UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Modelado de Calor utilizando Ray Tracing

Liber Dovat

Marcelo Gancio

Tutores:

Eduardo Fernández

José Pedro Aguerre

Proyecto de Grado en Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Noviembre de 2019

RESUMEN

Recientemente, diversos trabajos científicos han demostrado la viabilidad del uso de la técnica de ray tracing para modelar la transferencia de calor dentro de un cuerpo. La integración de esta técnica con algoritmos de aceleración (por ejemplo, estructuras jerárquicas) y procesadores altamente paralelizables (GPU) permite reducir considerablemente el tiempo de cálculo.

En el presente trabajo se toma un caso de estudio presentado por Caliot *et al.* (2018) que calcula la distribución de calor en una pared de caras sólidas e interior poroso, considerando sólo conducción y radiación en régimen estacionario. Dicho caso de estudio utiliza Monte Carlo y ray tracing para simular el movimiento de partículas "portadoras de calor", para calcular temperaturas a intervalos regulares dentro de la pared.

Se desarrollaron dos variantes del algoritmo que permiten resolver el problema estudiado. La primera corresponde a la implementación en GPU del algoritmo presentado en el caso de estudio, y la segunda es una extensión de la primera en donde se reutiliza la información intermedia generada por caminos aleatorios. Ambas variantes se implementaron utilizando la biblioteca de ray tracing OptiX, que permite enfocarse en la solución del problema sin preocuparse por los detalles de implementación sobre GPU.

Los resultados obtenidos con la primera variante son comparables en precisión a los presentados en la propuesta de Caliot *et al.* (2018). Con la segunda variante se obtuvieron mejoras tanto en el tiempo de cómputo como en el número de operaciones en comparación con la primera.

Se puede acceder al código fuente a través del enlace:
https://github.com/liber-dovat/proyecto_calor_raytracing

Palabras claves:
Método de Monte Carlo, Medio poroso, Geometría compleja, Ray Tracing, Transferencia de calor conductiva y radiativa, GPU, Optix.

Tabla de contenidos

Capítulo 1

Introducción

El modelado de la distribución de calor dentro de un cuerpo es analíticamente complicado, ya que implica resolver la ecuación del calor (ecuación diferencial de segundo orden) para dicho cuerpo y su interacción con el ambiente en el que se encuentra. En este contexto, se han desarrollado diferentes métodos computacionales para estimar la transferencia de calor en geometrías complejas. La principal estrategia utilizada corresponde a subdividir la geometría en varias regiones discretas y estimar la temperatura de cada una de ellas. Para determinar dichas temperaturas, se resuelve un sistema de ecuaciones con tantas variables como regiones de la geometría.

Una de las principales desventajas de los métodos *discretos* es que requieren subdividir la geometría en elementos más pequeños; en esta subdivisión se pueden llegar a perder ciertos detalles de la estructura original y aumenta el consumo de memoria debido al incremento en el número de variables y ecuaciones. Por eso surge el interés de utilizar técnicas basadas en el método de Monte Carlo que permiten trabajar directamente con la geometría original. El método de Monte Carlo utiliza variables aleatorias combinadas con aspectos geométricos simples (por ejemplo, el cálculo de trayectorias rectilíneas y colisiones) para estimar soluciones numéricas que modelan fenómenos complejos (Kroese *et al.* (2014)).

Por otra parte, desde el punto de vista computacional, dos avances significativos sugieren que los métodos basados en Monte Carlo pueden acelerarse significativamente:

1. En computación gráfica se ha desarrollado una técnica llamada ray tracing, que es comúnmente utilizada para generar imágenes simulando el

comportamiento de los rayos de luz en una escena. La idea es seguir la trayectoria de los rayos registrando dónde chocan con los objetos de la escena a representar (Pharr *et al.* (2018)). Esta técnica ha sido objeto de estudio y optimización algorítmica por más de tres décadas.

2. En los últimos años se han desarrollado unidades de cálculo en paralelo programables llamadas GPUs, que permiten un aumento considerable de rendimiento y ahorro de tiempo en algoritmos paralelizables.

El presente trabajo consiste en implementar uno de los algoritmos publicados recientemente para el cálculo de intercambio de calor basado en dichas técnicas. Para su implementación se utilizarán bibliotecas gráficas especializadas en ray tracing y otras herramientas basadas en la arquitectura de las GPUs y en conceptos de computación gráfica.

1.1. Organización del documento

Este documento se organiza de la siguiente manera: En el capítulo 2 se presentan conceptos preliminares, el estado del arte en este campo junto con el caso de estudio seleccionado y se analizan las herramientas factibles para realizar los cálculos necesarios. En el capítulo 3 se presenta la implementación realizada, en el capítulo 4 se muestran los resultados obtenidos, y por último el capítulo 5 presenta las conclusiones y las líneas de trabajo futuras.

Capítulo 2

Estado del arte

En este capítulo se detallan los conceptos, técnicas y trabajos relacionados con este proyecto. Comenzando con algunos conceptos preliminares sobre el modelado de calor y de técnicas de computación aplicables a dicho modelado. Luego se presenta un breve resumen de los trabajos relacionados y se realiza un análisis del trabajo seleccionado. De igual manera se hace un resumen de las herramientas disponibles para la implementación, y por último se presenta una descripción de la herramienta seleccionada.

2.1. Conceptos preliminares

Para simular el comportamiento del calor en un cuerpo, primero se deben entender los fundamentos físicos de la transferencia de calor, y luego conocer técnicas que permitan implementar una simulación de dicho fenómeno en una computadora. Para ello, se repasan los fundamentos del modelado del calor, técnicas de simulación del comportamiento de la luz y se presenta un método estadístico aplicable a dicho modelado.

2.1.1. Transferencia de calor

La transferencia de calor es un fenómeno muy estudiado debido a su importancia en muchas ramas de la ingeniería. Estudia el transporte de energía entre materiales debido a la diferencia de temperatura entre ellos (Lewis *et al.* (2004)).

Se definen tres modos de transferencia de calor (Bejan (1993), Incropera *et al.* (2006)): conducción, convección y radiación.

El modo de transporte de conducción ocurre debido al intercambio de energía entre moléculas. Este modo de transporte depende de las propiedades del medio y ocurre en sólidos, líquidos y gases si existe una diferencia de temperatura.

Las moléculas presentes en líquidos y gases tienen libertad de movimiento, y al moverse de una región caliente a una fría transportan energía con ellas. La transferencia de calor debido al movimiento macroscópico del líquido o gas, junto con la energía transportada por conducción dentro del fluido se llama *transferencia de calor* por convección.

Todos los cuerpos a una temperatura por encima del cero absoluto emiten radiación térmica en forma de ondas electromagnéticas. Éste es el único modo que no necesita un medio material para que ocurra la transferencia de calor. La naturaleza de la radiación térmica es tal que la propagación de energía se emite desde la frontera del cuerpo. Cuando estas ondas electromagnéticas llegan a otro cuerpo, parte de la onda es reflejada, otra parte es transmitida, y la restante es absorbida.

Las leyes de la transferencia de calor

Para cuantificar la cantidad de energía transferida por unidad de tiempo se utilizan ecuaciones basadas en el flujo de calor en el cuerpo.

Para la conducción de calor, la ecuación es conocida como la *Ley de Fourier* que expresada en una dimensión es

$$q_x = -k\frac{dT}{dx}$$

en donde q_x es el flujo de calor en la dirección x, (W/m^2); k es la conductividad térmica, (W/mK) la cual es una propiedad del material, y dT/dx es el gradiente de temperatura, (K/m).

Para la convección de calor se utiliza la *Ley del enfriamiento de Newton* que se expresa como

$$q = h(T_W - T_a)$$

siendo q el flujo de calor convectivo, (W/m^2); $(T_W - T_a)$ es la diferencia de temperatura entre el muro y el fluido, y h es el coeficiente de transferencia de calor de convección, (W/m^2K).

El flujo emitido por radiación desde la superficie de un cuerpo negro (objeto teórico o ideal que absorbe toda la luz y toda la energía radiante que incide sobre él) está determinado por la *Ley de Stefan–Boltzmann*

$$q = \sigma T_W^4$$

siendo q el flujo de calor radiativo, $(^W/_{m^2})$; σ la constante de Stefan–Boltzmann, $(\approx 5,670373 \times 10^{-8}\ ^W/_{m^2K^4})$, y T_W es la temperatura en la superficie, (K).

El flujo emitido por una superficie gris difusa es menor que la de una superficie negra, y está determinado por la ecuación

$$q = \varepsilon \sigma T_W^4$$

siendo ε el coeficiente de emisividad de la superficie. El intercambio neto de energía radiativa entre dos superficies 1 y 2 se calcula como

$$Q = F_\varepsilon F_G \sigma A_1 (T_1^4 - T_2^4)$$

en donde F_ε es un factor que tiene en cuenta la naturaleza de las superficies; F_G es un factor que tiene en cuenta la orientación geométrica de las superficies, y A_1 es el área de la superficie 1.

Discretización de las ecuaciones

Las ecuaciones de transferencia de calor son muy útiles para resolver un gran número de problemas, pero si el objeto y las condiciones de borde son muy complejas es muy difícil conseguir una solución analítica. Para estas situaciones han surgido métodos numéricos que permiten realizar aproximaciones de las soluciones. Mientras que una solución analítica permite calcular la temperatura en cualquier parte del sólido, una solución numérica permite calcular la temperatura solamente en puntos discretos. Esto se realiza dividiendo la región de interés en cierto número de regiones más chicas. Cuando se calcula la temperatura en cada una de las regiones en realidad se obtiene una distribución de temperatura en cada región. Los métodos más empleados son Diferencias finitas, Volúmenes finitos y Elementos finitos.

Existen muchas técnicas para realizar simulaciones de un fenómeno físico no

visible como el cálculo de distribución de la radiación térmica, y es de especial interés la aplicación de técnicas de iluminación global provenientes de la rama de la computación gráfica. Éstas simulan el comportamiento de la luz en la escena, permitiendo tomar en cuenta tanto su emisión como su interacción con los objetos. Dentro de ellas se destaca el ray tracing por ser muy versátil, y adaptable para resolver un gran número de problemas.

2.1.2. Emisión lambertiana

Al modelar un emisor puntual de luz la cantidad de energía emitida depende del ángulo desde donde se ve el emisor; o lo que es lo mismo, del ángulo del vector de salida de la luz desde el emisor, siendo máximo en la dirección normal a la superficie y nulo en la dirección paralela a la superficie.

Como se muestra en la figura 2.1 si consideramos un emisor puntual A la intensidad de luz emitida es proporcional al coseno del ángulo entre el vector de salida y el vector normal a la superficie. Debido a esto una dirección al azar tomada uniformemente se debe ajustar según el coseno de dicho ángulo para lograr una dirección realista del vector de salida de la luz (Pharr *et al.* (2018)).

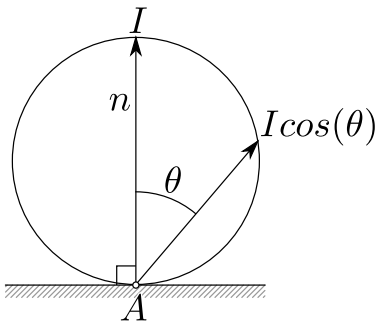


Figura 2.1: Esquema que muestra como la intensidad de luz emitida por un punto A en una dirección diferente a la normal es proporcional al coseno del ángulo que forma con ella.

Esto significa que la distribución de probabilidad del ángulo de salida de un rayo también debe seguir una distribución coseno.

Una vez resuelto el problema de la emisión de la luz, se debe resolver el comportamiento de esos rayos de luz en el ambiente y su interacción con los objetos.

2.1.3. Ray tracing

Como se ve en la figura 2.2, el algoritmo de ray tracing es en realidad muy simple; Se basa en seguir el camino de un rayo de luz a través de una escena, mientras rebota e interactúa con los objetos y el entorno (Pharr *et al.* (2018)). A pesar de que hay muchas formas de escribir un Ray Tracer, todos estos sistemas simulan al menos lo siguientes objetos y fenómenos (Foley *et al.* (1996)):

Cámaras: ¿Cómo y desde dónde se ve la escena? Las cámaras generan rayos desde el punto de vista hacia la escena.

Intersecciones objeto–rayo: Se debe poder decir con precisión dónde interseca un rayo con un objeto geométrico dado. Además, necesitamos determinar ciertas propiedades geométricas del objeto en el punto de intersección, como una superficie normal o su material. La mayoría de los Ray Tracer tienen alguna facilidad para encontrar múltiples intersecciones de los rayos con los objetos de una escena, devolviendo además la intersección más cercana a lo largo de un rayo.

Distribución de la luz: Sin la iluminación, tendría poco sentido representar una escena. Un Ray Tracer debe modelar la distribución de la luz a lo largo de la escena, incluyendo no sólo las ubicaciones de las luces en sí, pero también la forma en que distribuyen su energía a lo largo del espacio.

Visibilidad: Para saber si una luz dada deposita energía en un punto en una superficie, debemos saber si hay un camino ininterrumpido desde el punto hasta la fuente de luz. Afortunadamente, esta pregunta es fácil de responder en un Ray Tracer, ya que simplemente puede construir el rayo de la superficie a la luz, encontrar la intersección del objeto de rayo más cercano, y comparar la distancia de intersección a la distancia de la luz.

Dispersión de la superficie: Cada objeto debe proporcionar una descripción de su apariencia, incluyendo información sobre cómo la luz interactúa con la superficie del objeto, así como la naturaleza de la luz redirigida (o dispersada). Por lo general interesan las propiedades de la luz que se dispersa directamente hacia la cámara. Los modelos para la dispersión de superficie suelen ser parametrizables para que puedan simular una variedad de apariencias.

Ray tracing recursivo: Debido a que la luz puede llegar a una superficie

después de rebotar o de pasar a través de varias otras superficies, generalmente es necesario trazar rayos adicionales que se originen en la superficie para capturar completamente este efecto. Esto es particularmente importante para superficies brillantes como metal o vidrio.

Propagación del rayo: Se necesita saber qué sucede con la luz que viaja a lo largo de un rayo mientras pasa por el espacio. Si estamos renderizando una escena en el vacío, la energía luminosa permanece constante a lo largo de un rayo. Aunque la mayoría de las escenas modeladas no están en el vacío, este es el supuesto típico hecho por la mayoría de los Ray Tracer. Modelos más avanzados contemplan el transporte de la luz en medios participativos como la niebla, el humo y la atmósfera de la Tierra, entre otros.

En la figura 2.2 se puede ver un esquema del comportamiento de los rayos. Cuando un rayo llega a una superficie, puede ser reflejado o transmitido; entonces nuevos rayos son generados recursivamente y se repite el proceso hasta que se llega a un número de recursiones prefijado. Si no llega a ninguna superficie, el rayo toma el color dado para el ambiente y regresa en la recursión.

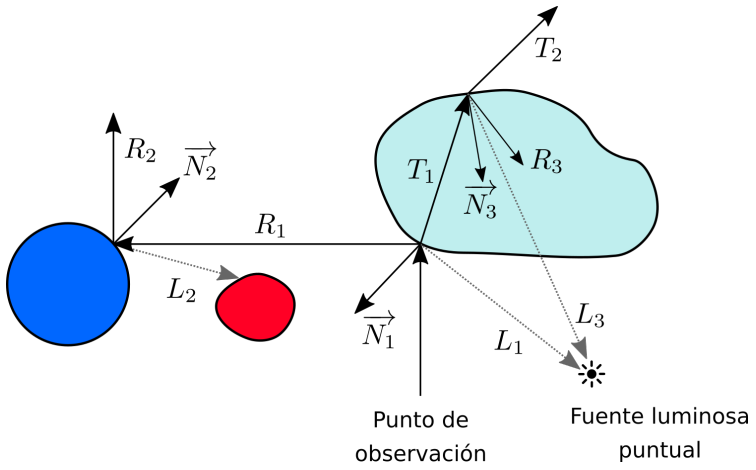


Figura 2.2: Esquema del comportamiento de los rayos. En la figura los rayos se corresponden de la siguiente manera: $\overrightarrow{N_i}$ Normal a la superficie, R_i Rayo reflejado, L_i Rayo de sombra y T_i Rayo transmitido. Esquema tomado de Foley *et al.* (1996) *Figura 14.35.*

La técnica de ray tracing y el modelado de transferencia de calor pueden ser mejorados si se repite el muestreo de los datos un gran número de veces. Uno de los métodos más utilizados para integrar los datos es el método de

Monte Carlo.

