Paralelización del algoritmo Monte Carlo Ray Tracing para comunicaciones ópticas submarinas

Víctor Guerra Yánez

Tutores:
Dr. Eduardo Rodríguez Barrera
Dr. Domingo Benítez Díaz

Julio 2012

Índice

1. Introducción **5**
- 1.1. Comunicaciones ópticas submarinas no guiadas . . . 5
- 1.2. Simulación de respuesta al impulso en canales ópticos . . . 6
- 1.3. Paralelización con GPU . . . 7

2. Objetivos **9**

3. Arquitectura del computador paralelo **10**
- 3.1. Prestaciones de la GPU . . . 10

4. Tecnologías de paralelización **11**
- 4.1. OpenMP . . . 11
  - 4.1.1. Programación . . . 11
- 4.2. CUDA . . . 12
  - 4.2.1. Programación . . . 13

5. Modelo matemático **16**
- 5.1. Fenómenos relevantes en UWOC . . . 16
  - 5.1.1. Absorción del agua . . . 16
  - 5.1.2. Índice de refracción . . . 17
  - 5.1.3. Interacción con partículas grandes . . . 18
  - 5.1.4. Scattering de Mie . . . 18
  - 5.1.5. Modelo de colisión con partículas . . . 20
  - 5.1.6. Modelo de la superficie . . . 20
  - 5.1.7. Modelo del fondo . . . 22
- 5.2. Modelo del emisor . . . 23
- 5.3. Aportaciones de potencia de cada rayo . . . 24

6. Diseño del código C **25**
- 6.1. Topología del problema . . . 25
- 6.2. Estructuras de datos . . . 26
- 6.3. Generación de los rayos aleatorios . . . 29
- 6.4. Generación de la normal aleatoria a la superficie . . . 29
- 6.5. Descripción del algoritmo . . . 29
- 6.6. Comentario sobre la resolución de los datos . . . 30
- 6.7. Comentario sobre el número máximo de saltos . . . 31

7. Implementación **32**
- 7.1. Código secuencial . . . 32
  - 7.1.1. Cálculo de la base asociada al rayo . . . 33
  - 7.1.2. Determinación de la distancia recorrida y del tipo de colisión . . . 34
  - 7.1.3. Actualización del estado del rayo tras un evento . . . 35
  - 7.1.4. Colisión con la superficie . . . 36
  - 7.1.5. Colisión con el fondo . . . 36
  - 7.1.6. Colisión con partícula pequeña . . . 37
  - 7.1.7. Llegada al receptor . . . 38
  - 7.1.8. Particularidades de la versión recursiva . . . 38
  - 7.1.9. Particularidades de la versión iterativa . . . 40

7.2. Código OpenMP 41
7.3. Código para GPU 42
7.3.1. Generación de números aleatorios 43

8. Resultados **44**
8.1. Resultados del simulador de canal 44
8.2. Resultados de la paralelización 54
8.2.1. OpenMP 54
8.2.2. CUDA 57

9. Conclusiones **63**

10. Líneas futuras **64**

11. Referencias **65**

1. Introducción

En este capítulo se realiza un breve análisis del estado del arte sobre aquellos aspectos que influyen directamente en el objeto de este trabajo. Se abordarán los temas de comunicaciones ópticas no guiadas en el medio submarino, los algoritmos de trazado de rayos para cálculo de respuesta al impulso y la paralelización utilizando arquitecturas multinúcleo.

1.1. Comunicaciones ópticas submarinas no guiadas

Las comunicaciones ópticas submarinas no guiadas (UWOC - Underwater Wireless Optical Communication) son un campo novedoso de aplicación de las comunicaciones en luz visible. Tradicionalmente, las comunicaciones en este entorno han estado basadas en emisiones electromagnéticas de LF (Low Frequency) o ultrasonidos. En el caso de las emisiones de radiofrecuencia, la propagación submarina es prácticamente imposible en todas las bandas, salvo para frecuencias extremadamente bajas (que llevan asociadas una longitud de onda extremadamente larga). Para estas frecuencias LF el agua presenta un comportamiento menos hostil, pero conlleva velocidades de transmisión muy bajas ya que la capacidad del canal es muy reducida, además de precisar unas antenas de longitudes que resultan impracticables en muchas aplicaciones. En lo referente a ultrasonidos, las ondas de presión tienen alcances kilométricos, con lo que este tipo de tecnología permite enlaces de baja velocidad a distancias muy elevadas. Sin embargo, un inconveniente en ciertas aplicaciones es que las potencias acústicas transmitidas tienen efectos potencialmente nocivos para la salud de la fauna marina.

Las UWOC son una buena alternativa a las dos tecnologías anteriores en escenarios donde no se precisen distancias muy largas pero donde sean necesarias tasas de transmisión elevadas, como por ejemplo las comunicaciones en redes de sensores submarinas (UWSN - Underwater Wireless Sensor Networks) y las comunicaciones con UAVs (Unmanned Automatic Vehicles) [1]. Desde el punto de vista del canal submarino, se ha demostrado que la ventana donde la propagación de la luz presenta mejores características es la del verde-azul (450 nm – 500 nm), debido a los efectos más reducidos de absorción y dispersión espacial del agua [2]. La principal característica del canal submarino en lo que respecta a su respuesta en longitud de onda es su bajo tiempo de coherencia, es decir, los intervalos donde el canal puede ser considerado cuasi-estacionario son muy reducidos. Esto hace que las aproximaciones de canal lineal y temporalmente invariante no sean válidas a priori, como se demuestra en [3]. En lo referente a las señales que llegan al receptor, los caminos ópticos que siguen los rayos emitidos por los emisores, ya sean provenientes de fuentes láser o LED, dependen de la variación del índice de refracción dentro del medio, asi como por la colisión con diferentes partículas que produzcan dispersión espacial. Ya que el índice de refracción del agua depende de la salinidad, la temperatura, la longitud de onda transmitida y la presión [4], podría ocurrir que el apuntamiento entre fuente y receptor se dificulte dada la imprevisibilidad de estos parámetros. Existe otro fenómeno que se produce en las capas más cercanas a la superficie y que afecta al índice de refracción, las turbulencias. Como se demostró en [5], las células turbulentas generadas en la atmósfera afectan a este parámetro y posiblemente los modelos se puedan extrapolar al medio de nuestro interés.

La naturaleza aleatoria de los fenómenos ópticos que se producen en el agua, convierten a este medio en un perfecto *testbench* donde probar diferentes modulaciones y codificaciones.

Figura 1: Modem AquaOptical II desarrollado en el MIT

Se han llevado a cabo múltiples estudios acerca del comportamiento de diferentes técnicas de modulación y codificación en el entorno submarino, como en [6] [7] y [8]. Sin embargo, las modulaciones multiportadora como la FFT-OFDM o la WPT-OFDM [9] no han sido aún probadas aplicadas a UWOC, lo que abre una interesante línea de análisis. La fabricación de prototipos de comunicaciones es otro reto, ya que deben tratarse aspectos como la estanqueidad, el modelado de lentes y el *fauling* óptico (proliferación de algas en las superficies de los emisores y receptores que afecta a las comunicaciones); y se está llevando a cabo en múltiples grupos de investigación. Un ejemplo es el módulo AquaOptical II del MIT (figura 1). Existen publicaciones sobre implementaciones en aplicaciones militares de sistemas capaces de transmitir a tasas de 5 Mbps [10] utilizando fuentes LED y otras como los enlaces ópticos basados en láseres azules y verdes que la US Navy asegura haber desarrollado de manera conjunta a SA Photonics [11].

1.2. Simulación de respuesta al impulso en canales ópticos

En el ámbito de las comunicaciones ópticas en espacio libre (FSO - Free Space Optics), la simulación de la respuesta impulsiva de los canales en interiores ha sido una línea de investigación muy importante desde la aparición de los primeros modelos en infrarrojo (IR - InfraRed). Barry y Khan [12], propusieron el primer algoritmo de cálculo de respuesta al impulso para canales *indoor* utilizando un modelo de trazado de rayos (Ray Tracing). Este tipo de algoritmo posee un coste computacional muy elevado, por lo que posteriormente surgieron en un lapso de tiempo relativamente corto diferentes estrategias de simulación basadas en el método de Monte Carlo [13], que reducían notablemente el tiempo de cálculo, despreciando aquellos rayos que no aportan una energía significativa en el receptor óptico.

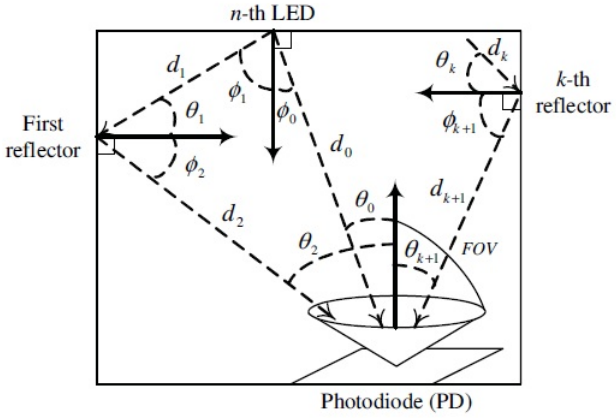

Figura 2: Geometría típica en canales *indoor*

Para UWOC todavía no existen estudios de simulación sobre el comportamiento de la respuesta impulsiva del canal, ya que se ha considerado siempre que el tiempo de coherencia (tiempo en el cual la respuesta del canal no varía) es muy pequeño y que la aproximación LTI (Linear and Temporary Invariant) no es aplicable al canal. Sin embargo, en este trabajo se implementa un simulador de canal que propone un algoritmo de cálculo de la respuesta al impulso teniendo en cuenta los efectos más significativos en la propagación de la luz en el agua. Ya que el tiempo de coherencia del canal es muy bajo y se produce gran cantidad de desvanecimientos por segundo [3], los resultados del simulador pueden ser aplicados al cálculo probabilístico de cobertura y ancho de banda disponible en función de una serie de parámetros del agua como la turbidez (concentración de partículas en suspensión), la agitación de la superficie del mar o la longitud de onda de trabajo empleada.

El número de rayos necesario para que el error cuadrático medio (MSE - Minimum Square Error) de la solución obtenida por un simulador de respuesta impulsiva sea más bajo que un umbral, es una incógnita que no se conoce a priori, ya que depende de la geometría del problema y de los parámetros que se introduzcan. Sin embargo, este error esta acotado según el número de rayos como se demuestra en [14]. Este número de rayos puede ser tan elevado que implique un coste computacional impracticable, pues con el modelo que se presenta en este trabajo se pretende obtener estadísticos (media y varianza por ejemplo) en función de las características del canal una vez se hayan realizado medidas empíricas, a fin de comparar el modelo con la realidad.

1.3. Paralelización con GPU

El escalado en frecuencia supone un problema en los procesadores, ya que las conexiones entre las diferentes partes del circuito integrado comienzan a comportarse como líneas de transmisión, produciendo reflexiones y por tanto, teniendo que adaptar cada línea a la impedancia de entrada de los transistores con el consiguiente aumento de área. Además, esta adaptación reduce en si misma el ancho de banda, y deberían emplearse técnicas de diseño UWB (Ultra Wide Band)[15], generando más restricciones de área y potencia aún. Este hecho ha centrado los esfuerzos de ingeniería en la generación de arquitecturas multiprocesador, como es el caso de [16] [17] [18] y de las GPU(Graphics Processing Unit) [19] [20], utilizadas tanto el gráficos como en procesado.

Las GPU fueron introducidas a finales de los noventa a modo de coprocesador para la aceleración de la simulación y visualización de imágenes 3D. Desde que surgieron estas arquitecturas

gráficas, este tipo de sistemas se han extendido de manera generalizada, formando parte de gran cantidad de equipos de sobremesa y siendo introducidos cada vez más en plataformas servidoras.

Las primeras arquitecturas estaban constituídas por unidades de procesado de imagen independientes (Shader, Vertex, Pixel, etcétera). Sin embargo, el modelo ha cambiado y dichas unidades han sido incorporadas en un único chip. Con el fin de utilizar la flexibilidad de los diseños, los fabricantes de GPU han desarrollado entornos de programación intuitivos, permitiendo que el usuarios tenga mayor control sobre la segmentación de la GPU y la jerarquía de memoria. Esta coyuntura ha conducido a que la potencia de cálculo de las GPU esté siendo explotada con fines científicos [21] [22] además de para visualización y renderizado de imágenes 3D.

La paralelización de los algoritmos de Ray Tracing se ha llevado a cabo en muchas ocasiones, utilizando arquitecturas multiprocesador, tanto en CPU como GPU [23] [14], con resultados muy positivos de aceleración y eficiencia, debida a la alta independencia entre rayos.

En este trabajo se propone un algoritmo de cálculo de respuesta al impulso del canal submarino y la paralelización del mismo. Se utilizarán dos estrategias de parelelización: una basada en OpenMP [24] (arquitectura CPU) y otra en CUDA [25], utilizando una tarjeta gráfica NVIDIA Tesla M2050, la cual dispone de 575 núcleos de procesamiento [26]; con el fin de realizar una comparativa donde se refleje la mejora de tiempo de cómputo, la aceleración y la eficiencia para cada caso.

2. Objetivos

Este Trabajo Fin de Máster (TFM), tiene como objetivo principal la implementación utilizando una arquitectura paralela de un simulador Monte Carlo Ray Tracing de respuesta al impulso para UWOC.

Los objetivos desde el punto de vista de la paralelización del algoritmo son los siguientes:

- Estudiar las diferentes arquitecturas de paralelización que existen.
- Elegir la arquitectura que mejor se ajuste al problema.
- Implementar el código paralelo utilizando la arquitectura elegida.
- Extraer resultados para diferentes parámetros de simulación y analizarlos.
- Que el tiempo de simulación sea lo más reducido posible.

A pesar de que el objetivo principal se centra en la paralelización del algoritmo, desde el punto de vista del modelo matemático que se propone, se pretende que el simulador sea capaz de modelar los siguientes fenómenos físicos producidos en el canal:

- Absorción del agua a cada longitud de onda.
- Velocidad de propagación a cada longitud de onda.
- Dispersión de Mie debida a partículas pequeñas (*plankton*, microorganismos, etcétera).
- Destrucción fotónica debida a la presencia de partículas grandes (arena, burbujas de aire, etcétera).
- Reflexión especular en la superficie del agua.
- Efecto difusivo del fondo marino.

Existen otros objetivos no principales que debe tratar de cumplir, estos son los siguientes:

- Que el entorno de simulación sea parametrizable (dimensiones, concentraciones de partículas, etcétera).
- Generar un código modular.

3. Arquitectura del computador paralelo

En esta sección se presentan las prestaciones de la máquina en la cual se han llevado a cabo los experimentos a la hora de obtener resultados.

Se ha empleado la estación de cálculo SIE LADÓN XEON 808 WESTMERE, cuyas características son las siguientes.

- 2 procesadores Xeon Westmere E5620 a 2.4 GHz de cuatro núcleos, caché de 12 Mb y 5.86 GT/s.
- 24 GB DDR3 a 1.33 GHz.
- Disco duro de 1 TB. 7200 rpm y 32 Mb de caché.
- 2 NVidia TESLA M2050 GPU.

En este trabajo se ha empleado únicamente una GPU, pero el problema es paralelizable de tal forma que podría emplearse cualquier número de unidades de procesamiento sin afectar al resultado final.

3.1. Prestaciones de la GPU

A continuación se presentan las características de procesamiento básicas del módulo GPU utilizado.

Nvidia TESLA M2050	
Procesadores	448
Velocidad	1150 MHz
Memoria	3072 MB
Ancho de banda	148.4 GB/s
Ancho de bus	384 bits
Capacidad de cómputo	2.0

Cuadro 1: Prestaciones de la Nvidia TESLA M2050

4. Tecnologías de paralelización

En esta sección se comentarán las principales características de las dos tecnologías de paralelización empleadas en este TFM: OpenMP y CUDA. Existe otro tipo de tecnologías como MPI (*Message Passing Interface*), orientadas a la arquitecturas multicomputador, pero ya que la estación de trabajo empleada no es un clúster de procesamiento, se ha optado por dos tecnologías basadas en un único *host*.

4.1. OpenMP

OpenMP usa un modelo de tipo *fork-join* para realizar ejecución el paralelo. Está pensado para aplicaciones basadas en arrays de grandes dimensiones. La intención de OpenMP es soportar código capaz de ejecutarse tanto en paralelo (múltiples hilos y soporte completo) como secuencialmente (directivas de OpenMP ignoradas). Sin embargo, es posible diseñar código exclusivamente para ejecución el paralelo.

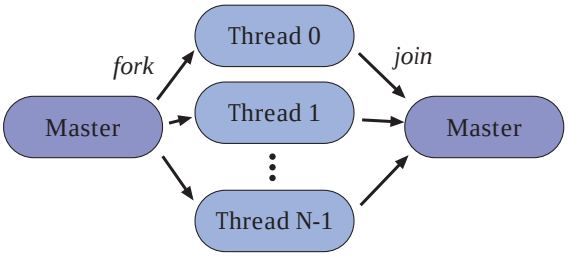

Figura 3: Paradigma fork-join en OpenMP

Un programa escrito en OpenMP comienza con la ejecución como un único hilo, llamado hilo principal. Este hilo es ejecutado de manera secuencial hasta la primera construcción en paralelo encontrada. Cuando se encuentra una sección de código en paralelo, el hilo princial crea un grupo de hilos que se ejecutan en paralelo, siendo el hilo principal el encargado de la gestión.

Cada hilo del grupo ejecuta las instrucciones de la región paralela, excepto para aquellas instrucciones en las que se utilicen directivas de control. Al final de las regiones de código paralelo existe una barrera implícita que controla que la ejecución del código no prosiga hasta que todos los hilos hayan finalizado.

Por defecto, todas las variables que se emplean dentro de una sección de código paralelo se comportan como variables compartidas, siendo accesibles y modificables por todos los hilos en ejecución. Esto provoca que si no se sincronizan los hilos, el estado final puede quedar indeterminado. Para solventar este problema, existen múltiples directivas de preprocesador con las que realizar sincronización o que definen variables como privadas (propias y no compartidas) para cada hilo.

4.1.1. Programación

La programación con OpenMP se basa en el uso de directivas. El formato de las directivas es el siguiente:

```
#pragma omp directiva [opciones]
```

Las principales directivas son las siguientes:

- **parallel**. Define una región paralela, la cual va a ser ejecutada por múltiples hilos. Es la directiva fundamantal que comienza la ejecución en paralelo.
- **for**. Esta directiva identifica una sección iterativa cooperativa, donde se especifica el número de iteraciones del bucle asociado que se ejecutará en paralelo. Las iteraciones son distribuciones según el número de hilos que vayan a ejecutarse.
- **critical**. Indica que la sección de código posterior es crítica, es decir, que los hilos deben sincronizarse para ejecutarla.

El número de hilos que deben ejecutar las secciones de código paralelo se indican con la función **omp_set_num_threads()**. El número de hilos puede ajustarse dinámicamente también, no de forma determinista, pero no es objeto de estudio en este trabajo.

Para los bucles for, las opciones posibles son las siguientes:

- **private(variables)**. Hace que las variables indicadas sean privadas para cada hilo.
- **firstprivate(variables)**. Indica que el estado inicial se la variable compartida es el que tenía justo antes del código paralelo.
- **lastprivate(variables)**. El valor de salida de la variable privada será el de la última iteración del código paralelo entendido en el sentido secuencial.
- **reduction(op:variables)**. Realiza la reducción de variables escalares empleando el operador indicado.
- **ordered**.
- **schedule(tipo,CHUNK_SIZE)**. Define de qué manera se organizan las iteraciones dentro del conjunto de hilos. Existen cuatro tipos: **static** (cada hilo ejecuta CHUNK_SIZE iteraciones antes de volver a buscar más iteraciones), **dynamic** (en este tipo, cada hilo no espera por la sincronización para obtener más iteraciones), **guided** (ejecuta las iteraciones de manera parecida a dynamic pero con requiere menor sincronización)) y **runtime** (se emplea cuando la planificación óptima depende de los datos de entrada).
- **nowait**. Hace que la barrera implícita al final de una directiva for sea ignorada.

