Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona (FIB)

Trabajo de Fin de Grado

Jordi Gil González

Análisis del método de renderizado Path Tracing en CPU y GPU

Entrega 1: Contexto y alcance del proyecto

Departament de Ciències de la Computació

Director del Trabajo de Fin de Grado:	Chica Calaf, Antoni
Tutor de GEP:	Barrabes Naval, Fernando
Programa de estudio:	Computación
Especialización:	Computación Gráfica

Curso Académico 2019/2020

7 de octubre de 2019

Índice general

*

CAPÍTULO 1

CONTEXTUALIZACIÓN Y ALCANCE DEL PROYECTO

1.1. Introducción

En la actualidad, las imágenes generadas por ordenador están muy presentes tanto en el entorno profesional como en el lúdico. La creación de imágenes realistas mediante el uso de computadoras se ha convertido en una necesidad a la orden del día. Industrias como el cine o los videojuegos requieren de algoritmos capaces de reproducir el mundo real en un entorno virtual y, si siempre que se pueda, en el menor tiempo posible.

El estudio de métodos que permiten renderizar imágenes realistas no es nuevo. Entre principios y mediados de los años 70 comenzaron a publicarse los primeros artículos científicos en referencia a la simulación de la luz y el color sobre superficies modelos tridimensionales. Para poder entender como funcionan estos métodos debemos tener presente la representación de modelos 3D.

Para representar un modelo 3D se utiliza una ”malla de polígonos”, popularmente conocida como *Mesh* y generalmente estos polígonos suelen ser triángulos. Ésta consiste en un conjunto de vértices conectados por aristas formando caras. Para cada una de estas caras podemos definir un vector normal ortogonal a ésta.

Volviendo a los métodos de cálculo de iluminación, el más simple de todos es el conocido como *Flat Shading*. Para calcular el color de cada una de las caras de nuestra malla solamente tiene en cuenta un vértice de los que la conforman y la normal de ésta, aunque una malla compuesta por triángulos es común utilizar el centroide. El color se interpola

para todos los vértices de la cara utilizando el color calculado al inicio dando así un resultado uniforme para toda la cara. Debido a no tener en cuenta las caras adyacentes esto produce resultados diferentes entre ellas. En la Figura 1.1 podemos observar el efecto generado por este método. Podríamos pensar que añadir más vértices a nuestra malla los resultados mejorarían, pero no es una propuesta adecuada debido al mayor uso de memoria requerido y el problema no quedaría resuelto. Si hiciéramos *zoom in* en el modelo, volveríamos a apreciar el efecto conocido como ”bandas de Mach” (Lotto et al., 1999).

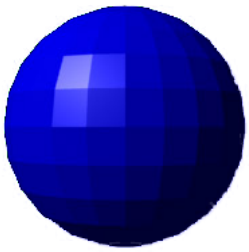

Figura 1.1: Ejemplo de *Flat Shading*. Fuente: Wikipedia

En los métodos de sombreado suave (*Smooth Shading*), el color cambia de pixel a pixel y no de cara a cara, resultando así en transiciones suaves entre las diferentes caras adyacentes. En 1971 Henri Gouraud nos presenta en su artículo *Continuous Shading of Curved Surfaces* (Henri, 1971) el sombreado de Gouraud (*Gouraud Shading*). Este método nos permite añadir mayor continuidad al sombreado respecto al método de *Flat Shading*. El gran avance respecto al método presentado anteriormente es que no precisa de una malla de gran densidad para lograr simular una mayor continuidad. Para cada pixel se determina su intensidad por interpolación de las intensidades definidas en los vértices de cada polígono.

- Para cada vértice se define una normal como promedio de las normales de los polígonos a los que pertenece dicho vértice.
- Mediante el uso de algún modelo de iluminación como, por ejemplo, el modelo de reflexión de Phong, se calcula la intensidad de cada vértice utilizando la normal obtenida en el punto anterior.
- Para cada pixel, se interpola la intensidad en los vértices para obtener la intensidad de éste.

Como podemos observar en la Figura 1.2, los resultados obtenidos respecto el método anterior son notablemente superiores, pero no acaba de representar de forma correcta los reflejos especulares. Éstos puede suponer un problema grave si se presentan en el centro de un polígono (cara) de gran tamaño.

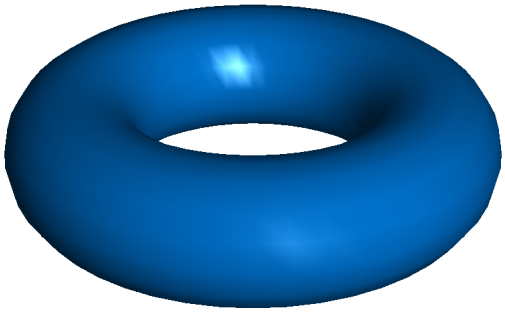

Figura 1.2: Ejemplo de *Gouraud Shading*. Fuente: Wikipedia

Más tarde, Bui Tuong Phong en su tesis doctoral (Phong, 1975) nos presentaba el sombreado de Phong. En el método presentado por Phong en vez de calcular la intensidad en el vértice, primero se define la normal de éste, se interpola y normaliza para cada pixel y es entonces cuando haciendo uso de algún modelo de iluminación se determina la intensidad final. El modelo resulta más costoso computacionalmente, debido a que el calculo se hace a nivel de fragmento (pixel) y no a nivel de vértice.

Phong menciona en su artículo publicado por la ACM (Phong, 1975, p. 311) que su objetivo no era simular la realidad, sino más bien añadir cierto grado de realismo:

> "*In trying to improve the quality of the synthetic images, we do not expect to be able to display the object exactly as it would appear in reality, with texture, overcast shadows, etc. We hope only to display an image that approximates the real object closely enough to provide a certain degree of realism.*"

A pesar de que estos métodos supusieron, en lo concerniente al realismo, un avance, no pretenden simular la realidad. Además, éstos solamente tienen en cuenta la luz ambiente, difusa y especular. No tienen presente la iluminación indirecta de la escena, factor importante a la hora de crear imágenes que produzcan efectos realistas tales como, por ejemplo, reflejos.

No fue hasta los años 80 en que aparecieron los primeros métodos capaces de renderizar imágenes realistas. Turner Whitted nos presentaba en la sexta conferencia anual sobre *Computer graphics and interactive techniques (SIGGRAPH)* el método de trazado de rayos, conocido popularmente por su nombre en inglés, *Ray Tracing* (Whitted, 1980). Este método está basado en el algoritmo de *Ray Casting*, presentado por (Appel, 1968), consistente en trazar rayos desde el observador, uno por pixel, para determinar cual es el objeto más cercano. Además, una vez el rayo impacta en una superficie, basándonos en las propiedades de los materiales definidos en el objeto y las propiedades de la luz, se calcula el color. Se puede hacer uso de *texture maps* para simular efectos como sombras.