2.1.4. Método de Monte Carlo

El método de Monte Carlo (MC) es en esencia la generación de objetos o procesos aleatorios utilizando una computadora. Estos objetos pueden surgir del modelado de un sistema real; sin embargo, en muchos casos son creados de forma artificial para poder resolver problemas determinísticos. En cualquier caso, la idea es repetir el muestreo suficientes veces para obtener valores estadísticamente confiables mediante el uso de la ley de grandes números y otros métodos de inferencia estadística (Kroese *et al.* (2014)).

Usos típicos

Se utiliza para estudiar un sistema físico, ya sea tomando muestras aleatorias del modelo del mismo o estimando valores relacionados a través de variables aleatorias. También se utiliza para estudiar posibles optimizaciones de funciones complejas.

Ventajas

Los algoritmos de MC tienden a ser simples, flexibles, escalables y paralelizables. Pueden reducir la complejidad del análisis y de la simulación de sistemas físicos.

Áreas de aplicación

Este método se utiliza para realizar simulaciones de modelos, aplicaciones a problemas de investigación operativa, simulaciones de sistemas físicos (por ejemplo, transporte de partículas), análisis financieros y análisis estadísticos de big data, entre otros.

2.2. Trabajos relacionados sobre simulación de calor usando Monte Carlo Ray Tracing

En los últimos años, ha tomado relevancia el uso del método de Monte Carlo (MC) junto con ray tracing (RT) para el cálculo de simulaciones de transferencia de calor (Csébfalvi (1997)). Dichos métodos (MCRT) se han combinado con

otros, por ejemplo, la utilización de volúmenes finitos en los trabajos de Ravishankar *et al.* (2010), Soucasse *et al.* (2012) y Kuczynski and Bialecki (2014) además de radiosidad en Kramer *et al.* (2015). La versatilidad del MCRT ha permitido aplicarlo a variadas situaciones, como al modelado del flujo de calor dentro de un medio con fibras estudiado por Arambakam *et al.* (2012), modelos con medios porosos como en los trabajos de Vignoles (2015) y Rong *et al.* (2013), e incluso con geometría compleja como en Caliot *et al.* (2018). También se ha comparado el MCRT con otras técnicas como se puede ver en Jacques *et al.* (2015) y Caliot *et al.* (2018) dando muy buenos resultados.

Otro aspecto a considerar en MCRT es la dificultad del modelado de la convección, lo que ha implicado en general el uso de modelos que sólo toman radiación y conducción, analizando los efectos de acople como se estudian en Perraudin and Haussener (2017). También ha habido intentos de realizar estos cálculos agregando convección como se puede ver en Fournier *et al.* (2016).

Por su naturaleza, el MCRT es factible de paralelizar para mejorar los resultados, por ejemplo, utilizando MPI como estudian Duarte Santos and Lani (2016), y también la aplicación de GPU para su cálculo como se ha utilizado en los trabajos de Kramer *et al.* (2015), Rong *et al.* (2013), Szénási and Felde (2016) e incluso utilizando múltiples GPU como demuestran Peterson *et al.* (2018).

Para este proyecto se ha tomado como caso de estudio el trabajo de Caliot *et al.* (2018) por tener un nivel descriptivo lo suficientemente detallado para lograr su implementación usando GPU y porque presenta resultados que permiten una fácil comparación utilizando un enfoque teórico acotado.

2.3. Trabajo de Caliot *et al.* (2018)

En Caliot *et al.* (2018) se estudia la transferencia de calor en una pared infinita, obteniendo la temperatura de la misma en un estado estacionario. Cada lado de la pared se encuentra a una temperatura fija, las cuales llamaremos T_{min} y T_{max}.

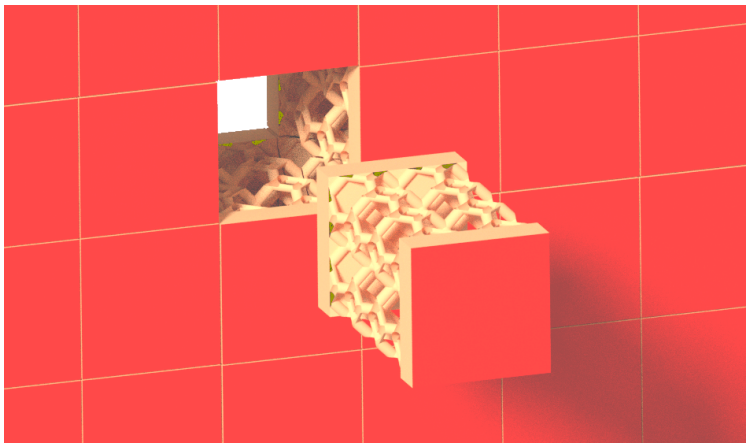


Figura 2.3: Diagrama de la pared y una sección.

2.3.1. Geometría

La pared se compone de tres capas (figura 2.3):

- Dos secciones sólidas, con uno de sus lados expuesto a una temperatura impuesta.
- El núcleo de la pared. Es una zona porosa que conecta las secciones sólidas mediante los tubos sólidos de celdas Kelvin. La celda Kelvin es un poliedro que permite llenar el espacio completamente mediante la repetición de sí mismo (similar a un cubo).

Las secciones sólidas junto con los tubos sólidos de las celdas Kelvin conforman la parte sólida de la pared. El espacio entre los tubos de las celdas Kelvin y a ambos lados de la pared se considera al vacío. Por lo tanto, no hay ningún medio participativo y no existe el efecto de convección. Para estudiar la transferencia de calor de la pared se la considera formada por secciones idénticas en forma y comportamiento (figura 2.4). Cada sección tiene un largo y alto de 8 mm

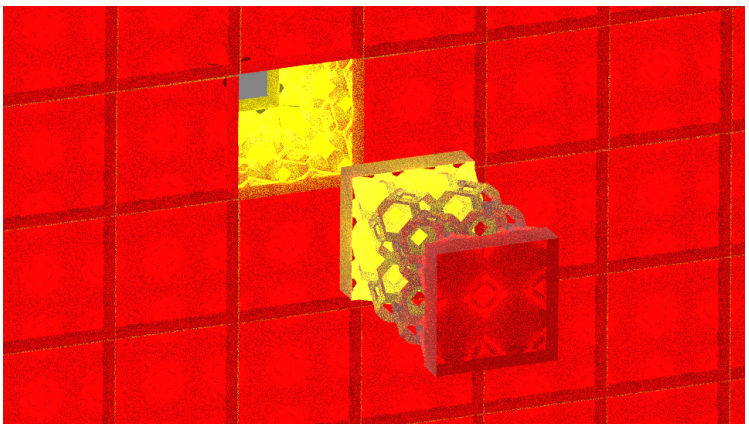


Figura 2.4: Pared y una sección con flujo de calor.

cada una, y tiene un grosor de 16 mm. Las secciones sólidas tienen un grosor de 2 mm cada una, y la zona porosa tiene un grosor de 12 mm.

Los tubos de la zona porosa son elementos geométricos regulares construidos a partir de celdas Kelvin (Figura 2.5). Cada celda Kelvin tiene 4 mm de diámetro y los tubos de sus aristas tienen un diámetro de 0,5 mm. En el anexo C se explica la construcción del modelo geométrico de la sección de la pared.

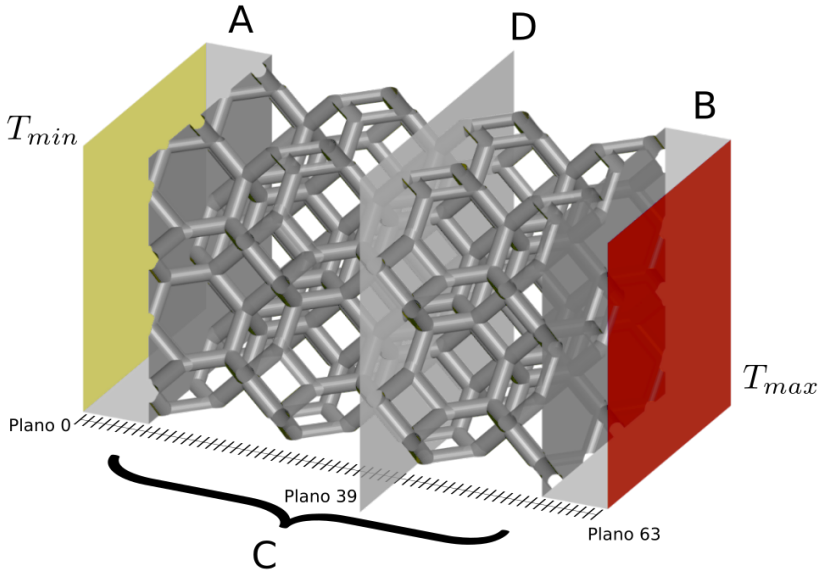


Figura 2.5: Sección de la pared y entramado de celdas Kelvin. A y B son las secciones sólidas, C es la zona porosa, y D (el plano número 39) es uno de los planos desde donde comienzan los caminos.

2.3.2. Algoritmo de cálculo de temperaturas

Para calcular la temperatura de la sección de la pared se va a subdividir la misma en subsecciones de mismo tamaño y se va a calcular la temperatura de cada una de ellas. Se toman 64 intervalos regulares a lo largo de la sección, y para cada intervalo se toma un plano que pasa por el medio del mismo.

Desde cada uno de los planos se lanzan múltiples caminos al azar desde puntos que se encuentran en la intersección del plano con el sólido; cuando dichos caminos llegan a una cara con temperatura T_{min} o T_{max} se le asigna esa temperatura al inicio del camino, y para ese plano se van guardando los valores de todos los caminos lanzados. Finalmente, para calcular la temperatura en cada plano se promedian los valores de los puntos guardados en el mismo.

En la figura 2.6 se muestra un esquema de avance de un camino utilizando pasos discretos, desde el punto x_0. Los caminos al azar comienzan todos dentro del sólido, y van avanzando en una serie de pasos discretos a azar dentro del mismo, y en un solo paso cuando atraviesan el vacío. A medida que los caminos van generando su recorrido pueden llegar a colisionar con la frontera del sólido. En ese caso se debe modelar el intercambio de calor con el medio. La radiación sólo se modela en el espacio al vacío en el interior de la pared.

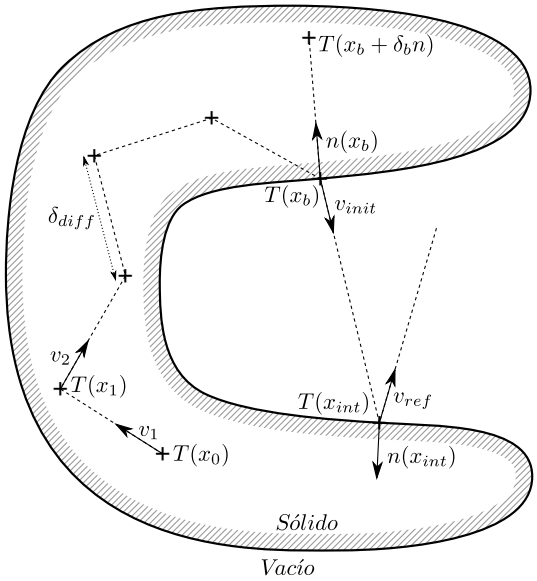


Figura 2.6: Esquema de avance de un camino usando pasos discretos y de comportamiento en la frontera. La figura fue extraída de Caliot *et al.* (2018), *Fig. 1.*

Para simular la pared infinita, el algoritmo considera que el calor que se

transmite por un lado de una sección (exceptuando los lados en contacto con T_{min} y T_{max}) pasa al lado opuesto de la sección adyacente. Como la sección adyacente se comporta igual que la sección actual, crea el efecto neto de una *traslación* del camino en el punto que cruza de una sección a otra, como se ve en la figura 2.7.

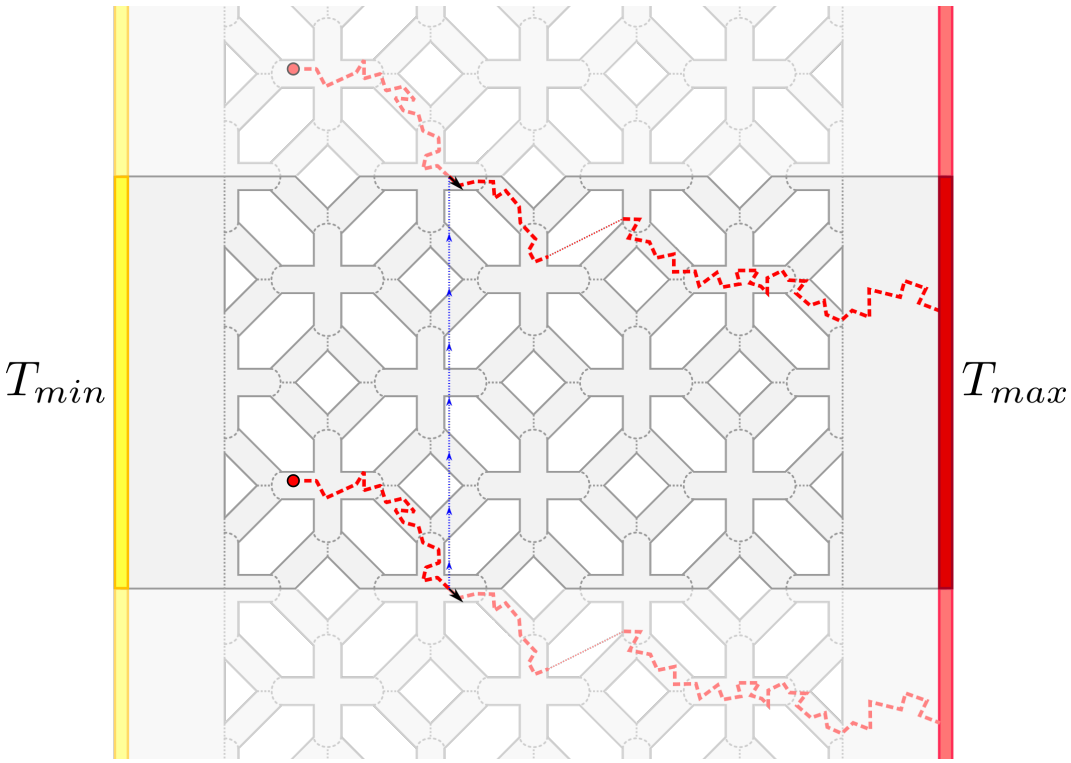


Figura 2.7: Representación de la traslación de un camino. Se muestran con colores más tenues las secciones y los caminos que simulan el efecto de la pared infinita.

Considerando el medio vacío, se pueden tomar aproximaciones a la ecuación de intercambio de calor a través del uso del coeficiente de emisividad y la probabilidad de difusión.

Como se ve en la figura 2.6, cuando el camino llega a un punto x_b en la frontera del sólido, la temperatura dentro de un entorno δ_b puede expresarse como:

$$T(x_b) = \frac{\frac{\lambda}{\delta_b}}{\frac{\lambda}{\delta_b} + h_r} T(x_b - \delta_b n) + \frac{h_r}{\frac{\lambda}{\delta_b} + h_r} T_{rad}(x_b) \tag{2.1}$$

donde $h_r = 4\varepsilon_h \sigma T_{ref}^3$ es el coeficiente linearizado de intercambio de radiación, siendo n la normal en dicho punto, y λ el coeficiente de conductividad

térmica. En la formulación de h_r, ε_h es el coeficiente de emisividad, σ es la constante de Stefan–Boltzmann y T_{ref} es la temperatura de referencia calculada como el promedio entre T_{min} y T_{max}. El valor de δ_b es una constante menor al diámetro de los tubos de la zona porosa.

La ecuación 2.1 muestra que en un punto de la frontera hay dos fuentes de temperatura a considerar; la primera $T(x_b - \delta_b n)$ debido a la difusión y la segunda $T_{rad}(x_b)$ debido a la radiación. Cada una tiene asociada una probabilidad, que en el caso de difusión es:

$$p_{diff} = \frac{\frac{\lambda}{\delta_b}}{\frac{\lambda}{\delta_b} + h_r} \tag{2.2}$$

y en el caso de radiación es:

$$p_{rad} = \frac{h_r}{\frac{\lambda}{\delta_b} + h_r} = 1 - p_{diff} \tag{2.3}$$

Observar en la ecuación 2.1 que si h_r es muy pequeño comparado con $\frac{\lambda}{\delta_b}$ se toma la temperatura obtenida por difusión, y en caso contrario se toma la temperatura obtenida por radiación.