La medición de tiempos se realiza utilizando la función **omp_get_wtime()**.

4.2. CUDA

Una GPU es un procesador diseñado para acelerar el renderizado de las imágenes que se muestran en los displays de los ordenadores. Los detalles de las arquitecturas de las GPUs varían considerablemente y en la literatura, los componentes se describen normalmente utilizando terminología heredada de la segmentación empleada en gráficos (*rasterizer*,*shader*, etcétera). Sin embargo, con el fin de simplificar las estructuras, las GPU puede suponerse que están confomadas por dos o más "multiprocesadores", cada uno compuesto por varios *cores* que a su vez contienen ALUs (*Arithmetic Logic Unit*).

Cada multiprocesador tiene una cantidad de memoria relativamente pequeña compartida entre los *cores* que la forman. Además, existe una memoria global que esta compartida por toda la GPU. Cuando un dato es transferido entre la CPU y la GPU, por defecto es almacenado o extraído de la memoria global.

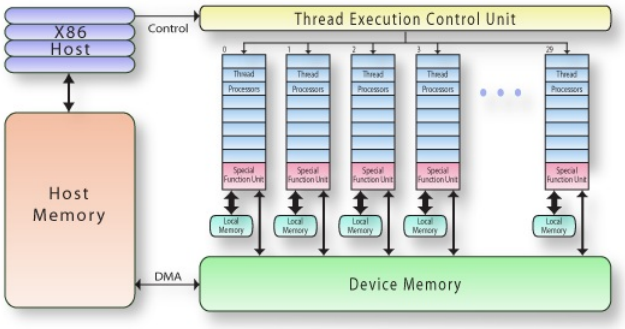

Figura 4: Arquitectura NVidia Tesla

La figura anterior muestra la arquitectura de las GPU Tesla. La memoria principal del host y la memoria principal de la GPU se comunican a través de DMA, controlando las transacciones de memoria en el host. La GPU está formada por grupos de procesadores que ejecutan las instrucciones al mismo tiempo. Cada grupo de procesadores comparte una memoria local y una unidad de cálculo para funciones trascendentes (Special Function Unit). Los hilos son gestionados por la unidad de control de la GPU (Thread Execution Unit)

La tecnología CUDA de NVidia lidera el mercado de la computación paralela basada en GPUs. La arquitectura CUDA implica una curva de aprendizaje relativamente corta y posteriormente, un acortamiento en el tiempo de desarrollo en aplicaciones paralelas basadas en GPU. Existen APIs para diferentes lenguajes como C/C++, Fortran, Java, Python y el Framework .NET de Microsoft.

La arquitectura del entorno CUDA está compuesta por varios componentes, los cuales se representan gráficamente en la figura 5.

CUDA permite programar en alto nivel utilizando sus librerías nativas. Para permitir la ejecución, se hace uso del *Runtime* de CUDA, el cual controla las llamadas al driver de la GPU que se tenga instalada. Este driver se comunicará a través del sistema operativo con los motores de computación paralela de la GPU.

4.2.1. Programación

En el programa principal, el *host* comienza la ejecución en la función *main*, declarando las variables (tanto en el lado *host* como en el lado *device*), reservando memoria e inicializándola. Las funciones del SDK de CUDA que se emplean para ello son las siguientes:

- **cudaMalloc()**. Reserva memoria en la GPU.
- **cudaMemcpy()**. Copia memoria en el sentido *host-device* o viceversa.

Tras la inicialización, la memoria está lista en la GPU para realizar el cómputo. La jerarquía existente en la arquitectura de la GPU, tiene su análogo a nivel de CUDA con los bloques de hilos (*blocks*) y los hilos (*threads*). Cada vez que se comienza un *kernel* de CUDA (código que se ejecuta en la GPU), es necesario especificar el número de bloques y el número de hilos por bloque.

Las llamadas a un kernel, definido mediante el atributo **__global__**, se especifican utilizando la notación de triple paréntesis agudo.

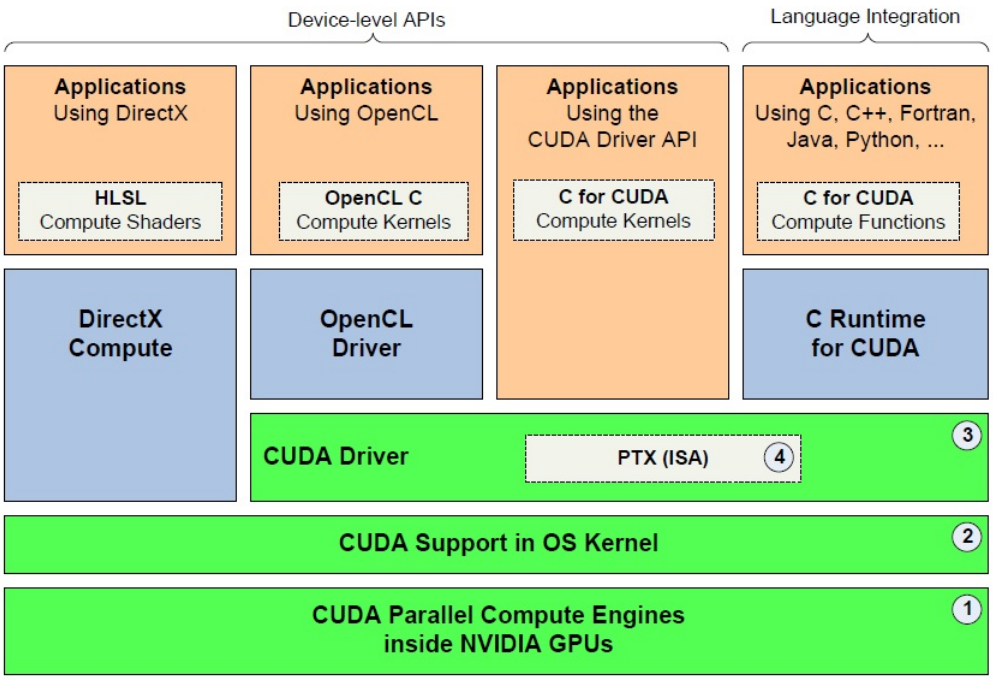

Figura 5: Arquitectura CUDA (fuente: www.nvidia.com)

```
kernel<<<Blocks,Threads>>>(argumentos)
```

Una función de kernel no puede hacer uso de funciones definidas en el espacio de *host*. Para poder hacer el código modular, se hace uso de la definición de funciones en la GPU mediante el atributo **__device__**. Además, con la arquitectura de GPU de la que se dispone, la recursividad no está soportada, por lo que las estructuras de tipo árbol deben ser recorridas de manera iterativa.

Los *cores* de una GPU NVidia ejecutan los hilos de manera SIMT (Single Instruction Multiple Thread), donde todos los núcleos del mismo grupo ejecutan la misma operación al mismo tiempo. Cuando existen saltos condicionales, algunos núcleos son desactivados, por lo que la eficiencia del código puede verse mermada.

SIMT es una forma de ejecución tipo SIMD (Single Instruction Multiple Data), donde se hace *fetch* de una instrucción para un grupo de threads que la ejecutan con su propio conjunto de datos [27]. Cuando existe un salto condicional o un conjunto de hilos está esperando debido a un acceso a memoria, mientras se resuelve su ejecución, los hilos son puestos en una cola y no se consideran para el *fetching* (son deshabilitados), lo cual implica un bloqueo en la ejecución.

Cuando se realiza la llamada a un kernel alojado en la GPU, el host no queda bloqueado, es decir, puede seguir ejecutando código en paralelo al procesador gráfico, de forma que únicamente quedará bloqueado en caso de hacer una petición de lectura de memoria de tipo **memcpy** de tipo *device-host*. Este hecho genera un problema a la hora de realizar mediciones de tiempo, ya que con instrucciones de procesador comunes como **clock()** no es posible medir los tiempos de cómputo en la GPU. Para ello, CUDA ofrece las siguientes funciones basadas en el tipo **cudaEvent_t**:

1. **cudaEventCreate()**. Crea un evento sobre el cual realizar medidas de tiempo.

2. **cudaEventRecord()**. Guarda una timestamp sobre el evento y un stream concreto.
3. **cudaEventSynchronize()**. Sincroniza un evento, es decir, espera a que termina la ejecución para guardar el timestamp.
4. **cudaEventElapsedTime()**. Calcula la diferencia de tiempos entre dos eventos.
5. **cudaEventDestroy()**. Destruye un evento creado anteriormente.

Gráficamente, el cálculo de tiempos se lleva a cabo de la siguiente manera:

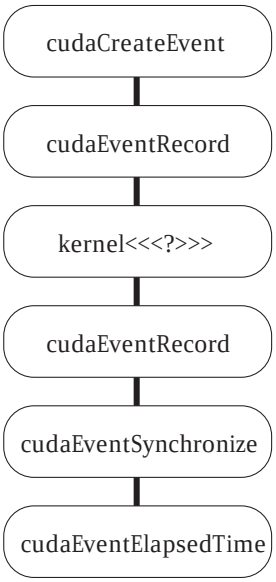

Figura 6: Medida de tiempos en CUDA

5. Modelo matemático

En esta sección se presenta el modelo matemático implementado en el simulador de canal. Este modelo ha sido desarrollado como un modelo teórico basado en los fenómenos que tienen lugar en el medio submarino. No es un modelo contrastado con datos empíricos, ya que todavía no han sido publicadas medidas referentes a este canal, pero es objeto de la tesis doctoral de la que este trabajo supone el primer resultado, realizar medidas que verifiquen o desmientan las simplificaciones y suposiciones que se han tenido en cuenta.

5.1. Fenómenos relevantes en UWOC

En UWOC, tienen lugar varios fenómenos físicos producidos por la interacción luz-materia que son de interés para el estudio de la propagación submarina de la luz. Existen efectos que influyen directamente sobre la potencia óptica de la señal, otros que afectan a la dirección de propagación y otros que afectan a su polarización. En este trabajo vamos a suponer fuentes de emisión tipo LED, por lo que la polarización no será tenida en cuenta. A continuación, se comentan estos fenómenos y se analiza su formulación matemática.

5.1.1. Absorción del agua

La interacción de la luz con las moléculas de agua y las sales disueltas produce dos efectos que se traducen en una pérdida de potencia óptica sobre una región del espacio. Se produce dispersión espacial y absorción de energía. Estos efectos dependen de la temperatura, de la salinidad del agua y de la longitud de onda de la radiación. En este trabajo se ha tenido en cuenta únicamente la dependencia de la absorción con la longitud de onda, ya que es el efecto más notable. Se ha empleado la curva de absorción obtenida por Wozniak y Dera [28].

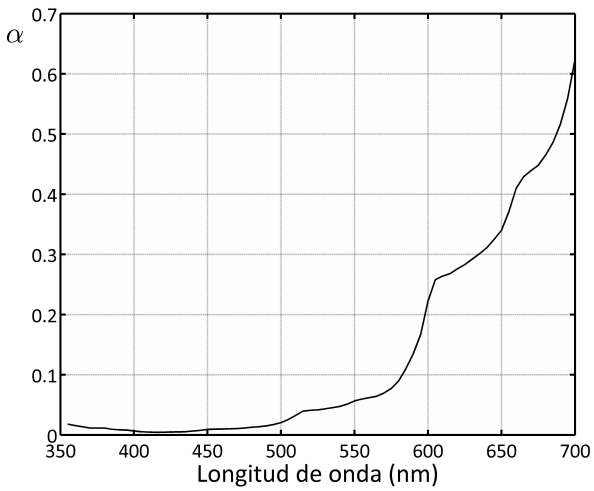

Figura 7: Curva de absorción obtenida por Wozniak y Dera

La curva de absorción define el parámetro de absorción α, el cual establece una relación de atenuación de potencia óptica con la distancia. Se define el factor de pérdidas por absorción de la siguiente manera.

$$L_{abs} = e^{-\alpha(\lambda)d} \tag{1}$$

Puede observarse que existe un mínimo de absorción en la región azul-verde del espectro visible. Además, las longitudes de onda infrarrojas y ultravioletas se ven atenuadas por factores muy elevados, de manera que la transmisión en estas ventanas queda desestimada a priori.

5.1.2. Índice de refracción

El índice de refracción es la relación entre la velocidad de propagación de la luz en el medio respecto de la velocidad de la luz en el vacío. Hace alusión a la oposición que ejerce el medio a la propagación de la energía. El índice de refracción del mar ha sido estudiado en profundidad, y varía con la longitud de onda, la salinidad y la temperatura. Aunque no puede definirse analíticamente, existen expresiones empíricas como las obtenidas por Matthäus [29] (expresión 2) y McNeil [30] (expresión 3).

$$\begin{aligned} n(\lambda, S, T) \quad = \quad & 1.447824 + 3.011 \cdot 10^{-4} S - 1.8029 \cdot 10^{-5} T - 1.6916 \cdot 10^{-6} T^2 - \\ & -0.48904\lambda + 0.728364 \cdot \lambda^2 - 0.383745\lambda^3 - S(7.9362 \cdot 10^{-7} T - \\ & -8.0597 \cdot 10^{-9} T^2 + 4.249 \cdot 10^{-4}\lambda - 5.847 \cdot 10^{-4}\lambda^2 + 2.812 \cdot 10^{-4}\lambda^3) \end{aligned} \tag{2}$$

$$n(\lambda, S, T) = 1.3247 - 2.5 \cdot 10^{-6} T^2 + S(2 \cdot 10^{-4} - 8 \cdot 10^{-7} T) + \frac{3300}{\lambda^2} - \frac{3.2 \cdot 10^7}{\lambda^4} \tag{3}$$

La velocidad de propagación de la luz para unas condiciones del medio determinadas y una longitud de onda de propagación, se define de la siguiente manera.

$$v = \frac{c_0}{n(\lambda, S, T)} \tag{4}$$

Donde:

- c_0 es la velocidad de propagación de la luz en el vacío ($3 \cdot 10^8 m/s$)
- λ es la longitud de onda de propagación.
- S es la salinidad del agua, normalmente expresada en tanto por uno.
- T es la temperatura del agua.

El índice de refracción del agua influye en varios aspectos en los sistemas UWOC. Ya que varía con la longitud de onda, produce dispersión cromática, esto es, la señal que llega al receptor óptico tiene un retardo diferente para cada longitud de onda (expresión 5). Este efecto sería más notable si las fuentes de emisión utilizadas fuesen de tipo WLED o RGB. Además, ya que los emisores y receptores ópticos deben tener carcasas transparentes que los aislen del contacto directo con el agua, los patrones de radiación y de visión de estos elementos pueden quedar ligeramente modificados por las relaciones entre índices de refracción en los interfases, produciéndose el fenómeno de la reflexión total interna y no transmitiéndose el total de energía radiada al medio de propagación. Por simplicidad, este último efecto no se ha tenido en cuenta.

$$\tau(\lambda) = \frac{d}{v} = \frac{n(\lambda, S, T) \cdot d}{c_0} \tag{5}$$

5.1.3. Interacción con partículas grandes

La radiación electromagnética interacciona con la materia. Uno de los efectos que se producen cuando existen partículas de dimensiones mucho mayores que la longitud de onda suspendidas en el agua, es la absorción de energía. Ya que en este modelo matemático se va a utilizar la aproximación de rayo para explicar el comportamiento de la luz sobre un ángulo sólido infinitesimal, puede suponerse que si una partícula de grandes dimensiones se interpone en la dirección de propagación de un rayo, la potencia óptica transmitida es nula, perdiéndose el rayo.

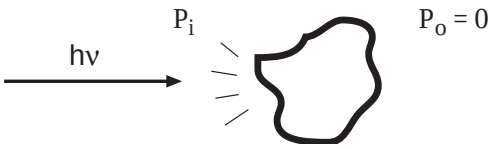

Figura 8: Absorción por partículas grandes

El hecho de que se suponga que las partículas absorben la energía del rayo, es porque dichas partículas son consideradas opacas a las longitudes de onda de trabajo.

En un medio con partículas grandes suspendidas y distribuídas uniformemente, la probabilidad de colisión de un rayo a una cierta distancia d sigue una distribución exponencial, de manera que la probabilidad de colisión con una partícula, sabiendo que es una partícula grande es:

$$p_{big} = 1 - e^{-\mu_b \cdot d} \tag{6}$$

Donde μ_b es el producto entre la concentración de partículas ρ_b y la sección de las partículas σ_b. A priori, la probabilidad de que un rayo impacte con una partícula grande será mayor con la distancia. Además, esta distribución define que la mayor parte de las colisiones con partículas grandes se producen más cerca del origen de la emisión.

5.1.4. Scattering de Mie

La teoria de Mie [31] define el comportamiento de la interacción de la radiación electromagnética con particulas esféricas. Esta teoria define los patrones de scattering que se producen debido a las múltiples refracciones y reflexiones que tienen lugar dentro y en el entorno de la partícula. Existen simplificaciones de esta teoría, cuya solución implica series infinitas, que son aplicables cuando la dimensión característica (expresión 7) de una partícula es muy inferior a la unidad, como la teoria de Rayleigh.

$$x = \frac{2\pi r \cdot n(\lambda)}{\lambda} \tag{7}$$

Ya que en el mar existen concentraciones de partículas micrométricas como algas, plankton y burbujas de aire. Los patrones de scattering deben ser descritos con la teoría de Mie y no con la simplificación de Rayleigh.

La formulación del scattering de Mie queda descrita a través de los polinomios de Legendre y las funciones de Ricatti-Bessel. La sección del scattering (expresión 8) hace alusión a la probabilidad de que el rayo interaccione con la partícula generando dispersión espacial.

$$\sigma_{sca} = \frac{\lambda^2}{8\pi^2}(i_1 + i_2) \tag{8}$$

Las intensidades i_1 e i_2 se definen como:

$$i_1 = \left| \sum_{m=1}^{\infty} \frac{2m+1}{m(m+1)} \left[a_m \pi_m(\cos\theta) + b_m \tau_m(\cos\theta) \right] \right|^2 \tag{9}$$

$$i_2 = \left| \sum_{m=1}^{\infty} \frac{2m+1}{m(m+1)} \left[a_m \tau_m(\cos\theta) + b_m \pi_m(\cos\theta) \right] \right|^2 \tag{10}$$

Las funciones angulares $\pi(\cos\theta))$ y $\tau(\cos\theta)$ se expresan según los polinomios de Legendre.

$$\pi_m(\cos\theta) = \frac{P_m^1(\cos\theta)}{\sin\theta} \tag{11}$$

$$\tau_m(\cos\theta) = \frac{dP_m^1(\cos\theta)}{d\theta} \tag{12}$$

Y los coeficientes a_m y b_m se definen como:

$$a_m = \frac{\Psi_m(x)\Psi'_m(nx) - n\Psi_m(nx)\Psi'_m(x)}{\xi(x)\Psi'_m(nx) - n\Psi_m(nx)\xi'(x)} \tag{13}$$

$$b_m = \frac{n\Psi_m(x)\Psi'_m(nx) - \Psi_m(nx)\Psi'_m(x)}{n\xi(x)\Psi'_m(nx) - \Psi_m(nx)\xi'(x)} \tag{14}$$

Donde $\Psi(x)$ y $\xi(x)$ son las funciones de Ricatti-Bessel.

A modo de ejemplo, el patrón de scattering producido por una partícula de índice de refracción $n = 2 - j0.2$, y de dimensión característica $x = 6.25$, para una longitud de onda de $470nm$ se puede observar a continuación.

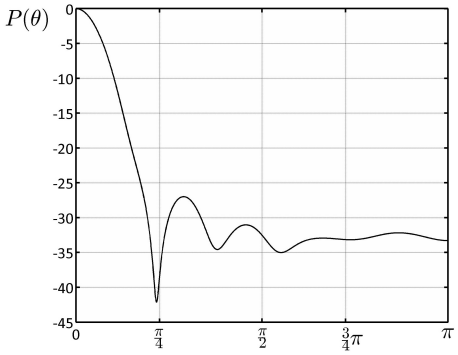

Figura 9: Patrón de scattering de la partícula descrita

Utilizando estas expresiones, puede obtenerse la sección de scattering total empleando la ecuación 8 de la siguiente manera:

$$\hat{\sigma_{sca}} = \frac{\lambda^2}{2\pi} \sum_{m=0}^{\infty} (2m+1)(|a_m|^2 + |b_m|^2) \tag{15}$$

Tal y como se definió en el apartado de interacción con partículas grandes, suponemos una distribución de partículas pequeñas análoga. De esta forma, existe una probabilidad de que se

colisione con una partícula, sabiendo que es una partícula pequeña, definida por la expresión siguiente:

$$p_{small} = 1 - e^{-\mu_s \cdot d} \tag{16}$$

Donde mu_s es el producto entre la concentración de partículas pequeñas ρ_s y la sección total del scattering $\sigma_s\hat{c}a$.

