA(\mathbf{x}_0) \cdots dA(\mathbf{x}_k) \qquad (7.5) \\
= & \int_{\mathcal{M}^2} L_e(\mathbf{x}_0 \leftarrow \mathbf{x}_1) G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) W_e^{(j)}(\mathbf{x}_{k-1} \leftarrow \mathbf{x}_k) dA(\mathbf{x}_0) dA(\mathbf{x}_1) \\
& + \int_{\mathcal{M}^3} L_e(\mathbf{x}_0 \leftarrow \mathbf{x}_1) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) f_s(\mathbf{x}_0 \leftarrow \mathbf{x}_1 \leftarrow \mathbf{x}_2) G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) \\
& \cdot W_e^{(j)}(\mathbf{x}_1 \leftarrow \mathbf{x}_2) dA(\mathbf{x}_1) dA(\mathbf{x}_2) \\
& + \cdots
\end{aligned}
$$

que comparando con (7.3) permite obtener la expresión de la *Measurement Contribution Function* vista en (7.4).

Respecto a los términos que aparecen en esta misma, como en la forma expandida vista en (7.5), G representa el cambio de variable mencionado, mientras que el término $W_e^{(j)}(\mathbf{x} \rightarrow \mathbf{x}')$ representa la importancia emitida desde $\mathbf{x}'$ hacia $\mathbf{x}$.

Un ejemplo del valor de $f_j(\bar{x})$ para un camino $\bar{x} = \mathbf{x}_0\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ sería, por tanto:

$$
\begin{aligned}
f_j(\bar{x}) = & L_e(\mathbf{x}_0 \rightarrow \mathbf{x}_1) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) f_s(\mathbf{x}_0 \rightarrow \mathbf{x}_1 \rightarrow \mathbf{x}_2) \\
& \cdot G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) f_s(\mathbf{x}_1 \rightarrow \mathbf{x}_2 \rightarrow \mathbf{x}_3) G(\mathbf{x}_2 \leftrightarrow \mathbf{x}_3) W_e^{(j)}(\mathbf{x}_2 \rightarrow \mathbf{x}_3) \qquad (7.6)
\end{aligned}
$$

como se puede observar en la Figura 7.4.

La integral que se presenta en esta aproximación ya no tiene el caracter recursivo que se encuentra en la ecuación de render, en la que se define la aportación de luz de parte de la ecuación como una aplicación recursiva de la misma.

Que esta aproximación presente una única integral es una ventaja. De esta manera se trata el problema de manera más completa y se permite, por ejemplo, la contrucción de caminos comenzando desde vértices intermedios, algo que con la ecuación de render no es posible, dado que el camino es construido con un lanzamiento de rayos recursivo desde el vértice inicial, la lente o cámara de la escena.

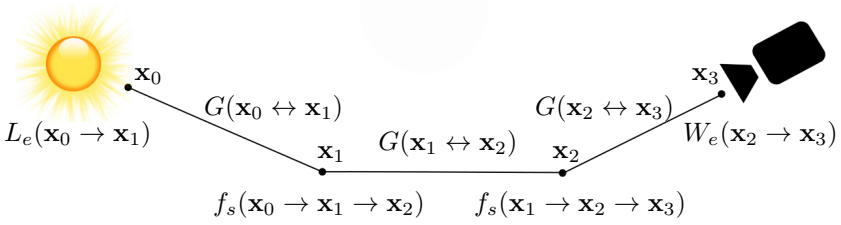


Figura 7.4: Ejemplo de un camino de longitud 3 según el concepto de *path integral*.

B. Integración numérica

En el ámbito del análisis numérico, se considera la integración numérica a los distintos algoritmos utilizados para determinar el valor numérico de una integral definida, de manera que habitualmente se utiliza por extensión este termino para referirse a la resolución de ecuaciones diferenciales.

