Implementación del algoritmo de path tracing en la GPU

Joaquim Romo

Treball Final de Grau / 2014

DIRECTOR
Javier Agenjo Departamento GTI

Resumen

El presente trabajo tiene como objetivo el estudio y la implementación de un algoritmo de renderizado con raytracing estocástico en la unidad de procesamiento de gráficos (GPU). Se ha elegido realizar la implementación en una arquitectura de este tipo debido a las ventajas que ofrece en cuanto a tiempo de ejecución, gracias a la gran capacidad de cómputo en paralelo que ofrecen las arquitecturas de GPU actuales.

La primera parte del trabajo se dedica al estudio teórico del algoritmo de pathtracing, se comentan algunos conceptos físicos básicos relacionados con el transporte de luz y su interacción con los materiales así como las ecuaciones matemáticas y las técnicas estadísticas necesarias para la comprensión y correcta implementación del algoritmo.

En una segunda parte se discuten las tecnologías involucradas, concretamente el uso que se hace de la arquitectura CUDA, la librería OptiX y su funcionamiento y la implementación del algoritmo que se ha realizado sobre estas.

Índice

Índice de figuras

Índice de cuadros

Capítulo 1

INTRODUCCIÓN

1.1. Contexto

En el entorno de la imagen generada por computador siempre ha sido un reto tratar de generar imágenes lo más realistas posibles. Para ello un gran número de investigadores se han dedicado a diseñar algoritmos que simulan o imiten el comportamiento y la interacción de la luz con los materiales. Estos algoritmos que tratan de simular de forma realista el comportamiento de la luz son generalmente conocidos como algoritmos de iluminación global.

Estos algoritmos, por lo general, suelen tener una complejidad computacional muy elevada y el tiempo de cómputo necesario para obtener un resultado satisfactorio en escenas complejas era un factor limitador en su aplicación práctica. Por ello las aplicaciones que hacen uso de gráficos 3D en tiempo real típicamente se centran en la iluminación local o directa de los objetos de la escena y simulan la iluminación indirecta mediante técnicas que aun sin tener un fundamento físico ofrecen una mayor credibilidad para el ojo humano. Estas técnicas suelen ser algoritmos de postprocesado que se aplican en espacio de pantalla, por ejemplo "ambient occlusion" o "directional occlusion".

Sin embargo en los últimos años se han realizado grandes avances en las arquitecturas de las unidades de procesamiento de gráficos (GPUs), en especial la gran capacidad de cómputo en paralelo debido al elevado número de microprocesadores que forman estos dispositivos. Con tal de aprovechar estos avances en el hardware, los fabricantes de GPU han desarrollado librerías de computo generico (OpenCL, CUDA) que ofrecen gran libertad al programador para implementar sus propios algoritmos.

Estas mejoras han permitido realizar implementaciones de algoritmos de iluminación global en las GPUs que son mucho más rápidos que las implementaciones típicas en la CPU permitiendo reducir el tiempo de cómputo de varias horas o

días a minutos e incluso a ratios interactivos dependiendo de la GPU y algoritmos utilizados.

1.2. Algoritmos de iluminación global

Se conoce como algoritmos de iluminación global aquellos que tratan de simular distintos aspectos del comportamiento de la luz en su interacción con los objetos de una escena tridimensional. Algunos de ellos están pensados y optimizados para fenómenos concretos mientras que otros tratan de recrear fielmente todos los aspectos del transporte de luz.

En esta sección revisaremos por encima algunos de los algoritmos clásicos. Téngase en cuenta que no es el objetivo de este trabajo dar una explicación detallada de cada uno de estos algoritmos. Si el lector desea mas información sobre alguno de ellos se han citado las fuentes originales a las que puede remitirse.

1.2.1. Ray tracing

Aunque no se trata de un algoritmo de iluminación global propiamente dicho, el algoritmo de ray tracing original, desarrollado primeramente por Appel (year) y posteriormente ampliado por Whitted (year), es relevante por la influencia que ha tenido en el campo de los gráficos generados por computador y por que ha servido de base para métodos de iluminación global desarrollados posteriormente.

1.2.2. Radiosity

Radiosity fue el primero de los algoritmos de iluminación global que se desarrollaron. Inicialmente el algoritmo fue desarrollado en los años 1950 para aplicarlo al problema de la transferencia de calor. En 1984 fue modificado y adaptado por Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg y Bennett Battaile, investigadores de la universidad de Cornell para su aplicación en la generación de imagen sintética.

Este algoritmo trata de resolver el problema de la iluminación indirecta entre superficies puramente difusas o Lambertianas sin tomar en cuenta la reflectancia especular.