5.1.5. Modelo de colisión con partículas

Los procesos estocásticos por los cuales se rige la probabilidad de colisión con partículas son procesos independientes. La probabilidad de colisión es la suma de las probabilidades de colisión, de manera que:

$$p_C = p_{small} + p_{big} - p_{big} \cdot p_{small} \tag{17}$$

Sustituyendo las funciones de probabilidad de las expresiones 6 y 16:

$$p_C = 1 - e^{-(\mu_s + \mu_b)d} \tag{18}$$

Como puede observarse, al tratarse de funciones exponenciales la tasa de colisiones μ es aditiva, es decir, si se consideran colisiones sin tener en cuenta el tipo de partícula, la tasa de colisiones por metro equivalente es la suma de las tasas de colisiones de cada proceso por separado, pues se ha supuesto que los procesos son independientes.

Tras una colisión, la probabilidad de que haya existido colisión con una partícula determinada se calcula como:

$$p_{b|C} = \frac{\mu_b}{\mu_b + \mu_s} \tag{19}$$

$$p_{s|C} = \frac{\mu_s}{\mu_b + \mu_s} \tag{20}$$

Y además:

$$p_{b|C} + p_{s|C} = 1 \tag{21}$$

5.1.6. Modelo de la superficie

La superficie del mar es, de manera general, un plano cuya normal no puede ser determinada en cada instante y en cada punto del espacio, pues es la superposición de infinitos armónicos producidos por diferentes fenómenos como la cortadura del viento, la fuerza de Coriolis, y las mareas generadas por la atracción gravitatoria de los cuerpos celestes que rodean a La Tierra. Las ecuaciones que rigen el oleaje y las mareas están bien definidas para cada componente en frecuencia [32], pero a pesar de ello, no puede conocerse en cada instante un vector normal al plano que lo defina en cada punto, por ello, se trata a la superficie del mar como una variable aleatoria.

Para modelar la superficie del mar, se ha supuesto que el vector normal que la define tiene naturaleza normal, dependiendo de un parámetro σ_{surf} que está relacionado con la agitación de la superficie. No es objeto de este trabajo profundizar en este efecto, pues a priori no tiene demasiada influencia en las comunicaciones submarinas. Sin embargo, en las comunicaciones agua-aire, donde la luz debe cambiar de medio para llegar al receptor, este efecto puede ser de suma importancia.

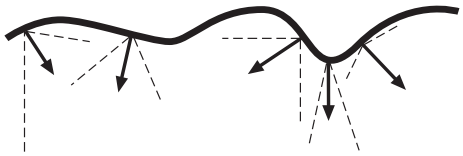

Figura 10: Ejemplo de superficie aleatoria

La normal aleatoria debe cumplir tres requisitos para resultar válida:

- Debe ser un vector unitario.
- La componente Y del vector debe ser negativa (apuntando hacia el fondo).
- La aleatoriedad queda definida en las tres direcciones (X, Y y Z).

Con estos requisitos, el vector normal se calcula como la superposición de tres componentes ortogonales con distribuciones normales de media nula y varianza σ^2_{surf}.

Al cambiar de medio, los rayos sufren desviaciones en sus direcciones de propagación debidos a la diferencia de índices de refracción. Según la ley de Snell, la relación de ángulos es la siguiente.

$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \tag{22}$$

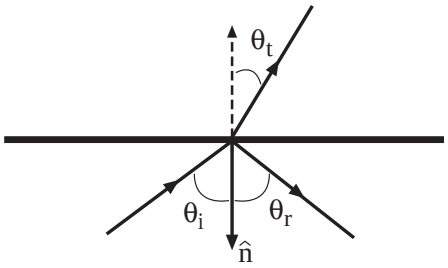

Figura 11: Ley de Snell

Además, cuando la radiación electromagnética pasa a un medio de índice de refracción menor, puede producirse el efecto de reflexión total interna a partir un ángulo crítico (ángulo de Brewster), no transmitiéndose energía al segundo medio.

$$\theta_{crit} = \arcsin\left(\frac{n_2}{n_1}\right) \tag{23}$$

La reflectividad relaciona la potencia incidente con la potencia reflejada por el interfaz en función del ángulo de incidencia. Para radiación sin polarización se calcula con la siguiente expresión.

$$R = \frac{1}{2}\left[\frac{\sin^2(\theta_T - \theta_i)}{\sin^2(\theta_T + \theta_i)} + \frac{\tan^2(\theta_T - \theta_i)}{\tan^2(\theta_T + \theta_i)}\right] \tag{24}$$

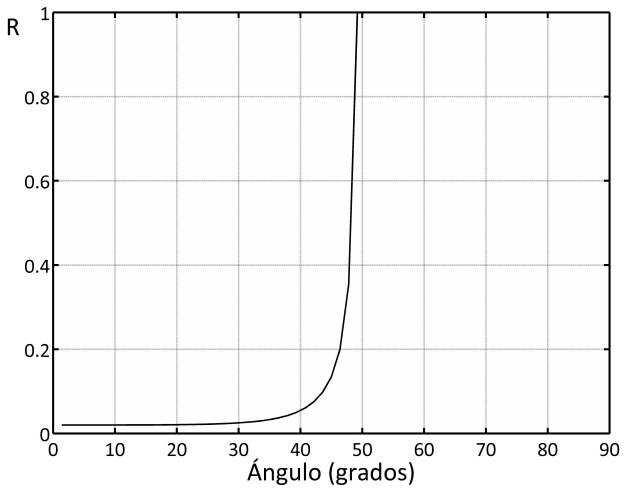

Figura 12: Curva de reflectividad entre el agua ($n \approx 1.33$) y el aire ($n \approx 1$)

5.1.7. Modelo del fondo

El fondo marino puede ser modelado como un difusor, es decir, como una superficie sobre la cual, al impactar un rayo, se emite un patrón de radiación lambertiano que depende del ángulo de incidencia y de la reflectividad del material del suelo marino. En este trabajo se va a suponer el fondo como un difusor lambertiano, igual que en los modelos utilizados por Barry [33]. Existen otros modelos de difusión más precisos y que se ajustan mejor a realidad, como los modelos de Phong [34], Torrance [35] o LaFortune [36], donde se tienen en cuenta, además de las componentes difusas del material, reflexiones y otros efectos.

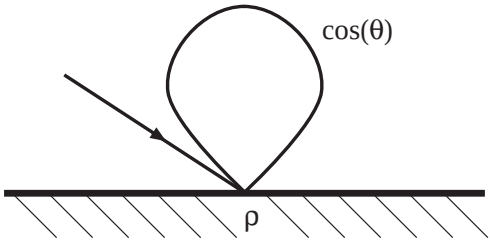

Figura 13: Efecto difusivo del fondo

La radiación emitida por el fondo puede ser modelada con la siguiente expresión.

$$R(\theta_i, \rho_{bottom}) = \rho_{bottom} \cdot cos(\theta_i) \cdot cos(\theta) \tag{25}$$

Donde θ_i es el ángulo de incidencia con el fondo, ρ_{bottom} es la reflectividad del suelo y θ es el ángulo de salida de la radiación difusa.

5.2. Modelo del emisor

En comunicaciones ópticas existen dos tipos de fuente, las fuentes tipo LED y láser. Los LED están basados en el fenómeno de emisión espontánea, generando luz monocromática no coherente a una longitud de onda que depende del material. Las fuentes láser emiten luz de manera estimulada, con un ancho espectral muy reducido y de manera coherente. En general, una fuente LED tiene un patrón de radiación que puede suponerse lambertiano (ver expresión 26), mientras que los láser no tienen patron de radiación, pues se supone que generan emisiones con ancho de haz nulo (a pesar de que existan fenómenos que produzcan dispersión espacial).

$$R(\theta, m) = \frac{m+1}{2\pi} \cos^m \theta \tag{26}$$

En este trabajo únicamente se han considerado fuentes tipo LED. Desde el punto de vista de la simulación de Monte Carlo, deben generarse rayos de manera aleatoria con el fin de cubrir todo el ángulo sólido radiado por el emisor. Para generar estos rayos, se utiliza una función densidad de probabilidad que se ajusta al patrón de emisión, de forma que habrá más concentración de rayos en un cierto ángulo sólido Ω. La dirección de salida de un rayo puede ser determinada por dos ángulos definidos en coordenadas esféricas θ y ϕ. El ángulo polar θ tiene una distribución de probabilidad que se ajusta al patron lambertiano de emisión, mientras que el ángulo acimutal ϕ tiene una distribución uniforme $U[0, 2\pi]$.

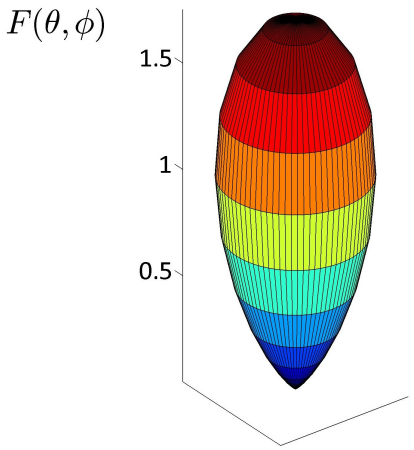

Figura 14: Ejemplo de emisor lambertiano de índice 10

Los rayos se generan acorde a las distribuciones de probabilidad comentadas. Ya que la potencia total de emisión debe ser P_T, cada rayo tendrá una potencia $P_i = P_T/N$, siendo N el número de rayos generados. El emisor queda definido, por tanto, por los siguientes parámetros:

- Potencia total de emisión P_T.
- Número de rayos generados N.
- Índice del patrón de radiación lambertiano m.

5.3. Aportaciones de potencia de cada rayo

Tras la generación de los N rayos en el emisor, cada uno con una dirección de propagación aleatoria definida por las distribuciones de probabilidad comentadas, al receptor llegarán M rayos provenientes de las múltiples colisiones con partículas, superficie y fondo. Cada rayo, además, llegará con un retardo que dependerá del camino óptico que haya seguido hasta llegar al receptor, definiendo una respuesta al impulso.

$$h(t,\lambda) = \sum_{i=1}^{M} P_i \cdot \delta\left(t - \frac{d_i \cdot n(\lambda, S, T)}{c_0}\right) \tag{27}$$

En general esta respuesta al impulso depende de la longitud de onda, pues todos los parámetros dependen de ella. Técnicamente, en un canal como el submarino, definir una respuesta al impulso implica que se considera el canal como lineal y temporalmente invariante. La linealidad se cumple debido a que no existen términos cuadráticos que puedan modificar las respuestas armónicas, sin embargo, la invarianza temporal únicamente puede asumirse en tiempos de coherencia que a priori están determinados por aquellos fenómenos aleatorios que tienen lugar debajo del agua (superficie del mar, movimiento de las partículas en suspensión, fauna marina, etcétera).

Cada rayo aportará al receptor una potencia que puede ser calculada por la siguiente expresión:

$$P_i = \frac{P_T}{N} L_{abs} \cdot \prod_{j=1}^{k} L_j \tag{28}$$

Donde:

- L_{abs} es la expresión de la absorción del agua (fórmula 1).
- k es el número de "saltos" que da el rayo hasta llegar al receptor.
- $L_k = \frac{A\cos\varphi}{\pi d_k^2}$ siendo $A\cos\varphi$ el área efectiva del receptor.
- L_j depende del tipo de colisión que se haya producido, las cuales pueden ser del siguiente tipo:
  - Colisión con la superficie. $L_j = \frac{R}{\pi d_j^2}$
  - Colisión con el fondo. $L_j = \frac{\rho_b \cos\theta_i \cos\theta}{\pi d_j^2}$
  - Scattering de Mie. $L_j = \frac{Mie(\theta)}{\pi d_j^2}$

Además, la distancia total recorrida por cada rayo será:

$$d = \sum_{i=1}^{k} d_i \tag{29}$$

6. Diseño del código C

En esta sección se comentan los aspectos relacionados con el diseño del algoritmo de Monte Carlo que se pretende implementar, así como las estructuras de datos que se van a incluir y el diagrama de flujo del algoritmo. Además, se comentará la decisión de abordar el problema en punto flotante en simple precisión con el fin de poder trabajar de manera coherente a la precisión que emplea la GPU.

6.1. Topología del problema

El problema de la simulación de la respuesta al impulso en el canal submarino puede ser abordado de manera modular, teniendo en cuenta un marco de referencia común que son las condiciones de contorno del escenario de simulación.

El objetivo del algoritmo es obtener un vector de retardos y un vector de potencias que se correspondan a las aportaciones de cada colisión y reflexión producida durante la propagación para cada rayo emitido en el emisor. Para ello, se propone la siguiente topología, donde cada rayo sigue un camino aleatorio que depende de la probabilidad de colisión con partículas y del escenario.

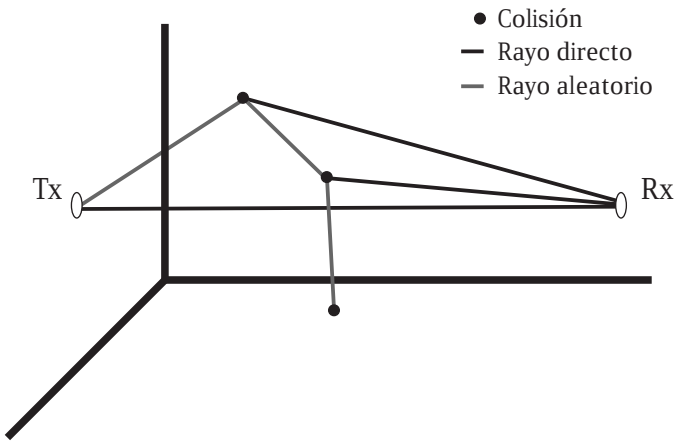

Figura 15: Topología del problema

En la figura del ejemplo se observa que cada rayo sufre desviaciones tras una colisión, las cuales influyen en el retardo y la potencia que aportarán al receptor.

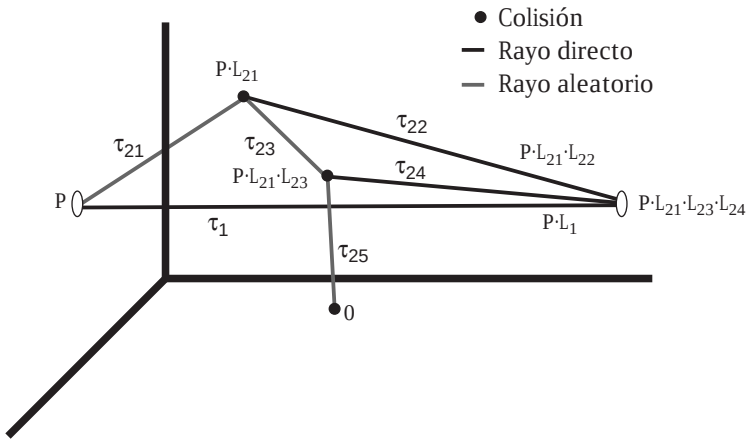

Figura 16: Topología del problema con potencias y retardos asociados

La respuesta temporal que generarían los rayos anteriores es la siguiente.

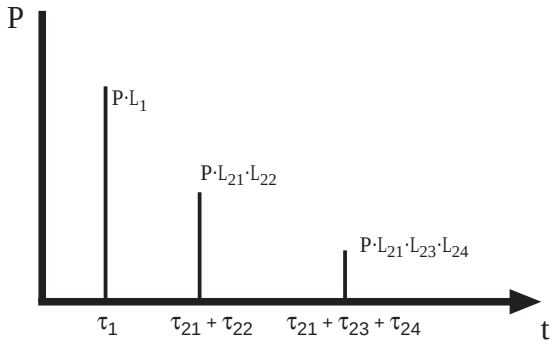

Figura 17: Respuesta temporal asociada a la topología de la figura 16

Tras cada impacto con un obstáculo (fondo, superficie o partículas), el punto de colisión se comporta como un nuevo emisor de rayos, quedando de manifiesto la fractalidad del problema. Este tipo de problemas se abordan de manera muy sencilla utilizando algoritmos recursivos, donde el criterio de parada lo determina el número de saltos máximo permitido o la pérdida de un rayo.

6.2. Estructuras de datos

Los algoritmos de Monte Carlo Ray Tracing dan lugar a estructuras de datos tipo árbol. En general, cuando ocurren colisiones, el punto de colisión pasa a comportarse como un nuevo emisor, y de él surge una nueva distribución de rayos que puede tratarse como un problema exactamente igual que el original, originándose un árbol con M ramas por colisión. Sin embargo, en este trabajo se ha considerado la generación de dos rayos únicamente cada vez que ocurre una colisión, de manera que el árbol generado es un árbol binario para cada rayo de salida.

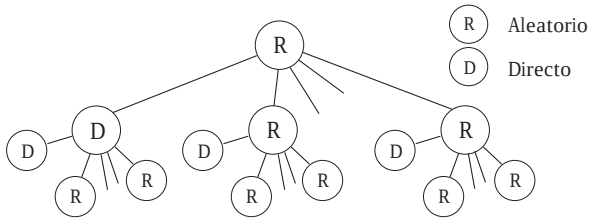

Figura 18: Estructura de árbol general

Como cada árbol tiene una profundidad máxima definida por el número máximo de saltos permitido (MAXHOPS) y cada rayo puede dar lugar a otros dos rayos tras una colisión, el tamaño máximo en memoria que ocupará un árbol es el siguiente.

$$M_{Tree} = 2^{MAXHOPS} \cdot M_{Ray} \tag{30}$$

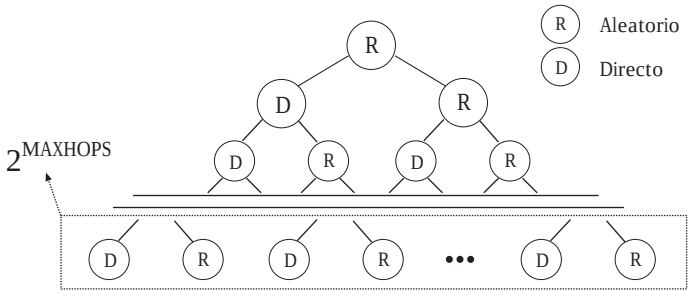

Figura 19: Árbol binario considerado en el trabajo

Los elementos que interesan desde el punto de vista del cálculo son los nodos hoja del árbol. Cada elemento, además, debe ir almacenando los parámetros necesarios para que se definan las propiedades del rayo en cada punto de colisión. Estas propiedades son las siguientes:

- Posición. Coordenadas cartesianas de la posición de la colisión.
- Vector director. Vector unitario que define la dirección de propagación del rayo.
- Indicador de rayo directo o rayo aleatorio. Es necesario conocer si el rayo se dirige al receptor o no, ya que únicamente se considerarán aportaciones, aquellos rayos directos que lleguen al receptor.s
- Distancia. La distancia recorrida es necesaria para poder calcular el retardo de propagación con la fórmula 27.
- Factor de potencia. Con cada colisión, la potencia del rayo se reduce en un factor que depende del tipo de colisión y de la distancia recorrida, tal y como se explicó en el modelo matemático.
- Salto actual. El salto actual define si el rayo debe perderse o no, ya que se impone un máximo de saltos permitidos para que el algoritmo tenga un tiempo de cómputo finito.

- Indicador de nodo hoja. Este parámetros será utilizado por una rutina para saber qué elementos del árbol implican aportación de potencia.