Esta integral definida supone la suma de valores dentro del rango establecido con un diferencial entre los mismos, de manera que se puede expresar de la siguiente manera:

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(\xi_i)\Delta x_i$$

donde $\Delta x_i = x_i - x_{i-1}$,$x_{i-1} \leq \xi_i \leq x_i$.

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.

Debido al caracter integral del problema de renderizado mediante la Ecuación de Render o mediante la *Path Integral*, el cálculo integral juega un papel muy importante en la simulación de imágenes 3D por ordenador.

B.1. Técnicas de cuadratura

Estas integrales se resuelven normalmente en la práctica calculando una aproximación suficientemente precisa de la misma haciendo uso de las llamadas técnicas de cuadratura. Estos métodos utilizan de manera general una combinación de evaluaciones del integrando para obtener aproximaciones de la misma.

Las técnicas de cuadratura siguen una fórmula general para realizar las aproximaciones de las integrales definidas:

$$\int_a^b f(x) \approx (b-a) \sum_{i=1}^{n} \omega_i f(x_i) \tag{7.7}$$

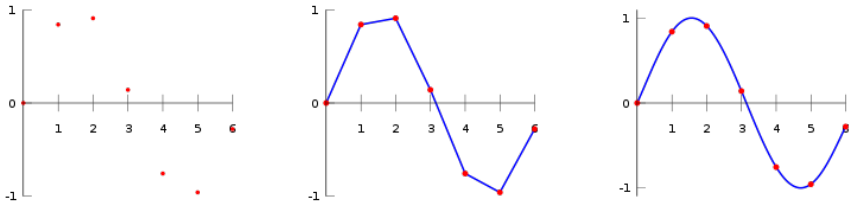
donde ω_i, x_i y n son los párametros que varian entre cada algoritmo de aproximación.

De entre ellos, n indica el número de evaluaciones que se ha hecho del integrando, por lo que normalmente es deseable utilizar métodos que permitan cálculos precisos con pocas evaluaciones, dado que así se reducirá el tiempo de cálculo que implica el realizarlas.

Las distintas técnicas existentes se pueden agrupar en aquellas que están basadas en funciones de interpolación y en métodos adaptativos.

Aquellas que utilizan funciones de iterpolación se basan en utilizar una función de interpolación, fácilmente integrable, de la función original de la que se desea calcular la integral.

La mencionada interpolación es la técnica del análisis numérico que permite la construcción de nuevos puntos partiendo de un conjunto conocido de puntos. Esto permite en muchas ciencias que a partir de un conjunto de muestras tomadas experimentalmente, generar una función que se ajuste a la distribución de los mismos.



(a) Puntos a interpolar (b) Interpolación linear (c) Interpolación polinómica

Figura 7.5: Ejemplo de distintas funciones de interpolación posibles aplicadas a un conjunto de datos.

El problema ligado estrechamente con este es la aproximación de funciones complejas por funciones más sencillas. Este es el problema que compete en las técnicas de cuadratura, puesto que interesa simplificar la función de estudio por otra más sencilla cuya integral sea facilmente calculable.

Por tanto, se hace uso de la interpolación para conseguir esa función simplificada utilizando datos muestreados, de manera que se intente ajustar de estructura general conocida.

Normalmente, en el caso de las técnicas de cuadratura, estas funciones de interpolación utilizadas son polinomios, y puesto que los polinomios de grado elevado tienden a tener unas variaciones extremas, en la práctica se hace uso de polinomios de menor grado, normalmente lineares o cuadráticos.

