Computación Gráfica Avanzada

Obligatorio 1
Curso: 2023

Estudiante	CI
Santiago Diaz	4.933.880-4
Fernando Mangold	5.101.020-8

Índice

Introducción

Definición del problema

El problema planteado en este laboratorio es el de implementar el algoritmo de iluminación global de photon mapping para iluminar una escena.

Análisis del problema

Los principales puntos de análisis para la resolución del problema fueron:

- Programar en CPU vs GPU.
- Que formato de escena usar.
- Como almacenar los fotones antes de agregarlos al kd-tree.
- Como manejar los datos entre CPU y GPU.

Diseño de la solución

Para la solución elegimos utilizar la librería optix de Nvidia, en conjunto con la librería OWL, para utilizar la oportunidad para aprender sobre programación en GPU. La solución se basa sobre el proyecto optix7course disponible en el repositorio de OWL. Este proyecto implementa un ray tracer, realiza el cargado de la escena, define luces, cámaras y nos sirvió de ejemplo para construir el emisor de fotones.

Para el formato de la escena, se utiliza un archivo obj para la descripción de la escena y un archivo mtl para los materiales. Basamos esta decisión en la familiarización adquirida con este tipo de archivos en el primer obligatorio del curso de introducción a la computación gráfica y que además el proyecto optix7course tiene el cargador Tiny object loader que puede cargar este tipo de archivos.

Para la emisión de fotones se utiliza una función cuda que se ejecuta paralelamente en varios cuda cores de las GPU Nvidia. Elegimos que la función se ejecute sobre 1200 cuda cores, número que consideramos razonable teniendo en cuenta la cantidad de cuda cores que tienen las tarjetas de gama baja y media. Cada hilo de ejecución emite una cantidad configurable de fotones, que se guardan al impactar en una superficie en un buffer.

Para calcular la radiancia de los fotones en cada punto de la escena utilizamos una esfera de radio configurable para promediar el color de los fotones cercanos a un punto de la geometría.

Arquitectura

La aplicación se puede separar entre la parte que ejecuta en CPU y la que lo hace en GPU.
Por el lado del CPU tenemos:

- main.cpp: Inicializa la carga de la escena, la cámara y la luz. Define un struct
- SampleWindow que se encarga del manejo de input y movimiento de la cámara.
- Model.cpp: se encarga del cargado de la escena.
- LaunchParams.h: se definen structs necesarios para el programa (como photon, frame, camera,..) y se inicializan los valores configurables.
- SampleRenderer.cpp: inicializa optix y manda a ejecutar funciones a la GPU.

En la GPU tenemos un solo archivo que es el devicePrograms.cu, en código cuda donde se emiten los fotones y se calcula la luz directa e indirecta.

Implementación

Descripción del entorno utilizado

Elementos utilizados en el entorno de desarrollo.

- IDE: Visual Studio 2022.
- Compilador v143
- Sistema operativo: Windows 11.
- CMake GUI.
- Blender.

Algoritmos y técnicas gráficas implementadas

Photon Mapping

La técnica de photon mapping consiste en emitir fotones desde una fuente de luz hacia una escena, guardar en que punto impacta y trazar sus rebotes. Esto es lo que se denomina la primera pasada. En la segunda pasada, para cada punto de la escena se calcula la radiancia de los fotones cercanos.

En nuestra implementación utilizamos una esfera de radio configurable para calcular la radiancia.