Además, para poder calcular en cada caso las probabilidades de colisión y poder realizar los cálculos correctamente, el árbol de rayos debe tener acceso a los parámetros de simulación. Estos parámetros son:

- Concentración de partículas grandes.
- Concentración de partículas pequeñas.
- Distancia a la superficie.
- Distancia al fondo.
- Coordenadas del receptor.
- Desviación típica de la normal de la superficie.
- Sección equivalente del scattering de Mie.
- Diámetro de las partículas grandes.
- Área del fotodiodo.
- Funciones de distribución de probabilidad asociadas al emisor y a las partículas que producen scattering.
- Reflectividad del suelo
- Parámetros del emisor (índice del emisor lambertiano, número de rayos, etc...)
- Ventana de integración. Este parámetro define el retardo máximo a considerar de los rayos que llegan al receptor.

De esta forma, se identifican tres estructuras de datos principales, cuya jerarquía se muestra gráficamente a continuación.

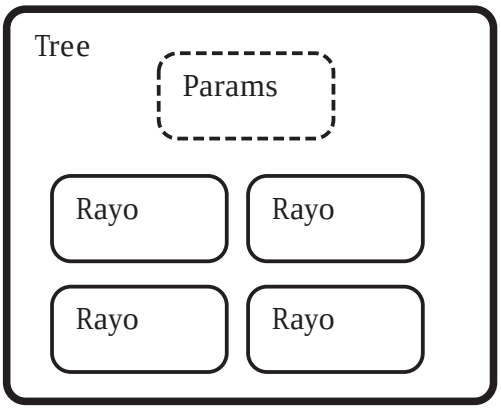

Figura 20: Dependencia entre las estructuras de datos

6.3. Generación de los rayos aleatorios

La dirección de propagación de los rayos queda definida, en coordenadas esféricas, por dos ángulos. Estos dos ángulos quedan determinados por el patrón de radiación del emisor. En este trabajo se ha considerado únicamente emisores lambertianos, los cuales quedan definidos por su índice de degeneración.

Como se comentó en el capítulo del modelo matemático, la distribución del ángulo polar se corresponde con el patrón de radiación, mientras que el ángulo acimutal, con una distribución uniforme, pues se radia con igual probabilidad en dicho ángulo. Considerando que se dispone de una función que obtiene números en coma flotante pseudoaleatorios con distribución uniforme en el intervalo [0,1], r, los ángulos pseudoaleatorios generados pueden calcularse como:

$$\theta = F^{-1}(r) \tag{31}$$

$$\varphi = 2\pi r \tag{32}$$

Donde F es la distribución de probabilidad calculada como la integral del patrón de radiación.

6.4. Generación de la normal aleatoria a la superficie

Cuando un rayo colisiona con la superficie del mar, debe generarse aleatoriamente la normal para poder obtener la reflectividad de la superficie y el ángulo de salida que se calculará mediante la ley de Snell.

Se ha definido que cada componente del vector normal a la superficie siga una distribución normal, forzando la componente Y a un valor siempre negativo.

Utilizando el mismo concepto de generación de números aleatorios a través de la distribución uniforme, para cada componente se obtiene la función gaussiana inversa.

$$x_i = \frac{\sigma_{surf}}{\sqrt{\ln r}} \tag{33}$$

6.5. Descripción del algoritmo

Se ha estructurado el algoritmo de forma que a cada rayo generado en el emisor le corresponda un árbol de rayos, donde se va almacenando el recorrido de cada rayo. Tras realizar el cálculo asociado a un árbol, una rutina recopilará aquellos nodos marcados como aportaciones válidas de potencia.

Para facilitar la programación, se ha planteado un esquema recursivo para rellenar el árbol de rayos, donde cada bifurcación del árbol es generada en función de la ocurrencia de un evento de colisión (superficie, fondo o partícula). Para conocer el tipo de colisión que ha ocurrido, se calcula la distancia desde la posición actual del rayo hasta los planos de interés (plano XY del receptor y planos de superficie y fondo) y se compara con una distancia de colisión generada aleatoriamente con la distribución de probabilidad de la fórmula 18.

Los pasos que sigue el algoritmo para evaluar el comportamiento de cada rayo, dada una dirección de propagación inicial, son los siguientes:

1. Comprobación del número de saltos, si se ha llegado al máximo, se desestima el rayo.
2. Cálculo de las distancias a los planos de interés.
3. Generación de la distancia aleatoria de colisión con partículas.

4. Determinación de la distancia de colisión y del tipo de colisión (superficie, fondo o partícula). En caso de tratarse de una colisión con una partícula, se determina con qué tipo de partícula se ha colisionado.

5. En función del tipo de colisión, se ejecuta un código u otro:

   - Si no hay colisión con partículas:
     - Si es un rayo directo al receptor, se calcula la aportación de potencia y se marca para "cosechar".
     - Si es un rayo aleatorio:
       - Si va a la superficie se calcula la reducción de potencia, la nueva dirección reflejada con la normal aleatoria y se hace una llamada recursiva como rayo aleatorio.
       - Si va hacia el plano del receptor se considera que el rayo se pierde.
       - Si va hacia el fondo:
         a) Se calcula la reducción de potencia.
         b) Se genera un rayo directo al receptor y se hace una llamada recursiva.
         c) Se genera un rayo aleatorio y se hace una llamada recursiva.
   - Si hay colisión con partícula grande se desestima el rayo, ya que se supone que la energía es absorbida.
   - Si hay colisión con una partícula pequeña:
     a) Se calcula la reducción de potencia.
     b) Se genera un rayo directo al receptor y se hace una llamada recursiva.
     c) Se genera un rayo aleatorio y se hace una llamada recursiva.

Una vez el árbol ha "crecido", una rutina recorre la estructura de datos generada, y en función del retardo asociado al rayo obtiene la posición correspondiente del vector de resultados. Una vez conocida la posición del vector, se suma la aportación de potencia a la ya existente.

La posición del vector se calcula como sigue:

$$pos = \lfloor t_{Rayo}(fs) - t_{min}(fs) \rfloor \tag{34}$$

6.6. Comentario sobre la resolución de los datos

Cuando se implementan algoritmos de simulación de la respuesta al impulso de un canal mediante trazado de rayos, la resultante compuesta por las aportaciones de cada rayo no debe ser integrada, pues cada rayo se supone que aporta potencia en un único instante temporal. Sin embargo, para poder trabajar con señales, es recomendable que estén muestreadas de manera equiespaciada en el tiempo, por lo que surge una controversia.

Los datos en coma flotante permiten resoluciones de 6.5 dígitos decimales, de forma que únicamente puede expresarse una millonésima de cualquier unidad (μs en el caso del tiempo). Como la velocidad de propagación de la luz en el agua está en el orden de cientos de miles de kilómetros por segundo, los retardos que se generarán serán más pequeños que la resolución de la coma flotante, por lo que la representación quedaria truncada y los datos falseados. Sin embargo, si se expresan las unidades temporales en femtosegundos a través de una definición de la velocidad de la luz en metros por femtosegundo, se alcanzan resoluciones de femtosegundo sin problema.

Desde el punto de vista del simulador, al quedar los retardos truncados al femtosegundo, puede existir la posibilidad de que algunos rayos generen aportaciones de potencia en el mismo instante. En este caso, la potencia se suma y se produce un efecto de integración que genera un ligero error, el cual no se tiene en cuenta.

De esta manera, se ha optado por una representación de los datos en simple precisión con el fin de optimizar el uso de la GPU.

6.7. Comentario sobre el número máximo de saltos

El parámetro de simulación MAXHOPS define el número máximo de saltos que se le permite dar a un rayo antes de ser desestimado.

Este parámetro además define las necesidades de memoria para almacenar las estructuras de datos definidas para la simulación. En el caso de la estructura Árbol, el número de rayos que deberá almacenar es $2^{MAXHOPS}$. Dependiendo de las condiciones del entorno de simulación, puede ocurrir que un número de saltos dado sea demasiado pequeño y no se generen aportaciones válidas debido a que se producen más colisiones que saltos permitidos.

Puede establecerse un criterio para definir el valor mínimo de MAXHOPS según las condiciones de simulación. Sea un escenario donde únicamente existen partículas pequeñas en suspensión (que generan colisiones a distancias d_{col}) y existe una distancia entre emisor y receptor D, MAXHOPS debe ser como mínimo:

$$MAXHOPS >= \lceil \frac{D}{E[d_{col}]} \rceil \tag{35}$$

Ya que las colisiones con partículas quedan definidas por una ley exponencial, la distancia media de colisión será $1/\mu$, quedando la expresión anterior como:

$$MAXHOPS >= \lceil D \cdot \mu \rceil \tag{36}$$

7. Implementación

En esta sección se comentan las particularidades de la implementación del código en secuencial y en paralelo. Respecto del código secuencial, se han considerado dos implementaciones distintas, una recursiva y otra iterativa. Esto es debido a que la GPU no permitía llamadas recursivas y fue necesario implementar la pila de estado manualmente. A pesar de que las implementaciones recursivas son muy intuitivas y fáciles de programar, son ineficientes en rendimiento y velocidad. Por ello, se han incluido en la comparativa las diferencias entre el código secuencial recursivo y el iterativo.

7.1. Código secuencial

El código secuencial implementado es la base de este trabajo, y dependiendo de su implementación, el código paralelo será más o menos eficiente, por lo que es necesario centrar los esfuerzos en desarrollar un código secuencial optimizado. En esta sección no se presentan secciones de código, ya que están disponibles como anexo a este trabajo. Se comentará la programación del algoritmo según el esquema de la sección de descripción del algoritmo del capítulo anterior.

El algoritmo realiza los siguientes pasos antes de llevar a cabo la simulación del canal propiamente dicha:

- Lectura de los parámetros de simulación. Rutina *inputData*.
- Cálculo de la absorción asociada a la longitud de onda de trabajo. Rutina *getAbsorption*.
- Cálculo del índice de refracción. Rutina *getRefractionIndex*.
- Cálculo de la sección del scattering por partículas pequeñas y el patrón de scattering de Mie. Rutina *bhmie*. Esta rutina implementa el algoritmo de Bohren-Huffman [37], y se ha adaptado a partir de un código libre disponible en la base de datos de código de Google.
- Inicialización de la estructura de parámetros (Params) que define el escenario de simulación. Rutina *initializeParams*.

Tras realizar las acciones anteriores, donde únicamente se han definido las condiciones de contorno del escenario de simulación, se procede al cálculo de la respuesta impulsiva del canal. Para ello, se realiza el siguiente proceso:

- Inicialización del árbol. Rutina *initializeTree*.
- Cálculo de las aportaciones válidas del rayo directo. Para ello se fija el campo *isRandom* del primer rayo a 0.
- Cálculo de las aportaciones válidas de los rayos aleatorios. Se generan N rayos aleatorios dentro de un bucle (campo *isRandom* a 1) y se obtienen las aportaciones correspondientes.

Para calcular las aportaciones de potencia válidas se han usado dos rutinas: la que hace "crecer" el árbol, *growTree*, y la rutina que "cosecha" aquellos nodos hoja que implican potencia en el receptor, *harvestTree*.

Independientemente de si el código es implementado de manera recursiva o iterativa, el proceso que se sigue para discernir si un rayo colisiona o no y el camino óptico que sigue, es el mismo. A continuación se comenta paso por paso el algoritmo.

7.1.1. Cálculo de la base asociada al rayo

Ya que el problema trata la propagación en 3D de un rayo y que tras una colisión, las direcciones de los rayos aleatorios generados dependen del vector director del rayo actual, será necesario definir una base vectorial que quede definida por el vector director, por ello, se ha implementado una estructura de datos denominada "Base" con tres campos que se corresponden a los tres vectores ortonormales de la base. Matemáticamente, la base vectorial **B** determinada por el vector director del rayo, quedaría definida como sigue.

$$B = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{pmatrix} \tag{37}$$

Para encontrar la base se ha empleado el algoritmo de ortogonalización de Gram-Schmidt, implementado en la rutina *gramSchmidt*. Para definir la base se ha empleado como vectores de referencia el vector director del rayo y los vectores $\vec{e_1}$ y $\vec{e_2}$ de la base canónica. Ya que el vector director puede corresponderse con alguno de los vectores de la referencia, si se sigue el algoritmo directamente, podría ocurrir que la base resultante no generara todo el espacio $\Re^3$. Por ello, se comprueba previamente si el vector director se corresponde con alguno de los vectores mencionados de la base canónica con la rutina *isCanonical*. En tal caso, se genera directamente la base canónica rotada debidamente.

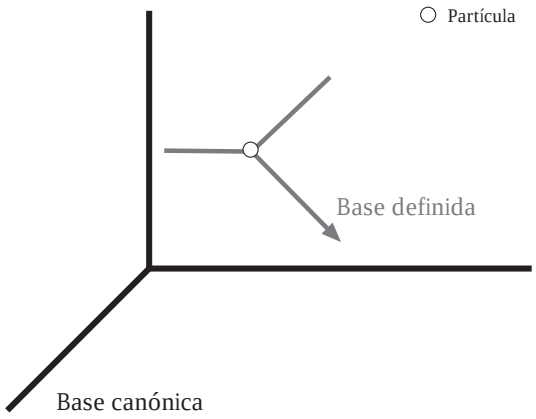

Figura 21: Ejemplo de definición de la base del espacio transformado

La operación para pasar de la base canónica a la base generada es la siguiente, y se lleva a cabo en la rutina *toCustomSpace*.

$$\vec{v}_{custom} = B \cdot \vec{v}_{canonical} \tag{38}$$

La operación inversa, paso del espacio generado a la base canónica (*toCanonicalSpace*), se lleva a cabo calculando la matriz inversa, la cual se corresponde con la traspuesta al tratarse de una matriz ortogonal, en la rutina *trasposeMatrix*.

$$\begin{array}{rcl} \vec{v}_{canonical} & = & B^{-1} \cdot \vec{v}_{custom} \\ \vec{v}_{canonical} & = & B^{T} \cdot \vec{v}_{custom} \end{array} \tag{39}$$

7.1.2. Determinación de la distancia recorrida y del tipo de colisión

Para cada rayo, definido geométricamente por una posición y un vector director, debe calcularse la distancia a la siguiente colisión y el tipo de colisión que se produce.

El escenario de simulación queda definido por tres planos como ya se comentó en el capítulo del modelo matemático: el plano del receptor, el plano de la superficie y el plano del fondo. Para la dirección de propagación de un rayo, debe obtenerse la distancia hasta el plano del receptor y la distancia hasta el plano de superficie o fondo (según indique la componente Y del vector director).

Estas distancias se calculan de la siguiente manera:

$$d_{planoRec} = \left| \frac{z_{rec} - z_{ray}}{v_z} \right| \tag{40}$$

$$d_{SurfBottom} = \left| \frac{y_{plano} - y_{ray}}{v_y} \right| \tag{41}$$

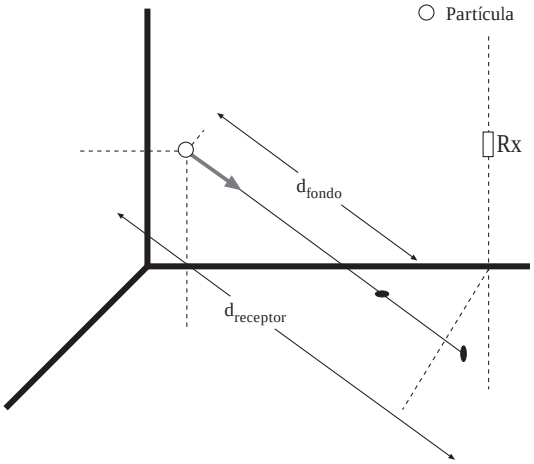

Figura 22: Distancias a los planos de interés ara un escenario dado

En el caso de que estas distancias sean infinitas, pues la dirección de propagación puede implicar que no exista punto de corte con alguno de los planos, se indica con -1.0.

Una vez obtenidas estas distancias, la máxima distancia que podrá recorrer el rayo será la mínima de las anteriores, siempre y cuando no sean infinitas las dos, donde en tal caso se indica que la distancia es -1.0. La dirección de propagación indicará también si el rayo colisiona con el plano del receptor o uno de los planos paralelos mediante un indicador denominado "tipo" (0 para el plano del receptor, 1 para la superficie y 2 para el fondo). Los casos son los siguientes:

Una vez obtenida la distancia a los planos, debe generarse aleatoriamente la distancia de colisión con partículas, para ello se emplean las fórmulas 18, 19 y 20. Una vez obtenida la distancia de colisión aleatoria, si esta distancia es menor que la distancia anteriomente calculada, se fija esta distancia como la distancia del siguiente evento sobre el rayo, en otro caso, implica que no ha habido colisión con partícula. El tipo de partícula con el que colisiona se indica con la variable "tipoColision" (0 no hay colisión con partículas, 1 colisión con partícula grande y 2 colisión con partícula que produce scattering de Mie).

d/tipo	$d_{Rx} < d_{Plano}$	$d_{Plano} < d_{Rx}$
$v_y > 0$	$d_{Rx}/0$	$d_{Plano}/1$
$v_y < 0$	$d_{Rx}/0$	$d_{Plano}/2$
$v_y = 0$	$d_{Rx}/0$	-1/0

Cuadro 2: Tabla de decisión de distancias

d/tipo de colisión	$d < d_{col}$	$d_{col} < d$
Big	$d/0$	$d_{col}/1$
Small	$d/0$	$d_{col}/2$
None	$d/0$	-1/0

Cuadro 3: Tabla de decisión de tipo de colisión con partículas

Una vez obtenida la distancia y definido el tipo de colisión que se ha producido, se procede a procesar el efecto producido sobre el rayo según el tipo de evento.

Sea cual sea el tipo de evento que se produzca, excepto aquellos en los que el rayo se desestime (absorción por partícula grande por ejemplo), el estado del rayo (posición , potencia, dirección, etcétera) debe actualizarse. A continuación se comenta este aspecto.

7.1.3. Actualización del estado del rayo tras un evento

La actualización del estado del rayo es necesaria para definir las condiciones sobre las que operar para calcular el siguiente evento que se produzca. Esta actualización se lleva a cabo antes de generar el siguiente evento, y deben actualizarse todos los campos de la estructura tipo Rayo. Estos cálculos son los siguientes:

- Dirección de propagación. La dirección se calcula dependiendo del tipo de evento que se ha producido.
- Tipo de rayo (aleatorio o directo). Este indicador se modifica dependiendo también del tipo de evento. Para la reflexión superficial únicamente se genera rayo aleatorio, pero para los demás evento se generarán los dos tipos de rayo.s
- Posición. La posición del estado $n+1$ se calcula segun la siguiente expresión:
  $x_i^{n+1} = x_i^n + d \cdot v_i$
- Distancia. La distancia recorrida por el rayo se calcula añadiendo la distancia de colisión calculada en el apartado anterior a la ya existente en la estructura.
- Factor de potencia. Dependiendo del tipo de evento, el factor de potencia se pondera según el tipo de función correspondiente, y además, siempre, por las pérdidas por propagación esférica para la distancia recorrida en el vano.

Ya que la fórmula de las pérdidas por propagación esférica es de tipo hiperbólica al cuadrado, puede darse el caso de que si la distancia es muy pequeña, la potencia del rayo se amplifique. Este hecho es físicamente imposible, y es una limitación debida a la aproximación de rayo (potencia concentrada en un ángulo sólido infinitesimal). Para solventar este hecho, ya que pueden producirse dos colisiones consecutivas a corta distancia, se ha definido la función de pérdidas del vano de la siguiente manera.

$$L_{vano} = \begin{cases} L_{col} & \text{si } d < \sqrt{\frac{L_{col}}{\pi}} \\ L_{col}/(\pi d^2) & \text{en otro caso} \end{cases} \quad (42)$$

Siendo L_{col} las pérdidas debidas a la colisión y asegurando así que el rayo nunca queda amplificado.

7.1.4. Colisión con la superficie

Si un rayo colisiona con la superficie del mar, la cual queda definida por el vector normal a la superficie, el cual ha sido modelado como una variable aleatoria (expresión 33), se produce una reflexión que a priori, genera un rayo aleatorio que no va dirigido al receptor.

Se ha modelado la reflexión superficial como un rayo aleatorio siempre, ya que la probabilidad de que el rayo vaya dirigido al receptor es muy pequeña, por lo que el error se desprecia.