De esta manera, el método más sencillo sería utilizar un polinomio de grado 0, es decir una función constante, que pasara en este caso por el punto medio del intervalo de la función a interpolar, $((a+b)/2, f((a+b)/2))$. Este método es conocido como regla del punto medio o Regla del Rectángulo (Figura 7.6a).

$$\int_a^b f(x)dx \approx (b-a)f\Big(\frac{a+b}{2}\Big) \tag{7.8}$$

La siguiente aproximación posible sería utilizar un polinomio de grado 1 que pasara por los puntos $(a, f(a))$ y $(b, f(b))$. Esta aproximación se conoce como Regla del Trapecio (Figura 7.6b).

$$\int_a^b f(x)dx \approx (b-a)\frac{f(a)+f(b)}{2} \tag{7.9}$$

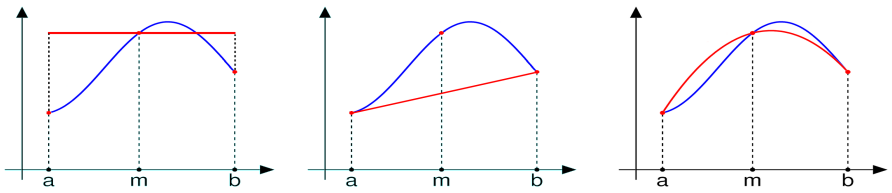
Otra aproximación posible sería la conocida como Regla de Simpson (Figura 7.6c), que utiliza como base un polinomio de grado 2.

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\Big[f(a)+4f\Big(\frac{a+b}{2}\Big)+f(b)\Big] \tag{7.10}$$

Para cada una de las reglas expuestas se puede conseguir una aproximación más precisa si el intervalo de estudio $[a, b]$ se subdivide en n subintervalos, calculando la aproximación para cada uno y sumando los resultados. Esto se conoce como Regla Compuesta. Un ejemplo de regla compuesta sería, en el caso de la Regla del Trapecio sería la regla compuesta del trapecio,

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\left(\frac{f(a)}{2}+\sum_{k=1}^{n-1}\left(f\left(a+k\frac{b-a}{n}\right)\right)+\frac{f(b)}{2}\right) \tag{7.11}$$

donde, si $h = (b-a)/n$ y $K = 0, 1, 2, ..., n-1$, los distintos intervalos tendrían la forma $[Kh, (K+1)h]$.



(a) Regla del Rectángulo (b) Regla del Trapecio (c) Regla de Simpson

Figura 7.6: Comparativa de las distintas reglas de cuadratura (rojo) para una misma función (azul).

Otra técnica de cuadratura distinta a las mencionadas es el uso del Método de Monte Carlo . El Método de Monte Carlo es una técnica matemática que hace uso de números aleatorios para la resolución de problemas.

En este caso, la integración por el Método de Monte Carlo es la técnica de integración numérica que hace uso de muestras aleatorias para resolver integrales definidas.

De tal manera el Método de Monte Carlo aproximaría una integral I haciendo uso de muestras aleatorias.

$$I = \int_a^b f(x)dx \approx \frac{(b-a)}{N} \sum_{i=1}^{N} f(x_i) \tag{7.12}$$

Esta técnica se menciona con más detalle en el siguiente apartado, dada su utilidad, precisamente, en el cálculo de integrales multidimensionales.

B.2. Cuadratura multidimensional

En la anterior subsección se han comentado distintas técnicas de cuadratura, pero estas son aplicables a integrales de una dimensión.

Para resolver integrales de múltiples dimensiones, la primera aproximación posible sería expresar la integral multidimensional que se desea calcular como una repetición de integrales unidimensionales, utilizando el Teorema de Fubini.

$$\int_X \left(\int_Y f(x,y)dy \right) dx = \int_Y \left(\int_X f(x,y)dx \right) dy = \int_{X \times Y} f(x,y)d(x,y) \tag{7.13}$$

El uso de este concepto será también importante para el algoritmo como se explicará en sucesivas secciones para poder realizar cálculos de los valores de la integral sobre una de las dimensiones del problema.