En 1986, David Immel et al. y James T. Kajiya, investigadores de la Cornell University y del California Institute of Technology (Caltech) respectivamente, presentaban de forma conjunta, en la décima tercera conferencia anual sobre *Computer graphics and interactive techniques (SIGGRAPH)*, la *Rendering Equation* (Kajiya, 1986; Immel et al., 1986). Dicha ecuación integral trata de resumir en una sola fórmula como la luz interactúa cuando impacta con una superficie haciendo uso de funciones probabilísticas llamadas "función de distribución de la reflectividad bidireccional", (BRDF por sus siglas en inglés). Ésta también tiene presente el ángulo en el que incide el rayo, la cantidad de fotones que llegan, los fotones emitidos desde otros puntos de la escena (iluminación indirecta), etc.

Otros métodos que nos permiten generar imágenes realistas a partir de calcular aproximaciones de la RE son: *Bidirectional Path Tracing* presentado por (Lafortune et al., 1993), *Photon Mapping* formulado por (Jensen, 1996) y *Metropolis light Transport* introducido por (Veach et al., 1997).

1.2. Contextualización

1.2.1. Contexto

Durante los estudios del grado, son varias las asignaturas dedicadas a la computación gráfica. En estas asignaturas se nos presentan los métodos de renderización realistas. Pero más allá de la introducción teórica a éstos, nunca se llegan a poner en práctica. De aquí nace la idea de realizar el presente proyecto, poder profundizar más en el tema de renderización realistas y así crear un aplicación en función a lo aprendido. Se pondrán también en práctica otros aspectos de la informática vistas en otras asignaturas como, por ejemplo, la creación de aplicaciones paralelas tanto en CPU como en GPU.

Como hemos indicado en la sección anterior, la renderización de imágenes realistas es un área de gran interés en el campo de la computación gráfica. Uno de los principales objetivos de ésta es ser capaces de renderizar imágenes indistinguibles de las del mundo real como, por ejemplo, fotografías. Siguiendo estas coordenadas, poder reproducir el comportamiento de la luz en un entorno virtual supone una tarea una importancia capital. Es primordial tener presente la iluminación global de una escena para poder obtener un alto grado de realismo. La iluminación global de una escena se compone de: I) luz directa , II) luz indirecta .

I) La luz directa es aquella que incide en un punto desde el foco de luz.
II) La luz indirecta es aquella que incide en un punto proveniente de la luz que rebota en otros puntos de la escena.

Recuperando lo expuesto en la introducción, el primer método capaz de renderizar imágenes realistas fue el *Ray Tracing*, basado en la técnica de *Ray Casting* de trazar rayos desde el observador a todos los pixeles de la imagen. La gran novedad respecto al algoritmo presentado por (Appel, 1968) es la incorporación de la recursividad. Cuando un rayo impacta contra un superficie puede generar tres tipos de rayos nuevos: I) rayo de reflexión , II) rayo de refracción y III) rayo de sombra. Al trazar los rayos nuevos somos capaces de conseguir efectos como reflejos, sombras, etc. debido a que para calcular el color estamos teniendo en cuenta como los demás objetos de la escena se afectan entre si. Una gran desventaja de este método es la dependencia que éste tiene respecto a los polígonos de la escena; a más compleja es, más ineficiente será. A pesar de ofrecernos un alto grado de realismo al ser capaz de tratar con precisión efectos ópticos como la refracción, reflexión, los resultados obtenidos en una imagen renderizada mediante *Ray Tracing* no son necesariamente foto-realistas. Para conseguir renderizar imágenes foto-realistas debemos aproximar la RE, un buen ejemplo de ello es el método de *Path Tracing* presentado por (Kajiya, 1986).

El algoritmo de *Path Tracing* surge como mejora del *Ray Tracing* con el objetivo de dar una solución a la RE mediante la integración de Monte Carlo. Es gracias a ésto que el algoritmo es capaz, de forma natural, representar efectos como *Motion Blur*, *Ambient Oclussion* e iluminación indirecta sin necesidad de posprocesado. A diferencia del *Ray Tracing*, el *Path Tracing* es indiferente al número de polígonos presentes en la escena. Como hemos mencionado anteriormente, en el primer método se traza un rayo por cada polígono de la escena, en cambio en el método de Kajiya el rayo se traza por pixel. Debido a que cada uno es independiente de los demás, tenemos un algoritmo con una alta capacidad de paralelismo. En consecuencia, podemos explotar la capacidad de concurrencia que nos proporcionan las CPUs y GPUs; y poder así calcular más de un pixel de la imagen al mismo tiempo.

En el presente proyecto trataremos de crear un aplicación que implemente el método de *Path Tracing* para generar imágenes de carácter realista. Dicha aplicación calculará cada pixel de la imagen final de forma secuencia, es decir, uno tras otro y nos servirá de base para crear una solución paralela haciendo uso de la GPU y otra haciendo uso de la CPU. El porque de realizar estas tres aplicaciones es para ser capaces de realizar un análisis del rendimiento que nuestro algoritmo obtiene explorando en profundidad la capacidad de concurrencia que nos ofrecen a día de hoy las CPUs y GPUs; y ver así que arquitectura nos ofrece mayor rendimiento.

Como base, partiremos de la implementación propuesta por Peter Shirley en su ”saga” de libros sobre *Ray Tracing* (Shirley, 2019c; Shirley, 2019b; Shirley, 2019a) para implementar nuestro *Path Tracing*. La idea principal es desarrollar tres aplicaciones (una secuencia, una paralela en CPU y otra paralela en GPU) partiendo de la base que nos presenta Shirley, para poder hacer un análisis que como es el rendimiento de un *Path Traing*

explotando la capacidad de concurrencia de una CPU contra la de una GPU.

No solo nos centraremos en el apartado de renderizado, aunque se trata de la trama central de este proyecto, sino que estudiaremos también que técnicas de aceleración son comúnmente utilizadas para tratar de mejorar al máximo nuestro algoritmo. Es por ellos que veremos también cual es la mejor manera de representar de forma interna la escena que queremos renderizar siguiendo la idea que nos presenta (Karras, 2012) en su artículo.