Para calcular el factor de potencia del rayo reflejado se utiliza la fórmula de Fresnel (expresión 24). El ángulo de incidencia del rayo con la superficie se calcula como sigue (teniendo en cuenta que las normas de los vectores son unitarias):

$$cos(\theta_i) =< \vec{v}_{ray}, \hat{n} > \quad (43)$$

Siendo $\vec{v}_{ray}$ el vector director del rayo.

Para calcular la dirección de salida del rayo, calculamos la proyección de la dirección del rayo respecto del plano normal y restamos dos veces la aportación del vector director en la dirección normal, obteniendo así, el rayo reflejado.

$$\vec{v}_{ref} = \vec{v}_{ray} - 2 < \vec{v}_{ray}, \hat{n} > \hat{n} \quad (44)$$

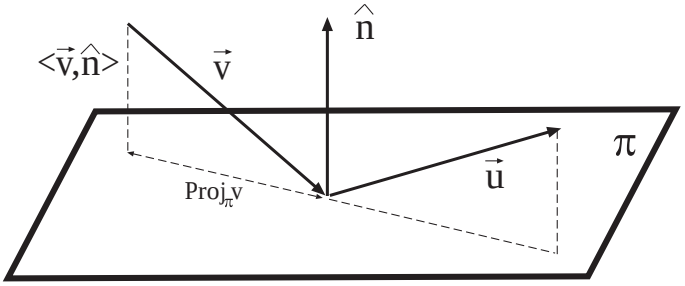

Figura 23: Geometría del cálculo del rayo reflejado

7.1.5. Colisión con el fondo

Al impactar un rayo con el fondo, se generarán dos rayos: uno que representa la aportación directa al receptor y otro que hace alusión a la energía que se dispersa por difusión (rayo aleatorio). La potencia de ambos rayos se pondera segun la expresión 25.

Para el rayo directo, el programa sigue los siguientes pasos:

- Se obtiene la base del espacio transformado respecto de la normal al fondo, es decir, al vector (0,1,0).
- Se calcula el vector directo normalizado al receptor como $\vec{v}_D = \frac{1}{D_{p-Rx}} \sum x_{rec,i} - x_i$.

- Se transforma el vector directo del espacio canónico al espacio transformado y se expresa en esféricas con la rutina *toSphericals*.
- Se calcula el factor de ponderación conociendo la theta incidente (componente Y del vector inicial del rayo) y la theta del ángulo de salida.
- Se actualiza el estado del rayo y se introduce como entrada del siguiente proceso de cálculo.

Para el rayo aleatorio, se procede de manera análoga:

- Se obtiene la base del espacio transformado respecto de la normal al fondo, es decir, al vector (0,1,0).
- Se obtiene aleatoriamente un ángulo polar de salida (θ) sabiendo que se difunde la energía con un patrón lambertiano de orden uno ($\cos\theta$). Se obtiene a través de función inversa de la integral del patrón de radiación (función de probabildad), es decir, con la función $\arcsin(r)$. El ángulo acimutal (ϕ) se obtiene con una distribución de probabilidad uniforme.
- Se transforma el vector obtenido en esféricas (θ,ϕ,1) a coordenadas rectangulares con la rutina *toRectangulars*.
- Ya que el ángulo generado está referido a la base definida por la normal al fondo, debe expresarse respecto de la base canónica, para ello se utiliza la rutina *toCanonicalSpace*.
- Se calcula el factor de ponderación conociendo la theta incidente (componente Y del vector inicial del rayo) y la theta del ángulo de salida.
- Se actualiza el estado del rayo y se introduce como entrada del siguiente proceso de cálculo.

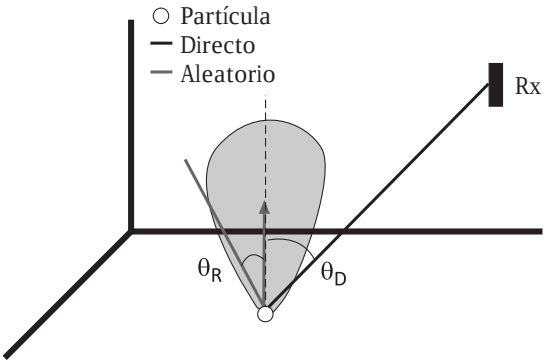

Figura 24: Geometría de colisión con el fondo

7.1.6. Colisión con partícula pequeña

Cuando existe una colisión con una partícula pequeña se sigue el mismo proceso que con la colisión con el fondo. En este caso, la función de probabilidad empleada ha sido calculada anteriormente a través de los resultados de la rutina de Bohren-Huffman y es la empleada para obtener el ángulo polar de salida de los rayos aleatorios y ponderar las energías. Además, en este

caso, el espacio transformado es calculado tomando como referencia el vector director original del rayo.

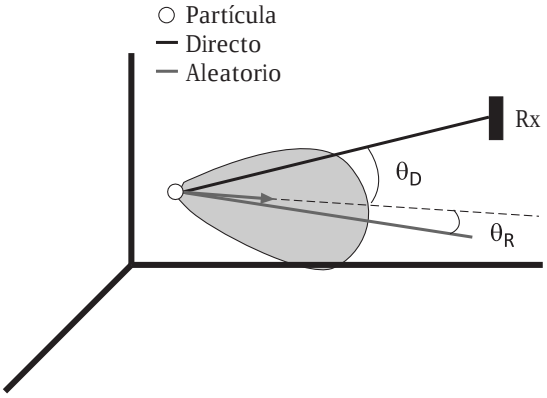

Figura 25: Geometría de la colisión con una partícula pequeña

7.1.7. Llegada al receptor

Cuando el rayo llega al receptor, se actualiza la distancia recorrida y se obtienen las pérdidas totales que ha sufrido el rayo a través de la ecuación 28.

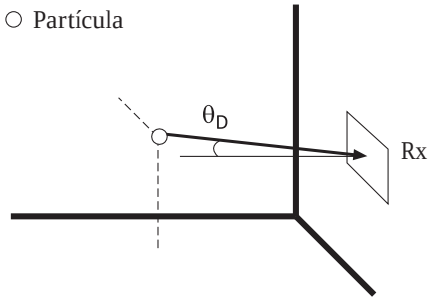

Figura 26: Geometría de la llegada al receptor

Con el estado final del rayo fijado, se marca para "cosechar".

A continuación se comentarán las diferencias entre la implementación recursiva inicial y la implementación iterativa que tuvo que realizarse forzosamente para que fuese posible compilar el código para la GPU.

7.1.8. Particularidades de la versión recursiva

En esta versión se realizan llamadas recursivas sobre la estructura Árbol inicializada con cada rayo cada vez que ocurre un evento de colisión. Ya que se trata con estructuras lógicas como las descritas en la figura 19, el número de elementos que tendría un árbol lleno serian:

$$\text{máx}\, Elems = \sum_{i=0}^{MAXHOPS} 2^i = 2^{MAXHOPS+1} - 1 \tag{45}$$

Sin embargo, el tipo Árbol instancia únicamente un vector de Rayos, el cual se corresponde con el último nivel de profundidad del árbol, por lo que se reserva memoria para $2^{MAXHOPS}$ rayos.

En esta versión, la cual fue programada como un prototipo inicial, primero se hacían todas las llamadas recursivas y se marcaban aquellos rayos que habían llegado al receptor, para posteriormente recorrer el vector e ir almacenando dichas aportaciones. Respecto del rendimiento del código, las llamadas recursivas implican guardar en la pila de programa todo el estado de la función antes de la llamada, lo cual supone un coste de cómputo y memoria adicional respecto del codigo iterativo.

A continuación se muestra el diagrama de flujo del código:

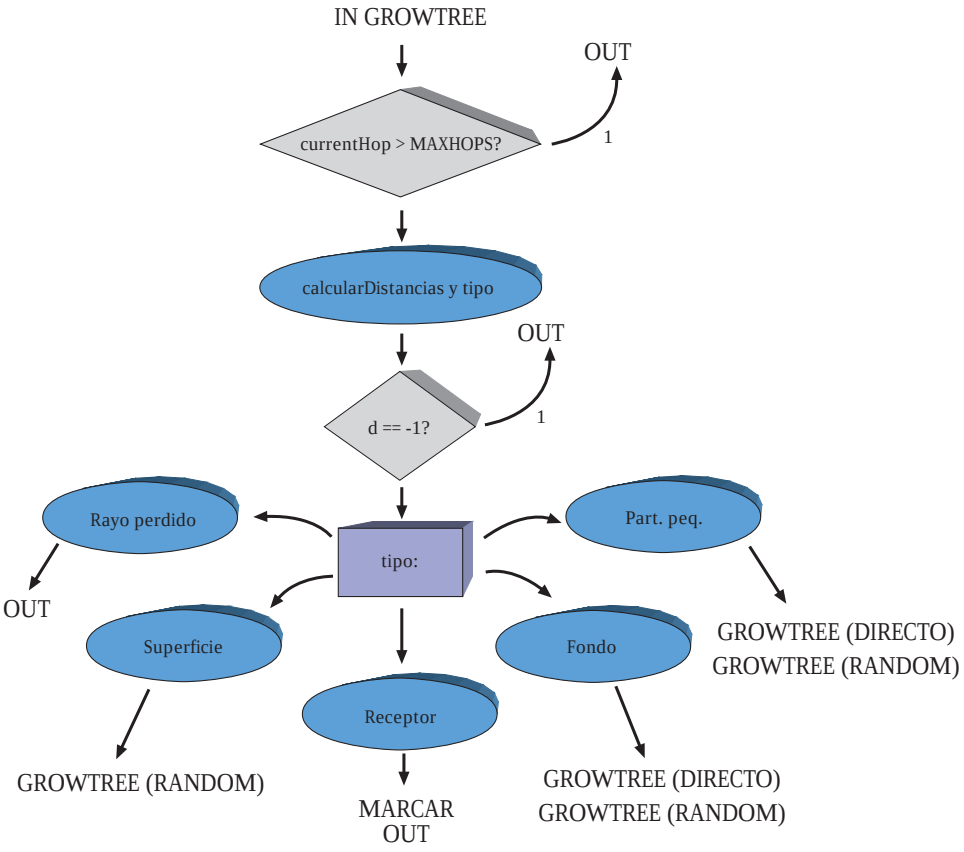

Figura 27: Diagrama de flujo del algoritmo recursivo

7.1.9. Particularidades de la versión iterativa

En esta versión, se sustituyeron las rutinas *growTree* y *harvestTree* por una única rutina que almacena las aportaciones en el momento que son calculadas (*growAndHarvestTree*). Además, ya que se sustituyen las llamadas recursivas por un bucle, debe implementarse la pila manualmente. Respecto del código recursivo, de manera somera, donde existia una llamada recursiva se guarda el estado en la pila, se actualiza el rayo y se realiza otra iteración, mientras que donde existía una sentencia return, se ha realizado una extracción del contenido de la pila.

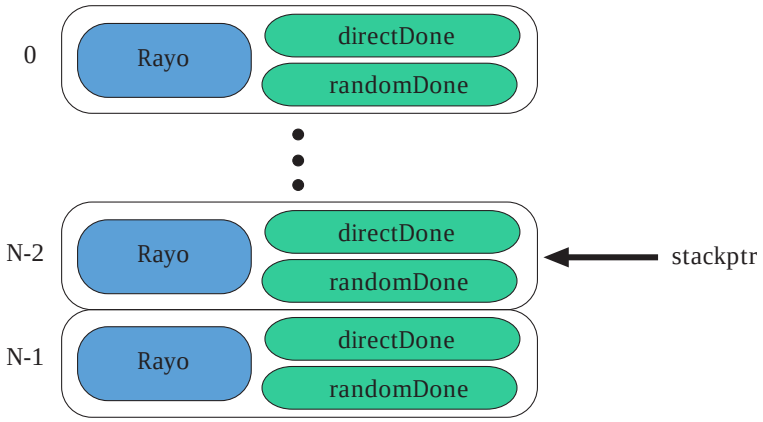

Figura 28: Implementación de la pila

En la versión recursiva se precisaba de una estructura de tipo array donde almacenar los rayos a la hora de calcular sus aportaciones. Sin embargo, en esta versión, con un único Rayo se puede realizar el cálculo, por lo que desaparece la entidad Árbol. Las variables que deben almacenarse en la pila son: el rayo original y dos indicadores booleanos para saber si ya ha sido calculado el rayo directo y el rayo aleatorio.

La rutina *save* almacena el estado actual en la cabeza de la pila, mientra que la rutina *pop*, carga el primer elemento de la pila. Cuando el *stackptr* (variable que indica la siguiente posición libre de la pila) está a cero, si el algoritmo indica que se debe hacer otro *pop*, se termina la ejecución.

Los variables que indican que el rayo directo y el rayo aleatorio ya han sido generados son *randomDone* y *directDone*. La rutina *save* debe almacenar estos valores, pero en el inicio de cada iteración, deben estar fijados a cero, ya que o se genera un nuevo rayo o se realiza una extracción de la pila.

A continuación se muestra el diagrama de flujo del código:

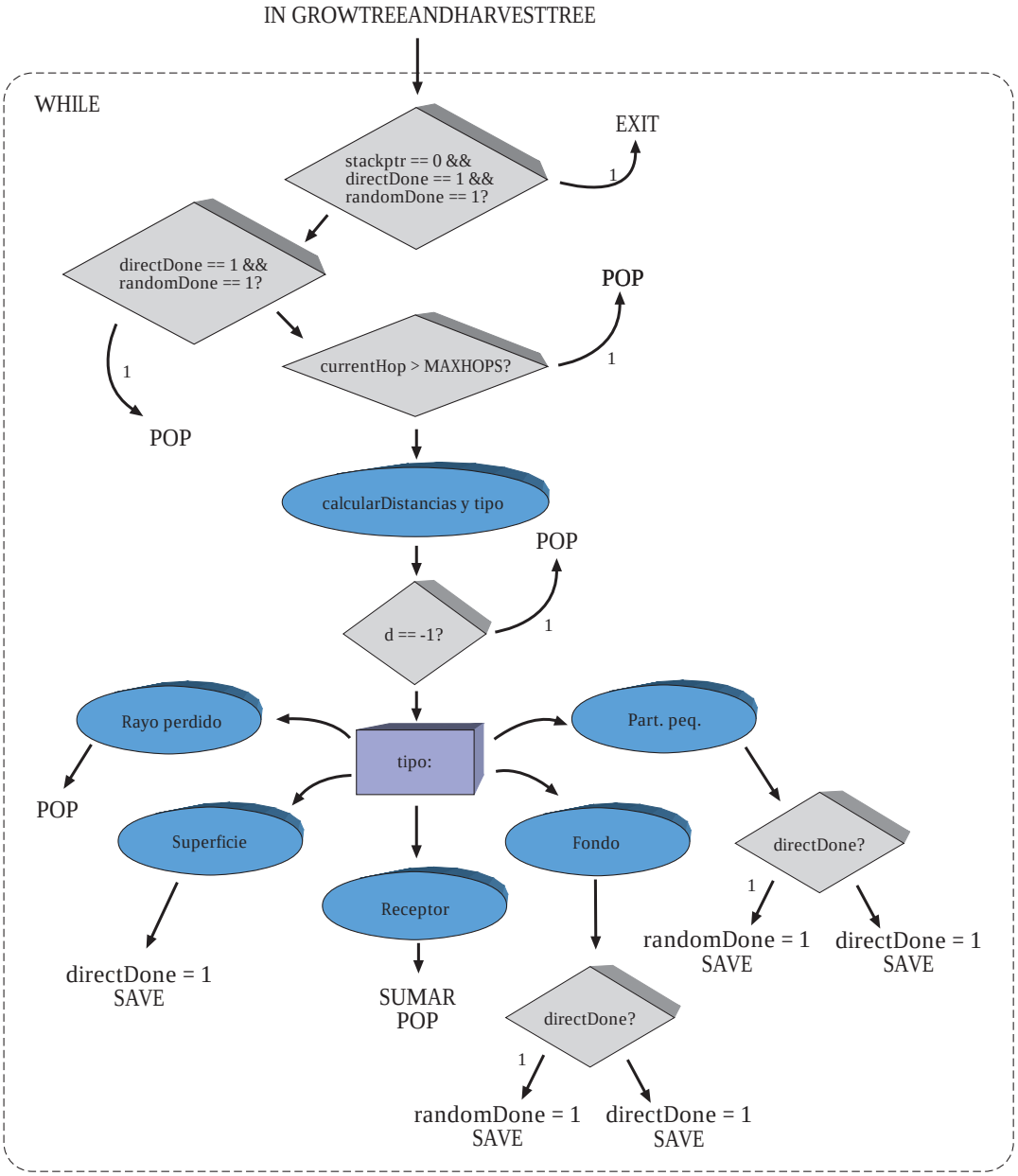

Figura 29: Diagrama de flujo del algoritmo iterativo

7.2. Código OpenMP

El código del simulador de canal se basa en generar dentro de un bucle el número de rayos aleatorios determinado por el archivo de parámetros. Cada rayo es independiente del otro, es decir, los caminos ópticos que siguen los rayos están determinados únicamente por la dirección aleatoria de salida y las colisiones aleatorias que se producen hasta llegar al receptor, por lo que la paralelización es *a priori* muy sencilla e inmediata.

El código secuencial que se ha paralelizado ha sido el código iterativo, ya que es el que es más eficiente en tiempo de cómputo y requisitos de memoria. En este apartado se comentan los aspectos básicos de la paralelización empleando OpenMP.

Ya que los rayos son independientes y su cálculo asociado se realiza dentro de un bucle, la paralelización se basa únicamente en distribuir las iteraciones según el número de hilos que van a emplearse en el cálculo. Para indicar el número de procesos generados, se ha añadido un campo al archivo de parámetros: "Número de threads".

Ya que OpenMP considera como variables compartidas todas aquellas variables existentes antes de la paralelización, cada hilo deberá poseer un rayo de manera privada, y además, ya que los ángulos aleatorios de salida (acimutal y polar) se generan dentro del bucle, estas variables también deberán ser declaradas privadas. Sin embargo, el vector donde se almacenan las contribuciones de potencia es compartido, ya que así se mejora la eficiencia del algoritmo. Otra posibilidad es declarar dicho vector como privado y sumar elemento a elemento los vectores de cada hilo. Se ha elegido la primera opción porque la probabilidad de que 2 o más hilos accedan a la misma posición de memoria (considerando la probabilidad de acceder a una posición equiprobable) es relativamente pequeña. Para evitar problemas a la hora de actualizar una posición de memoria, se ha declarado la sección donde se actualiza la posición de memoria como crítica.

La probabilidad mencionada, para M hilos y un vector de N elementos es ($p = 1/N$ y $q = 1-p$):

$$p_{N,M} = \sum_{i=2}^{M} p^i q^{M-i} \tag{46}$$

Ya que N es un número muy grande, q tenderá a la unidad, por lo que la probabilidad de acceso múltiple puede simplificarse de la siguiente manera:

$$p_{N,M} = \sum_{i=2}^{M} p^i q^{M-i} \approx \sum_{i=2}^{M} p^i \tag{47}$$

Ya que es una serie geométrica, la probabilidad anterior se calcula como:

$$p_{N,M} \approx \frac{p^2 - p^{M+1}}{1-p} = \frac{p^2(1-p^{M-1})}{1-p} \approx p^2 \tag{48}$$

A modo de ejemplo, para un vector de aportaciones de potencia de cien mil elementos, esta probabilidad queda acotada aproximadamente a 10^{-10}.

Debe tenerse en cuenta que este resultado es una aproximación, ya que se considera que el acceso a una posición sigue una distribución equiprobable, cuando en la realidad la probabilidad de acceso a una posición depende del escenario de simulación, pues existirá siempre mayor concentración de accesos en las posiciones cercanas al origen.

El código se ha paralelizado utilizando las dos siguientes directivas de OpenMP.

- **#pragma omp parallel for private(ray, theta, phi)**. Situada justo antes del bucle donde se generan los rayos.
- **#pragma omp critical**. Situado justo antes de la actualización de la posición de memoria dentro de la rutina *growAndHarvestTree*.