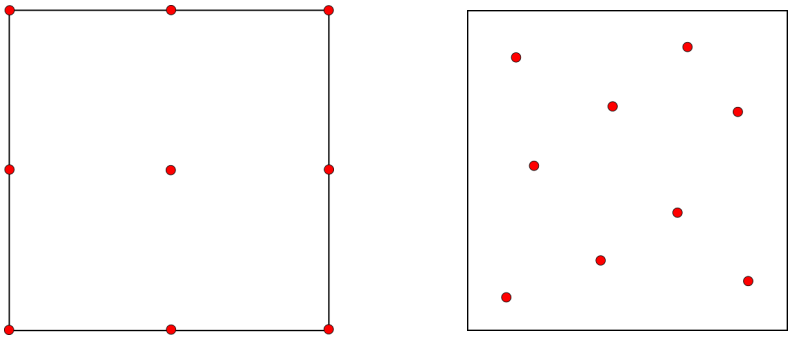
$$\begin{aligned} \int_{a_x}^{b_x} \int_{a_y}^{b_y} f(x,y)dydx &= \int_{a_y}^{b_y} \int_{a_x}^{b_x} f(x,y)dxdy \\ &\approx (b_x - a_x) \frac{(b_y - a_y)\left(\frac{f(a_x,a_y)+f(a_x,b_y)}{2} + \frac{f(b_x,a_y)+f(b_x,b_y)}{2} \right)}{2} \\ &\approx (b_y - a_y) \frac{(b_x - a_x)\left(\frac{f(a_x,a_y)+f(b_x,a_y)}{2} + \frac{f(a_x,b_y)+f(b_x,b_y)}{2} \right)}{2} \end{aligned} \tag{7.14}$$

Esta evaluación de múltiples integrales para la resolución de la integral múltiple tendría el problema de que el número de evaluaciones de la función que habría que realizar crecería de manera exponencial según el número de dimensiones se incrementara. Este problema es conocido como la maldición de la dimensionalidad, y existen varios métodos que intentan evitar este problema, de entre los cuales el más conocido y utilizado es el mencionado anteriormente Método de Monte Carlo (7.12)

El Método de Monte Carlo es un método ampliamente utilizado para la resolución de problemas de cuadratura multidimensional, debido a que la complejidad de sus aproximaciones no escala respecto al número de dimensiones contempladas en la integral definida como si que hacen otras reglas de cuadratura. Así mismo, permite una fácil parametrización del número de muestras

utilizadas en la aproximación.

Sin embargo, la propia aleatoriedad de las muestras que se usan en este método le confiere de un defecto que otras reglas de cuadratura presentan en menor medida. Las aproximaciones que se realizan utilizando este método poseen un ruido[9] causado por esta aleatoriedad, que aunque puede ser disminuido mediante distintas técnicas, sigue siendo una característica poco deseable para una regla de cuadratura.



(a) Método de Simpson

(b) Método de Monte Carlo

Figura 7.7: Comparativa de la distribución de muestras necesarias para el cálculo de una integral de 2 dimensiones con 2 métodos distintos

B.3. Cuadratura adaptativa

Las técnicas de cuadratura adaptativas se basan en las técnicas de cuadratura mencionadas previamente, utilizando subintervalos del dominio de integración de estudio escogidos adaptativamente.

Esto quiere decir que el número de muestras que estas técnicas utilizan se ajusta para conseguir la precisión necesaria. De esta manera, cada subintervalo es cálculado con un número de muestras acorde a la complejidad del mismo.

Para considerar los algoritmos adaptativos es necesario considerar una estimación del error que se comete al realizar la aproximación de la integral, que podría ser a priori o posteriori, aunque normalmente es usado a posteriori, pues es calculado tras haber calculado la aproximación de la integral. Esta estimación del error se calcula normalmente como la diferencia de dos aproximaciones distintas de la integral, de manera que se considera el error entre ambas como el error estimado de la aproximación que se esta realizando para la integral (Algoritmo 2).

[9] En el ambito matemático y físico se denomina ruido a aquellos datos o información no correcta que se encuentra entre aquella correcta provocando datos erróneos o incorrectos. En el caso de la integración hace a datos incorrectos como resultado de la integración.

