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 i 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

21 de septiembre 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 en nuestro día a día, ya sea en un entorno de trabajo o en un lúdico. La creación de imágenes realistas mediante el uso de computadoras se ha convertido en una necesidad de actualidad. Industrias como el cine o los videojuegos requieren de algoritmos capaces de poder reproducir el mundo real en un entorno virtual 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 sobre iluminación realista. El sombreado Gouraud (*Gouraud Shading*), el sombredado Phong (*Phong Gouraud*) y el sombreado Blinn-Phong (*Blinn-Phong Shading*) son ejemplo de ello. Estos algoritmos tienen en cuenta solamente la luz ambiente, la luz difusa y la luz especular.

En computación gráfica existe un conjunto de métodos que nos permiten generar imágenes realistas: *Ray Tracing* [1], *Path Tracing* [2], *Bidirectional Path Tracing* [3], *Photon Mapping* [4], *Metropolis light Transport* [5], entre otros. A excepción del *Ray Tracing*, el resto de métodos listados tratan de aproximar la *Rendering Equation* presentada por Kajiya et al. [2].

En éste proyecto nos centraremos en la implementación del *Path Tracing* presentada por [6] en su libro.

Nos planteamos tres grandes objetivos:

1. Búsqueda de información sobre técnicas de optimización usadas en éste tipo de métodos
2. Implementación para CPU y GPU de les técnicas vistas en el punto 1

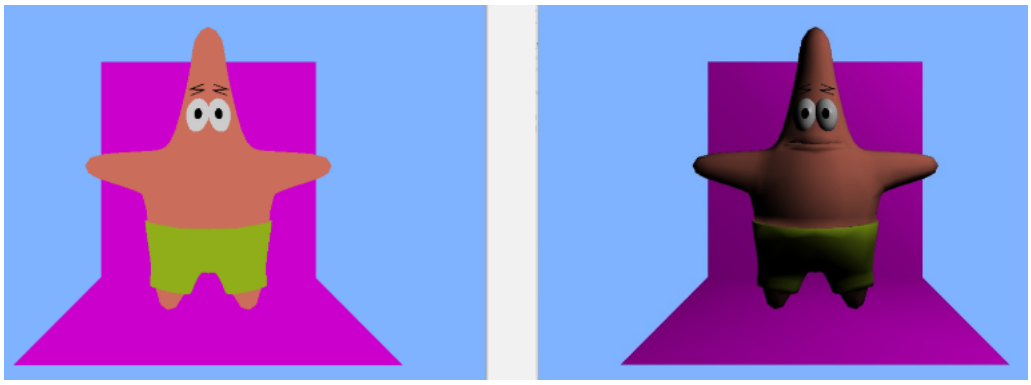

Figura 1.1

3. Análisis y comparación de los resultados obtenidos en ambas implementaciones.

1.2. Contextualización

1.2.1. Contexto

La renderización de imágenes realistas es un área de gran interés en el campo de la computación gráfica. Un de los principales objetivos de ésta, es ser capaces de renderizar imágenes que sean indistinguibles de las del mundo real, como por ejemplo fotografías. Es por eso, que es muy importante poder reproducir el comportamiento de la luz en un entorno virtual. Para poder obtener este realismo es necesario la iluminación global, es decir, hemos de ser capaces de poder simular la luz directa (luz que incide de forma directa a un objeto de la escena desde el foco de luz) i la luz indirecta (luz que incide a un objeto de los rebotes en otros objetos de la escena).

El algoritmo de *Ray Tracing* es un de los métodos más populares de este conjunt. Se trata de una técnica que consiste en trazar rayos desd el foco de luz (*Forward Ray Tracing*) a los diferentes objetos de la escena, o des de el ojo, o cáaera virtual, (*Backward Ray Tracing*). Éste es muy sensible al nombre de polígonos que contiene las escena, a más compleja es más infeciente será. Todo y proporcionarnos un alt grado de realismo, no es capaz de conseguir un efecto foto-realista debido a no tener en cuenta la iluminación indirecta. Para conseguir este efecte, hemos de tener presente tanto la iluminación directa com la indirecta, es por eso que Kajiya et al. presentó la *Rendering Equation* [2]. A pesar de tener ciertas limitaciones y no representar todos los efectos que la luz produce en el mundo real como por ejemplo el *Subsurface Scattering*, un método que aproxime esta ecuación ya nos permitirá renderizar imágenes foto-realistas.