La forma en que representemos nuestra escena tendrá un gran impacto a la hora de calcular el color de la imagen final. Como hemos comentado, la base del método es trazar rayos por la escena para calcular el color de cada pixel. Si nuestra representación de la escena consiste en almacenar todos los objetos en una estructura de datos de tipo lista o vector ordenados por orden de creación, a la hora de calcular un punto de la imagen en el peor caso estaremos recorriendo todo el conjunto de polígonos de la escena para determinar el color final. Tratar de renderizar una escena que, muy posiblemente, esté compuesta de millones de polígonos puede traducirse en horas y horas de procesado. Es por eso que haremos uso de una estructura de datos aceleradora que nos permita representar la escena de una forma más inteligente, para que a la hora de determinar si un rayo impacta o no un polígono se determine de la forma más rápida posible.

1.2.2. Stakeholders

En esta sección presentaremos cuales son los diferentes actores implicados en un proyecto.

Desarrollador

Este actor es el encargado de realizar la planificación del proyecto, búsqueda de información, documentación, desarrollo del software requerido, solución de posibles obstáculos y/o problemas que puedan aparecer a lo largo del desarrollo y realización y análisis de los experimentos. Debe trabajar de forma conjunta y coordinada con el director, y co-director y/o ponente si lo hay, y es la última persona encargada del cumplimiento de los términos establecidos.

Director del proyecto

Este actor es el encargado de guiar al desarrollador en caso de dificultades, así como del asesoramiento de posibles soluciones.

Usuarios beneficiados

Aunque este proyecto no tiene la intención de crear un producto, no quiere decir que no existan beneficiarios. Explorar diferentes vías de optimización haciendo uso de arquitecturas paralelas puede ser útil para investigadores y desarrolladores en el campo de la computación gráfica.

1.3. Justificación

Como bien hemos indicado en la sección anterior, partiremos de los libros presentados por Peter Shirley y el artículo sobre *Bounding Volume Hierarchy* de Tero Karras para construir nuestras aplicaciones. Puesto que no se trata de un trabajo de investigación, no pretendemos innovar sobre como el *Path Tracing* calcula el color de un pixel o de que manera construimos el BVH. En este trabajo lo que pretendemos es estudiar como podemos explotar al máximo las diferentes arquitecturas que tenemos en nuestra computador (CPU y GPU) y ver en cual de ellas un método de renderizado de imágenes realistas se comporta de manera más eficiente. No podremos comparar nuestra solución aportada frente otras aplicaciones, como por ejemplo RenderMan de Pixar Animation Studios (Christensen et al., 2018), que implementan el mismo método o uno similar debido a que éstas implementan un mayor número de características (como *Subsurface scattering)* que nosotros no podemos asumir en este proyecto.

Por esto mismo, realizaremos tres versiones del mismo algoritmo: I) versión secuencial, II) versión paralela en CPU y III) versión paralela en GPU. De esta forma podremos hacer una comparativa de que arquitectura nos brinda una mayor ventaja para nuestra implementación.

En la actualidad son muchas las librerías orientadas a la programación en GPU. Tenemos librerías como `OpenCL` y `OpenACC`, que nos permiten una mayor portabilidad entre tarjetas gráficas de distintos fabricantes como por ejemplo `AMD` y `NVIDIA`. Pero para este proyecto hemos decidido escoger el entorno de `CUDA` desarrollado por `NVIDIA` específicamente para sus tarjetas gráficas y aceleradores. Decidimos usar esta API debido a que se trata de software propietario, mejor optimizado para las tarjetas de dicha empresa como bien nos indican (Karimi et al., 2010) y (Fang et al., 2011) respectivamente en sus artículos científicos, las cuales utilizaremos en este proyecto.

Es posible que, citadas tarjetas/aceleradores `NVIDIA` y, teniendo presente que el proyecto está enfocado al tema de renderizado de gráficos realistas, al lector del presente Trabajo de Fin de Grado le venga a la mente la nueva gama de tarjetas RTX diseñada por `NVIDIA`. En un inicio se planteó guiar el proyecto hacia el uso de tarjetas con tecnología

RTX debido a que éstas han sido diseñadas específicamente para el uso de *Ray Tracing* en tiempo real. Esta idea fue descartada en seguida debido al elevado precio de éstas. El rango de precios de las tarjetas de esta gama oscila entre los 350€ en los modelos más económicos, hasta los varios miles de euros en modelos destinados a entornos profesionales. Finalmente, aunque el autor de dicho trabajo adquirió una tarjeta gráfica NVIDIA RTX 280 Super con 8Gb de memoria, se decidió no guiar el proyecto a utilizarla de forma exclusiva, estudiando y poniendo en práctica las nuevas mejoras que ésta ofrece (RT Cores, Tensor Cores, Mesh Shaders, etc.), frente a otras tarjetas de gamas inferiores que no incluyen, debido al corto plazo de tiempo para el desarrollo. Es por eso que esta tarjeta gráfica será usada para testar nuestra aplicación, pero no en un sentido exclusivo.

Como hemos comentado al inicio de esta sección, `CUDA` es un API que está muy bien optimizada para hardware de `NVIDIA`. Esto nos da un punto a favor debido a que la aplicación que vamos a desarrollar será probada en diferentes entornos que utilizan la tecnología de NVIDIA:

1. Computador portátil - Lenovo Legion Y520 con Nvidia GTX1050 Mobile - 4GB.
2. Computador personal - Nvidia RTX 2080 Super - 8Gb.
3. Cluster docencia BOADA - 4 GPUs Nvidia Tesla K40c.

Al usar tarjetas en entornos diferentes podremos analizar como nuestra aplicación responde en cada uno de ellos y estudiar así cómo es el rendimiento cuando utilizamos varias tarjetas pensadas para un entorno de investigación/profesional, en contraposición con otras dos pensadas para un uso más cotidiano. También podremos ver como es el rendimiento en una tarjeta gráfica de gama media (Nvidia GTX1050 Mobile) y una de gama alta (Nvidia RTX 2080 Super); y hacer así una comparativa entre ellas.

1.4. Alcance del proyecto

Para solucionar el problema presentado en nuestro proyecto necesitamos una aplicación que sea capaz de renderizar imágenes realistas. Como hemos comentado unas secciones más atrás, en (Shirley, 2019c; Shirley, 2019b; Shirley, 2019a) se nos presentan las bases para crear un *Ray Tracing*. A partir de esta base extenderemos nuestra aplicación a una versión paralela haciendo uso de la CPU y otra haciendo uso de la GPU.