Procedure `integrate(f, a, b, tau)`
  $Q \approx$ aproximation1(f, a, b);
  $\epsilon \approx |Q -$aproximation2(f, a, b) | ;
  si ($\epsilon >$tau) **entonces**
    m = (a+b)/2 ;
    Q = `integrate(f, a, m, tau)` + `integrate(f, m, b, tau)`
  fin
  devolver Q ;

Algoritmo 2: Esquema general del algoritmo adaptativo para aproximación de integrales

El algoritmo adaptativo realiza, por tanto, una aproximación de la integral a calcular en el intervalo de estudio, así como una estimación del error cometido en dicha aproximación.

Si el error calculado en la aproximación es mayor que un valor de tolerancia escogido τ, el intervalo de estudio se subdivide, y se aplica el propio algoritmo sobre cada uno de los intervalos que se obtienen.

De esta manera la aproximación que se obtiene de la integral es, o bien la aproximación que se calculo inicialmente, o la suma del calculo recursivo de los subintervalos obtenidos.

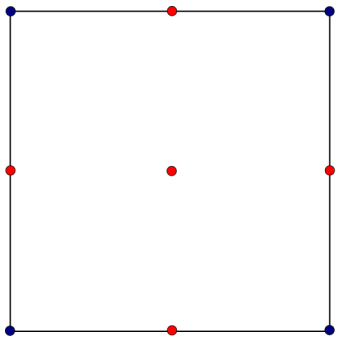


Figura 7.8: Las muestras necesarias para los cálculos que conlleva la Regla de Simpson (aquí en 2D) incluyen las muestras que necesita una regla de menor grado como la Regla del Trapecio (muestras en azul).

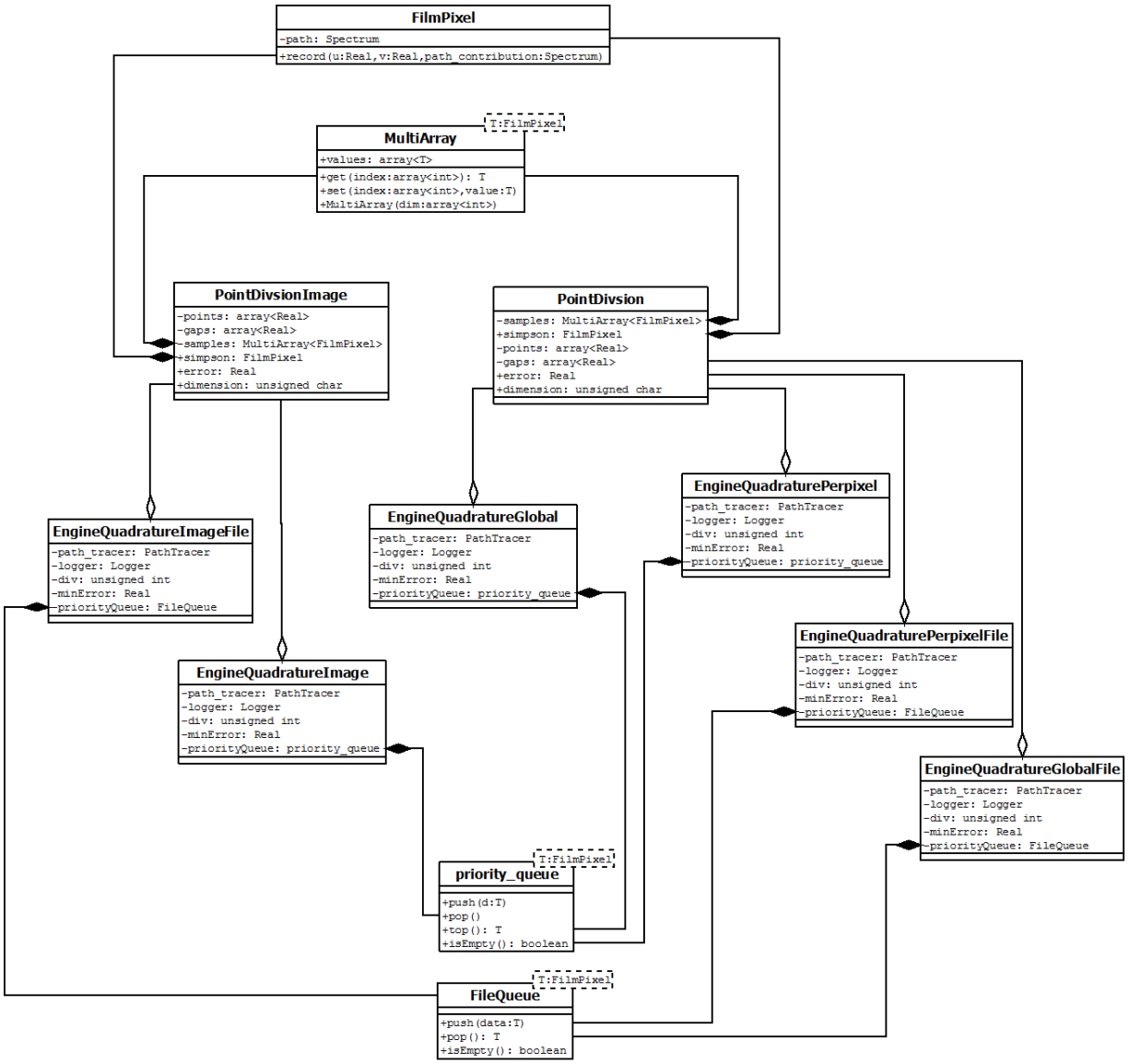
En la práctica, es importante observar un hecho importante si queremos ahorrar cálculos a la hora de utilizar los métodos adaptativos.