El algoritmo de *Path Tracing* surge como mejora del *Ray Tracing* con el objetivo de dar una solución a la *Rendering Equation* mediante la integración de Monte Carlo. Gracias a ésto, el algoritmo es capaz, de forma natural, representar efectos como *Motion Blur*, *Ambient Oclussion* e iluminación indirecta sin

necesidad de uso de shaders a posterior. En el *Path Tracing* es indiferente el número de polígonos que conforman una escena, pero su mayor desventaja es el ruido. Se necesitan muchas muestras por cada píxel de la imagen para que la imagen converja, es por eso que resulta un algoritmo muy costoso para aplicaciones en tiempo real como videojuegos. En cambio, se lleva utilizando muchos años en industrias como el cine de animación dónde el tiempo de procesado de un frame no es tan importante en comparación a la calidad de imagen deseada.

Debido a que cada píxel es independiente de los demás, tenemos un algoritmo con una alta capacidad de paralelismo. Gracias a ésto, podemos explotar la capacidad que tiene la CPU y la GPU para ejecutar aplicaciones en paralelo y calcular más de un píxel al mismo tiempo.

1.2.2. Stakeholders

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

Desarrollador

Este actor es el encargada 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 i/o ponente en caso de haber, y es la última persona encargada del cumplimiento de los términos establecidos.

Director del proyecto

Este acto es el encargada 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

En la actualidad son muchas las librerías orientadas a la programación en GPU. Tenemos librerías como `OpenCL` u `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 la API CUDA implementada por NVIDIA específicamente para sus tarjetas gráficas y aceleradores. Decidimos usar esta API debido a que al ser sotfware propietario de NVIDIA éste está mejor optimizado para las tarjetas de dicha empresa, que son las que utilizaremos en este proyecto.

Es posible que al comentar que usaremos tarjetas/aceleradores NVIDIA y el proyecto está enfocado al tema de renderizado de gráficos realistas nos venga a la mente la nueva gama de tarjetas RTX. En un inicio se planteó guiar el proyecto hacia el uso de tarjetas con tecnología RTX debido a que 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 está entre los 350€ en los modelos más barato hasta los varios miles de euros en modelos destinados a entornos profesionales. Finalmente, se adquirió una tarjeta Nvidia RTX 280 Super de 8Gb de memoria, pero debido al corto periodo de tiempo de desarrollo no guiamos el proyecto a utilizar dicha tarjeta de forma exclusiva estudiando y poniendo en práctica las nuevas mejoras que otras tarjetas de gamas inferiores no incluyen (RT Cores, Tensor Cores, Mesh Shaders, etc.). Es por eso que esta tarjeta gráfica será usada para probar 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 el la aplicación que vamos a desarrollar será probada en diferentes entornos:

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

Al usar tarjetas en entornos diferentes podremos analizar como nuestra aplicación responde en dichos entornos. Estudiar como es el rendimiento cuando utilizamos 4 tarjetas pensadas para un entorno de investigación en contra posición con dos tarjetas gráficas pensadas para el uso cotidiano. También podremos ver como es el rendimiento en una tarjeta gráfica de gama media (Nvidia GTX1050) y un gráfica de gama alta (Nvidia RTX 2080 Super).

1.4. Alcance del proyecto

Para solucionar el problema presentado en nuestro proyecto necesitamos una aplicación que sea capaz de renderizar imágenes realistas. En [6], [7] i [8] se presentan las bases para crear un *Path 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. Se hará uso de diferentes técnicas de optimización comúnmente utilizadas en este tipo de aplicaciones, como por ejemplo el uso de estructuras de datos aceleradoras [9] para representar la escena de forma interna. En concreto, la estructura de datos que utilizaremos para genera la escena se trata de una *Bounding Volume Hierarchy*, que se trata de una estructura de tipo árbol, ya sea binario o n-ario, donde cada hoja representa la caja englobante

(*Bounding Box*) de cada primitiva que existe en la escena, y cada nodo intermedio representa la caja englobante de sus hijos.

1.4.1. Alcance

En este apartado se presentan los diferentes objetivos y posibles obstáculos del proyecto.

1.4.2. Objetivo

El objetivo principal de este proyecto es desarrollar una aplicación que dada un escena, la renderice mediante *Path Tracing* y analizar el rendimiento que éste nos da en su versión paralela en CPU y en GPU.

La gestión de memoria en una aplicación paralela, ya sea en CPU o en GPU, es muy importante y tener una mala gestión de ésta puede suponer que el rendimiento de nuestra aplicación no sea el esperado. Por eso definiremos como objetivo secundario, pero no menos importante, estudiar y llevar a cabo cuales son las mejores prácticas en este campo.