También, como hemos comentado en la sección sobre el contexto, el uso de estructuras de datos aceleradoras es un factor importante en este tipo de aplicaciones y se llevan estudiando durante muchos años (p.e. (Rubin et al., 1980)) por parte de la comunidad científica. En este proyecto haremos uso de la *Bounding Volume Hierarchy*, que se trata de una estructura de tipo árbol, ya sea binario o n-aria, en el cual cada hoja representa la

caja delimitadora (*Bounding Box*) de cada primitiva de la escena; y cada nodo intermedio representa la caja delimitadora de sus hijos. De este modo, estamos representando la escena como un conjunto de cajas que nos permitirá saber de forma eficiente si un rayo impacta o no con un polígono y con cual de todos los que componen la escena.

Existen muchas formas de construir un BVH, en nuestro caso nos hemos decantado por la versión presentada por (Karras, 2012; Karras et al., 2013). La forma en la que construimos el árbol nos permite explotar al máximo la concurrencia que nos proporciona la CPU y la GPU ya que es posible construir cada nodo del árbol de forma independiente.

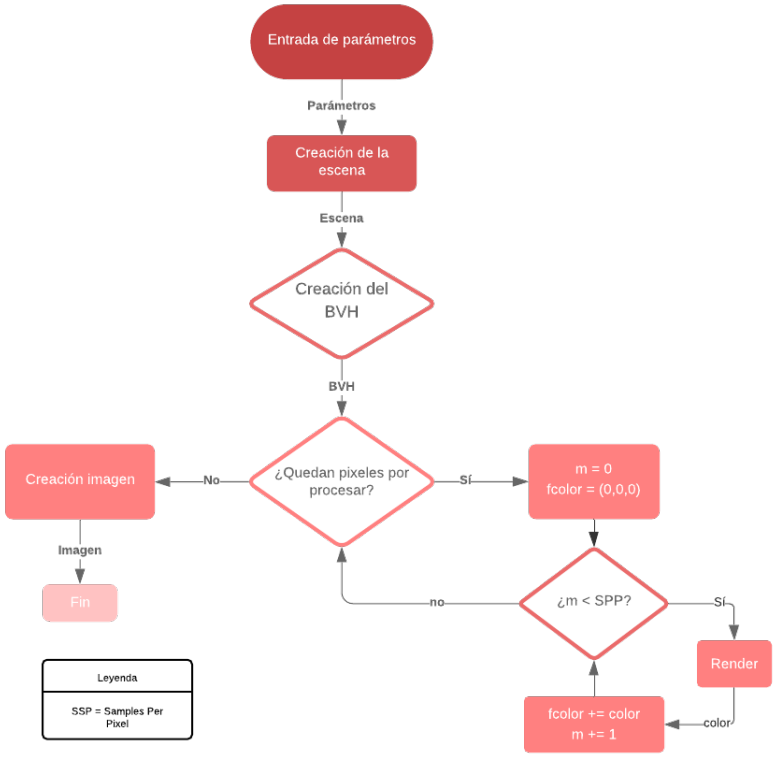

Figura 1.3: Esquema aplicación. Fuente: Propia

En la Figura 1.3 podemos observar un esquema, a grandes trazos, de cual comportamiento es el comportamiento de nuestra aplicación.

1.4.1. Objetivos

Objetivos principales

Son varios los objetivos principales que nos proponemos en este proyecto. El primero se trata de desarrollar tres aplicaciones (secuencia, paralela en CPU y paralela en GPU) que dada una escena la renderice mediante el método de *Path Tracing* y analizar el rendimiento que éste nos ofrece entre las diferentes versiones con la intención de analizar que arquitectura nos brinda una mayor rendimiento en este tipo de aplicaciones.

Un segundo objetivo principal que nos proponemos en nuestro proyecto es estudiar y poner en práctica cuales son las mejores prácticas en la gestión de memoria en una aplicación paralela se refiere. Lo identificamos como un objetivo principal por que una mala gestión de ésta puede suponer un gran obstáculo en el desarrollo del proyecto.

Objetivos secundarios

Como objetivos secundarios, pero no menos importantes, tenemos:

1. Implementar de forma eficiente el método presentado por P. Shirley y T. Karras.
2. Buscar información sobre técnicas de optimización en el cálculo de intersección rayo-objeto.
3. Implementar de forma eficiente los cálculos de intersección rayo-objeto.
4. Extender la aplicación para renderizar modelos 3D.
5. Creación de una aplicación interactiva.

Los objetivos secundarios 4 y 5 se realizarán en función del tiempo restante para la finalización del proyecto.

1.4.2. Obstáculos y riesgos del proyecto

Los temas principales en los que se centra el presente proyecto han sido muy estudiados por parte de la comunidad científica. No obstante, esto no implica que el desarrollo de éste sea una tarea sencilla, pues son muchos los problemas u obstáculos a los cuales podemos enfrentarnos.

Programa principal

Como bien hemos comentado en la sección de objetivos, la gestión de memoria es un factor muy importante. Una mala gestión de ésta puede provocar errores que no permitan el correcto funcionamiento de la aplicación.

Algoritmo utilizado

El algoritmo utilizado calcula el color a partir de la intersección de los rayos emitidos desde la cámara a los diferentes pixeles de la escena. Al tratarse operaciones que se llevarán a cabo miles de millones de veces en la creación de una imagen, tener una mala implementación de éstas puede afectar al rendimiento de nuestra aplicación de forma negativa.

1.5. Metodología y rigor

En esta sección veremos el conjunto de herramientas que se usarán a lo largo del desarrollo del presente proyecto. Para poder llevar un buen desarrollo de éste nos organizaremos de la siguiente forma: I) Reuniones cada 15 días con el director del presente proyecto para comentar el estado de éste, resultados obtenidos, carencias, objetivos cumplidos y no cumplidos y acordar los siguientes pasos a realizar. II) Reuniones semanales de cara a la fase final del desarrollo.

1.5.1. Metodología

La metodología de trabajo que se seguirá en el presente proyecto es la conocida como *agile*. Esta metodología se caracteriza por se iterativa. Es decir, en cada iteración hay una serie de tareas y cuando se finalizan estas se inicia una nueva iteración, conocidas como *sprint*. En nuestro proyecto cada iteración del ciclo de vida corresponderá con el inicio de una reunión con el director del proyecto hasta el inicio de la siguiente. En cada reunión se especificará una serie de objetivos, que deberán cumplirse de cara a la siguiente vista y en la que se evaluarán cuales han sido cumplidos y cuales no. En la Figura 1.4 podemos observar un esquema representativo del funcionamiento de esta metodología.

Figura 1.4: Esquema metodología ágil. Fuente: OpenWebinars

1.5.2. Herramientas de desarrollo

El desarrollo de nuestra aplicación se llevará a cabo en C++ haciendo uso de las librerías `OpenMP` y `CUDA`.