2.3.3. Casos de estudio

En el artículo de Caliot *et al.* (2018) se consideran cuatro casos de estudio los cuales se muestran en la tabla 2.1. Para calcular el valor de la probabilidad de difusión (p_{diff}) se utiliza la ecuación 2.2

$$p_{diff} = \frac{\frac{\lambda}{\delta_b}}{\frac{\lambda}{\delta_b} + h_r}, \quad \text{con } h_r = 4\varepsilon_h \sigma T_{ref}^3 \tag{2.4}$$

En la ecuación 2.4 los valores de λ y T_{ref} toman diferentes valores según el caso de estudio. Para todos los casos se utilizan los mismos valores de δ_b y ε_h, tomando $\delta_b = 0.1mm$ y $\varepsilon_h = 0{,}85$.

En el caso 1a debido a que el valor de p_{diff} es casi del 100 % la conducción es casi total (y la radiación es casi nula); por lo tanto, la gran mayoría de los caminos deben recorrer el "laberinto" de los tubos hasta llegar a una cara con temperatura conocida. Por esta razón, como se verá en la sección 4.3.2, este caso también es el que requiere mayor tiempo de cómputo.

El caso 2a presenta el escenario opuesto al mencionado anteriormente: la

Caso	λ (W/mK)	$T_{min} - T_{max}$ (K)	p_{diff}
1a	40	$300 - 310$	$\sim 1{,}00$
1b	0,001	$300 - 310$	$\sim 0{,}65$
2a	0,0042	$1.000 - 1.500$	0,10
2b	0,03765	$1.000 - 1.500$	0,50

Tabla 2.1: Tabla con los casos de estudio considerados.

radiación es del 90 %.

Por último, en los casos 1b y 2b las probabilidades de conducción son 65 % y 50 % respectivamente. Estos casos presentan un comportamiento intermedio respecto a los casos 1a y 2a.

En el trabajo de Caliot *et al.* (2018) se grafican las temperaturas calculadas en cada caso para observar su comportamiento dentro de la pared. También se comparan los resultados obtenidos con los del método de volúmenes finitos utilizando el software ANSYS Fluent. En la figura 2.8 se muestra un ejemplo de dichas gráficas; para más detalles ver el capítulo 4.

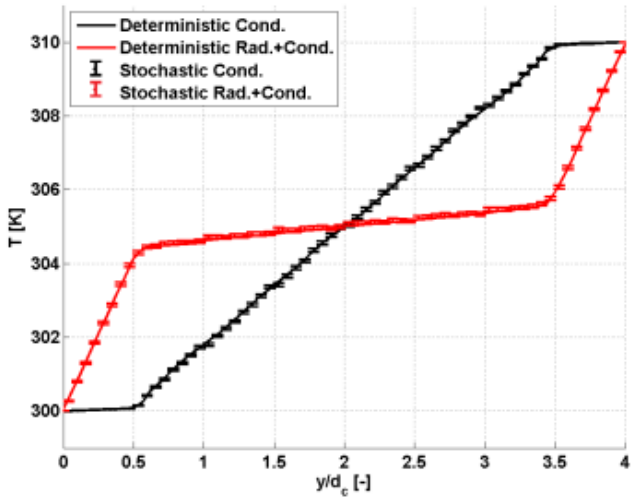


Figura 2.8: Resultados de Caliot *et al.* (2018), *Fig. 7.* Se muestran los casos 1b (Cond.+Rad.) y 1a (Cond.).

2.3.4. Pseudo códigos de los algoritmos de Caliot *et al.* (2018)

A continuación, se presentan los pseudo códigos de los algoritmos propuestos en Caliot *et al.* (2018).

El algoritmo comienza en un punto interno del sólido x_0 (ver figura 2.6). Aplicando el Código 2.1 el cual genera un paso al azar de largo δ_{diff} llega al punto x_1. Si el destino del paso está dentro del sólido se continúa de forma recursiva. En caso de alcanzar la frontera del sólido (punto x_b) se aplica el Código 2.2. La distancia δ_{diff} se calcula como $\delta_{diff} = {}^{\delta_b}/_2$.

Comprobar que la posición ingresada x_0 pertenece al sólido
Considerar la distancia del salto igual a δ_{diff}
Tomar una dirección al azar v tomada uniformemente en 4π sr
Evaluar la posición $x_1 = x_0 + \delta_{diff} \cdot v$
if x_1 *pertenece al interior de un sólido* **then**
    $T(x_0) = T(x_1)$ (Llamada recursiva al Código 2.1)
else
    Identificar la posición x_b (intersección entre x_1–x_0 y la frontera del sólido)
    $T(x_0) = T(x_b)$ (Llamada al Código 2.2)
end

Código 2.1: Algoritmo para evaluar $T(x_0)$ dentro del sólido.

El Código 2.2 decide según el valor de p_{diff} si el camino sigue por dentro del sólido (y se continúa con el Código 2.1) o si sale al vacío en forma de radiación. En caso de ingresar al sólido lo hace con una dirección normal $\vec{n}$ a la superficie $n(x_b)$ y genera un paso de largo δ_b. En caso de salir al vacío se proyecta un rayo en dirección al azar según una distribución coseno v_{init} y se sigue el cálculo con el Código 2.3 en el punto destino x_{int}.

El Código 2.3 determina en función de la probabilidad de emisividad si evaluar la temperatura de la frontera (y en ese caso se utiliza el Código 2.2), o si sale al vacío con una dirección v_{ref} al azar según una distribución coseno. En caso de salir al vacío se volverá a usar el 2.3 en el nuevo punto destino.

La figura 2.9 presenta un diagrama de estados donde se muestra la evolución de un camino aleatorio desde que comienza dentro del sólido hasta que llega a una cara con temperatura conocida.

Comprobar que la posición ingresada x_b está en la frontera
Considerar la distancia del salto igual a δ_b
Tomar un valor al azar r uniformemente dentro del rango $[0, 1]$
if $r < \frac{\lambda/\delta_b}{\lambda/\delta_b + h_r(x_b)}$ **then**
    $T(x_b) = T(x_b - \delta_b n)$ (Llamada al Código 2.1)
else
    $T(x_b) = T_{rad}(x_b)$ (Llamada al Código 2.3)
end

Código 2.2: Algoritmo para evaluar $T(x_b)$ en la frontera.

Tomar una dirección de emisión v_{init} según $\frac{|v_{init} \cdot n(x_b)|}{\pi} p_\varepsilon(v_{init})$
Lanzar un rayo desde x_b, con dirección v_{init}
Identificar la primera intersección x_{int} entre el rayo y la frontera de un sólido
Tomar un valor al azar r uniformemente dentro del rango $[0, 1]$
if $r < \varepsilon_h$ **then**
    $T_{rad}(x_b) = T(x_{int})$ (Llamada al Código 2.2)
else
    Tomar una dirección de reflexión v_{ref} según $\frac{|v_{ref} \cdot n(x_{int})|}{\pi} p_\varepsilon(v_{ref})$
    $T_{rad}(x_b) = T_{rad}(x_{int})$ (llamada recursiva al Código 2.3)
end

Código 2.3: Algoritmo para evaluar $T_{rad}(x_b)$ desde el vacío hacia el sólido.

El valor de δ_b se elige arbitrariamente al comienzo de la ejecución y se mantiene el mismo valor en todas las pruebas.

En todos los casos, cuando un camino llega a una cara con temperatura conocida el algoritmo termina; cuando esto ocurre se desenvuelven las llamadas recursivas llevando la información de la temperatura en cada caso ($T(x_i)$, $T(x_b)$, $T_{rad}(x_b)$ y $T_{rad}(x_{int})$) hasta llegar al punto de origen $T(x_0)$.

2.3.5. Comentarios

En el algoritmo propuesto por Caliot *et al.* (2018) la temperatura calculada sólo se asigna al punto de origen de cada camino, desperdiciando la información que aporta un camino cuando atraviesa otros planos además del de origen. Se originan 100.000 caminos en cada uno de los planos. Para elegir un punto en un

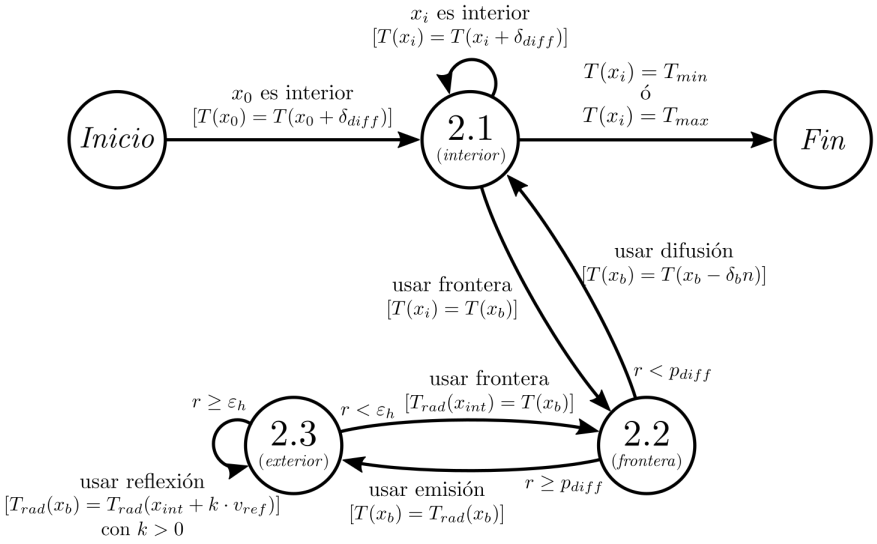


Figura 2.9: Diagrama de estados de los códigos que describen la evolución de un camino. Los números corresponden a los códigos 2.1, 2.2 y 2.3.

plano se toma una distribución desigual ya que se fuerza a que todos los puntos pertenezcan al interior de la intersección del plano con el sólido; debido a esto las secciones de menor área tienen mayor densidad de puntos respecto a las de mayor área. Además, se garantiza que la cantidad de orígenes configurada para cada simulación sea igual en todos los planos.

2.4. Herramientas para ray tracing

Para poder implementar el algoritmo descrito en Caliot *et al.* (2018) se buscó una herramienta que brinde un motor de ray tracing incorporado y que sea eficiente.

Se compararon varias herramientas para poder encontrar la adecuada para implementar los algoritmos necesarios y obtener el mejor desempeño.

Biblioteca Embree

Es una colección de kernels para realizar ray tracing en CPU con muy alto rendimiento desarrollado por Intel. Los kernels están optimizados para los últimos procesadores de Intel que soportan los conjuntos de instrucciones especiales para manejo de vectores y matrices (SSE, AVX, AVX2 y AVX-512).

La documentación de los ejemplos es nula, sólo se tiene de la API. (Intel Embree (2018))

Raytracer Mitsuba

Es un renderizador basado en física (PBR: Physically Based Rendering) orientado a la investigación. Está escrito en C++ portable y está altamente optimizado para las arquitecturas actuales de CPU. Es modular y contiene un pequeño conjunto de bibliotecas básicas y muchos complementos para generar las luces, materiales y algoritmos de renderizado. Tiene una curva de aprendizaje alta, con una documentación enfocada al uso en lugar de al desarrollo de aplicaciones. (Mitsuba (2018))

Raytracer PBRT

Es un sistema de renderizado basado en las propiedades físicas tomando como base el algoritmo clásico de ray tracing. Dicho sistema está descrito extensivamente en el libro homónimo el cual está fuertemente enfocado para un uso educativo, no tanto en el rendimiento. (Pharr *et al.* (2018))

Biblioteca OptiX

Es un framework de ray tracing de óptimo rendimiento en GPU. Provee un pipeline simple, recursivo y flexible para acelerar algoritmos de ray tracing. Esta biblioteca está disponible solamente para GPUs de NVIDIA y permite programar cada etapa del algoritmo (generación de rayos, cálculo de intersecciones, sombreado, etc.). Tiene una excelente documentación tanto de la API como de los ejemplos, facilitando la creación de aplicaciones. (NVIDIA OptiX (2018))

Dado que el objetivo es implementar los algoritmos de Caliot *et al.* (2018) en GPU, utilizaremos OptiX para su desarrollo por tener muy buen rendimiento junto con una excelente documentación. Además, la experiencia previa en el uso de herramientas de NVIDIA para programación en GPU (CUDA) redujo tiempos de aprendizaje en el uso de la biblioteca.

2.5. Descripción del funcionamiento de OptiX

El framework OptiX encapsula CUDA (la herramienta de programación de NVIDIA) para abstraer el manejo de la GPU del desarrollo de la aplicación que utiliza ray tracing.

Define un contexto de operación para el lanzamiento de los rayos y los eventos globales de la escena (por ejemplo, cuando un rayo no toca ningún objeto) dentro del cual se utiliza una estructura jerárquica de aceleración. Ofrece funciones para cargar objetos (como geometrías 3D) en dicha estructura; a cada objeto se le puede asociar un conjunto de rutinas que serán ejecutadas según los eventos ocurridos entre un rayo y los objetos, por ejemplo, la intersección más cercana o cuando un rayo no interseca a un objeto.

También ofrece funciones para el intercambio de datos entre el Host y la GPU y una manera propia de acceso a la memoria de GPU por lo que no es recomendable utilizar CUDA directamente a riesgo de interferir con el funcionamiento de OptiX (NVIDIA OptiX (2018)).

Los módulos que ejecuta una GPU se llaman kernels en la terminología de CUDA. En cada kernel pueden codificarse muchas funciones y rutinas auxiliares. Las funciones de cada kernel son ejecutadas por varios hilos en paralelo.

El modelo de proceso presentado por OptiX consiste en una rutina de ray tracing que se inicializa y luego gerencia la traza de los rayos solicitados mediante las respectivas funciones desde los kernels. En dichos kernels el programador debe implementar tanto las rutinas de lanzamiento de rayos como las de respuesta a los eventos de su interés, posiblemente solicitando dentro de dichas rutinas el trazado de nuevos rayos.

Ofrece varios ejemplos de diferentes usos de ray tracing integrándolo con otras técnicas. Para este proyecto se tomó como base un ejemplo de photon mapping progresivo del sitio de tutoriales de OptiX (NVIDIA DesignWorks Samples (2018)), al cual se le agregó el uso de números aleatorios generados en GPU NVIDIA cuRAND (2018).

Capítulo 3

Diseño e Implementación propuesta

El algoritmo planteado en el capítulo 2 sólo considera la información obtenida en los orígenes descartando todos los cálculos de los puntos intermedios de cada camino. A partir de esto surge una posible mejora que consiste en aprovechar dichos puntos, aumentando la cantidad de información disponible para calcular el promedio de la temperatura de cada plano.

3.1. Diseño

Para calcular la distribución de temperatura de la sección de la pared, se utilizan las temperaturas de todos los planos. Para hallar la temperatura de cada plano se toman varios puntos de origen (como se describió en la sección 2.3.2). Para cada uno de esos puntos de origen se calcula un camino como una serie de pasos al azar hasta llegar a una cara con temperatura conocida. Durante su recorrido, cada camino va marcando por cuales planos va pasando. Cuando un camino termina se asigna su valor de temperatura a cada uno de los planos por donde pasó.

Este enfoque permite calcular resultados diferentes utilizando dos variantes del algoritmo:

Caliot OptiX (CO) Cuando se consideran solo las temperaturas de los puntos de origen de cada camino.

Reúso de Caminos OptiX (RCO) Cuando se consideran los aportes de temperatura de todos los caminos que cortaron cada plano.

Cada uno de estos métodos tiene dos fases. La primera fase permite generar los orígenes de los caminos, y la segunda fase hace posible calcular el camino para dichos orígenes.

Para generar los orígenes en cada plano se toman puntos al azar que deben verificar además su pertenencia al interior del sólido. La cantidad de puntos de origen determina el número de caminos a calcular.

Para calcular cada uno de los caminos se generan una serie de pasos; en cada paso se evalúan los códigos de la sección 2.3.4 para decidir los parámetros del siguiente paso a dar (dirección, longitud, etc.), y luego se procesa el paso siguiente. Cuando en uno de los pasos se llega a una zona con temperatura conocida (por ejemplo, las caras con temperaturas T_{min} o T_{max}), el camino termina y se asigna ese valor al punto de origen en la variante CO y al camino completo en la variante RCO.