Si los cálculos para la aproximación de la integral se utiliza una regla de cuadratura determinada, supongamos que una regla de cuadratura de grado N, podría ahorrarse utilizar muestras adicionales si la regla de cuadratura que se utiliza para, junto a la primera, aproximar el error cometido es de grado $N-1$.

$$\epsilon = reglaGradoMayor(f, a, b) - reglaGradoMenor(f, a, b) \tag{7.15}$$

De esta manera para el cálculo del error no se realizará ningún cálculo adicional al ya realizado para únicamente tener una aproximación de la integral. Esto es conocido como técnicas anidadas, y se utilizará tal y como se explica en secciones posteriores para ahorrar cálculo de muestras y por tanto tiempo de computación innecesario.

C. Diagrama de clases



Se incluye a continuación una breve descripción de las clases expuestas en este diagrama de clases:

- **FilmPixel**: Clase encargada de almacenar la información de color asociada a un único pixel.
- **MultiArray**: Clase que permite definir *arrays* multidimensionales.
- **PointDivision**: Encargada de almacenar los datos de las subdivisiones en el algoritmo de subdivisión por pixel.
- **PointDivision**: Encargada de almacenar los datos de las subdivisiones en el algoritmo de subdivisión de la imagen.
- **EngineQuadratureGlobal**: Clase que implementa el algoritmo de subdivisión por pixel, con uso de montículo global.
- **EngineQuadraturePerpixel**: Clase que implementa el algoritmo de subdivisión por pixel, con uso de montículo por pixel.
- **EngineQuadratureImage**: Clase que implementa el algoritmo de subdivisión de la imagen.
- **EngineQuadratureGlobalFile**: Versión de la clase EngineQuadratureGlobal, implementada haciendo uso de la estructura alternativa para la cola de prioridad.
- **EngineQuadraturePerpixelFile**: Versión de la clase EngineQuadraturePerpixel, implementada haciendo uso de la estructura alternativa para la cola de prioridad.
- **EngineQuadratureImage**: Versión de la clase EngineQuadratureImage, implementada haciendo uso de la estructura alternativa para la cola de prioridad.
- **priority_queue**: Clase perteneciente a las librerias STL de C++ que implementa la cola de prioridad por montículo.
- **FileQueue**: Clase propia que implementa la cola de prioridad haciendo uso de la estructura alternativa explicada en la sección 4.5.2.