`OpenMP` es una API diseñada para añadir concurrencia a programas escritos en `C`, `C++` y `Fortran`. La principal ventaja de usar esta API, en contra de otras de características similares, es que nos permite escribir un código portable entre diferentes sistemas operativos como podría ser Linux, Windows o MAC y nos permite crear de forma sencilla aplicaciones paralelas haciendo uso de la CPU.

`CUDA` es una plataforma de computación paralela y una API desarrollada por `NVIDIA` que nos permite acceder al conjunto de instrucciones y elementos de cómputo de las tarjetas gráficas y aceleradores de `NVIDIA` para poder crear aplicaciones paralelas haciendo uso de la GPU.

1.5.3. Herramientas de seguimiento

Por tal de llevar a cabo un buen seguimiento del desarrollo del presente proyecto se hará uso de `git`. Esta herramienta nos permite llevar un control de versiones que nos permitirá consultar versiones anteriores de nuestro código en caso de ser necesario.

CAPÍTULO 2

PLANIFICACIÓN TEMPORAL

2.1. Planificación y programación

Este capítulo trata la planificación del proyecto y describe las tareas realizadas, a través de un plan de acción, para cumplir los plazos relativos a la entrega del proyecto. Veremos también los recursos utilizados y se tendrán en cuenta los posibles obstáculos que puedan modificar la planificación.

El proyecto se inició a principios de julio de 2019 con vista a entregarlo el 13 de enero de 2020.

2.2. Descripción de tareas y recursos utilizados

2.2.1. Descripción de tareas

Estudio de conceptos

Antes de comenzar el proyecto, fue necesario familiarizarse con los conceptos básicos que rigen nuestro proyecto. Durante el semestre previo hubo un par de reuniones con el director del proyecto para definir el tema en el cual se centraría y poder así encaminarlo. También, el autor del presente proyecto se matriculó de la asignatura de Tarjetas Gráficas y Aceleradores (TGA por sus siglas) para introducirse en la programación en el entorno de

CUDA y así agilizar el proceso de desarrollo del proyecto su etapa inicial. Dado que esta tarea se realiza fuera del proyecto sumado a la dificultad de determinar una estimación en horas, no se tendrá en cuenta en la planificación.

Configuración del sistema

Antes de entrar propiamente en el desarrollo en si del proyecto, debemos configurar las herramientas necesarias y garantizar su correcto funcionamiento.

Para poder desarrollar el proyecto será necesario tener instalado en nuestros sistemas la API de CUDA con tal de poder crear el programa principal de nuestra aplicación paralela en GPU. Las librerías de OpenMP vienen instaladas por defecto en los sistemas Linux, por lo que no será necesaria una instalación, pero sí comprobar que funcionan correctamente. Por último, haremos uso de una plataforma, *TexMaker*, que nos permita compilar código LaTeXcon tal de generar la documentación requerida.

Esta configuración debe hacerse tanto en el computador de sobremesa como en el portátil. La parte de *CUDA* en el cluster de docencia no será necesario de configurar debido a que ya está previamente configurado por el DAC (Departament d'Arquitectura de Computadors).

Planificación del proyecto

Esta es la tarea que se está desarrollando actualmente. Esencialmente trata sobre todo el contenido cubierto por el curso de GEP. La podemos dividir en tres subtareas:

1. Contextualización y alcance.
2. Planificación temporal.
3. Presupuesto y sostenibilidad.

Desarrollo

Esta tarea es la más importante de todo el proyecto. Cubre el desarrollo de las aplicaciones, experimentación y documentación de los resultados de cada una de las partes. Dividiremos el desarrollo en cuatro ciclos de 14 días cada uno, coincidiendo con las reuniones programadas con el director del presente proyecto. Cada uno de los ciclos los dividiremos en cuatro tareas:

- **Desarrollo versión secuencial:** Se trata de la versión más básica de todas. Sirve como base para desarrollar tanto la versión paralela en CPU (`OpenMP`), como la versión paralela en GPU (`CUDA`).
- **Desarrollo versión paralela CPU:** Una vez implementada la versión secuencial pasaremos a implementar una versión paralela de ésta haciendo uso de la librería de `OpenMP`.
- **Desarrollo versión paralela GPU:** Una vez implementada la versión secuencia pasaremos a implementar una versión paralela de ésta haciendo uso de `CUDA`.
- **Experimentación:** Crearemos la misma escena en cada una de las versiones comentadas en los puntos anteriores y analizaremos el rendimiento de ambas, teniendo en cuenta el tiempo de creación requerido para generar la imagen final.

Las dependencias entre las diferentes tareas son fáciles de describir. Hasta que no tengamos la versión secuencial no podremos empezar a programar las versiones paralelas de ésta. Tampoco podremos empezar un ciclo nuevo sin haber terminado el anterior.

Etapa final

En esta etapa estructuraremos toda la documentación de las etapas anteriores y prepararemos la presentación final para la defensa del proyecto.

2.2.2. Tabla resumen y Estimación

Tarea	Tiempo empleado (horas)
Configuración del sistema	10
Planificación del proyecto	130
Desarrollo	330
Etapa final	120
Total	590

Cuadro 2.1: Resumen de horas

Con una estimación de 35 horas a la semana (3 horas de lunes a jueves, 10 horas sábados y domingos), y teniendo presentes un total de 18 semanas de trabajo, tenemos: $35h \cdot 18 = 630$ horas en total. Esto nos deja un margen de tiempo para posibles inconvenientes.

2.2.3. Recursos utilizados

Software

- Linux, usado en todas las tareas.
- LaTeX, para realizar la documentación.
- `C++`,`OpenMP` y `CUDA`, para el desarrollo de las aplicaciones.
- `git`, utilizado para el control de versiones y compartir el código entre diferentes entornos (computador de sobremesa y el portátil).