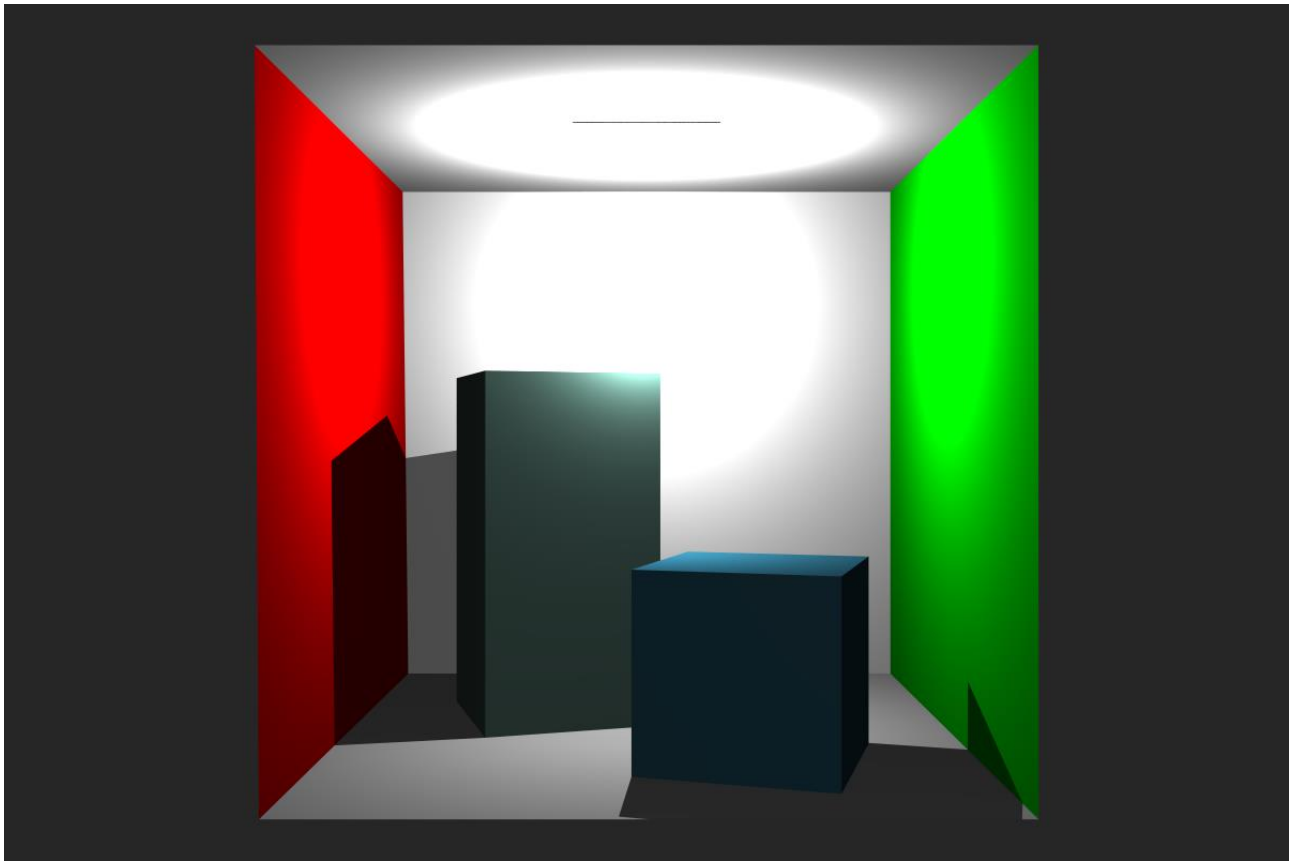
Librerías utilizadas

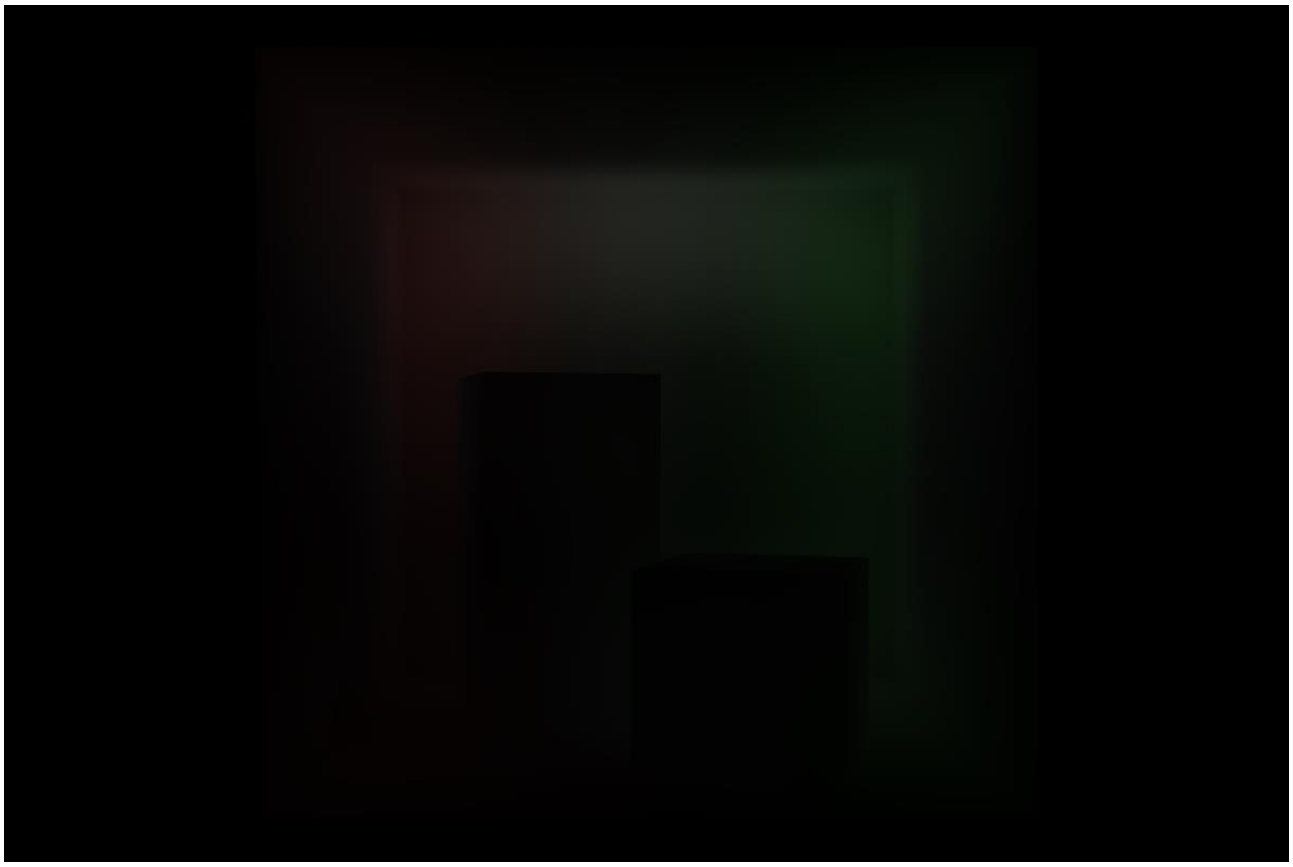
- Optix: framework de ray tracing en GPU
- Owl: librería que facilita el uso de Optix
- Tiny object loader: librería para cargar objetos obj.