7.3. Código para GPU

La versión para GPU del simulador se ha llevado a cabo definiendo todas las funciones de las que hace uso la función **growAndHarvestTree** de tipo **__device__**, de manera que se ejecuten

en la tarjeta gráfica y no en el host. Además, se han definido dos *kernels* distintos, uno secuencial para llevar a cabo el cálculo de la aportación del rayo directo (un único hilo) y otro paralelo que se ejecutará en múltiples hilos en la GPU.

Ha tenido que modificarse la definición de la estructura de parámetros (Params), ya que referenciaba con punteros los arrays donde estaban almacenadas las funciones de distribución de probabilidad para el scattering y el emisor. Se ha cambiado por un almacenamiento por valor, para que al cargar la estructura en la GPU, no se referencie memoria de host sino memoria global de la GPU.

Otro aspecto que tuvo que modificarse fue la gestión de memoria dentro del código del simulador. En las versiones ejecutadas en el host, se hacía uso de la función **memcpy**, y al pasar a la GPU, estas funciones no son interpretadas por el compilador y son ignoradas. Por ello, las copias se han efectuado manualmente.

7.3.1. Generación de números aleatorios

CUDA no dispone de un generador de números aleatorios como el **rand()** de C, por lo que la generación de números aleatorios se ha tenido que llevar a cabo de manera externa. Ya que cada hilo va a hacer uso de números aleatorios para realizar el cálculo, se ha implementado la siguiente rutina de tipo **__device__**.

```
__device__ float frand(uint *m_z, uint *m_w){
   uint dummy1;

   *m_z = 36969 * (*m_z & 65535) + (*m_z >> 16);
   *m_w = 18000 * (*m_w & 65535) + (*m_w >> 16);
   dummy1 =  (*m_z << 16) + *m_w;  /* 32-bit result */

   return ((float)dummy1/FRAND_MAX);

}
```

El código anterior es un generador de números pseudoaleatorios con distribución uniforme en el intervalo (0,1) [38], el periodo del código generado es del orden de 2^{20}, es decir, que la secuencia de números generados no se repetirá hasta que se haya llamado a la función un número de veces igual al periodo.

Como puede observarse, los parámetros **m_z** y **m_w** se pasan como punteros, con el fin de hacer que cada hilo posea una semilla diferente y genere valores incorrelados respecto de los demás hilos en ejecución. El esquema seguido para hacer uso de los números aleatorios es el siguiente:

1. Generar en el host un número aleatorio por cada hilo.
2. Llamar al kernel pasando a cada hilo su número aleatorio.
3. Ejecutar el código paralelo utilizando la función anterior.

8. Resultados

En esta sección se presentan los resultados tanto de la respuesta al impulso del canal en función de los parámetros como de aceleraciones y eficiencias según la paralelización del código.

Se han realizado simulaciones de canal con diferentes parámetros, con el fin de observar la influencia de dichos parámetros en la respuesta al impulso. Se ha definido un **escenario base** con los siguientes parámetros:

- Emisor lambertiano puro. Orden 1.
- Radio de las partículas pequeñas: $8\mu m$
- Concentración de partículas pequeñas: $4 \cdot 10^9 parts/m^3$
- Concentración de partículas grandes: 0
- Distancia a la superficie: $5m$
- Distancia al fondo: $5m$
- Distancia al receptor: $5m$
- Número de rayos: 10^5
- Reflectividad del fondo: $\rho = 0.5$
- Agitación de la superficie: $\sigma = 2.5$

8.1. Resultados del simulador de canal

A continuación se comenta la influencia de los parámetros de simulación sobre la respuesta al impulso resultante de la simulación. Debe tenerse en cuenta que dos iteraciones diferentes del simulador proporcionan dos resultados diferentes, ya que las colisiones con partículas son modeladas mediante variables aleatorias.

Como primer resultado, se observa la variación de la respuesta al impulso en función de la distancia. Todos los resultados están representados en unidades logarítmicas (dBm).

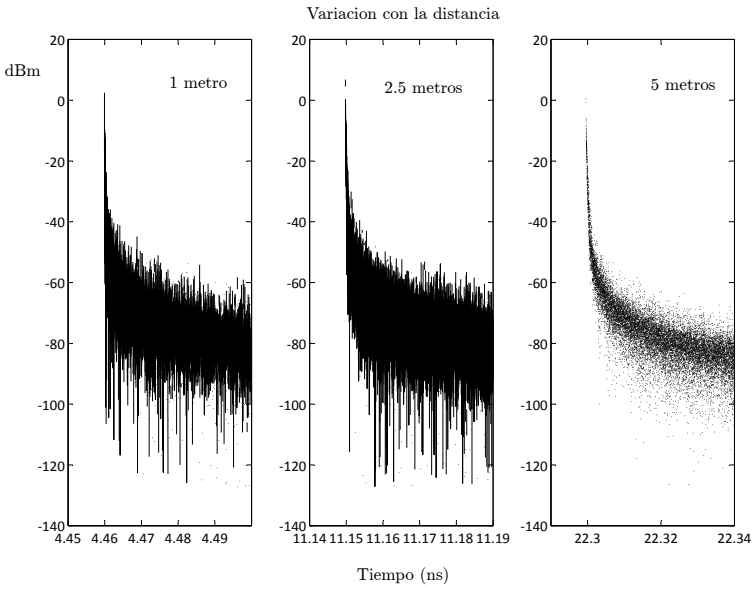

Figura 30: Variación de la respuesta al impulso con la distancia al receptor

Se observa que el máximo de la respuesta al impulso es mayor para el caso de 2.5 metros que para el de 1 metro. Esto es debido a que la distancia media de colisiones para el tipo de partículas y concentración definidas es de 60 centímetros, con una varianza de 35 centímetros aproximadamente, por lo que existe una probabilidad significativa de que los rayos se pierdan en el caso de 1 metro. Además, se observa que la gráfica está punteada, esto es debido a la representación logarítmica. Como el vector que contiene la respuesta al impulso tiene a priori bastantes elementos nulos, a la hora de representarlos se corta la curva en dichos puntos.

Se ha calculado el esparcimiento del retardo para los tres casos. Es intutivo pensar que a mayor distancia, la cola de la respuesta al impulso se verá más esparcida temporalmente. Los resultados son los siguientes (calculados a un 95 % de la potencia de la señal):

$d_{Rx}(m)$	τ_{rms}
1	$153fs$
2.5	$206fs$
5	$274fs$

Cuadro 4: *Delay spread* en función de la distancia

Como se observa, el esparcimiento es del orden de décimas de picosegundo, por lo que en comunicaciones ópticas submarinas (normalmente a MHz), la respuesta va a poder considerarse como una delta de Dirac. Sin embargo, en el caso de comunicaciones tipo láser, este esparcimiento puede comenzar a ser significativo, reduciendo el ancho de banda disponible en el canal.

Se ha simulado también la influencia de la distancia a la superficie en la respuesta impulsiva.

Se observa que para un enlace a 25 centímetros de profundidad, existen aportaciones de potencia que hacen alusión a las reflexiones en la superficie. Igual que en el caso anterior, se ha analizado el esparcimiento del retardo para poder visualizar la influencia de la distancia a la

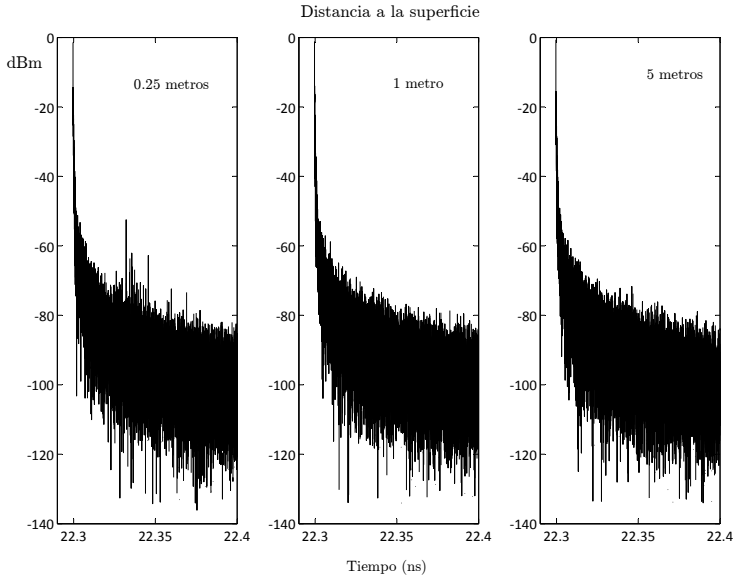

Figura 31: Variación de la respuesta al impulso con la distancia a la superficie

superficie en el ancho de banda disponible.

$d_{Rx}(m)$	τ_{rms}
1	$286fs$
2.5	$280fs$
5	$274fs$

Cuadro 5: *Delay spread* en función de la distancia a la superficie

Puede concluirse que existe una pequeña influencia con la distancia a la superficie. Debe destacarse que las aportaciones de potencia debidas a la superficie son aleatorias, y que la gráfica se corresponde con una posible distribución.

La agitación de la superficie hace que los rayos reflejados en el interfaz agua-aire, tengan direcciones más dispersas a medida que el coeficiente de dispersión σ crece. A priori, se concentrará más potencia debido a las aportaciones por reflexión cuanto más en calma esté la superficie del mar. La figura 32 hace referencia a una simulación a 25 centímetros de profundidad.

Se observa que con el mar totalmente en calma, existe aparentemente una segunda cresta de potencias localizada a 0.3 nanosegundos de la contribución principal. Esto es debido a que son contribuciones referentes a las reflexiones que hacen que exista potencia significativa en el receptor. A medida que la agitación del agua crece, esta contribución de potencia tiende a dispersarse temporalmente.

El fondo ha sido modelado como un difusor lambertiano puro. En la figura 33 se observa la influencia de la distancia al fondo.

No existe prácticamente diferencia entre las respuestas obtenidas para 1 metro y 5 metros de distancia al fondo, esto es debido a que no hay reflexiones ni difusiones significativas a dicha

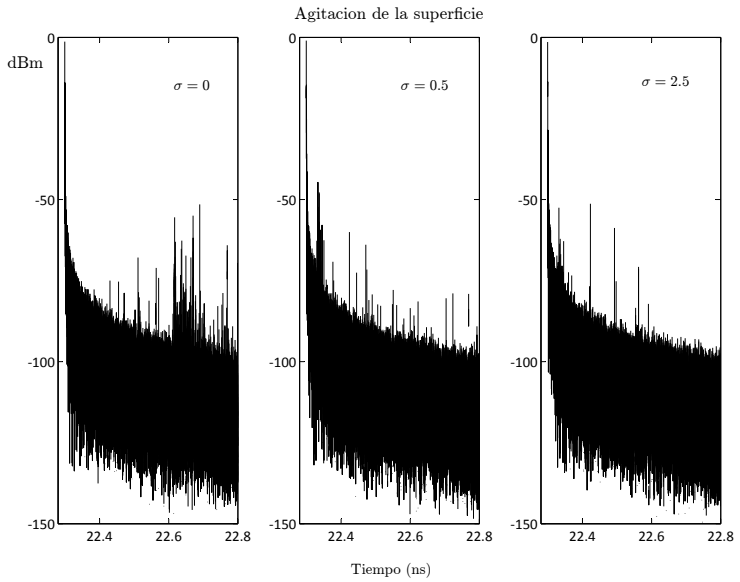

Figura 32: Variación de la respuesta al impulso con la agitación de la superficie

distancia. Sin embargo, a 25 centímetros de distancia, las contribuciones de potencia son muy significativas, haciendo que exista una segunda cresta que, a pesar de estar 50 dB por debajo de la contribución principal, se extiende más de 1 nanosegundo en el tiempo.

La reflectividad del fondo tiene también una influencia significativa en la respuesta al impulso si la distancia al fondo es pequeña, de forma que un fondo muy especular generará más aportaciones que un fondo totalmente absorbente.

El resultado es el esperado (figura 34). Para un fondo totalmente absorbente la forma de la respuesta al impulso se parece bastante al de un enlace en mar profundo (5 metros), sin embargo, a medidad que la reflectividad crece, la segunda cresta mencionada anteriormente comienza a ganar presencia.

El radio de las partículas modifica el patrón de scattering que generan. Cuanto mayor es el radio, la luz se ve menos afectada por la presencia de la partícula y la dirección de propagación no se ve prácticamente afectada. Sin embargo, al reducirse la dimensión normalizada de la partícula, empiezan a existir más lóbulos secundario en el patron de scattering y la luz se dispersa en mayor medida.

A medida que el radio disminuye, la sección del scattering también disminuye, y por lo tanto, la probabilidad de colisión con una particula pequeña, por lo que el número de colisiones será menor para un escenario dado. Sin embargo, cuanto menor es la dimensión normalizada de una partícula, el scattering que produce es mayor, desviando los rayos en mayor medida (figura 35).

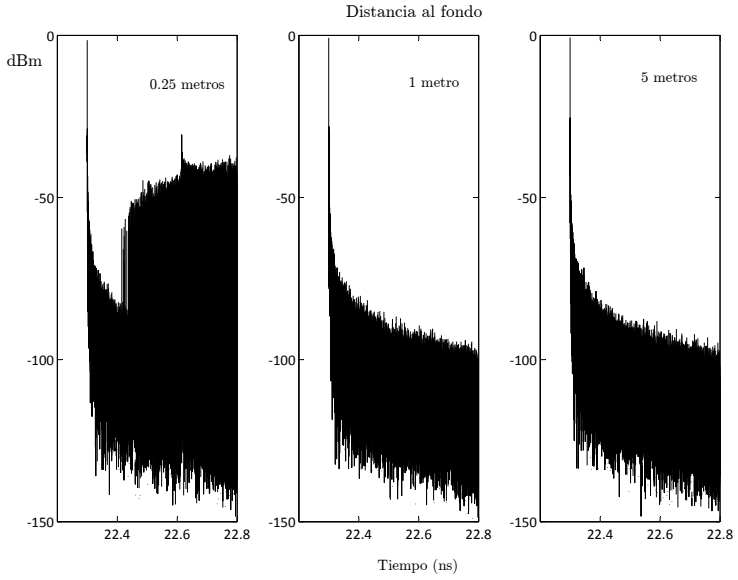

Figura 33: Variación de la respuesta al impulso con la distancia al fondo

$R(\mu m)$	τ_{rms}
1	$28.312ps$
4	$1.38ps$
8	$274fs$

Cuadro 6: *Delay spread* en función del radio de las partículas

Se observa que la dispersión temporal de la respuesta depende fuertemente del radio de las partículas (cuadro 6).

Cuanto mayor sea la concentración de partículas suspendidas en el medio, la luz se verá más dispersada y a priori, la cola de la respuesta al impulso se verá ensanchada.

En la figura 37 se observa que a medida que la concentración de partículas aumenta, el número de aportaciones de potencia significativas disminuye (señal más punteada). Esto es debido a dos efectos, por un lado, el simulador tiene una limitación en el número máximo de saltos que permite (a mayor densidad de partículas existirán más saltos); y por otro lado a mayor número de saltos, menor potencia aportada.

El *delay spread* asociado a este efecto es el siguiente:

$conc(x1e9)$	τ_{rms}
2	$274fs$
4	$274fs$
8	$286fs$

Cuadro 7: *Delay spread* en función de la concentración

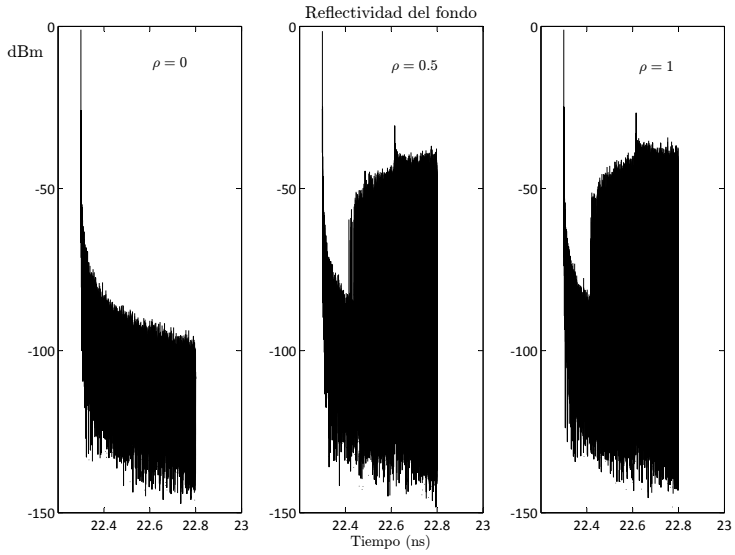

Figura 34: Variación de la respuesta al impulso con la reflectividad del fondo

Se observa que para el tipo de partículas simuladas (y el escenario en general), no existe una influencia significativa con la concentración, ya que la potencia dispersada por cada partícula es muy pequeña.

El número de rayos influye en la precisión de la respuesta al impulso obtenida. Un número pequeño de rayos puede no ser suficiente para cubrir todas las aportaciones de potencia que se generan en el receptor. Por ello se ha simulado el mismo escenario con tres densidades de rayos distintas. Debe tenerse en cuenta que tal como se ha planteado el simulador, un único rayo puede llegar a generar $2^{MAXHOPS-1}$ aportaciones de potencia según los parámetros de simulación.

La única diferencia observada en la figura 38, es el grado de definición de la curva obtenida. Esto pone de manifiesto que en función del escenario, será necesario un número mínimo de rayos para obtener una respuesta al impulso aceptable.

Por último, se ha simulado la influencia del patrón de radiación de la fuente emisora en la respuesta impulsiva del canal (figura 39). Es lógico pensar que cuanto más directiva sea la fuente, menor influencia existirá con los parámetros de simulación, ya que la emisión tenderá a parecerse a un único rayo.

Se observa que a medida que se concentra la emisión en un ángulo sólido menor, se aumenta la potencia de la señal recibida. En este caso no se observa que la dispersión del retardo se disminuya, pero a en la figura 40 se observa la misma simulación que la anterior pero para partículas de radio $1\mu m$.

En este caso se observa que si existe una dependencia con el índice de emisión, ya que el ángulo sólido efectivo que dispersa potencia por partículas de 1 micrómetro es mayor que para el caso de partículas de 8 micrómetros, donde la presencia de dichas partículas no desvía la dirección de propagación de los rayos prácticamente. El *delay spread* para este caso se muestra en la tabla 8

Por último, para observar el efecto absorbente de las partículas grandes, se ha variado la concentración de estas particulas (de 0.1 mm de radio) a 100 partículas por metro cúbico y se

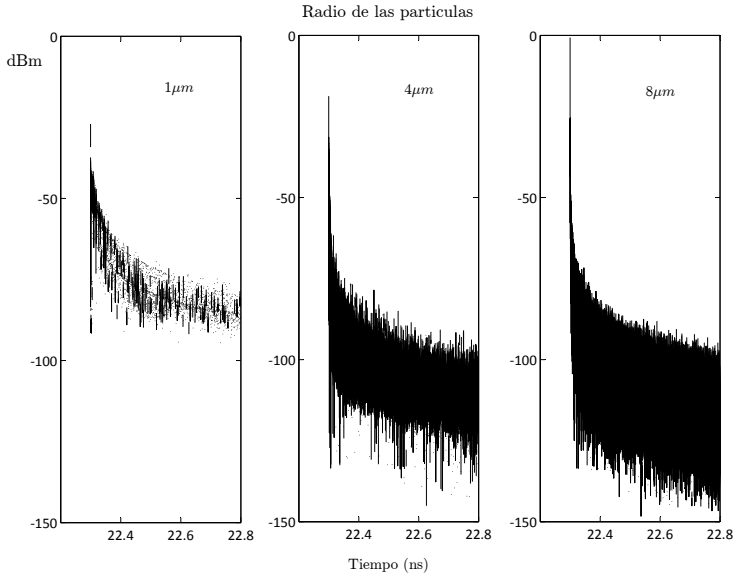

Figura 35: Variación de la respuesta al impulso con el radio de las partículas

m	τ_{rms}
1	$28.312ps$
10	$23.74ps$
100	$20.41ps$

Cuadro 8: *Delay spread* en función del índice de emisión para partículas de $1\mu m$

han obtenido tres realizaciones del experimento.