Hardware

- PC Sobremesa con las siguientes características: i7 7700 3.6 GHz, NVIDIA GeForce RTX 2080 SUPER, 24GB RAM, 250 SSD, 1TB SSD, 1TB HDD
- PC Portátil con las siguientes características: i7 7700HQ 2.8GHz, NVIDIA GeForce 1050 Mobile, 8GB RAM, 500GB SSD
- Cluster de docencia con las siguientes características: Intel Xeon E5-2620 v2 2.10GHz x2, NVIDIA Tesla K40c x4, 64GB RAM, 1TB x2

2.3. Diagrama de Gannt

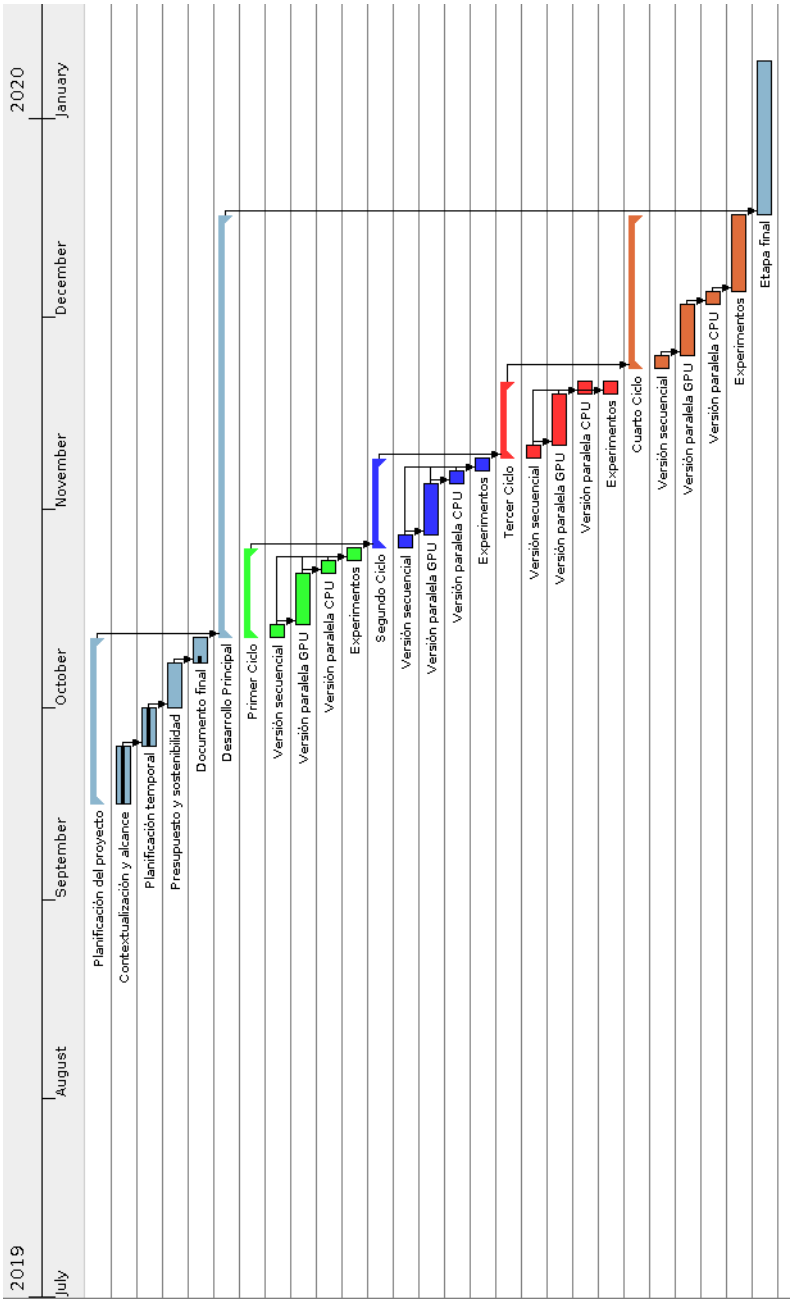

Figura 2.1: Diagrama de Gannt completo. Fuente: Propia

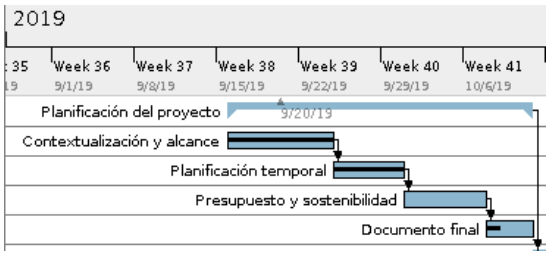

Figura 2.2: Etapa de gestión.
Fuente: Propia

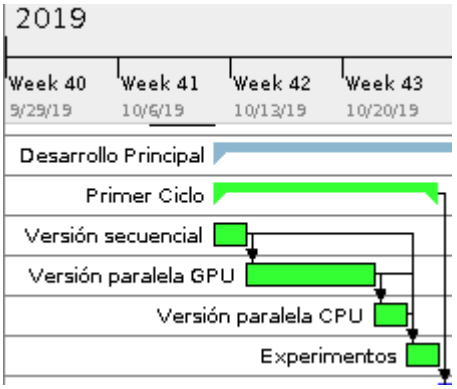

Figura 2.3: Primer ciclo.
Fuente: Propia

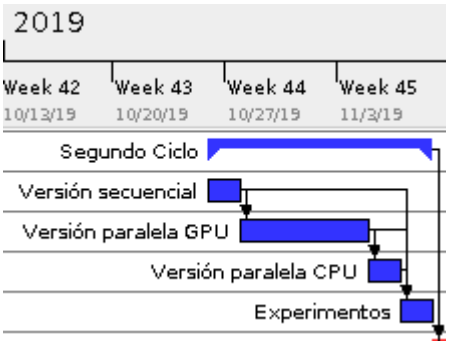

Figura 2.4: Segundo Ciclo. Fuente: Propia

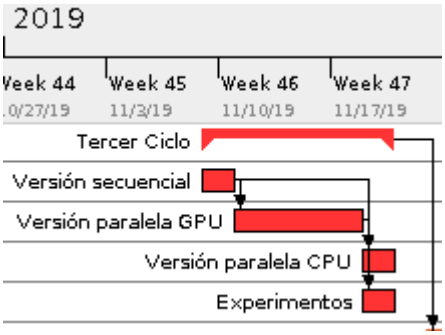

Figura 2.5: Tercer ciclo.
Fuente: Propia

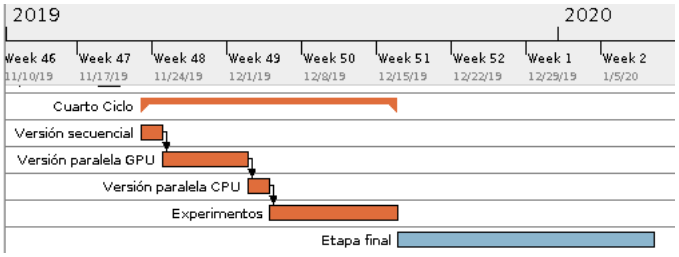

Figura 2.6: Cuarto ciclo y etapa final.
Fuente: Propia

2.4. Gestión de riesgos: Planes alternativos y obstáculos

Como podemos observar en la Figura 2.6, en el cuarto ciclo de desarrollo la tarea de experimentos ocupa más tiempo en comparación a ciclos anteriores. En este caso, hemos sobrestimado la duración de ésta para tener tiempo suficiente de resolver problemas que puedan aparecer, como los definidos en el capítulo anterior (Contextualización y alcance), en el curso del desarrollo del proyecto o experimentos que puedan surgir a última hora.