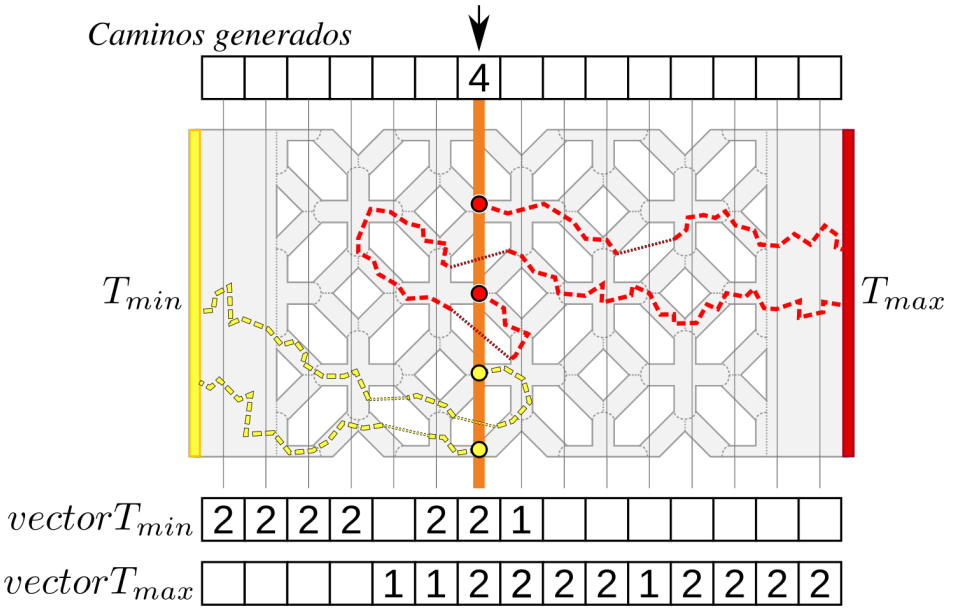


Figura 3.1: Ejemplo de marcado y conteo de temperaturas. El modelo de ejemplo fue dividido en 15 planos para que sea más fácil de apreciar.

En cada plano se almacena el conteo de los caminos que lo han cruzado en dos vectores; un vector para cada temperatura conocida y alcanzable (T_{min} y T_{max}). Con esta información se puede calcular el promedio de temperatura de cada plano como el promedio de las temperaturas de los caminos que lo cruzaron.

En la figura 3.1 puede verse un ejemplo con dos caminos, en la parte su-

perior el vector de caminos generados, y en la parte inferior los vectores de conteo para cada temperatura y plano. Entonces la temperatura de un plano P, es:

$$T(P) = \frac{T_{min} * vectorT_{min}(P) + T_{max} * vectorT_{max}(P)}{vectorT_{min}(P) + vectorT_{max}(P)}$$

Para poder calcular el promedio de temperaturas, se necesita que los valores a promediar de cada plano sean estadísticamente independientes.

En el caso de los puntos de origen, esto viene dado por la independencia de su generación; en cambio al considerar la evolución de los caminos puede ocurrir que un camino cruce repetidamente por el mismo plano. En este caso se toma sólo el primer cruce. La manera simple de asegurar esto es marcar los planos por los que cruza un camino (incluyendo al de origen). Las marcas no se acumulan si el camino vuelve a cruzar por un plano ya marcado por el mismo camino.

Entre planos cercanos no hay independencia estadística, pero es un fenómeno similar al que ocurre en las técnicas de iluminación global cuando se calcula la iluminación de puntos cercanos.

3.1.1. Cálculo de IC

En este caso el muestreo tiene una distribución binomial, ya que hay sólo dos valores posibles, T_{min} y T_{max}. Para un total de muestras n grande y por el Teorema Central del Límite (Canavos (1998)), se puede suponer que el intervalo de confianza de una proporción de la muestra p se comporta como una distribución normal.

Entonces, sea $\hat{p}$ un estimador de p, el intervalo de confianza de $\hat{p}$ es

$$\left[\hat{p} - z_c\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}, \hat{p} + z_c\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\right]$$

donde z_c es el valor crítico para la confianza c usando la distribución normal, y $\hat{p}$ puede estimarse como x/n siendo x el número de muestras de uno de los posibles valores del muestreo (por ejemplo, T_{min}) (Canavos (1998)).

El valor de z_c para una confianza del 99,7 % es 3 (Canavos (1998)). Aplicado a este caso tenemos:

$$n = \#Muestras\_T_{min} + \#Muestras\_T_{max}$$
$$x = \#Muestras\_T_{min}$$
$$\hat{p} = x/n$$

$$IC = 3 \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

Como el IC está referido al rango de la proporción $[0, 1]$ debe escalarse al rango de temperaturas usado $[T_{min}, T_{max}]$.

3.1.2. Arquitectura del programa

La arquitectura consiste en varios módulos que se ejecutan unos en CPU y otros en GPU. Como se ve en la figura 3.2, los módulos son los siguientes:

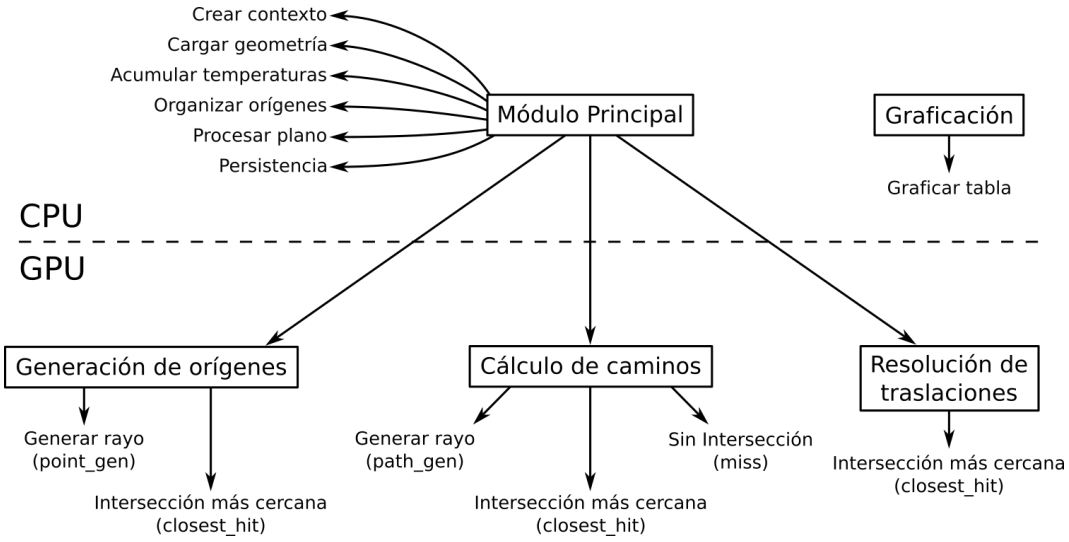


Figura 3.2: Diagrama de la arquitectura.

CPU

Módulo Principal Lanza la ejecución del algoritmo, recopila y guarda los resultados.

Graficación Calcula los promedios, intervalos de confianza y grafica los resultados.

GPU

Generación de orígenes Calcula los orígenes válidos para los caminos.

Cálculo de caminos Calcula los caminos como una serie de pasos.

Resolución de traslaciones Resuelve la traslación de un camino para simular la pared infinita.

3.2. Implementación

Para la implementación se tomó como base un ejemplo de photon mapping progresivo (descrito en el anexo A) del sitio de tutoriales de OptiX (NVIDIA DesignWorks Samples (2018)) el cual fue modificado principalmente de las siguientes maneras:

- En el módulo principal, se eliminó el uso del árbol KD y la interfaz gráfica de usuario. Se agregaron las rutinas para acumular y guardar los puntos de cada plano. Además, se modificaron las funciones para cargar los objetos del disco (geometría y caja).
- Las rutinas que trazaban rayos desde la cámara ahora generan los puntos de origen de los caminos.
- Las rutinas que eran responsables de trazar los rayos de los fotones en los sucesivos rebotes ahora calculan los saltos de avance de cada camino.
- Se agregó una rutina para la función de traslación de los rayos que salen por un lateral de la geometría.
- Se agregaron estructuras de datos para el conteo y detección de los cruces de los caminos con los planos.
- Se eliminó la rutina que generaba la imagen utilizando la información de los rayos de la cámara y del mapa de fotones.
- Se sustituyó el uso de la función estándar de generación de números aleatorios por el uso de la biblioteca NVIDIA cuRAND (2018). Esta biblioteca tiene como ventaja que puede ser llamada directamente desde las funciones de los kernels.

Se reutilizaron el resto de las rutinas y funciones.

Detalles de la implementación

El esquema de ejecución es el siguiente:

- Se inicializan las estructuras de cálculo.
- Se itera para cada plano:
  - Se lanza el módulo para la generación de orígenes.
  - Se optimiza el arreglo de orígenes.
  - Se lanza el módulo de cálculo de caminos.
  - Se acumulan las temperaturas para ambas variantes del algoritmo según la información de cada camino.
- La información generada se guarda en archivos de texto.
- Se calculan los IC y se generan las imágenes de las gráficas.

La geometría del sólido está modelada con dos objetos: uno que modela una sección de la pared y otro que modela la "caja" que la envuelve.

Como se muestra en la figura 3.3, cada temperatura ($Tmin$ y $Tmax$) tiene asociada una constante para identificarla cuando es asignada a un camino. Existe otra constante (Tninguna) para indicar que un punto no tiene temperatura asignada. También existe una matriz que almacena las marcas hechas por cada camino al cruzar cada plano, así como un registro para cada camino que indica a cuál temperatura llegó.

Matriz de marcas

			Planos							
Camino	Temperatura		1	2	...	7	8	...	15	16
1	$Tmax$	:			...	X	X	...	X	X
2	$Tmax$	:			...	X	X	...	X	X
3	$Tmin$	:	X	X	...	X	X	...		
4	$Tmin$	:	X	X	...	X		...		

Arreglos de conteo de la variante CO

Marcas $Tmin$	:	0	0	...	2	0	...	0	0
Marcas $Tmax$	:	0	0	...	2	0	...	0	0

Arreglos de conteo de la variante RCO

Marcas $Tmin$	:	2	2	...	2	1	...	0	0
Marcas $Tmax$	:	0	0	...	2	2	...	2	2

Figura 3.3: Estructuras de datos correspondiente al ejemplo de la figura 3.1. Se muestra la matriz de marcas y los arreglos de conteo de temperatura para cada variante.

Para el cálculo de los promedios de temperatura de cada plano, sólo es necesario saber cuántos puntos representantes de cada temperatura hay en

cada plano; esto se logra utilizando dos arreglos donde cada uno acumula los puntos de una determinada temperatura para cada plano. Para acumular los puntos de temperatura en cada arreglo, se usan la matriz de marcas y los registros de temperatura. Los conteos para cada variante se almacenan en arreglos separados, uno para la CO y otro para la RCO.

Cada hilo que se encarga de calcular el avance de un camino lo hace en una serie de pasos. Cada paso se implementa con un rayo de dirección y longitud apropiada, teniendo rayos identificados como de tipo δ_b, δ_{diff} para los pasos dados dentro del sólido, y de tipo *infinito* para los pasos dados en el vacío. El cálculo para determinar el paso siguiente en el avance de un camino se realiza según lo propuesto por los códigos de la sección 2.3.4.

El cálculo adicional requerido para la variante RCO (detección de cada cruce de plano y marcado del mismo) respecto a la variante CO, tiene un costo muy pequeño en comparación con las otras tareas comunes a ambas variantes.

Para obtener las direcciones al azar se utilizan diferentes métodos:

- En el caso de un paso dentro del sólido, se selecciona una dirección de una distribución uniforme en una esfera (Weisstein (2013), Simon (2015)).
- En el caso de ingreso de un rayo al interior del sólido, se usa una distribución uniforme de una hemiesfera orientada hacia adentro (Weisstein (2013), Simon (2015)).
- En el caso de salida de un rayo al exterior, se usa una distribución lambertiana (2.1.2).

Como cada fila de la matriz de marcas y cada registro es independiente para cada camino, se elimina la necesidad de compartir memoria entre hilos. El registro mencionado almacena además el estado del punto actual de cada camino. Dicho estado se compone de un indicador de la temperatura alcanzada, la posición y dirección tomada, el tipo de rayo utilizado, y variables aleatorias para decisiones posteriores (parámetros para elección de dirección, parámetros de emisividad para radiación, etc.).

Tanto la matriz de marcas como los registros de estado son usados por las rutinas de los kernels.

Para evitar bloqueos por excesivos rebotes, los caminos tienen una longitud máxima de pasos, así como una cantidad máxima de traslaciones. Ambos umbrales son configurables.

Las rutinas de los kernels tienen varios chequeos para validar los rayos lanzados y en caso de falla se *elimina* el camino en cuestión, descartándolo de los cálculos. Además, llevan la contabilidad de los caminos generados que lograron llegar a una fuente, así como los que fueron eliminados.

La implementación planteada ha requerido que se codifiquen tres kernels: uno que genera los orígenes, otro que calcula los caminos; y el tercero que se encarga de la traslación de los caminos. Cada kernel está compuesto por una o varias rutinas que implementan (si corresponde) la generación de rayos, el comportamiento respecto a colisiones, y el comportamiento respecto a no intersecar al sólido.

3.2.1. Módulo principal

Contiene las rutinas necesarias para la ejecución de los módulos de cálculo de orígenes y caminos, la recopilación de resultados y su almacenamiento para el módulo de graficado.

Las funciones más relevantes son:

Función *main*

Se encarga de procesar los argumentos de línea de comando, precalcular coeficientes de difusión y emisividad, inicializar el contexto de OptiX, cargar la geometría, realizar las iteraciones del algoritmo, recopilar los resultados y guardarlos en archivos.

Función *create_context*

Crea el contexto de OptiX responsable del estado de la escena.

Crea los buffers que contendrán los arrays con la información y estado de los caminos. El buffer *ppass_point_buffer* es de entrada/salida pues guarda los orígenes creados por la primera fase para pasarlos a la segunda. Los demás buffers sólo generan información de salida para acumular.

Además, asocia las rutinas a los eventos de *generación de rayos* y de *no intersección* de la escena para cada fase.

Funciones *create_geometry* y *create_bbox*

Cargan las geometrías del modelo y de la caja, asignando los eventos *intersección más cercana* correspondientes a las mismas.

Función *acumular_temperaturas*

Obtiene la información de los caminos generada en GPU para luego acumular los puntos de cada temperatura, tanto cuando se consideran sólo los orígenes como cuando se toma el camino completo.

Función *organizar_arreglo*

Prepara el arreglo de orígenes para la siguiente fase, agrupándolos y ordenándolos.

Función *launch_all*

Calcula los caminos desde un plano dado, acumulando las temperaturas generadas. Para ello lanza el cálculo de orígenes para dicho plano en GPU, organiza los arreglos de orígenes, lanza el cálculo de caminos en GPU y acumula las temperaturas.

3.2.2. Módulo de procesamiento y graficado

El módulo de graficación lee los archivos de texto generados por el módulo principal y utiliza las funciones de la biblioteca Matplotlib (Matplotlib (2019)) con Python (Python (2019)) para obtener los gráficos de temperatura.

Consiste en un script que contiene las funciones de formateo de información y la función principal de procesamiento *graficarTabla*. Recibe como parámetro el valor de lambda para determinar que archivos graficar. Con dicho valor se obtiene el nombre del archivo a cargar para procesar.

El procesamiento consiste en ir cargando los datos del archivo y calculando los promedios e intervalos de confianza para cada plano aplicando las fórmulas de la sección 3.1.1. Luego se grafican dichos datos guardando la gráfica generada.

La función *graficarTabla* es usada con cada uno de los casos de lambda mencionados en Caliot *et al.* (2018) para generar las correspondientes gráficas.

Las bibliotecas usadas permitieron calcular los promedios de temperaturas y los intervalos de confianza respectivos. También se utilizaron dichas herramientas para realizar las distintas gráficas que se pueden ver en el capítulo 4.

3.2.3. Generación de orígenes

Contiene la rutina de generación de rayos para generar los puntos de origen y la rutina de respuesta al evento de *intersección más cercana* para determinar la validez de un punto verificando si está adentro del sólido.

Rutina de *generación de rayos*

Inicializa el registro de estado y las variables aleatorias. Selecciona al azar un punto del plano y una dirección al azar de manera uniforme en una esfera. Para probar la validez del punto como origen de camino (es válido si está dentro de un sólido), se traza un rayo desde dicho punto con la dirección seleccionada. Si está adentro del sólido se guarda dicho punto como origen, en caso contrario, se selecciona otro punto, otra dirección y se repite el proceso.

Rutina del evento de *intersección más cercana*

Evalúa si el rayo incide desde adentro del sólido usando el producto punto entre la dirección del rayo y el vector normal de la superficie en la intersección, guardando el resultado en el registro de estado para su uso por la rutina de generación de orígenes.