La figura 41 muestra que la potencia máxima varía en un rango de $+-3dB$ y que aparecen y desaparecen aportacionse de potencia detrás del pico principal. Una mayor concentración de partículas implicaría una mayor reducción de la potencia, pudiendo llegar a anular incluso la llegada de rayos al receptor. El efecto que producen las partículas grandes en el receptor es el de generar centelleo (desvanecimientos profundos en períodos muy cortos).

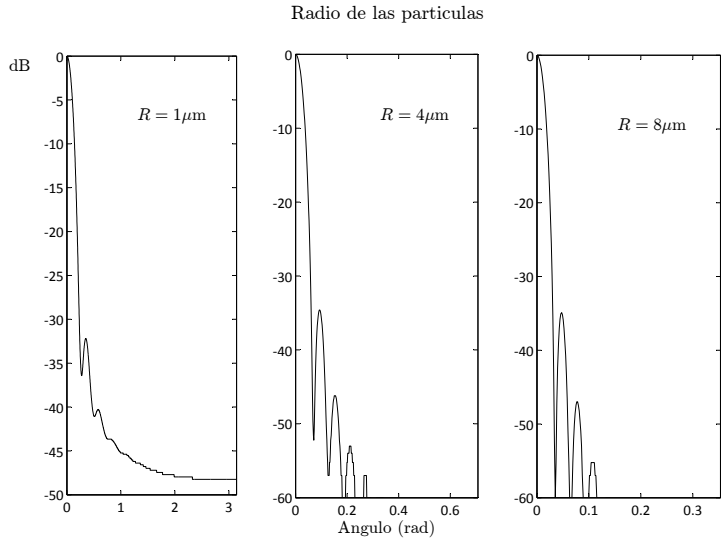

Figura 36: Patrones de scattering correspondientes a los radios anteriores

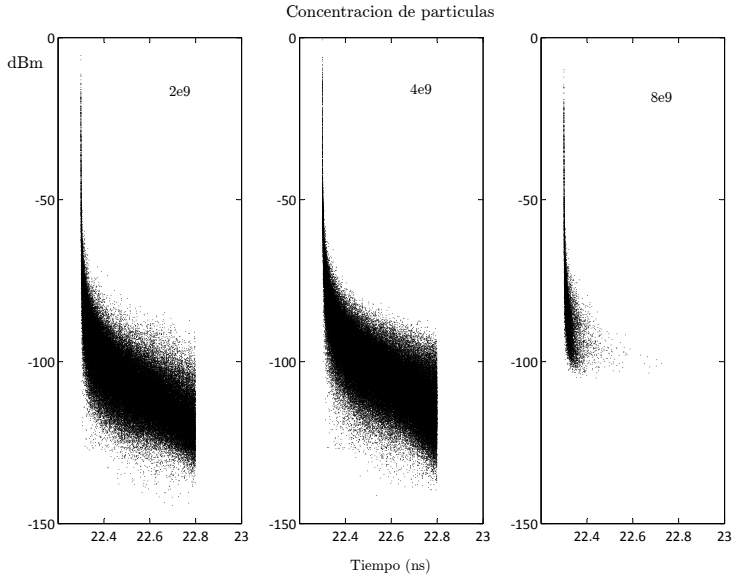

Figura 37: Variación de la respuesta al impulso con la concentración de partículas

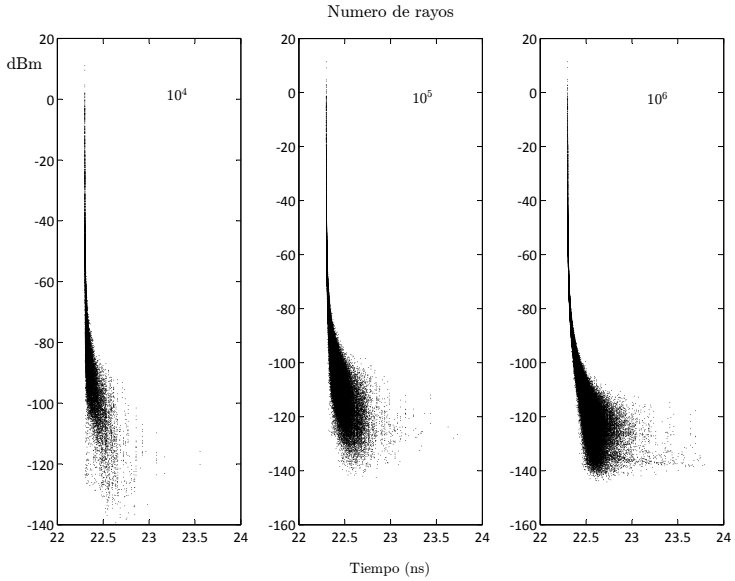

Figura 38: Variación de la respuesta al impulso con el número de rayos

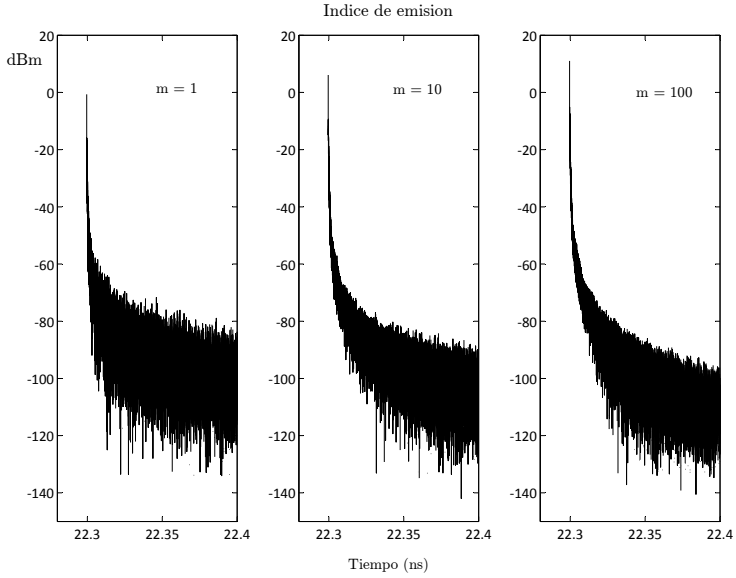

Figura 39: Variación de la respuesta al impulso con el patrón de radiación del LED

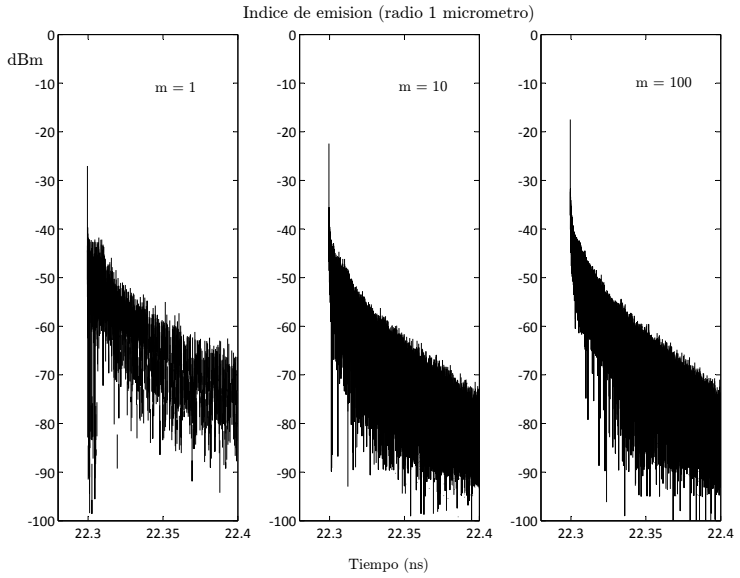

Figura 40: Variación de la respuesta al impulso con el patrón de radiación del LED con partículas de $1\mu m$s

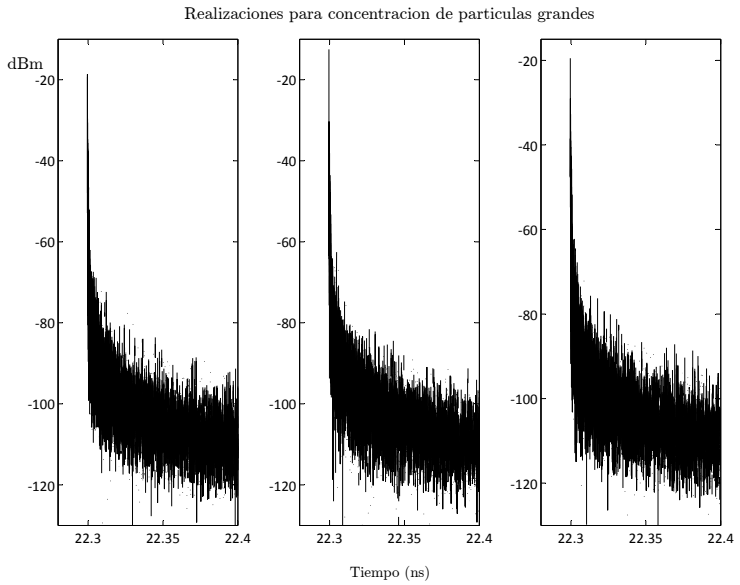

Figura 41: Tres realizaciones de la respuesta al impulso con concentración de partículas grandes

8.2. Resultados de la paralelización

En esta sección se presentan las aceleraciones y eficiencias relativas a la paralelización del código del simulador de canal. Como primer resultado, se presenta la aceleración debida a la programación iterativa del código frente a la programación recursiva. El simple hecho de variar esta programación ha hecho que el tiempo de cómputo se vea acelerado por un factor de 22.

En los resultados obtenidos, el escenario de simulación que se ha empleado es el descrito como **escenario base** en la sección anterior, ejecutándose en 68381.74 ms aproximadamente.

8.2.1. OpenMP

A continuación se presenta la aceleración del código al utilizar OpenMP.

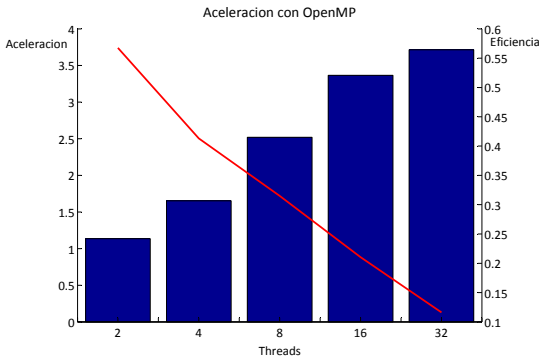

Figura 42: Aceleración y eficiencia para OpenMP

Utilizando OpenMP sin ningún tipo de planificación resulta bastante ineficiente. Las estructuras tipo árbol generadas durante el proceso de cálculo tienen profundidades y anchos aleatorios, por lo que es probable que un hilo termine mucho antes que otro y la ejecución del hilo maestro se vea frenada. Por ello se han empleado dos planificaciones, una estática y otra dinámica, con el fin de aumentar la eficiencia.

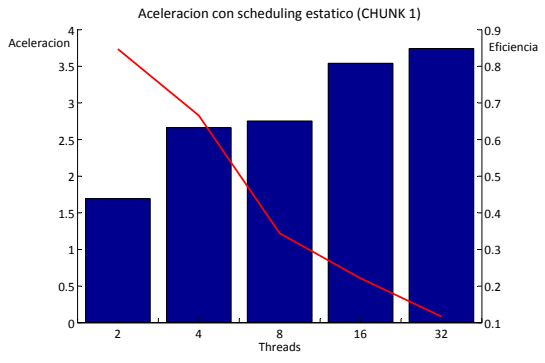

Figura 43: Aceleración y eficiencia para OpenMP empleando static scheduling (CHUNK 1)

Se observa que utilizando una estrategia de planificación estática, la eficiencia se ve aumentada, sin embargo, al utilizar un *chunk* (número de iteraciones a realizar por hilo antes de volver al conjunto total para seguir con la ejecución) tan bajo, el posible que el *overhead* generado, merme la eficiencia. Se han obtenido resultados para tamaños de chunk 10 y 100.

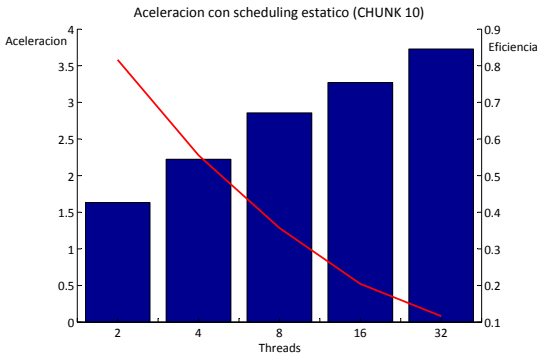

Figura 44: Aceleración y eficiencia para OpenMP empleando static scheduling (CHUNK 10)

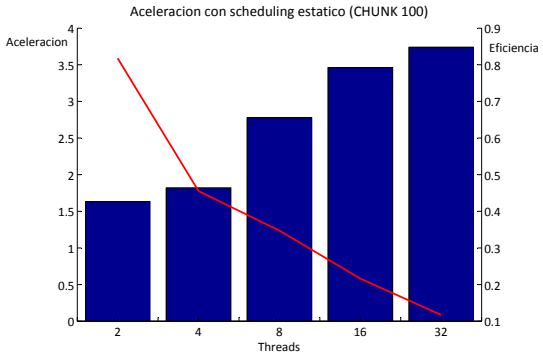

Figura 45: Aceleración y eficiencia para OpenMP empleando static scheduling (CHUNK 100)

El hecho de aumentar el tamaño del chunk no influye significativamente en la eficiencia del simulador. A priori, utilizando planificación de tipo *dynamic* la eficiencia del simulador debe aumentar, pues cada hilo no espera por la sincronización para obtener más iteraciones.

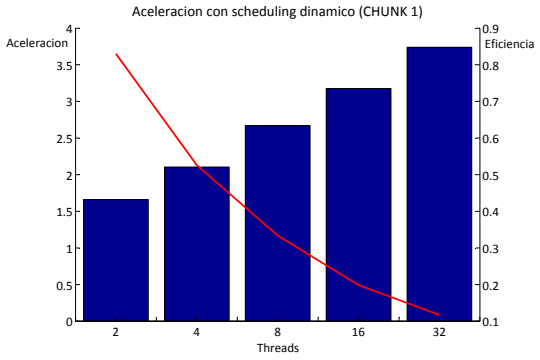

Figura 46: Aceleración y eficiencia para OpenMP empleando dynamic scheduling (CHUNK 1)

Para un tamaño de chunk tan bajo no es visible la mejora, sin embargo, para chunks mayores si es visible.

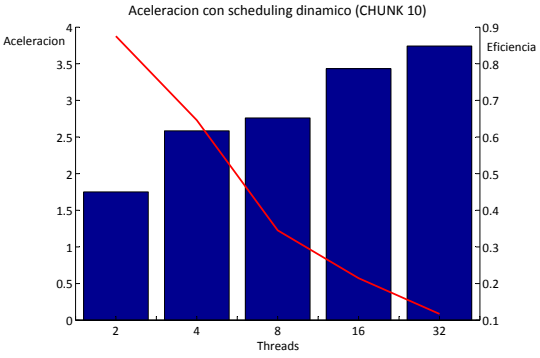

Figura 47: Aceleración y eficiencia para OpenMP empleando dynamic scheduling (CHUNK 10)

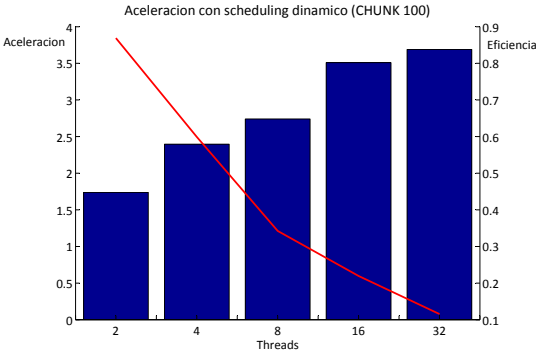

Figura 48: Aceleración y eficiencia para OpenMP empleando dynamic scheduling (CHUNK 100)

Puede observarse que para 2 hilos, en la planificación estática la eficiencia no alcanza el 85 por ciento, sin embargo, usando planificación dinámica, está cerca del 90 por ciento, lo cual pone de manifiesto que para este programa, la planificación dinámica es una buena elección.

8.2.2. CUDA

Al emplear CUDA y la GPU Tesla M2050, se dispone de 448 núcleos para realizar cálculos. Sin embargo, aceleraciones cercanas al número de núcleos no podrán ser alcanzadas ya que como se comentó anteriormente, las GPU operan de manera SIMT, y el código está implementado con saltos condicionales, por lo que muchos de los procesadores que conforman los multiprocesadores serán deshabilitados. Independientemente de esta limitación, las aceleraciones obtenidas con esta tecnología serán muy superiores a las de OpenMP. Esta limitación podría ser solventada probablemente haciendo una simplificación estadística del problema, es decir, modificando el algoritmo de forma que todos los núcleos ejecuten el mismo código exactamente (sin saltos condicionales); o utilizando CUDA a bajo nivel con el fin de optimizar el uso de los recursos disponibles en la GPU.

Los resultados se han obtenido utilizando 2^N bloques de hilos y 2^M hilos por bloque (con $N \in [0, 10]$ y $M \in [0, 8]$), con el objetivo de visualizar los efectos de la gestión de hilos en la GPU.

El tiempo de carga y descarga de datos en la GPU medido ha sido de $3ms$ aproximadamente, por lo que el tiempo de ejecución del kernel y el tiempo de ejecución del algoritmo completo son aproximadamente iguales.

La eficiencia de un código paralelo se mide en relación al número de procesadores activos durante la ejecución. Ya que se ha empleado el compilador cruzado de CUDA C de NVidia (NVCC), es complicado, si no imposible conocer el número de procesadores activos durante la ejecución del código paralelo. Por ello, para ofrecer una estimación de la eficiencia del código paralelizado mediante GPU, se ha supuesto que en todos los casos, el número de núcleos que intervienen en el cálculo es el máximo diponible (448), enmarcando el problema en el peor caso.

A continuación se muestran los resultados obtenidos gráficamente.

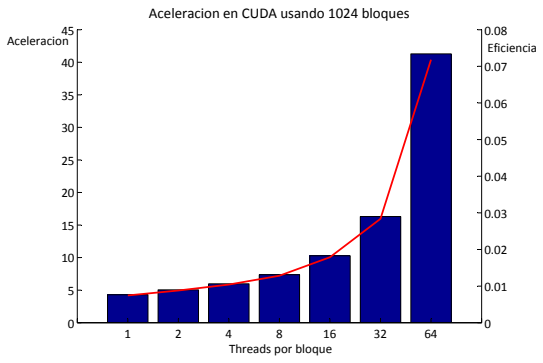

Figura 49: Aceleración y eficiencia para CUDA empleando 1024 bloques

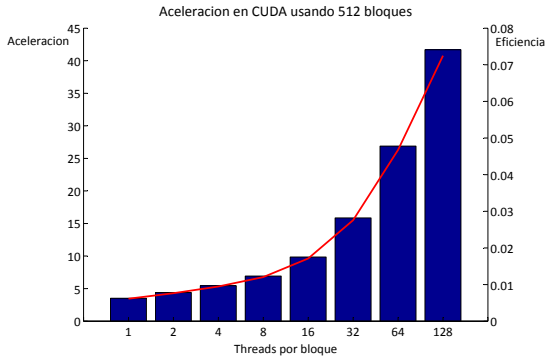

Figura 50: Aceleración y eficiencia para CUDA empleando 512 bloques

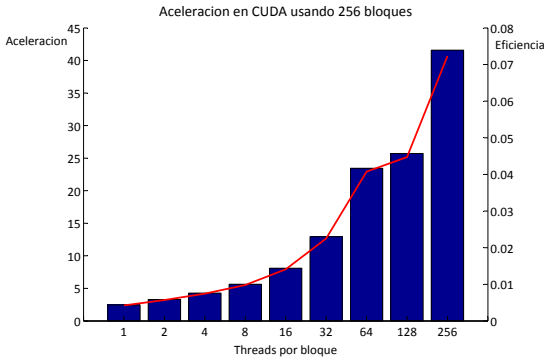

Figura 51: Aceleración y eficiencia para CUDA empleando 256 bloques

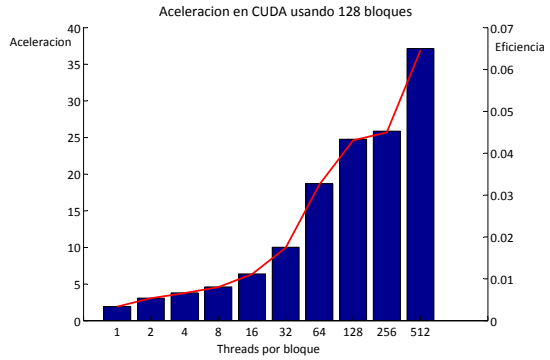

Figura 52: Aceleración y eficiencia para CUDA empleando 128 bloques

Tal y como se comentó, la eficiencia que se obtiene es muy pequeña, pero la aceleración que se consigue llega más allá de 40, esto implica que una simulación de 1 minuto pasa a ser de un

segundo y medio. El fin de este TFM es obtener un aumento de velocidad significativo en la simulación del canal, ya que se pretende obtener medias y varianzas en el futuro, lo cual implica realizar cientos de realizaciones del mismo experimento, por lo que la baja eficiencia es un aspecto secundario pero mejorable.