CAPÍTULO 3

PRESUPUESTO Y SOSTENIBILIDAD

En este capítulo trataremos los temas de presupuesto y sostenibilidad. Veremos una descripción detallada de los costes del proyecto, de los recursos tanto materiales como humanos, y un análisis de cómo podrían afectar a nuestro presupuesto los diferentes obstáculos que pueden acaecer en el trascurso del desarrollo. Haremos también una evaluación de la sostenibilidad del proyecto.

El presupuesto puede estar sujeto a modificaciones a lo largo del desarrollo del proyecto.

3.1. Presupuesto

En esta sección veremos como hacer una estimación del presupuesto por tal de poder desarrollar el proyecto. Podemos dividir el presupuesto en dos grandes secciones: I) Costes directos y II) Costes indirectos.

3.1.1. Costes directos

Los costes directos los dividiremos en función de los diferentes tipos de recursos que tenemos en este proyecto: I) software, II) hardware y III) humanos.

Para el cálculo de la amortización tendremos en consideración que el proyecto tiene una duración aproximada de cinco meses.

Recursos de Software

El software que vamos a utilizar en nuestro proyecto es de código libre. Y en caso de necesitar software adicional intentaremos que este sea también de código libre. La Tabla 3.1 muestra el coste del software utilizado en el desarrollo.

Producto	Precio	Vida útil	Amortización
Ubuntu 18.04	0,00€	-	0,00€
LaTeX	0,00€	-	0,00€
CUDA	0,00€	-	0,00€
OpenMP	0,00€	-	0,00€
C++	0,00€	-	0,00€
Total	0,00€	-	0,00€

Cuadro 3.1: Coste recursos de software

Recursos de Hardware

La Tabla 3.2 muestra el coste del hardware utilizado en el desarrollo.

Producto	Precio	Vida útil (en años)	Amortización
PC Sobremesa	2210,09€	5	92,09€
Lenovo Legion Y520	900,00€	5	37,50€
BOADA	3500.00€	5	145,83€
Total	6610.09€	-	275,42€

Cuadro 3.2: Coste recursos de hardware

En el coste del cluster de docencia, BOADA, no se tienen presente las cuatro tarjetas gráficas NVIDIA Tesla K40c puesto que fueron una donación de *NVIDIA*, por lo tanto su coste es nulo.

Recursos Humanos

Para cada tarea especificada le identificaremos el rol más adecuado.

- **Responsable del proyecto:** Planificación del proyecto y Fase final.
- **Desarrollador de software:** Desarrollo de las tres aplicaciones y experimentos.

- **Técnico informático:** Configuración del sistema.

Categoría	Salario neto €/h	Salario bruto €/h	Horas Totales	Coste estimado
Director de proyecto	18€/h	24,30€/h	80h	1944,00€
Responsable de proyecto	15€/h	20,25€/h	250h	5062,50€
Desarrollador de software	13€/h	17,55€/h	330h	5791,50€
Técnico informático	11€/h	14,85€/h	10h	148,50€
Total	-	-	670h	9590,50€

Cuadro 3.3: Coste recursos de RRHH

En la Tabla 3.3 podemos observar la relación del sueldo percibido por hora y el número de horas por cada uno de los roles. Para determinar el sueldo se ha consultado el portal web [1].

3.1.2. Costes indirectos

En esta sección tendremos en cuenta los costes que se relacionan con el proyecto de forma indirecta como el derivado del consumo energética y el coste de la red de Internet.

Teniendo presente las diferentes máquinas que se utilizan en el proyecto el consumo de cada una de ellas es el siguiente:

1. **PC Sobremesa:** 320W.
2. **Lenovo Legion Y520:** 160W.
3. **Cluster docencia BOADA:** 1500W.

A continuación, en la Tabla 3.4 podemos observar la relación entre las tareas realizadas y, la máquina utilizada en cada una de ellas.

Máquina	Tareas					Coste
	Configuración	Planificación	Desarrollo	Experimentos	Fase final	Estimad
PC Sobremesa	5h	130h	240h	30h	120h	525h
Lenovo Legion Y520	5h	-	-	30h	-	35h
BOADA	-	-	-	30h	-	30h

Cuadro 3.4: Horas por tarea y recurso

[1] https://tusalario.es

Teniendo presentes las horas totales por cada máquina a un precio de 0,1198€/kWh y los demás costes mencionados al inicio de esta sección, el coste indirecto total es:

Máquina	Precio	Unidades	Coste estimado
PC Sobremesa	0,1198€/kWh	168kWh	20,13€
Lenovo Legion Y520	0,1198€/kWh	5,6kWh	0,67€
BOADA	0,1198€/kWh	45kWh	5,40€
Internet	50€	6 meses	300,00€
Total	-	-	326,2€

Cuadro 3.5: Coste de consumo por máquina

3.1.3. Costes inesperados

En caso de alguna desviación en la planificación del proyecto, destinaremos una parte del presupuesto a los contratiempos que puedan surgir.

Categoría	Salario neto €/h	Salario bruto €/h	Horas Totales	Coste estimado
Director de proyecto	18€/h	24,30€/h	20h	486,00€
Responsable de proyecto	15€/h	20,25€/h	10h	202,50€
Desarrollador de software	13€/h	17,55€/h	20h	351,00€
Técnico informático	11€/h	14,85€/h	2h	29,70€
Total	-	-	-	1069,20€

Cuadro 3.6: Coste inesperados

3.1.4. Presupuesto total

Concepto	Coste estimado
Coste directo	
Software	00,00€
Hardware	6610,50€
RRHH	9590,50€
Coste indirecto	326,20€
Coste inesperado	1069,20€
Subtotal	17596,40€
Contingencia (10 %)	1759,64€
Total	19356,04€

Cuadro 3.7: Presupuesto total

3.2. Control de gestión

Con tal de hacer un control presupuestario, al final de cada una de las tareas haremos un recuento de las horas empleadas y el coste del material extra utilizado, si se diera el caso. Con esta información podremos hacer una comparativa con los datos estimados previamente y calcular la desviación.

$$\text{desviación en el coste} = (CE - CR) \cdot HR^{2}$$

$$\text{desviación en el consumo} = (HE - HR) \cdot CE^{3}$$

3.3. Sostenibilidad

3.3.1. Impacto ambiental

El impacto ambiental producido por el presente proyecto no es más que el resultado del consumo de electricidad por parte de las máquinas que utilizamos. Al tratarse de un proyecto experimental sobre un algoritmo en concreto, no estamos creando ningún tipo

[2]CE = Coste estimado; CR = Coste real;
[3]HE = Horas estimadas; HR = Horas reales;

de producto físico por lo que no se ha planteado el uso de material reciclado o material de proyectos similares anteriores.