3.2.4. Cálculo de caminos

Este módulo contiene una rutina de generación de rayos en donde se van generando los sucesivos puntos de un camino, y para cada uno se traza un rayo de longitud adecuada guardando el indicador de la temperatura a la que llega. El cálculo de los pasos y su dirección para los casos de los códigos 2.2 y 2.3 se realiza en la rutina de respuesta al evento de *intersección más cercana*. Para seleccionar la dirección del próximo paso se toma en cuenta si el rayo es interno o externo al sólido y la comparación de las variables aleatorias con los coeficientes de difusión y emisividad. El cálculo del caso del código 2.1 se implementa en la rutina del evento de *no intersección* mediante el rayo de

largo δ_{diff} y el empleo de la distribución esférica para el cálculo de la dirección del nuevo paso.

Rutina de *generación de rayos*

Inicializa el registro de estado del punto de construcción del camino con los datos de su punto de origen. También inicializa el array de uso que indicará los planos por donde pasa.

Para cada nuevo punto del camino se generan sus variables aleatorias y se lanza el rayo de tipo adecuado a su nuevo estado. Se repite el proceso hasta que su temperatura es conocida o se excede el tamaño máximo de camino, en cuyo caso dicho camino se elimina.

Rutina del evento de *intersección más cercana*

Calcula si el rayo es interno o externo al sólido, verificando que el tipo de rayo corresponda con su longitud (sólo los rayos con longitud δ_b o δ_{diff} son internos) informando en el registro de estado y eliminando el camino si ocurre una inconsistencia.

En caso de ser un rayo externo al sólido, decide (comparando su variable aleatoria de emisividad con el coeficiente de emisividad) si debe considerar la frontera o rebotar; si rebota lo hace seleccionando una dirección al azar en un hemisferio con distribución coseno. En caso de considerar la frontera, entonces decide (evaluando su variable aleatoria de difusión contra el coeficiente de difusión) si ingresar al sólido con dirección normal en el punto de intersección o salir con dirección al azar de la misma manera que al rebotar.

En caso de ser un rayo interno al sólido, decide (evaluando su variable aleatoria de difusión contra el coeficiente de difusión) si rebotar utilizando la dirección normal o salir con dirección al azar en un hemisferio con distribución coseno.

En todos los casos se ajusta el tipo de rayo de acuerdo su nuevo estado. El tipo de rayo para el caso de rayo interno es δ_b, e *infinito* en el otro caso.

Si un rayo interno cruzó el plano del intervalo correspondiente a su destino, se marca para el conteo y se incrementa el tamaño del camino.

Luego se actualiza el estado del camino (posición, dirección y tipo de rayo).

Rutina del evento de *no intersección*

Verifica que el tipo de rayo sea interno al sólido para continuar o eliminar el camino en caso contrario.

Calcula el destino del paso como el punto anterior más la distancia del rayo lanzado (ya sea δ_{diff} o δ_b). Calcula la nueva dirección eligiendo al azar uniformemente en una esfera.

Si el paso cruzó el plano del intervalo correspondiente a su destino, se marca para el conteo y se incrementa el tamaño del camino.

Se ajusta el tipo de rayo a δ_{diff} y se carga el nuevo estado del punto (posición, dirección y tipo de rayo).

3.2.5. Cálculo de traslaciones

Este módulo contiene únicamente una rutina de respuesta a eventos de *intersección más cercana*.

Rutina del evento de *intersección más cercana*

Calcula la traslación de un rayo hacia el otro lado de la caja de forma simétrica. Esto se logra trasladando el punto de choque, manteniendo la dirección original y ajustando la longitud según la distancia restante del rayo original al momento de chocar con la caja (como se explicó en 2.3.2 y se puede ver en la figura 2.7).

También se encarga de verificar si un camino llegó a una de las caras de temperatura conocida. Además, comprueba si el rayo es interno a la caja, y elimina el camino si la cantidad de traslaciones sucesivas supera el umbral configurado.

Capítulo 4

Resultados

En este capítulo se muestran los resultados obtenidos utilizando las variantes *Caliot OptiX* (CO) y *Reúso de Caminos OptiX* (RCO) del algoritmo implementado, comparándolos con los resultados del artículo de Caliot *et al.* (2018). Los casos analizados son los que se muestran en la tabla 2.1. Primero se comparan los resultados de la variante CO con el original de Caliot *et al.* (2018) para poder validar la implementación en GPU. Luego se comparan ambas variantes del algoritmo.

También se describe cómo se obtuvo una estimación de la cantidad de caminos necesarios para obtener intervalos de confianza similares y otras métricas de interés (número de caminos eliminados (sección 3.2) y tiempos de ejecución).

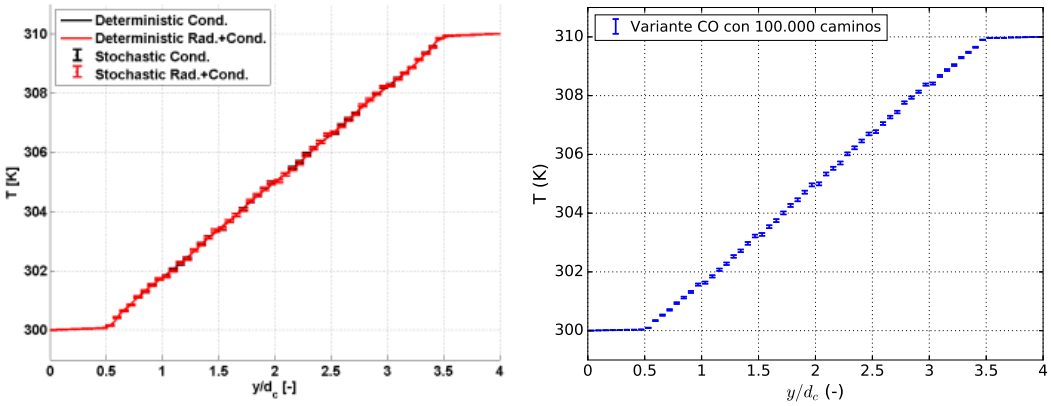
Para todas las pruebas las distancias de los pasos usadas son $\delta_b = 0.1mm$, y $\delta_{diff} = {}^{\delta_b}/_2$. El sólido se modela como un cuerpo gris difuso con un coeficiente de emisividad $\varepsilon_h = 0{,}85$.

4.1. Resultados de Caliot OptiX (CO)

En las gráficas 4.1, 4.2, 4.3 y 4.4 se pueden ver los resultados obtenidos con la variante CO implementada comparándolas con las del artículo de Caliot *et al.* (2018). En el eje horizontal de dichas gráficas se toma el diámetro d_c de una celda Kelvin (4 mm) como normalización adimensional (y/d_c (-)) de la escala. En Caliot *et al.* (2018) se compara su algoritmo con una solución calculada por el método de volúmenes finitos (utilizando el software ANSYS Fluent), etiquetadas en sus gráficas como *Deterministic*.

4.1.1. Caso 1a ($T_{min} = 300K$, $T_{max} = 310K$ y $\lambda = 40\ {}^{W}/_{mK}$)

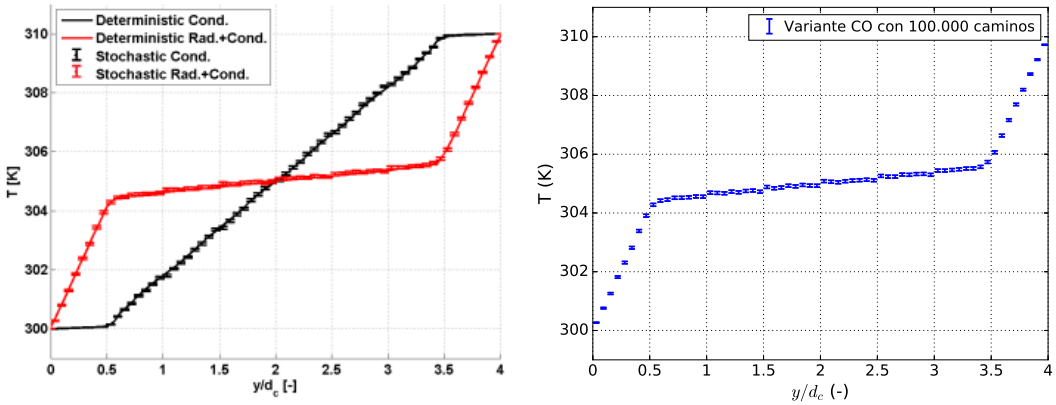
En el caso de la figura 4.1 se tiene una probabilidad de radiación casi nula, por lo que el fenómeno dominante es la conducción. Eso se aprecia al observar que las temperaturas son casi constantes en las partes sólidas y hay una evolución lineal de la temperatura en la zona porosa.



(a) Resultado de Caliot *et al.* (2018). **(b)** Resultado de la variante CO.

Figura 4.1: Comparación de los resultados para el caso 1a.

4.1.2. Caso 1b ($T_{min} = 300K$, $T_{max} = 310K$ y $\lambda = 0{,}001\ {}^{W}/_{mK}$)



(a) Resultado de Caliot *et al.* (2018). **(b)** Resultado de la variante CO.

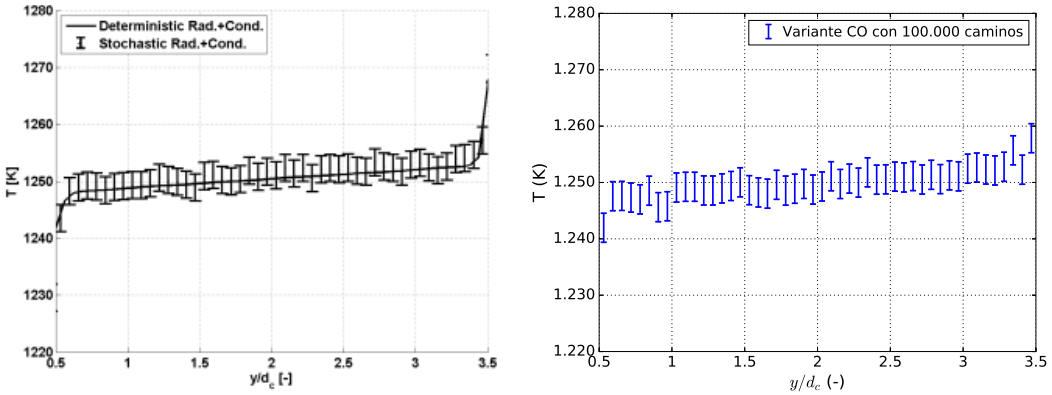
Figura 4.2: Comparación de los resultados para el caso 1b.

En el caso de la figura 4.2 la radiación tiene una influencia notoria. Lo que se observa con radiación es un aumento de la conductividad "efectiva" de

la zona porosa. Para este caso, las condiciones iniciales fueron elegidas de tal forma que la probabilidad de radiación/conducción son similares. En la figura 4.2a también aparece el caso 1a.

4.1.3. Caso 2a ($T_{min} = 1.000K$, $T_{max} = 1.500K$ y $\lambda = 0{,}0042\ ^{W}/_{mK}$)

En el caso de la figura 4.3 debido a las condiciones iniciales se tiene una probabilidad de radiación casi total, se grafican las temperaturas correspondientes a la zona porosa con más detalle para ver mejor los intervalos de confianza y el comportamiento de la temperatura.



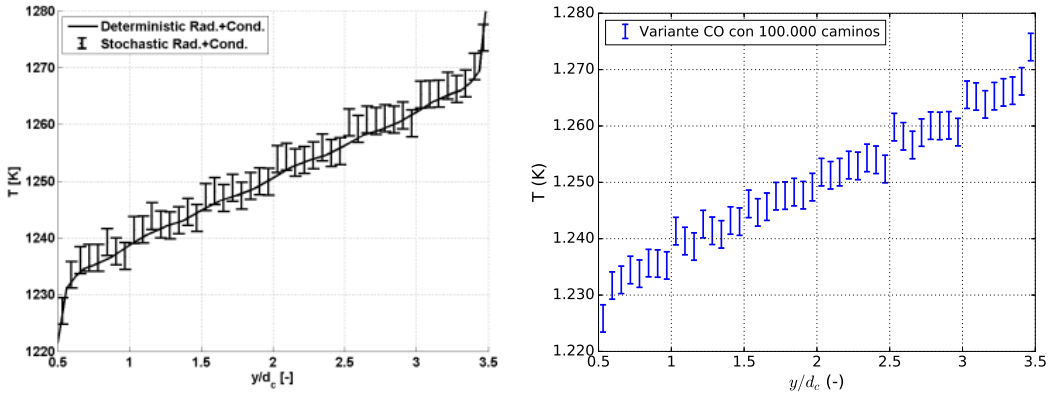
(a) Resultado de Caliot *et al.* (2018). **(b)** Resultado de la variante CO.

Figura 4.3: Comparación de los resultados para el caso 2a.

4.1.4. Caso 2b ($T_{min} = 1.000K$, $T_{max} = 1.500K$ y $\lambda = 0{,}03765\ ^{W}/_{mK}$)

El caso de la figura 4.4 es similar al caso 1b (figura 4.2), y por tanto la probabilidad de radiación/conducción son similares. Se grafican las temperaturas correspondientes a la zona porosa con más detalle para ver mejor los intervalos de confianza y el comportamiento de la temperatura.

En todos los casos se obtuvieron resultados similares a los del artículo de Caliot *et al.* (2018). También se observan pequeños saltos en las posiciones 1, 1.5, 2, 2.5 y 3. Estos saltos ocurren en las regiones de unión de las celdas Kelvin; allí los caminos encuentran una sección vertical similar a las secciones



(a) Resultado de Caliot *et al.* (2018). **(b)** Resultado de la variante CO.

Figura 4.4: Comparación de los resultados para el caso 2b.

sólidas, y por ende presentan un comportamiento similar, ya sea de temperatura constante cuando no hay radiación, o de aumento abrupto de temperatura cuando hay conducción y radiación.

4.2. Resultados de Reúso de Caminos en OptiX (RCO)

Al implementar RCO y por tanto aprovechar la información intermedia generada por los caminos, se obtiene una mayor densidad de muestras de temperatura por plano. Esto sugiere que se pueden obtener resultados con intervalos de confianza (IC) similares a los planteados en Caliot *et al.* (2018), pero utilizando menos caminos, y por lo tanto con mejor rendimiento.

Para calcular el mínimo de caminos necesarios en RCO para obtener la misma confiabilidad que Caliot *et al.* (2018), se utilizó una métrica global de confiabilidad de una ejecución para diferentes configuraciones de número de caminos. La métrica usada fue el promedio de los ICs de todos los planos. Notamos esta métrica por el símbolo $\overline{IC}$.

Se realizaron varias ejecuciones de la variante RCO del caso 1a variando el número de caminos, hasta encontrar aquella cuyo valor de $\overline{IC}$ fuese el más cercano al $\overline{IC}$ del algoritmo CO. Los resultados obtenidos se muestran en la figura 4.5. Allí se aprecia que, utilizando 100.000 caminos, los ICs en RCO son unas 4,7 veces inferiores a los obtenidos por CO. También se observa que para obtener en RCO ICs similares a los obtenidos por CO alcanza con 4.330

caminos. La curva obtenida a partir de los valores experimentales sugiere que posee la forma $\overline{IC} = \frac{2{,}015}{\sqrt{\#caminos}}$.

En la figura 4.5 se pueden ver los valores experimentales contrastándolos con la curva mencionada.