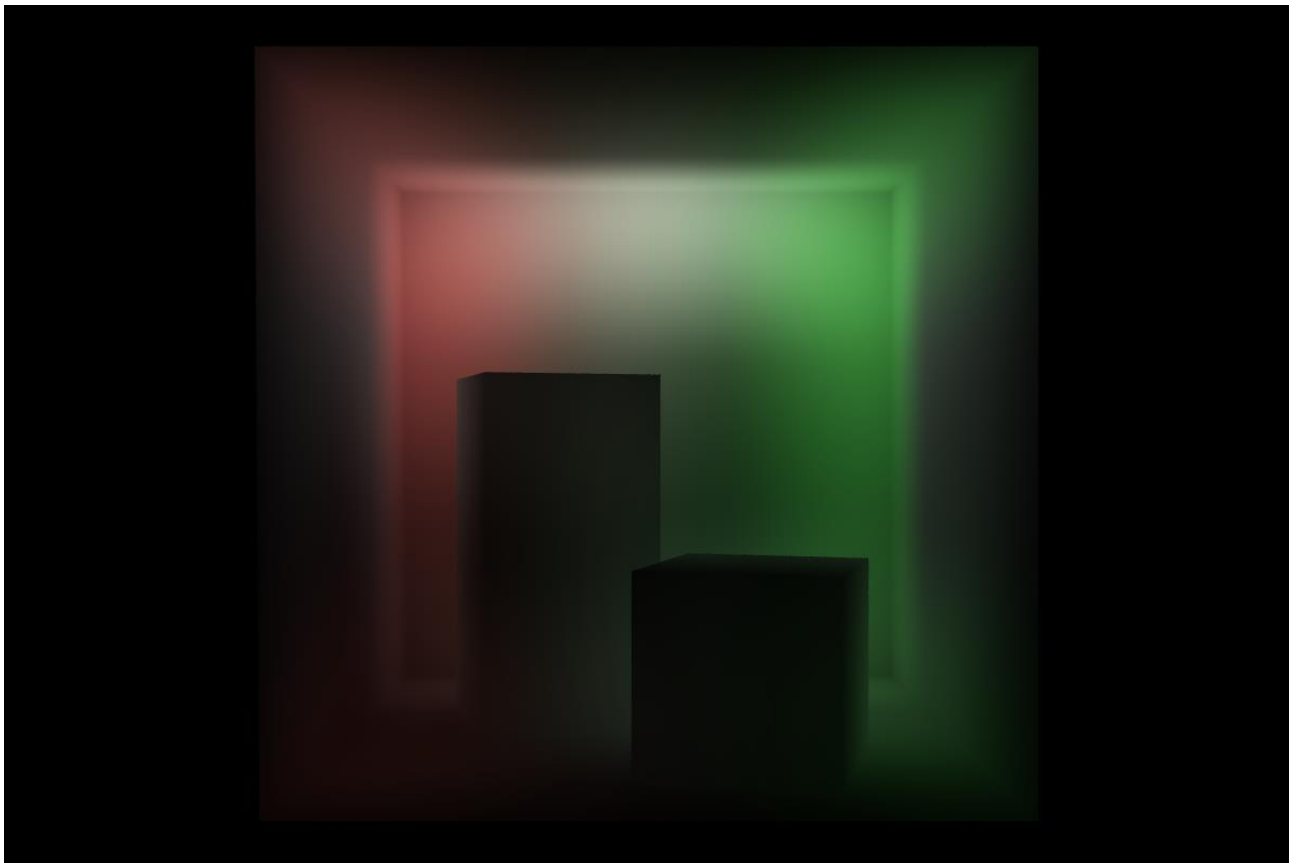
Código desarrollado por terceros

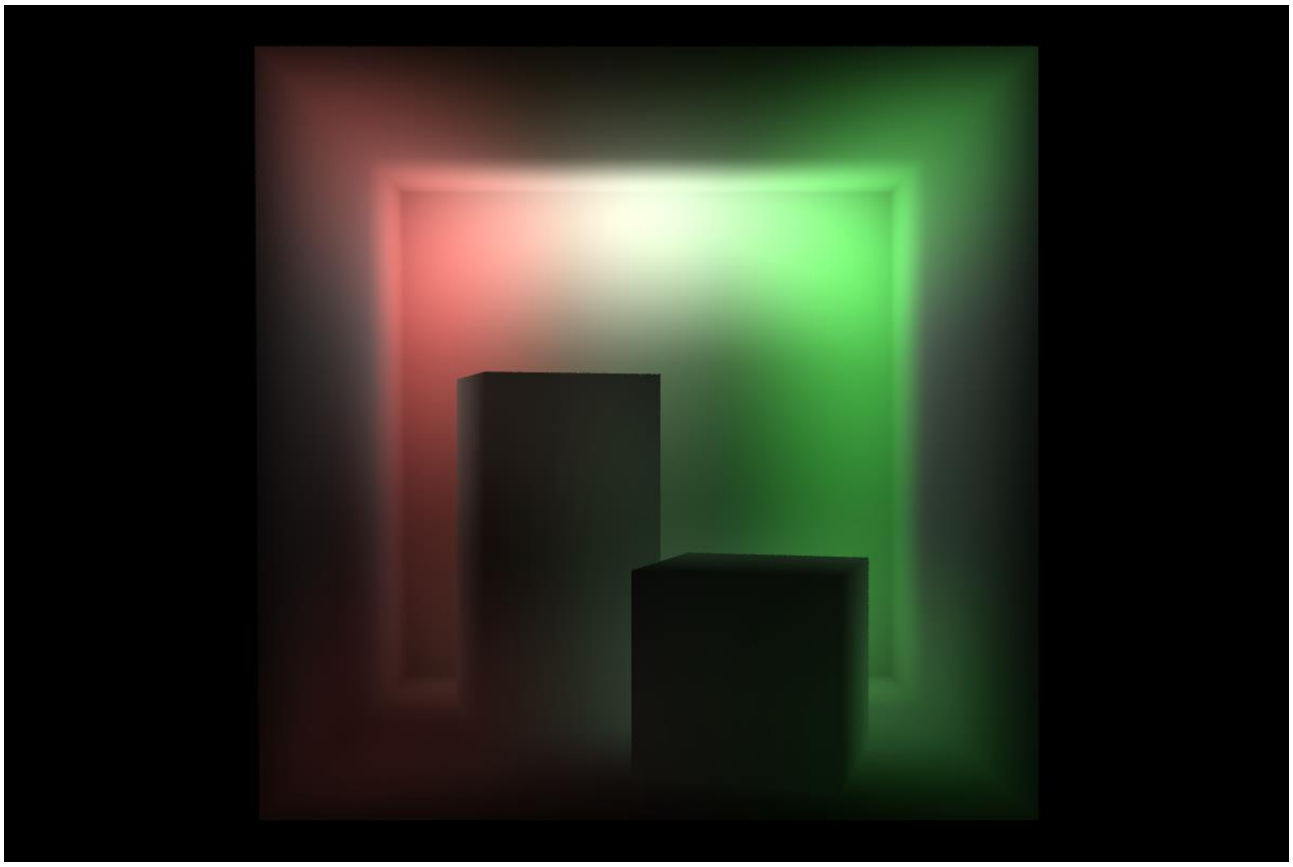
Además de las librerías mencionadas anteriormente se encuentra el código del proyecto optix7course del cual tomamos el ray tracer, el cargador de objetos y el manejador de ventanas.