En la Tabla 3.3 podemos observar que el total de horas que concierne a los recursos humanos son de 670h. Considerando que una persona en su rutina habitual tiene un consumo de 0,1kWh, el consumo total es de $0,1kWh \cdot 670h = 67kWh$.

Como hemos comentado en la sección anterior, utilizamos tres máquinas diferentes para realizar los experimentos, pero para el desarrollo íntegro del proyecto solamente estamos utilizando una de ellas. El proyecto se realiza en la máquina que tiene un consumo de 250W, teniendo en cuenta las horas utilizadas por esta máquina, según vemos en la Tabla 3.4, el consumo total es de $0,25kWh \cdot 525h = 131,25kWh$. Si bien es cierto que no se analizó el impacto ambiental que el presente proyecto pudiera tener para así poder minimizarlo, ahora, teniendo una mayor perspectiva podríamos haber optado por realizar el proyecto en la máquina de menor consumo energético y remitir las otras solamente a los experimentos (que requieren menos horas de trabajo y por ende un menor consumo energético).

Por último, cabe mencionar que el presente proyecto no ofrece ningún tipo de mejora ambiental respecto a proyecto/productos similares. El impacto ambiental que pueda suponer el consumo de electricidad depende exclusivamente del suministrador y no tiene el mismo impacto el uso de energías renovables que el uso de energías fósiles.

3.3.2. Impacto económico

Todos los costes relativos al presente proyecto han sido detallados en la sección anterior de los recursos utilizados, tanto humanos como de software y hardware. A pesar de haber varios roles definidos, estos serán llevados por una sola persona (a excepción del rol de "Director del proyecto"). Esto supone que el coste humano se reduce a solamente dos personas.

El principal coste económico que observamos en el proyecto es el de hardware. Éste podría reducirse haciendo, quizás, un menor uso de máquinas, pero limitando así los objetivos del proyecto. Es por eso que creemos que el gasto económico es el adecuado.

3.3.3. Impacto social

A título personal, este proyecto me permite adquirir nuevos conocimientos no vistos a lo largo del grado y definir también los pasos a seguir en el futuro, ayudándome a guiar mi futuro profesional y/o académico.

Una vez finalizado el proyecto no tendrá gran trascendencia en la sociedad en general. Como comentamos en capítulos anteriores, este proyecto no propone ninguna alternativa respecto a soluciones existentes sino más bien dar una serie de herramientas al autor de éste para adentrarse en el mundo de la computación gráfica realista. Por lo tanto, es difícil de determinar el impacto social que pueda tener más allá del divulgativo o académico.

BIBLIOGRAFÍA

Lotto, R. Beau, S. Mark Williams y Dale Purves (1999). “Mach bands as empirically derived associations”. En: *Proceedings of the National Academy of Sciences of the United States of America*. ISSN: 00278424.

Henri, Gouraud (1971). “Continuous Shading of Curved Surfaces”. En: *IEEE Transactions on Computers*. ISSN: 00189340.

Phong, Bui Tuong (1975). “Illumination for Computer Generated Pictures”. En: *Communications of the ACM* 18.6, págs. 311-317. ISSN: 15577317.

Whitted, Turner (1980). “An Improved Illumination Model for Shaded Display”. En: *Communications of the ACM*. ISSN: 15577317.

Appel, Arthur (1968). “Some techniques for shading machine renderings of solids”. En: *AFIPS Spring Joint Computing Conference*.

Kajiya, James T. (1986). “The rendering equation”. En: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986*. ISBN: 0897911962.

Immel, David S., Michael F. Cohen y Donald P. Greenberg (1986). “A radiosity method for non-diffuse environments”. En: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986*. ISBN: 0897911962.

Lafortune, Eric P. e Yves D. Willems (1993). “Bi-Directional Path Tracing”. En: *Proc. SIGGRAPH*. ISSN: 1098-6596. arXiv: arXiv:1011.1669v3.

Jensen, Henrik Wann (1996). “Global Illumination using Photon Maps”. En: *Rendering Techniques ’96*. Springer Vienna, págs. 21-30.

Veach, Eric y Leonidas J. Guibas (1997). “Metropolis light transport”. En: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997*. ISBN: 0897918967.

Shirley, Peter (2019c). *Ray Tracing in One Weekend*. Peter Shirley, pág. 41. URL: https://github.com/RayTracing/InOneWeekend.

– (2019b). *Ray Tracing: The Rest of Your Life*. Peter Shirley, pág. 53. URL: https://github.com/RayTracing/TheRestOfYourLife.

– (2019a). *Ray Tracing : The Next Week*. Peter Shirley, pág. 50. URL: https://github.com/petershirley/raytracingthenextweek.

Karras, Tero (2012). "Maximizing parallelism in the construction of bvhs, octrees, and k-d trees". En: *High-Performance Graphics 2012, HPG 2012 - ACM SIGGRAPH / Eurographics Symposium Proceedings*. ISBN: 9783905674415.

Christensen, Per et al. (2018). "RenderMan: An advanced path-tracing architecture for movie rendering". En: *ACM Transactions on Graphics*. ISSN: 15577368.

Karimi, Kamran, Neil G. Dickson y Firas Hamze (2010). "A Performance Comparison of CUDA and OpenCL". En: *Computing Research Repository - CORR* abs/1005.2.1, pág. 10. arXiv: 1005.2581. URL: http://arxiv.org/abs/1005.2581.

Fang, Jianbin, Ana Lucia Varbanescu y Henk Sips (2011). "A comprehensive performance comparison of CUDA and OpenCL". En: *Proceedings of the International Conference on Parallel Processing*. ISBN: 9780769545103.

Rubin, Steven M. y Turner Whitted (1980). "A 3-dimensional representation for fast rendering of complex scenes". En: *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1980*. ISBN: 0897910214.

Karras, Tero y Timo Aila (2013). "Fast parallel construction of high-quality bounding volume hierarchies". En: *Proceedings - High-Performance Graphics 2013, HPG 2013*. ISBN: 9781450321358.

Blinn, James F. (1977). "Models of light reflection for computer synthesized pictures". En: *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1977*.

Hunt, Warren y William R. Mark (2008). "Ray-specialized acceleration structures for ray tracing". En: *RT'08 - IEEE/EG Symposium on Interactive Ray Tracing 2008, Proceedings*, págs. 3-10. ISBN: 9781424427413.