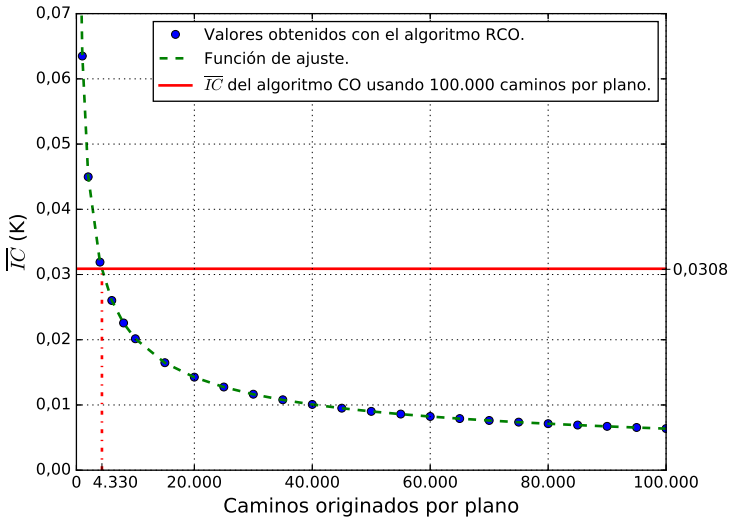


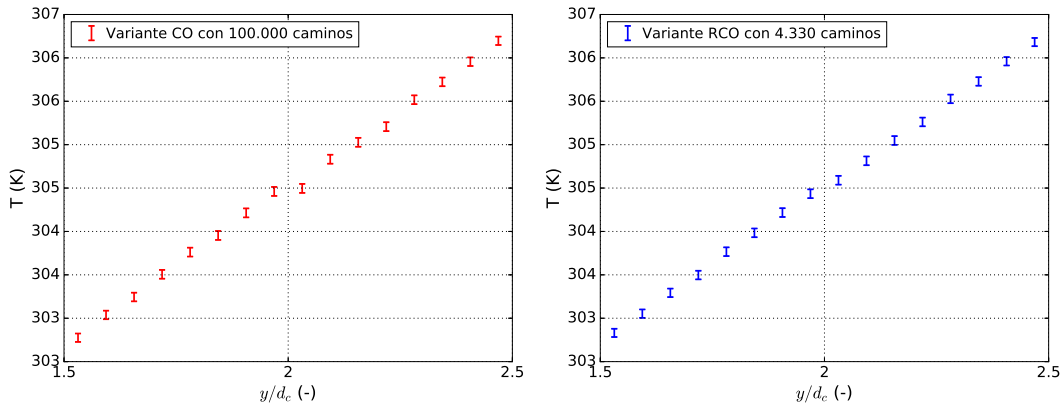
Figura 4.5: Comparación entre los valores experimentales de $\overline{IC}$ para el caso 1a con distintos números de caminos y la curva de ajuste $\overline{IC} = \frac{2{,}015}{\sqrt{\#caminos}}$.

Se repitió el análisis precedente para el resto de los casos. En la tabla 4.1 se pueden ver los resultados obtenidos. Obsérvese que los valores de los $\overline{IC}$ utilizando la variante RCO con 4.330 caminos son similares (levemente mejores) que CO con 100.000 caminos en todos los casos. Este resultado sugiere que reúso de caminos (variante RCO) conlleva un costo computacional aproximado de solo el 4,33 % en comparación con la variante CO, para la obtención de resultados similares.

Caso	$\overline{IC}$ (K) de CO con 100.000 caminos	$\overline{IC}$ (K) de RCO con 4.330 caminos
1a	0,0308	0,0306
1b	0,0453	0,0445
2a	2,4132	2,1705
2b	2,2859	2,1497

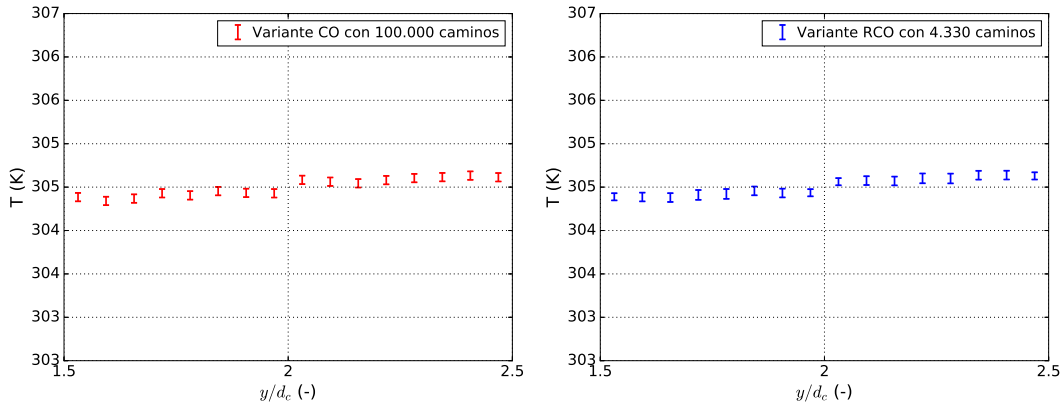
Tabla 4.1: Tabla comparativa de $\overline{IC}$.

En las figuras 4.6, 4.7, 4.8 y 4.9 se puede ver la comparación entre las dos variantes. En el eje horizontal de dichas gráficas se toma el diámetro d_c de una celda Kelvin (4 mm) como base de la escala de los segmentos. Cada gráfica muestra únicamente la región central del modelo para resaltar las posibles diferencias. En los casos 1a y 1b la ampliación es en la región porosa comprendida en el segmento $[1.5, 2.5]$. En los casos 2a y 2b la ampliación es en la región porosa comprendida en el segmento $[0.5, 3.5]$. En todos los casos los resultados de la variante RCO con 4.330 caminos parecen similares a los de la variante CO con 100.000 caminos.



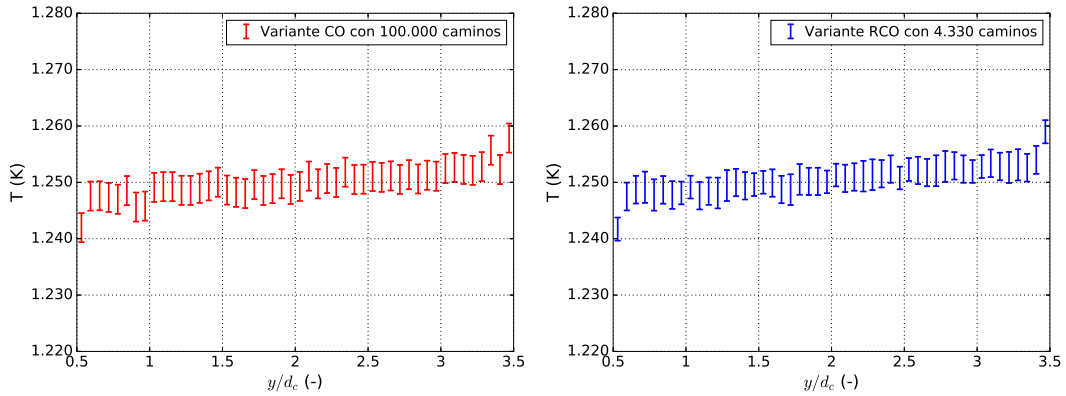
(a) CO utilizando 100.000 caminos. **(b)** RCO utilizando 4.330 caminos.

Figura 4.6: Comparación de las variantes CO vs RCO para el caso 1a.



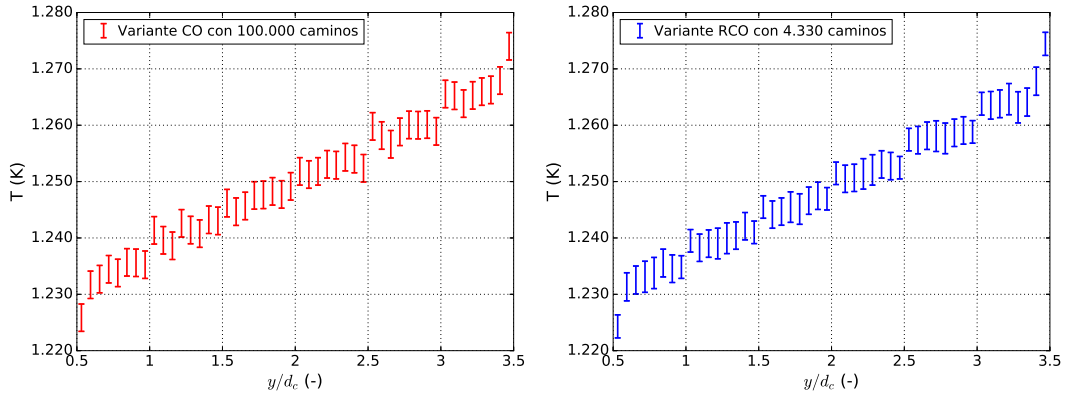
(a) CO utilizando 100.000 caminos. **(b)** RCO utilizando 4.330 caminos.

Figura 4.7: Comparación de variantes CO y RCO para el caso 1b.



(a) CO utilizando 100.000 caminos. **(b)** RCO utilizando 4.330 caminos.

Figura 4.8: Comparación de las variantes CO y RCO para el caso 2a.



(a) CO utilizando 100.000 caminos. **(b)** RCO utilizando 4.330 caminos.

Figura 4.9: Comparación de las variantes CO y RCO para el caso 2b.

4.3. Otras métricas obtenidas

Además de los resultados de las secciones 4.1 y 4.2 se obtuvieron otras medidas que muestran otros aspectos de la implementación planteada, como ser la cantidad de caminos eliminados, y el impacto en el rendimiento del algoritmo al aumentar la cantidad de caminos o al cambiar de GPU.

4.3.1. Estudio de caminos eliminados

En las pruebas realizadas fue observado que ciertos porcentajes de los caminos lanzados fueron eliminados (ver 3.2). Estos porcentajes se mantienen relativamente constantes para cada uno de los casos. Los porcentajes según el caso estudiado se muestran en la tabla 4.2.

Caso	% (aprox.)
1a	10,2
1b	4,4
2a	13,7
2b	4,8

Tabla 4.2: Tabla de porcentaje de caminos eliminados.

Una posible explicación del alto número de caminos eliminados para los casos 1a y 2a es que muchos de ellos llegan al límite de largo de camino sin haber llegado a una cara con temperatura conocida.

En el caso 1a, en donde la probabilidad de conducción es muy alta, los caminos deben recorrer el sólido viajando dentro de los tubos; dichos caminos deben recorrer el "laberinto" de la zona porosa hasta llegar a una cara con temperatura conocida. En el caso 2a se da la situación opuesta; la probabilidad de radiación es muy alta, y rara vez ingresa dentro del sólido. La única forma en la que un camino termina es ingresando al sólido y llegando a una cara con temperatura conocida. Debido a que es poco probable que un camino ingrese, el mismo "rebota" un gran número de veces, llegando a superar el límite de largo de camino. Para los otros dos casos, el valor de p_{diff} da un porcentaje de caminos eliminados menor. Hay buena probabilidad de radiación y conducción lo que permite llegar más fácilmente a una cara con temperatura conocida.

4.3.2. Relación entre el tiempo de ejecución y el número de caminos

Otro aspecto que se puede analizar de esta técnica es el tiempo necesario para obtener resultados con la variante RCO; comparado con la variante CO se obtuvieron en promedio diferencias de tiempo de sólo un 2 %, para simulaciones con la misma cantidad de caminos. Esto confirma el bajo impacto de los cómputos adicionales de RCO. Por otra parte, se tiene una relación casi lineal entre la cantidad de caminos calculados y el tiempo empleado. La pendiente dependerá de varios factores, por ejemplo, en la probabilidad de conducción, el modelo de la GPU utilizada, etc.

Las gráficas de las figuras 4.10 y 4.11 se obtuvieron utilizando una tarjeta de video Nvidia GeForce GTX 780 (NVIDIA (2013)).

En la figura 4.10 puede verse que los tiempos utilizando 4.330 caminos son

sustancialmente menores que utilizando 100.000 caminos. También se observa una tendencia lineal entre la cantidad de caminos y el tiempo utilizado para el cálculo.

En la figura 4.11 se muestran ejecuciones distanciadas de a 200.000 caminos entre ellas hasta llegar a 1.000.000 caminos. Se realizaron ejecuciones con grandes números de caminos para poder observar la tendencia lineal del tiempo de cálculo.

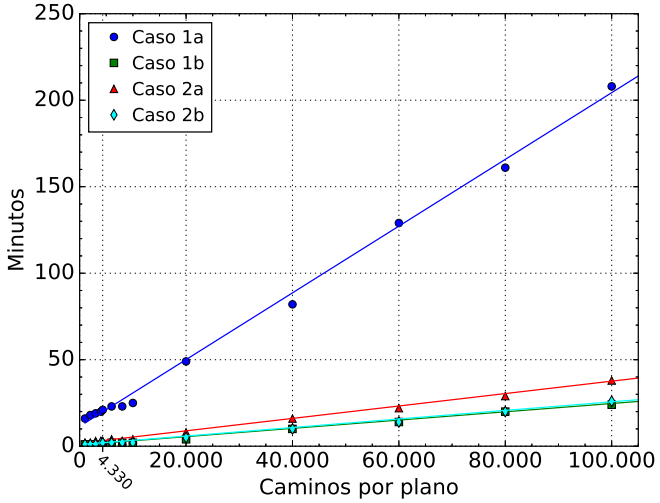


Figura 4.10: Gráfica comparativa de tiempos entre 1.000 y 100.000 caminos. Se muestran los datos experimentales junto con su recta de ajuste.

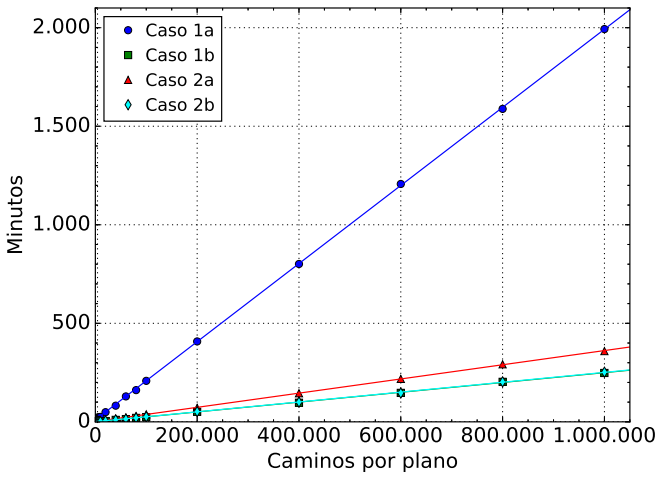


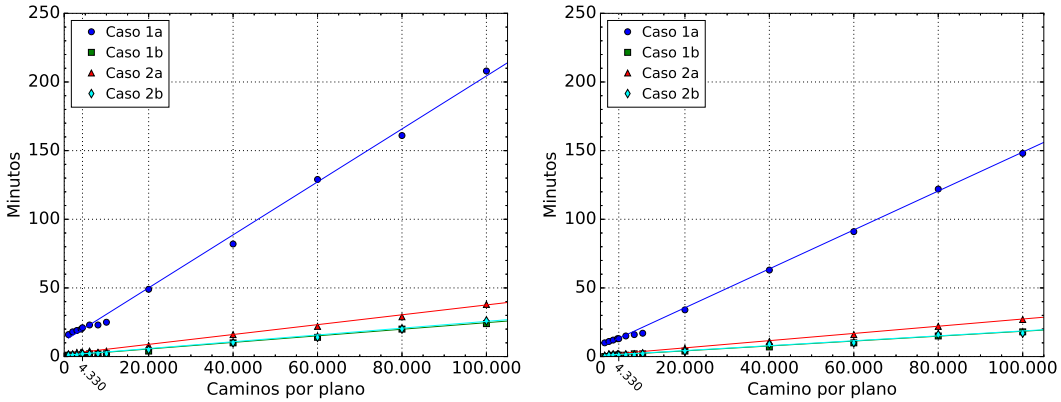
Figura 4.11: Gráfica comparativa de tiempos entre 1.000 y 1.000.000 caminos. Se muestran los datos experimentales junto con su recta de ajuste.

En las gráficas se puede ver que el caso 1a es el más lento. Como se comentó en la sección 4.3.1, debido a que los caminos deben recorrer el sólido a través de los tubos y dando saltos muy chicos, éstos demoran mucho en llegar a una fuente de temperatura. Sin embargo, el resto de los casos de estudio terminan su ejecución bastante rápido.

Es interesante destacar el caso 2a, que pese a ser el caso con menor valor de p_{diff} de todos, no necesariamente es el más rápido.

Comparación de tiempos de ejecución con diferentes tarjetas

En las gráficas de la figura 4.12 se comparan los diferentes tiempos de cómputo de los casos de estudio utilizando las tarjetas de video Nvidia GeForce GTX 780 (NVIDIA (2013)) y Nvidia GeForce GTX 970 (NVIDIA (2014)). El speedup de la tarjeta Nvidia GeForce GTX 970 es de aproximadamente 1,37.



(a) Nvidia GeForce GTX 780 **(b)** Nvidia GeForce GTX 970

Figura 4.12: Comparación de tiempos entre la tarjeta Nvidia GeForce GTX 780 y la tarjeta Nvidia GeForce GTX 970.

Modelo	GTX 780	GTX 970
Año	2013	2014
Arquitectura	Kepler	Maxwell
# CUDA Cores	2.304	1.664
Base Clock (MHz)	863	1.050
Memoria total (MB)	3.072	4.096
Memory Clock (Gbps)	6,0	7,0

Tabla 4.3: Tabla de especificaciones de tarjetas Nvidia utilizadas. En todos los casos la memoria es de tipo DDR5.