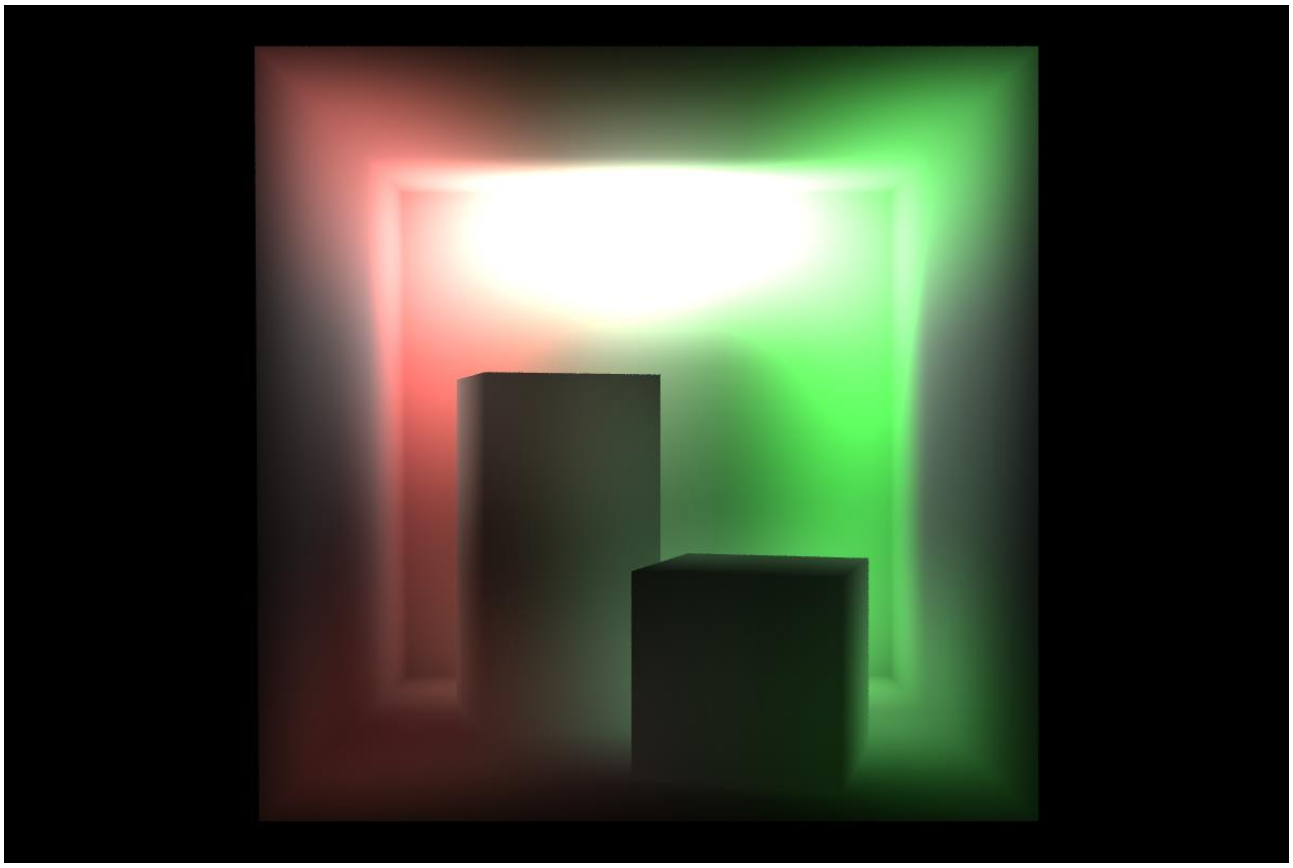
Resultados

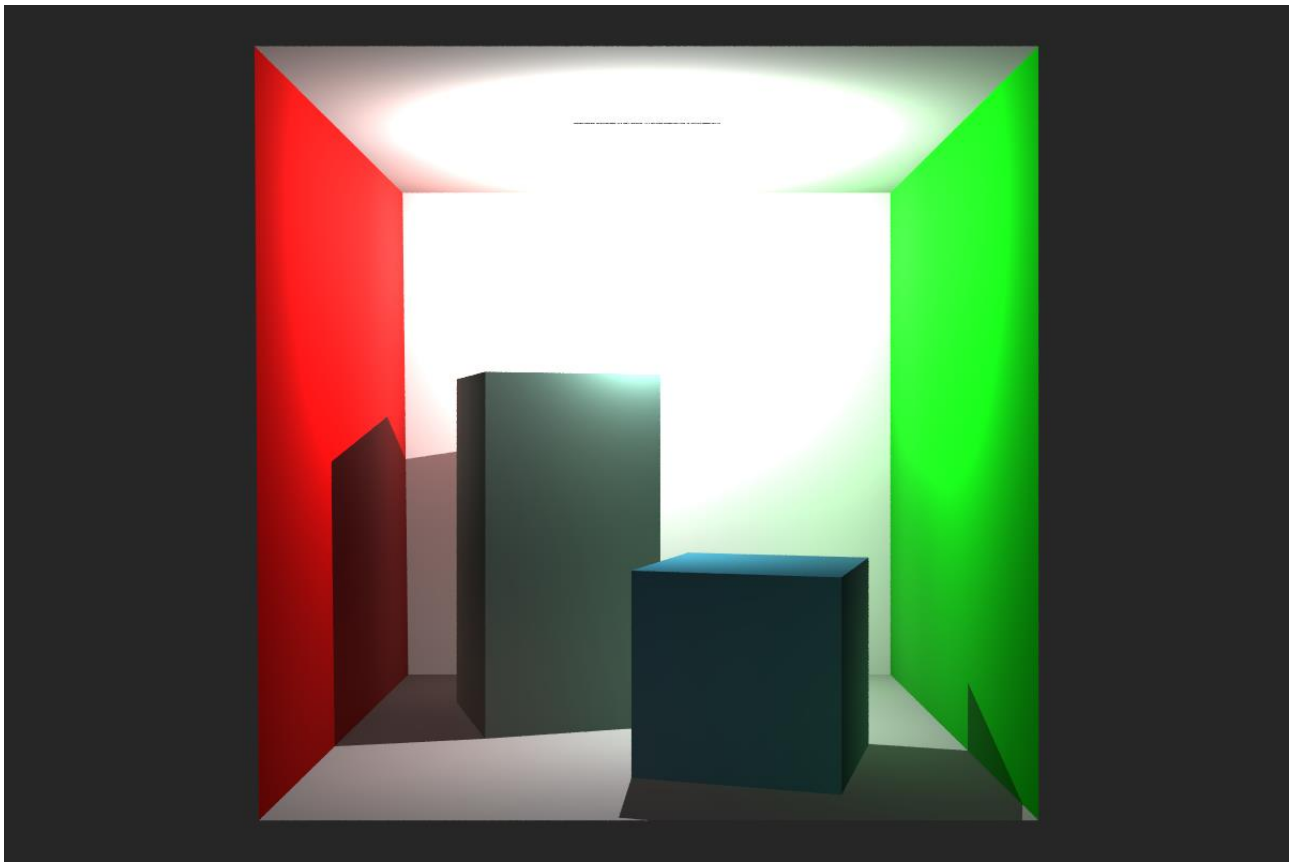












Análisis de performance

Descripción de las imágenes de arriba y detalles de la performance.
Desde arriba hacia abajo:
Imagen 0:

- No hay emisión de fotones, solo ray tracing.
- <1 segundo en generar la imagen.
- 661 MB de memoria utilizados.

Imagen 1:

- Buffer de fotones de máximo 120.000 fotones. Sin RT.
- 5 segundos para generar la imagen.
- 782 MB de memoria utilizados.

Imagen 2:

- Buffer de fotones de máximo 600.000 fotones. Sin RT.
- 18 segundos para generar la imagen.
- 1.2 GB de memoria utilizados.

Imagen 3:

- Buffer de fotones de máximo 900.000 fotones. Sin RT.
- 33 segundos para generar la imagen.
- 1.5 GB de memoria utilizados.

Imagen 4:

- Buffer de fotones de máximo 1.500.000 fotones. Sin RT.

- 54 segundos para generar la imagen.
- 2.1 GB de memoria utilizados.

Imagen 5:

- Buffer de fotones de máximo 600.000fotones. Con RT
- 20 segundos para generar la imagen.
- 1.3 GB de memoria utilizados.

Desarrollo del obligatorio

Cargado de información

La aplicación tiene dos archivos para la descripción de la escena, scene.obj y scene.mtl que se encuentran en la carpeta models.

Configuración

Se tiene un archivo de configuración llamado config.txt donde se pueden configurar los siguientes valores:

- Número de fotones emitidos por hilo de ejecución (1200 hilos en total).
- Número máximo de rebotes por fotón (sin incluir el primer rebote).
- Radio de la esfera para el cálculo de radiancia.
- Si se emiten fotones o no.
- Si se hace ray tracing o no.
- Número máximo de fotones en total (tamaño del buffer de fotones=

Características particulares de la solución

Dada la dificultad para implementar un kd-tree en GPU utilizamos un buffer lineal para guardar los fotones. Para guardar los fotones en el buffer utilizamos un índice que se incrementa de forma atómica, lo que permite que los hilos de ejecución puedan guardar en el buffer los fotones de manera contigua sin sobre escribir en el mismo índice.

Conclusiones

Concluimos que aun sin utilizar una estructura de datos que facilite el calculo de la radiancia para cada punto, gracias a la paralización de utilizar cores de GPU podemos obtener tiempos de photon mapping razonables.

Trabajo futuro

- Implementar el componente especular.
- Implementar mapa de causticas.
- Implementar el kd-tree en GPU.

Referencias

- https://developer.nvidia.com/rtx/ray-tracing/optix
- https://github.com/owl-project/owl
- A Practical Guide to Global Illumination using Photon Mapping, Jensen