Se observa que a medida que el número de bloques disminuye, la eficiencia del simulador aumenta. Este efecto se debe a que el sistema de gestión de la GPU tiene menor carga, ya que se encarga de gestionar un menor número de bloques de hilos.

A continuación se muestran los demás resultados obtenidos.

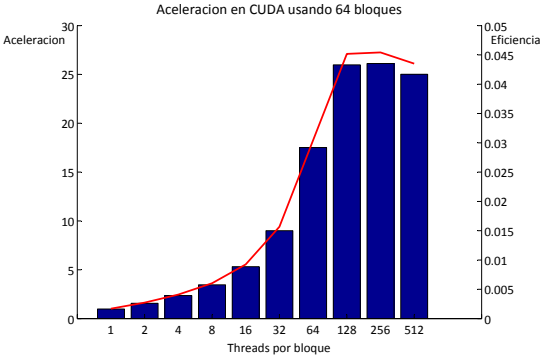

Figura 53: Aceleración y eficiencia para CUDA empleando 64 bloques

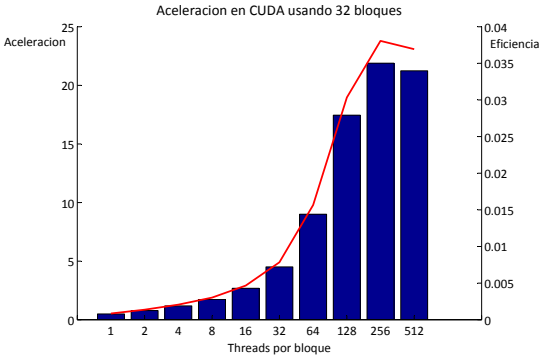

Figura 54: Aceleración y eficiencia para CUDA empleando 32 bloques

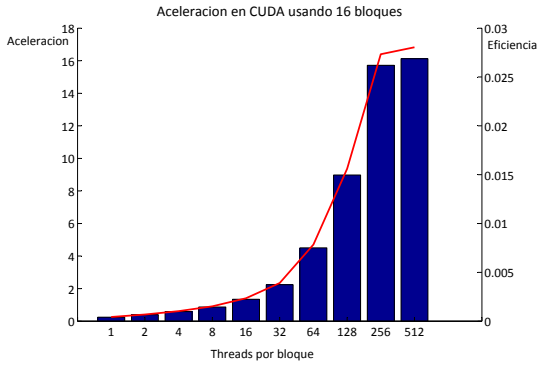

Figura 55: Aceleración y eficiencia para CUDA empleando 16 bloques

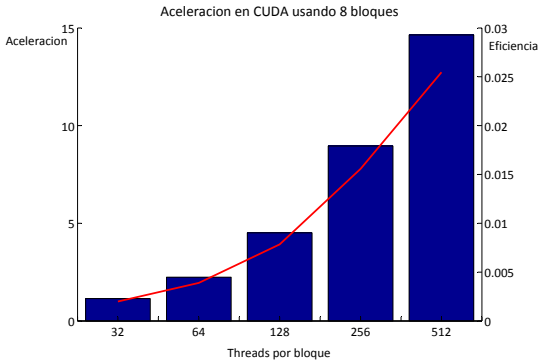

Figura 56: Aceleración y eficiencia para CUDA empleando 8 bloques

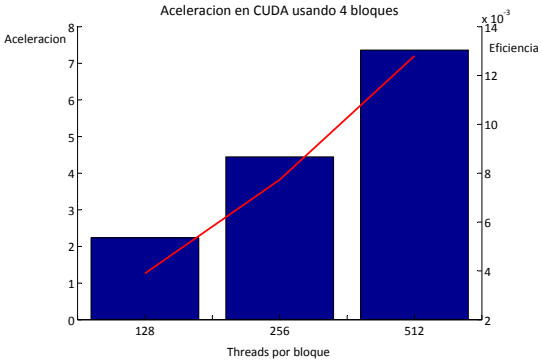

Figura 57: Aceleración y eficiencia para CUDA empleando 4 bloques

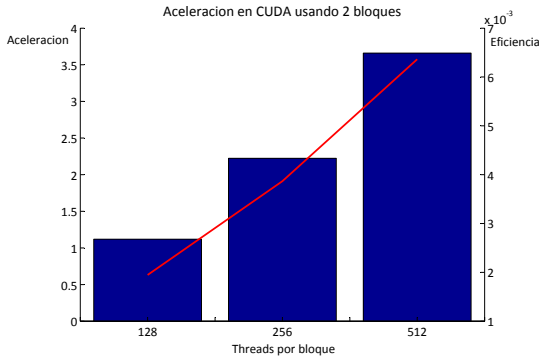

Figura 58: Aceleración y eficiencia para CUDA empleando 2 bloques

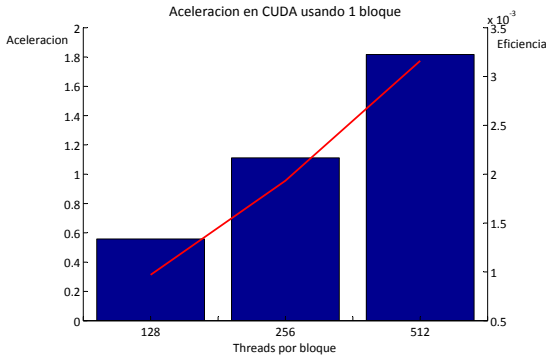

Figura 59: Aceleración y eficiencia para CUDA empleando 1 bloque

Estos últimos resultados ponen de manifiesto aún más la baja eficiencia del código en referencia al modelo SIMT de ejecución de la GPU. En el caso extremo de 1 único bloque de hilos de procesamiento, hasta que el número de hilos no llega a 256, el algoritmo se vuelve más lento en lugar de aumentar la velocidad.

A pesar de que el código es ineficiente en relación al modelo de computación que utiliza la GPU, se ha obtenido una aceleración bastante superior a la obtenida con OpenMP. Además, hablando en términos de eficiencia económica, la relación entre la aceleración obtenida con la GPU y su coste, es mucho mayor que la hipotética relación que se obtendría utilizando M procesadores como los empleados en este TFM.

Tecnología	Aceleración por unidad	Coste (K$)	Acc./Coste
Tesla M2050	40	2	20
2 Xeon (OpenMP)	4	0.8	5

Cuadro 9: Relación aceleración-coste

De manera cuantitativa, suponiendo incluso que la aceleración incrementase linealmente tomando como base los resultados obtenidos con OpenMP, para obtener una aceleración de 40,

serian necesarios aproximadamente 20 procesadores Xeon. La eficiencia económica de una solución de tipo GPU es aproximadamente 4 veces superior a la relativa a una solución basada en OpenMP.

9. Conclusiones

En este TFM se ha implementado un simulador de canal óptico submarino para la respuesta al impulso. Además, se ha llevado a cabo la paralelización del mismo con el fin de hacer viable la realización de experimentos que impliquen gran cantidad de realizaciones.

Se ha puesto de manifiesto que la paralelización con GPU es una estrategia de paralelización cuya eficiencia viene determinada por la correlación de las acciones que deben realizar los hilos que componen la ejecución, sin embago, los resultados absolutos de aceleración hacen que la solución propuesta sea muy efectiva, pasando de ejecuciones de 1 minuto a ejecuciones de 1.5 segundos.

Haciendo una comparación entre las dos técnicas de paralelización empleadas, OpenMP llega a presentar eficiencias cercanas a la unidad, mientras que CUDA presenta eficiencias muy pobres. Sin embargo, dada la densidad de núcleos de procesamiento presentes en la GPU, CUDA obtiene aceleraciones muy superiores a OpenMP. Además, en lo referente a la eficiencia económica, se demuestra que la solución basada en GPU es 4 veces superior a la solución basada en OpenMP.

Aunque la eficiencia empleando CUDA sea pequeña, existe un margen de mejora que abre líneas de investigación en este aspecto, trabajando sobre el algoritmo a nivel conceptual y a nivel de implementación.

El código secuencial desarrollado se ha implementado tanto de modo iterativo como recursivo. A pesar de que el código recursivo es más intuitivo y rápido de programar, es más lento a la hora de ejecutarse, por lo que los recorridos iterativos en preorden a través de estructuras tipo árbol son más rápidos.

Respecto de los resultados de simulación de canal, no existen aún medidas empíricas con las que compararlos, aunque se advierten prometedores. Se ha observado que los cuatro parámetros que influyen en mayor medida en la dispersión del retardo en este tipo de comunicaciones son: el tipo de partículas, la concentración de dichas partículas, la distancia entre emisor y receptor, y el tipo de emisor.

El tipo de partícula influye en la cantidad de potencia se dispersa tras una colisión, siendo inversamente proporcional. La concentración define la distancia media de colisión, introduciendo mayor dispersión a mayor concentración. La distancia influye directamente en la dispersión del retardo ya que a mayor distancia, mayor número de colisiones posibles. Y por último, si aumenta el índice del emisor lambertiano considerado, la potencia transmitida se concentra en un ancho de haz menor, por lo que la variación de retardos tiende a reducirse.

En los escenarios simulados, la dispersión de retardo máxima obtenida ha sido de 30 picosegundos, pero posiblemente pueda llegar a rondar el nanosegundo aumentando la concentración de partículas y reduciendo el radio de las mismas. Esto implica que en algunos escenarios podria existir una limitación de ancho de banda acotada a partir de 1 GHz, que para las comunicaciones ópticas inalámbricas puede entenderse como infinito.

El efecto de las particulas absorbentes (grandes partículas que no dejan pasar la luz), será el de producir centelleo en el receptor, es decir, desvanecimientos de potencia cortos y muy frecuentes, por lo que los resultados obtenidos pueden servir para ser introducidos en un simulador de centelleo.

Existen otros efectos que no han sido considerados en este TFM y que pueden influir notablemente en las comunicaciones, como por ejemplo las turbulencias en escenarios de emisión tipo láser. El efecto que se describe en comunicaciones atmosféricas es el de un centelleo en el receptor, y a priori, podría introducirse en la simulación de canal submarino a través de alguns ajustes.

10. Líneas futuras

El desarrollo de este TFM abre nuevas líneas de investigación apoyándose sobre los resultados obtenidos. Se ha observado que en lo relativo a la paralelización del algoritmo sobre GPU, existe un margen de mejora grande, pues la eficiencia obtenida pone de manifiesto que los recursos hardware no estan siendo aprovechados.

Este trabajo sugiere la posibilidad de modificar el paradigma de simulación y aprovechar las propiedades estadísticas de las colisiones que se generan durante la propagación de los rayos, con el fin de generar un algoritmo que sea capaz de ajustarse a la filosofía SIMT de la GPU.

Otro ámbito de actuación es la programación del algoritmo, ya que puede optimizarse el uso de los recursos de la GPU si se interviene directamente a nivel de procesador, es decir, no utilizando el compilador NVCC y programar directamente a nivel físico.

En lo referente a la simulación de canal, el resultado de este TFM permite generar gran cantidad de realizaciones de un experimento en mucho menor tiempo que en el caso del código secuencial. El objetivo último del simulador es obtener respuestas medias y varianzas.

Otro tipo de efectos no tenidos en cuenta en este trabajo puede ser incluído en versiones posteriores, como por ejemplo las turbulencias, el paso de fauna marina entre emisor y receptor o el fauling óptico.

Por último, el modelo matemático propuesto deberá ser validado con medidas experimentales, realimentando los resultados en la etapa de diseño a fin de corregir el modelo para ajustarlo a la realidad.

11. Referencias

[1] D. Anguita, D. Brizzolara, G. Parodi, and Q. Hu, "Optical wireless underwater communication for auv: Preliminary simulation and experimental results," in *OCEANS, 2011 IEEE - Spain*, pp. 1 –5, june 2011.

[2] J. Smart, "Underwater optical communications systems part 1: variability of water optical parameters," in *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pp. 1140 –1146 Vol. 2, oct. 2005.

[3] J. Simpson, B. Hughes, and J. Muth, "A spatial diversity system to measure optical fading in an underwater communications channel," in *OCEANS 2009, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges*, pp. 1 –6, oct. 2009.

[4] S. Jaruwatanadilok, "Underwater wireless optical communication channel modeling and performance evaluation using vector radiative transfer theory," *Selected Areas in Communications, IEEE Journal on*, vol. 26, pp. 1620 –1627, december 2008.

[5] A. Jurado, "Enlaces ópticos no guiados con técnicas de diversidad en canal atmosférico afectado por turbulencias," *[TESIS DOCTORAL] Universidad de Málaga*, September 2010.

[6] S. Meihong, Y. Xinsheng, and Z. Zhangguo, "The modified ppm modulation for underwater wireless optical communication," in *Communication Software and Networks, 2009. ICCSN '09. International Conference on*, pp. 173 –177, feb. 2009.

[7] B. Cochenour, L. Mullen, and A. Laux, "Phase coherent digital communications for wireless optical links in turbid underwater environments," in *OCEANS 2007*, pp. 1 –5, 29 2007-oct. 4 2007.

[8] M. Sui, X. Yu, and F. Zhang, "The evaluation of modulation techniques for underwater wireless optical communications," in *Communication Software and Networks, 2009. ICCSN '09. International Conference on*, pp. 138 –142, feb. 2009.

[9] A. Li, W. Shieh, and R. Tucker, "Wavelet packet transform-based ofdm for optical communications," *Lightwave Technology, Journal of*, vol. 28, pp. 3519 –3528, dec.15, 2010.

[10] J. Simpson, W. Cox, J. Krier, B. Cochenour, B. Hughes, and J. Muth, "5 mbps optical wireless communication with error correction coding for underwater sensor nodes," in *OCEANS 2010*, pp. 1 –4, sept. 2010.

[11] S. P. Webpage, "Underwater optical communications," *http://www.saphotonics.com/high-bandwidth-opticalcommunications/underwater/*.

[12] J. Barry, J. Kahn, W. Krause, E. Lee, and D. Messerschmitt, "Simulation of multipath impulse response for indoor wireless optical channels," *Selected Areas in Communications, IEEE Journal on*, vol. 11, pp. 367 –379, apr 1993.

[13] J. Lopez-Hernandez, R. Perez-Jimenez, and A. Santamaria, "Modified monte carlo scheme for high-efficiency simulation of the impulse response on diffuse ir wireless indoor channels," *Electronics Letters*, vol. 34, pp. 1819 –1820, sep 1998.

[14] O. Gonzalez, C. Militello, S. Rodriguez, R. Prez-Jimenez, and A. Ayala, "Error estimation of the impulse response on diffuse wireless infrared indoor channels using a monte carlo ray-tracing algorithm," *Optoelectronics, IEE Proceedings* -, vol. 149, pp. 222 – 227, oct/dec 2002.

[15] D. Goren, R. Gordin, S. Slafman, and R. Carmon, "The closed environment concept in vlsi on-chip transmission lines design and modeling," in *Signal Propagation on Interconnects, 2006. IEEE Workshop on*, pp. 201 –204, may 2006.

[16] P. Patton, "Multiprocessors: Architecture and applications," *Computer*, vol. 18, pp. 29 –40, june 1985.

[17] AMD, "Amd technologies." `http://www.amd.com/es/products/technologies/Pages/technologies.aspx`. Consultado el 2 de Julio de 2012.

[18] Intel, "Multiprocessor specification." `http://www.intel.com/design/archives/processors/pro/docs/242016.htm`. Consultado el 2 de Julio de 2012.

[19] NVidia, "Nvidia tesla architecture." `http://www.nvidia.com/object/tesla\_tech\_specs.html`. Consultado el 5 de Julio de 2012.

[20] NVidia, "Nvidia fermi architecture." `http://www.nvidia.com/object/fermi-architecture.html`. Consultado el 5 de Julio de 2012.

[21] K. Rocki and R. Suda, "Large-scale parallel monte carlo tree search on gpu," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pp. 2034 –2037, may 2011.

[22] P. Vidal and E. Alba, "A multi-gpu implementation of a cellular genetic algorithm," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pp. 1 –7, july 2010.

[23] B. Zhou, X. Hu, and D. Chen, "Memory-efficient volume ray tracing on gpu for radiotherapy," in *Application Specific Processors (SASP), 2011 IEEE 9th Symposium on*, pp. 46 –51, june 2011.

[24] OpenMP, "Openmp project." `http://www.openmp.org`. Consultado el 10 de Julio de 2012.

[25] NVidia, "Nvidia cuda 4.0 toolkit." `http://developer.nvidia.com/cuda-toolkit`. Consultado el 10 de Julio de 2012.

[26] NVidia, "Nvidia tesla m2050 specification." `http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_M2050_M2070_Apr10_LowRes.pdf`. Consultado el 10 de Julio de 2012.

[27] H. K. Nagesh B. Lakshminarayana, "Effect of instruction fetch and memory scheduling on gpu performance," *Georgia Institute of Technology*, 2011. Consultado el 17 de Julio de 2012.

[28] B. Wozniak and J. Dera in *Light Absorption in Sea Water*, vol. VIII, pp. 11 – 80, 2007.

[29] W. Matthäus in *Empirische Gleichungen für den Brechungsindex des Meerwassers*, no. 33, 1975.

[30] G. T. McNeil, "Metrical fundamentals of underwater lens system," in *Opt. Eng*, no. 16, pp. 128 – 139, 1977.

[31] C. Mätzler, "Mie scattering with and without difraction," in *Research Report, Institute of Applied Physics, University of Bern*, no. 2, 2004.

[32] C. Liang and Z. Zhuang, "Symbolic computation and construction of exact traveling wave solutions to the variant boussinesq equations," in *Advanced Computer Control (ICACC), 2011 3rd International Conference on*, pp. 570 –573, jan. 2011.

[33] K. Lee, H. Park, and J. Barry, "Indoor channel characteristics for visible light communications," *Communications Letters, IEEE*, vol. 15, pp. 217 –219, february 2011.

[34] A. Glassner, "Situation normal [gourand and phong shading]," *Computer Graphics and Applications, IEEE*, vol. 17, pp. 83 –87, mar/apr 1997.

[35] N. Tanaka, S. Tominaga, and T. Kawai, "Estimation of the torrance-sparrow reflection model from a single multi-band image," in *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, vol. 3, pp. 596 –599 vol.3, 2000.

[36] S. Ozan and S. Gumustekin, "Efficient representation of measured brdf data using lafortune model," in *Signal Processing and Communications Applications Conference (SIU), 2010 IEEE 18th*, pp. 684 –687, april 2010.

[37] G. Code, "Bohren-huffman scattering calculatin algorithm source code." `http://code.google.com/p/scatterlib/wiki/Spheres`. Consultado el 1 de Junio de 2012.

[38] G. Marsaglia, "Random numbers fall mainly in the planes," in *National Academy of Science, Proceedings of the*, vol. 61, pp. 25 – 28, 1968.