En la tabla 4.3 se presentan las especificaciones de los modelos de tarjetas Nvidia que se utilizaron para realizar los análisis de tiempos de ejecución.

En todos los casos se observa una mejora de los tiempos al emplear tarjetas más modernas, debido a la mejora de su arquitectura, clock y memoria.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

En este trabajo se implementó un algoritmo para el cálculo del intercambio de calor utilizando el método de Monte Carlo. Para su implementación se utilizó la biblioteca de trazado de rayos OptiX.

Los resultados obtenidos muestran que con CO se obtienen ICs similares a los obtenidos en el trabajo de Caliot *et al.* (2018). Para obtener con RCO los mismos intervalos de confianza que los obtenidos con CO, alcanzó con ejecutar el 4,3 % de los caminos. Por otra parte, como el costo promedio del cálculo de caminos es similar en ambas variantes, el menor número de caminos implica una reducción de 23× en los costos de cómputo. Por último, para 100.000 caminos se observaron en RCO intervalos de confianza 4,7 veces menores a los obtenidos en CO. El desarrollo de RCO en GPU muestra las mejoras en tiempo de cómputo de implementar MCRT en GPU para el caso estudiado.

Se observó una aceleración de aproximadamente 1,37× al utilizar GPUs con arquitecturas más modernas. Dicho desarrollo es escalable y aplicable a otros casos de uso de MCRT. Si bien la utilización de GPUs para acelerar el algoritmo de ray tracing ha sido objeto de estudio en los últimos años, los métodos estudiados se aplican generalmente en iluminación global o en temas de computación gráfica. Este trabajo demostró que las GPUs también permiten acelerar ray tracing en el contexto de simulación de transferencia de calor en geometrías complejas, particularmente para modelar la conducción de calor dentro de cuerpos sólidos.

5.2. Trabajo futuro

El trabajo presentado consiste en un análisis unidimensional de la transferencia de calor. Un posible trabajo futuro es expandir dicho análisis a tres dimensiones. De esta forma se tienen bloques tridimensionales, cada uno con un valor de temperatura representativo de su región. Con esto se mejora la visualización del interior del sólido tomando cortes transversales para ver la distribución de calor. Dado que los requerimientos de memoria podrían ser elevados, el uso de otras estructuras de datos más compactas (por ejemplo, octree (Vörös (1997))) que aprovechen las características espaciales del modelo de la sección (por ejemplo, si tiene mucho espacio vacío). El impacto de dicha implementación sería grande, debido a que afectaría a todas las rutinas de cálculo de caminos y conteo de temperaturas.

En la parte de visualización de los resultados, al tratarse de información tridimensional puede optarse por el formato usado por algún paquete de tomografía (generando imágenes de cada plano sagital de la sección). Otra manera puede ser la generación de un modelo 3D con los vóxeles coloreados según su temperatura, visualizable por un gran conjunto de programas de manejo 3D.

Otro caso de estudio sería agregar múltiples fuentes de calor lo que permitiría aplicarlo a casos más cercanos a la realidad. A su vez sería interesante utilizar un gradiente continuo de temperaturas en lugar de usar un conjunto discreto de temperaturas conocidas. En este caso se debe decidir adecuadamente la estimación de T_{ref}, la temperatura global conocida del sólido en estado estacionario.

Dado que el algoritmo desarrollado considera únicamente los fenómenos de conducción y radiación, sería importante introducir la simulación de convección al sistema. Para ello se podría incorporar la técnica conocida como red de Boltzmann (Guo and Zhao (2005)) que se utiliza comúnmente para resolver los cálculos de convección en transferencia de calor.

Lista de símbolos

δ_b	Largo infinitesimal usado en los cálculos de la frontera vacío–sólido, m
δ_{diff}	Largo infinitesimal usado en los cálculos dentro del sólido definido como $\delta_b/2$, m
λ	Conductividad térmica, W/mK
ε_h	Coeficiente de emisividad esférico
p_{diff}	Probabilidad de difusión
σ	Constante de Stefan–Boltzmann, $\approx 5,670373 \times 10^{-8}\ W/m^2K^4$
h	Coeficiente de transferencia de calor de convección, W/m^2K

Lista de figuras

Lista de tablas

Referencias bibliográficas

Arambakam, R., Tafreshi, H. V., and Pourdeyhimi, B. (2012). Analytical monte carlo ray tracing simulation of radiative heat transfer through bimodal fibrous insulations with translucent fibers. *International Journal of Heat and Mass Transfer*, **55**, 7234–7246.

Bejan, A. (1993). *Heat Transfer*. John Wiley & Sons. ISBN: 0-471-59952-2.

Blender (2017). Blender. https://www.blender.org/. Último acceso: Agosto del 2019.

Caliot, C., Blanco, S., Coustet, C., Hafi, M. E., Eymet, V., Forest, V., Fournier, R., and Piaud, B. (2018). Combined conductive–radiative heat transfer analysis in complex geometry using the monte carlo method. In *Eurotherm Seminar 110 – Computational Thermal Radiation in Participating Media – VI, At Cascais, Portugal*.

Canavos, G. C. (1998). *Probabilidad y estadística. Aplicaciones y métodos*. McGraw–Hill Interamericana de México, S.A. de C.V. ISBN: 968-451-856-0.

Csébfalvi, B. (1997). A review of monte carlo ray tracing methods. http://old.cescg.org/CESCG97/csebfalvi/index.html. Último acceso: Agosto del 2019.

Duarte Santos, P. and Lani, A. (2016). An object–oriented implementation of a parallel monte carlo code for radiation transport. *Computer Physics Communications*, **202**, 233–261.

Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. (1996). *Introducción a la graficación por computador*. Addison–Wesley Iberoamericana. ISBN: 0-201-62599-7.

Fournier, R., Blanco, S., Eymet, V., Hafi, M. E., and Spiesser, C. (2016). Radiative, conductive and convective heat–transfers in a single monte carlo algorithm. In *Eurotherm Conference 105 – Computational Thermal Radiation in Participating Media V*.

Guo, Z. and Zhao, T. S. (2005). A lattice boltzmann model for convection heat transfer in porous media. *Numerical Heat Transfer, Part B: Fundamentals*, **47**, 157–177.

Incropera, F. P., DeWitt, D. P., Bergman, T. L., and Lavine, A. S. (2006). *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons. ISBN: 0-471-45728-0.

Intel Embree (2018). Embree. https://github.com/embree/embree. Último acceso: Agosto del 2019.

Jacques, L., Masset, L., and Kerschen, G. (2015). Direction and surface sampling in ray tracing for spacecraft radiative heat transfer. *Aerospace Science and Technology*, **47**, 146–153.

Kramer, S., Gritzki, R., Perschk, A., Rösler, M., and Felsmann, C. (2015). Numerical simulation of radiative heat transfer in indoor environments on programmable graphics hardware. *International Journal of Thermal Sciences*, **96**, 345–354.

Kroese, D. P., Brereton, T., Taimre, T., and Botev, Z. I. (2014). Why the monte carlo method is so important today. *WIREs Computational Statistics*, **6**, 386–392.

Kuczynski, P. and Bialecki, R. (2014). Radiation heat transfer model using monte carlo ray tracing method on hierarchical ortho–cartesian meshes and non–uniform ational basis spline surfaces for description of boundaries. *Archives of thermodynamics*, **35**(2), 65–92.

Lewis, R. W., Nithiarasu, P., and Seetharamu, K. N. (2004). *Fundamentals Of The Finite Element Method For Heat And Fluid Flow*. John Wiley & Sons. ISBN: 0-470-84788-3.

Matplotlib (2019). Matplotlib. https://matplotlib.org/. Último acceso: Agosto del 2019.

Mitsuba (2018). Mitsuba documentation. https://www.mitsuba-renderer.org/releases/current/documentation.pdf. Último acceso: Agosto del 2019.

Morton, G. M. (1966). A computer oriented geodetic data base; and a new technique in file sequencing. https://domino.research.ibm.com/library/cyberdig.nsf/papers/0DABF9473B9C86D48525779800566A39/$File/Morton1966.pdf. Último acceso: Agosto del 2019.

NVIDIA (2013). GeForce GTX 780. Especificaciones técnicas. https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-780. Último acceso: Agosto del 2019.

NVIDIA (2014). GeForce GTX 970. Especificaciones técnicas. https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-970. Último acceso: Agosto del 2019.

NVIDIA cuRAND (2018). The API reference guide for cuRAND, the CUDA random number generation library. https://docs.nvidia.com/cuda/curand/index.html. Último acceso: Agosto del 2019.

NVIDIA DesignWorks Samples (2018). Optix advanced samples. https://github.com/nvpro-samples/optix_advanced_samples. Último acceso: Agosto del 2019.

NVIDIA OptiX (2018). Nvidia optix programming guide. http://raytracing-docs.nvidia.com/optix/guide/index.html#guide#. Último acceso: Agosto del 2019.

Perraudin, D. Y. and Haussener, S. (2017). Numerical quantification of coupling effects for radiation–conduction heat transfer in participating macroporous media: Investigation of a model geometry. *International Journal of Heat and Mass Transfer*, **112**, 387–400.

Peterson, B., Humphrey, A., Holmen, J., Harman, T., Berzins, M., Sunderland, D., and Edwards, H. C. (2018). Demonstrating gpu code portability and scalability for radiative heat transfer computations. *Journal of Computational Science*, **27**, 303–319.

Pharr, M., Jakob, W., and Humphreys, G. (2018). *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann. ISBN: 978-0-12-800709-9.

Python (2019). Python. https://www.python.org/. Último acceso: Agosto del 2019.

Ravishankar, M., Mazumder, S., and Sankar, M. (2010). Application of the modified differential approximation for radiative transfer to arbitrary geometry. *Journal of Quantitative Spectroscopy & Radiative Transfer*, **111**, 2052–2069.

Rong, F., Zhang, W., Shi, B., and Guo, Z. (2013). Numerical study of heat transfer enhancement in a pipe filled with porous media by axisymmetric tlb model based on gpu. *International Journal of Heat and Mass Transfer*, **70**, 1040–1049.

Simon, C. (2015). Generating uniformly distributed numbers on a sphere. http://corysimon.github.io/articles/uniformdistn-on-sphere/. Último acceso: Agosto del 2019.

Soucasse, L., Rivière, P., and Soufiani, A. (2012). Monte carlo methods for radiative transfer in quasi–isothermal participating media. *Journal of Quantitative Spectroscopy & Radiative Transfer*, **128**, 34–42.

Szénási, S. and Felde, I. (2016). Heat transfer simulation using gpus. In *2016 IEEE 20th Jubilee International Conference on Intelligent Engineering Systems (INES)*.

Vignoles, G. L. (2015). A hybrid random walk method for the simulation of coupled conduction and linearized radiation transfer at local scale in porous media with opaque solid phases. *International Journal of Heat and Mass Transfer*, **93**, 707–719.

Vörös, J. (1997). A strategy for repetitive neighbor finding in images represented by quadtrees. *Pattern Recognition Letters*, **18**(10), 955–962.

Weisstein, E. W. (2013). Wolfram: Sphere Point Picking. http://mathworld.wolfram.com/SpherePointPicking.html. Último acceso: Agosto del 2019.

ANEXOS

Anexo A

Descripción del ejemplo de photon mapping de OptiX

Aquí se describe el ejemplo usado de base para este proyecto.
Este ejemplo de photon mapping emplea tres etapas:

- Una que lanza los rayos desde la cámara.
- Otra que calcula los fotones para cargarlos en un mapa.
- La fase de gather que usa los datos de las anteriores para renderizar la escena.

Abajo se detallan las rutinas comenzando por el programa principal y luego las rutinas de los kernels.

A.1. Módulo principal

Se encarga de recibir los parámetros e inicializar las estructuras de OptiX para luego llamar las rutinas en GPU y mostrar la escena en la GUI al usuario, permitiendo el control de la misma en tiempo real.

Función *createContext*

- Crea el contexto de OptiX para la escena.
- Busca una segunda GPU para hacer los cálculos liberando la primera para el despliegue.
- Ajusta algunos parámetros del contexto (programas y variables de estado).

- Prepara los buffers para la transferencia de datos entre el Host y la GPU.
- Carga los valores de semilla para las variables aleatorias.
- Asigna las rutinas para generación de rayos y para los eventos de excepción, error de color y de no intersección.

Función *createGeometry*

Carga la geometría de la escena y asigna las rutinas de los eventos de intersección.

Función *createPhotonMap*

Carga los fotones generados en GPU en un árbol KD (estructura que organiza espacialmente los datos, permitiendo una eficiente búsqueda por localización espacial) para crear el mapa de fotones.

Función *buildKDTree*

Organiza un árbol KD, ofreciendo 3 posibles estrategias de organización del árbol:

1. Rotando el eje de corte entre X, Y y Z.
2. Tomando el eje de mayor varianza.
3. Tomando el eje más grande.

Función *launch_all*

- Traza los rayos de vista de la cámara.
- Traza los fotones y crea el mapa de fotones.
- Realiza el renderizado usando el mapa de fotones y los rayos de la cámara.

Función *glfwRun*

Gerencia la GUI.

Función *main*

- Procesa los parámetros de la línea de comando.

- Inicializa la interfaz de usuario.
- Crea el contexto y ajusta los parámetros de visualización (cámara y luces).
- Lanza el cálculo de la escena y la despliega (o la guarda en un archivo).

A.2. Kernel *RT_pass*

Lanza los rayos de la cámara para obtener su color, calculando sus reflejos.

Rutina *rtpass_camera*

Lanza un rayo desde la cámara.

Rutina *rtpass_closest_hit*

Registra el color para el rayo cuando choca contra una luz o un objeto difuso.

Lanza un nuevo rayo en caso de reflejo.

Rutina *rtpass_miss*

Toma el color de la imagen de fondo de la escena para los rayos que no intersecan la geometría.

A.3. Kernel *Photon_pass*

Genera los fotones desde las fuentes de luz.

Rutina *ppass_camera*

Inicializa los fotones y genera los rayos para calcularlos.

Rutina *ppass_closest_hit*

Registra el fotón cuando choca con un objeto y calcula su rebote al azar en una hemiesfera.

A.4. Kernel *Gather_pass*

Usa el mapa de fotones para calcular el color de un punto de la escena, incluyendo el cálculo de sombras.

Rutina *gather*

- Acumula los fotones usando el mapa de fotones.
- Calcula el flujo de luz indirecto.
- Calcula el flujo de luz directo y su atenuación.
- Traza un rayo para calcular la sombra.
- Calcula el color del punto de la escena.

Rutina *gather_any_hit*

Usado para terminar el rayo de sombra.

A.5. Kernel *sphere*

Usado desde OptiX para calcular la intersección con una esfera.

A.6. Kernel *triangle_mesh*

Usado desde OptiX para calcular la intersección con un triángulo de la malla de la geometría.

Anexo B

Evolución del proyecto

El trabajo presentado es el resultado de la evolución del desarrollo del proyecto, principalmente en lo que tiene que ver al procesamiento de la información de temperatura de los caminos.