Como a sub-objetivos tenemos:

1. Implementar de forma eficiente los algoritmos presentados en [6] y [10]
2. Implementar de forma eficiente los cálculos de intersección rayo-objeto

1.4.3. Obstáculos i riesgos del proyecto

Los temas principales que se tratan en el proyecto, como por ejemplo implementaciones del *Path Tracing* o diferentes métodos de construcción de un BVH, están muy estudiados y desarrollados. No obstante, eso no implica que el desarrollo del proyecto sea un camino sencillo, son muchos los problemas a los cuales podemos enfrentarnos por el camino.

Programa principal

Como bien hemos comentado en la sección de objetivos, la gestión de memoria es un factor muy importante. Una mala gestión puede provocar errores en nuestra aplicación provocando que ésta no funcione correctamente o directamente no funcione.

Algoritmo utilizado

El algoritmo utilizado calcula el color a partir de la intersección de los rayos emitidos desde la cámara a los diferentes objetos de la escena. El cálculo de esta intersección debe ser muy eficiente ya que se trata de operaciones que se llevarán a cabo miles de millones de veces en la creación de una sola imagen. Tener una mala implementación puede afectar de manera negativa en el rendimiento de nuestra aplicación y en los experimentos a realizar.

1.5. Metodología y rigor

En esta sección veremos el conjunto de herramientas que se usaran a lo largo del desarrollo del proyecto. Para poder llevar un buen desarrollo de nuestro proyecto nos organizaremos de la siguiente forma. Cada 15 días habrá una reunión con el director para comentar el estado del proyecto, resultados obtenidos, carencias, objetivos cumplidos, no cumplidos y acordar los siguientes pasos a realizar. De cara a la fase final las reuniones serán semanales.

1.5.1. Herramientas de desarrollo

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

`OpenMP` es una API diseñada para añadir concurrencia a programas escritos en `C`, `C++` y `Fortan`. 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 Linux, Windows o MAC. Nos permite crear de forma sencillas aplicaciones paralelas haciendo uso de la CPU.

`CUDA` es una plataforma de computación paralela y una API desarrollada por `NVIDIA` y nos permite acceder al conjunto de instrucciones y elementos de las tarjetas gráficas de `NVIDIA`. Una gran ventaja de esta API es la gran accesibilidad, para usuarios novel, que tiene en contra de otras APIs orientadas a programación en GPU como por ejemplo `OpenGL`, `OpenCL` y `DirectX`.

1.5.2. Herramientas de seguimiento

Para poder llevar a cabo un buen seguimiento del desarrollo se hará uso de `git`. Esta herramienta nos permite llevar un control de versiones para poder acceder a versiones anteriores de nuestro código en caso de ser necesario.

1.5.3. Validación de resultados

BIBLIOGRAFÍA

[1] Turner Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 1980. ISSN 15577317. doi: 10.1145/358876.358882.

[2] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986*, 1986. ISBN 0897911962. doi: 10.1145/15922.15902.

[3] Eric P. Lafortune and Yves D. Willems. Bi-Directional Path Tracing. *Proc. SIGGRAPH*, 1993. ISSN 1098-6596. doi: 10.1017/CBO9781107415324.004.

[4] Henrik Wann Jensen. Global Illumination using Photon Maps. 1996. doi: 10.1007/978-3-7091-7484-5_3.

[5] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997*, 1997. ISBN 0897918967. doi: 10.1145/258734.258775.

[6] Peter Shirley. *Ray Tracing in One Weekend*. 2018. URL `https://github.com/RayTracing/InOneWeekend`.

[7] Peter Shirley. *Ray Tracing : The Next Week*. 2018. URL `https://github.com/petershirley/raytracingthenextweek`.

[8] Peter Shirley. *Ray Tracing: The Rest of Your Life*. 2018. URL `https://github.com/RayTracing/TheRestOfYourLife`.

[9] Warren Hunt and William R. Mark. Ray-specialized acceleration structures for ray tracing. In *RT'08 - IEEE/EG Symposium on Interactive Ray Tracing 2008, Proceedings*, pages 3–10, 2008. ISBN 9781424427413. doi: 10.1109/RT.2008.4634613.

[10] Tero Karras. Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *High-Performance Graphics 2012, HPG 2012 - ACM SIGGRAPH / Eurographics Symposium Proceedings*, 2012. ISBN 9783905674415. doi: 10.2312/EGGH/HPG12/033-037.