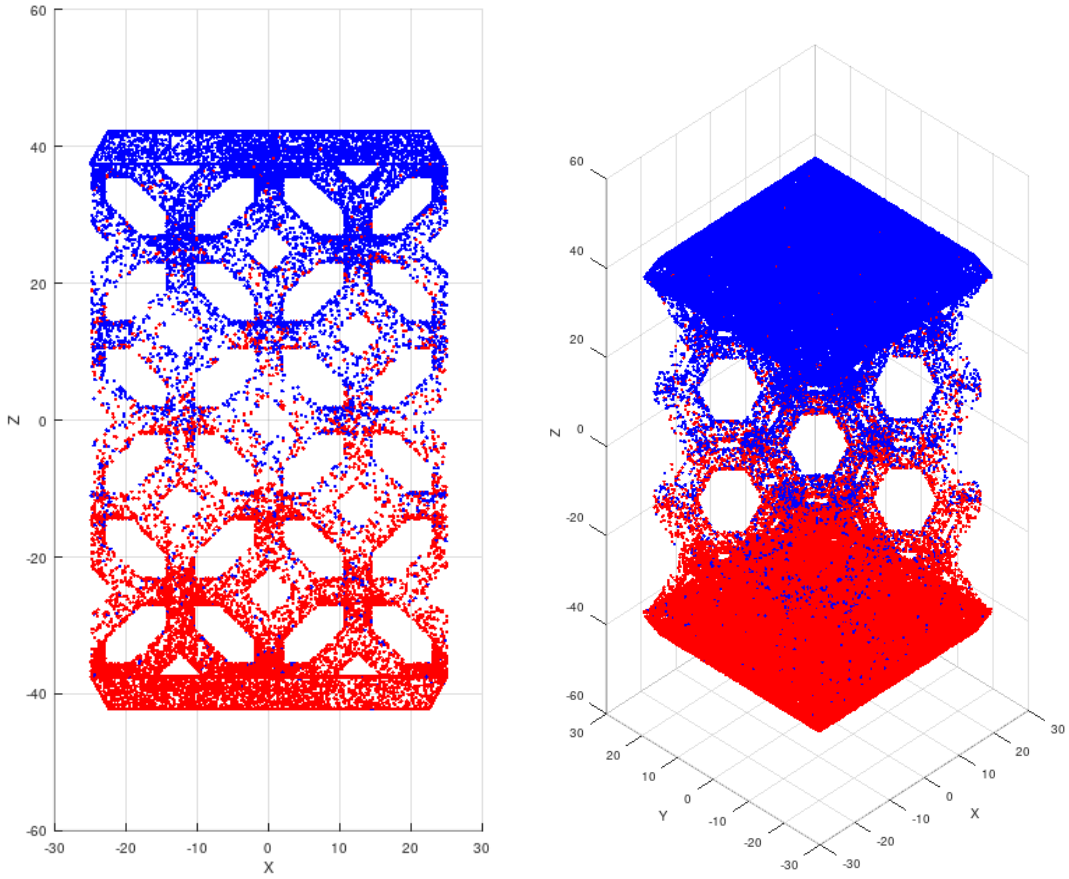
B.1. Generación de nube de puntos

Para calcular los promedios de temperatura de cada plano se guardaba la información de cada punto de intersección de los caminos con dicho plano, formando una nube de puntos. La información que se guardaba era la posición y la temperatura alcanzada. Luego se fraccionaba la nube según cada plano y se hacía el promedio correspondiente para graficar la distribución de temperatura. Eso permitía también graficar en 3D la forma de la nube de puntos para tener idea de la distribución de caminos, útil para depurar problemas geométricos con el modelo 3D, como se muestra en la figura B.1.

Se utilizaron dos enfoques para generar la nube de puntos:

1. Generar puntos al azar en los planos de temperatura conocida, generando un camino aleatorio cargando directamente la temperatura en cada punto hasta alcanzar el largo máximo de camino o un plano generador.
2. Generar puntos al azar en cualquier parte del volumen del sólido, y luego construir un camino aleatorio desde ellos hasta encontrar un plano de temperatura conocida. Cuando el camino finaliza se carga dicha temperatura a todos los puntos del camino.

En ambos enfoques se avanza según los códigos vistos en 2.3.4. Se esperaba que ambos enfoques dieran resultados similares.



(a) Vista frontal **(b)** Visa superior a 45°

Figura B.1: Nube de Puntos. Nótese la silueta de las celdas kelvin.

El segundo método es más cercano en esencia al método de Caliot *et al.* (2018), pero es ineficiente al desperdiciar tiempo y procesamiento en caminos que no llegan a temperatura conocida. Sin embargo, dicha ineficiencia pudo solventarse permitiendo caminos tan largos que eventualmente alcanzan una temperatura conocida. Por eso fue el seleccionado para el desarrollo final.

B.1.1. Z-Order

Se estudió la posibilidad de una mejora de rendimiento utilizando Z-Order. Esta técnica consiste en agrupar los orígenes espacialmente ordenándolos utilizando códigos Morton para realizar un ordenamiento de Z-Order (Morton (1966)); al estar cerca se supone que la coherencia espacial de los caminos hace que los hilos correspondientes mejoren los tiempos de cómputo.

Sin embargo, el Z-Order implementado no tuvo un impacto en el rendi-

miento digno de mención. Suponemos que debido a que los caminos aleatorios se van separando con el tiempo se pierde la coherencia espacial; además como se explica en la sección B.2.4, existen optimizaciones internas de OptiX que diluyen el impacto del Z-Order implementado.

B.2. Problemas encontrados al usar GPU

Durante el uso de OptiX para su implementación en GPU se encontraron problemas habituales al desarrollar en GPU, descriptos a continuación:

B.2.1. Problemas de divergencia

En la arquitectura de NVIDIA CUDA los hilos de ejecución se agrupan en conjuntos lógicos llamados *warps*. Cuando se desea ejecutar una secuencia de instrucciones de código de un kernel la misma instrucción es enviada a todos los hilos de un warp, y ellos los ejecutan sincronizadamente.

La divergencia dentro de la GPU ocurre cuando algunos hilos de un warp se ramifican en la línea de ejecución (por ejemplo, debido a sentencias condicionales o iteraciones con límites diferentes). Debido a que los hilos deben ejecutar la misma instrucción el planificador separa los hilos de cada rama de ejecución en subconjuntos diferentes, y luego despacha cada conjunto separadamente. Esto provoca que los hilos de los subconjuntos queden esperando su turno a ejecutar y así se desperdicia capacidad de cálculo.

En nuestra implementación esto ocurría cuando se tenían las etapas de generación de orígenes y cálculo de caminos combinadas en la misma rutina. Algunos hilos de ejecución demoraban en la generación de orígenes haciendo esperar a los que habían terminado esa etapa para poder continuar.

Otra fuente de divergencias ocurría debido a que los hilos eran asignados en secuencia uno para cada plano de división del sólido. Esto crea una carga de trabajo diferenciada para los diferentes hilos de un mismo warp, ya que los hilos que estaban cerca de las fuentes de temperatura tenían menos trabajo que los hilos que tienen que recorrer la zona porosa.

Se pudo reducir en gran medida el problema de divergencia asignándole a los hilos de un mismo warp los caminos cuyos puntos de origen se encuentran en el mismo plano. De esta manera en promedio todos los hilos tienen la misma cantidad de trabajo y hay menos hilos esperando la finalización de otros. De

todas maneras siempre van a haber cierto porcentaje de hilos que divergen debido a la aleatoriedad del método de Monte Carlo.

B.2.2. Saturación de stack (pila de ejecución)

El stack es un área de memoria de tamaño limitado para almacenar datos de retorno de una rutina. Es muy exigida en la ejecución de algoritmos recursivos como el Ray Tracing.

En una de las primeras versiones de la implementación el estado actual de un camino se guardaba en el stack, siguiendo la misma idea que el Ray Tracing, pero dada la gran cantidad de puntos que genera un camino resultó insuficiente.

Para resolverlo se cambió el algoritmo a una versión iterativa. En la nueva versión se almacena el estado de los caminos en un vector y se reutiliza la variable del estado actual del camino; no es necesario almacenar el historial de dicho estado ya que sólo interesa saber que planos cortó.

B.2.3. Falta de memoria

Inicialmente se guardaban las coordenadas y la temperatura alcanzada para todos los pasos realizados por un camino (sin considerar los planos de división del sólido). Al principio fue útil para depurar problemas con el modelo 3D (por ejemplo, caminos que se salían del modelo), pero luego esos problemas fueron resueltos aplicando controles de validación de los caminos (como se comenta en la sección 3.2). Además, limitaba la cantidad de caminos según la memoria disponible y ralentizaba seriamente el procesamiento de dicha información para graficarla.

Por ello se decidió calcular parte de la estadística en el módulo principal (el conteo de caminos que pasan por un plano) y los hilos marcan los planos por los que pasan los caminos ya que cada camino tiene asociado un vector de planos acotado. Las coordenadas de los pasos se utilizan internamente en los kernels y no guardan. Eso permitió un aumento dramático en los tiempos de procesamiento y una disminución considerable de los requerimientos de memoria, permitiendo generar muchos más caminos y más largos.

B.2.4. NVIDIA Visual Profiler

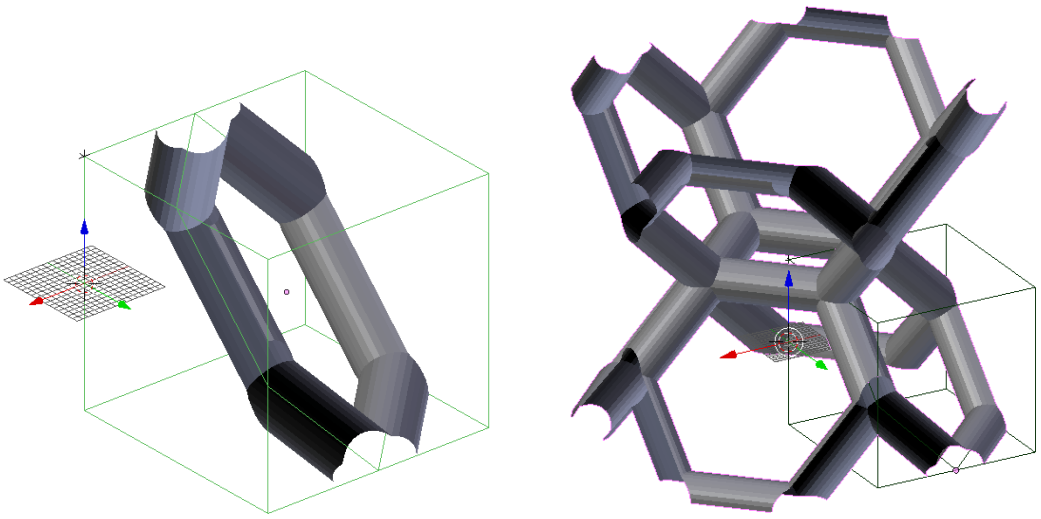
Se utilizó la herramienta NVIDIA Visual Profiler para analizar posibles mejoras en el rendimiento de los diferentes kernels, pero no se obtuvieron datos relevantes. Al abstraer la utilización de CUDA, OptiX implementa un gran número de subrutinas y kernels que son difíciles de analizar por separado.

Entre las rutinas que implementa OptiX están el cálculo de la jerarquía de volúmenes acotantes (BVH de sus siglas en inglés) usado para acelerar el cálculo de intersecciones con modelos 3D, y el ordenamiento de los caminos utilizando Z-Order (B.1.1), el cual agrupa los caminos espacialmente y ofrece una mejora de rendimiento de las GPUs.

Anexo C

Modelado 3D de la Celda Kelvin

Para hacer los resultados comparables al trabajo de Caliot *et al.* (2018), se recreó la geometría usada en dicho trabajo. Para generar los modelos 3D se utilizó el software Blender (Blender (2017)) con una escala de 0.1 mm (décimas de milímetro) por unidad. De esta manera la sección de la pared tiene un largo y alto de 80 unidades cada una, y un grosor de 160 unidades. Las secciones sólidas tienen un grosor de 20 unidades cada una, y la zona hueca tiene un grosor de 120 unidades. El diámetro de los tubos de la zona porosa es de 5 unidades.



(a) Hexágono Base **(b)** Celda Base

Figura C.1: Fases de construcción de la celda kelvin base de la zona porosa.

La parte más complicada fue la creación de las celdas Kelvin de la zona porosa. Para crearlas, se partió del hexágono formado por la unión de los

puntos medios de las aristas de un cubo (figura C.1a). Usando dicho hexágono se colocaron los tubos de la celda kelvin, recortándolos a las caras del cubo.

Luego se aplicaron simetrías en los 3 ejes, logrando formar una celda kelvin.

Entonces se multiplicaron las celdas en una matriz de 2 x 2 x 3 y se agregaron las tapas interiores. Este objeto junto con la caja forman la geometría

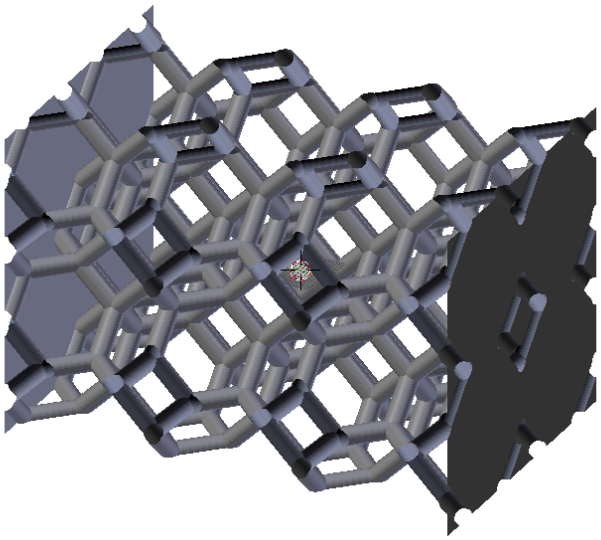


Figura C.2: Celdas kelvin con tapas interiores.

utilizada.

Notar que las normales de los polígonos que forman las tapas apuntan hacia la zona vacía igual que los polígonos de los tubos.

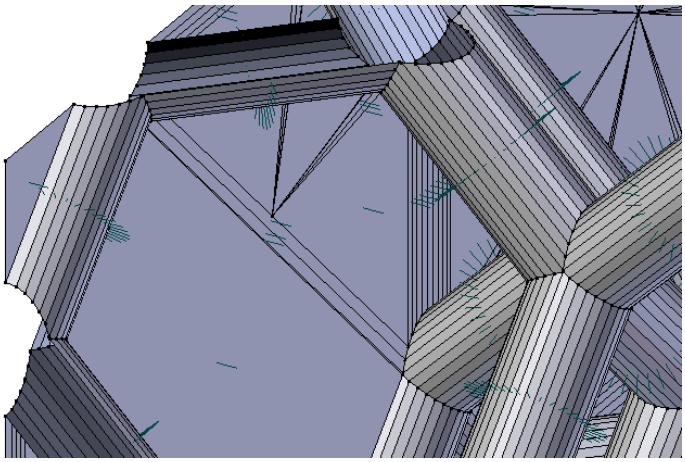


Figura C.3: Normales de los polígonos (en verde).

Anexo D

Instalación y uso

D.1. Instalación

Los requisitos de instalación son los mismos que los del ejemplo de OptiX de photon mapping en que está basado. La instalación puede hacerse desde las fuentes, utilizando las mismas instrucciones que el ejemplo de photon mapping, siempre y cuando se respete la estructura de directorios. La geometría de la pared y la caja están en el directorio *(directorio de instalación)/src/assets*. La geometría de la caja está en el archivo *bbox_80x80x160.obj*.

D.2. Uso

El uso de la solución comprende el programa *optixProgressivePhotonMap* para generar los archivos de resultados, y del script *plot.py* para la generación de las gráficas.

optixProgressivePhotonMap

Se ejecuta mediante el comando:

`optixProgressivePhotonMap [`*`parámetros`*`]`

donde los parámetros pueden ser (se indica el *valor por omisión*):

`-pl, --path-length` *1500000*

Largo máximo del camino, en pasos.

`-i, --numero-iteraciones` *1*

Número de iteraciones del algoritmo sobre todos los planos.

`-nh, --numero-hilos` *4330*
: Cantidad de hilos a lanzar, que se corresponde también con la cantidad de caminos a calcular.

`-db, --delta-b` *1*
: Valor de δ_b.

`-tmin, --tmin` *300*
: Valor de temperatura mínima.

`-tmax, --tmax` *310*
: Valor de temperatura máxima.

`-e, --emisividad` *0.85*
: Valor de emisividad ϵ_h.

`-l, --lambda` *40*
: Valor de λ.

`-du, --densidad-uniforme` *(no usar)*
: Activa el uso de densidad uniforme.

`-z, --use-zorder` *(no usar)*
: Activa el uso del ordenamiento Z.

`-on, --object-name` *kelvinCell_paper.obj*
: Nombre del archivo de geometría a cargar.

Su ejecución genera dos archivos en el directorio actual cuyos nombres son de la forma:

```
lambda_difusion_duX_zY_numero-hilos_object_nameNumero. \
    txt
```

```
lambda_difusion_duX_zY_numero- \
    hilos_cyril_object_nameNumero.txt
```

para los resultados generados usando el algoritmo mejorado y el de Caliot *et al.* (2018) respectivamente.

`Numero` es un número al azar usado para separar las diferentes corridas que utilizan los mismos parámetros.

Ejemplo:

```
optixProgressivePhotonMap -i 1 -nh 100000 -pl  \
    1500000 -db 1 -tmin 1000 -tmax 1500 -e 0.85 -- \
    lambda 0.001 -on kelvinCell.obj
```

Invoca al programa fijando varios valores de los parámetros (en este ejemplo coinciden con los valores por defecto), generando los archivos:

```
0.001_0.646411_du0_z0_4330_kelvinCell.obj1762505870.txt
```

```
0.001_0.646411_du0_z0_100000_cyril_kelvinCell. \
    obj1612651852.txt
```

plot.py

Se invoca en el mismo directorio que contiene los archivos de resultados, generando las gráficas en formato *png* y *eps*.