El funcionamiento del algortimo, en lineas generales, se basa en dividir la escena en pequeñas unidades de area, llamadas parches, que deberian funcionar como diferenciales de area. Luego a traves de una serie de iteraciones se intenta balancear el flujo de luz emitido, reflejado y absorbido entre todos estos parches.

1.2.3. Path tracing

El algoritmo de path tracing [Kajiya, 1986] sea posiblemente el primer algoritmo capaz de solucionar completamente la ecuación de renderizado.

Este algoritmo empieza como el ray tracing clásico, lanzado rayos desde la cámara hacia la escena, pero cuando un rayo intersecciona con un objeto se lanza un rayo en una dirección aleatoria para tener una estimación de cuanta luz indirecta llega a ese punto. Este rayo aleatorio a su vez es evaluado recursivamente siguiendo el mismo algoritmo.

Evidentemente este proceso de trazar un rayo desde la cámara y hacerlo rebotar por la escena para obtener un estimación de la luz es muy impreciso, por lo que es necesario repetir el proceso varias veces y hacer la media entre los resultados obtenidos para obtener una solución satisfactoria.

1.2.4. Bidirectional path tracing

El algoritmo de Bidirectional path tracing [Lafortune and Willems, 1993] fue desarrollado como una extensión al algoritmo de path tracing de Kajiya. En esta modalidad los rayos primarios no solo se lanzan desde la camara sino también desde las fuentes de luz. Estos caminos de luz, se calculan del mismo modo que los de la camara. Se guardan los puntos de intersección de los caminos de la camara y los de la luz y en una ultima fase se unen estos dos grupos de puntos para obtener la evaluación final del camino.

La principal mejora de este algoritmo respecto a su antecesor es que es capaz de funcionar mejor y converger mas rapido hacia una solucion correcta en escenas complejas en las que las fuentes de luz no son facilmente visibles desde la mayoria de puntos de la escena.

1.2.5. Metropolis light transport

Siguiendo en la linea de los dos algoritmos anteriores otra notable mejora llego con el llamado Metropolis light transport [Veach, 1997]. Este algoritmo parte de la base del path tracing bidireccional pero en vez de confiar en crear muchos paths hasta converger a una solución aceptable utiliza un método conocido como algoritmo de Metropolis-Hastings para generar varias mutaciones del mismo path.

Este algoritmo desata todo su potencial cuando se trata de renderizar interacciones complejas entre materiales que normalmente serian muy costosas de renderizar con los algoritmos que hemos comentado anteriormenteme. Por ejemplo causiticas, interreflecciones especulares-difusas, etc.

1.2.6. Photon mapping

Todos los algoritmos que estamos viendo tratan la luz como partículas y no como ondas, pero este algoritmo lo hace de un modo aun mas explicito. El algoritmo de Photon mapping [REFERENCIA JENSEN] empieza lanzando rayos (fotones) desde las fuentes de luz. Cuando estos fotones interseccionan con un objeto de la escena se decide aleatoriamente y segun las propiedades (BRDF) del material si el foton sera absorbido, dispersado especularmente o dispersado difusamente. Las posiciones finales donde los fotones son absorbidos se guardan en un mapa (kd-tree) de fotones para la siguiente fase.

La siguente fase, llamada final gathering, realiza un ray tracing de la escena y en cada intersección consulta el mapa de fotones para ver la cantidad de luz que llega a ese punto.

Este algoritmo sobresale entre todos los demás cuando se trata de renderizar causticas. Pero en cambio puede producir errores cuando se renderizan superficies difusas si el numero de fotones no es muy grande.

1.2.7. Instant radiosity

Este algoritmo, desarrollado por Keller en 1997, combina las ideas de los algoritmos de radiosity y photon mapping. Igual que el radiosity original, este algoritmo, en principio, solo funciona para superficies puramente difusas.

La idea general consiste en lanzar fotones desde las fuentes de luz (como en foton mapping) e ir guardando sus posiciones. La diferencia principal radica en que en la fase de renderizado, estos fotones son tratados como luces puntuales (VPLs, del inglés Virtual Point Lights) con orientación, es decir que tienen un vector normal. La ventaja es que una vez generados estos VPL es posible renderizar la escena mediante una API gráfica tradicional acelerada por hardware como OpenGL o DirectX.

1.2.8. Irrandiance caching

1.3. Tecnologías involucradas

El propósito de esta sección es explicar las principales tecnologías utilizadas durante el desarrollo de este proyecto. Debido al alcance de este trabajo la mayoría de tecnologías que comentaremos en este apartado son tecnologías especificas de las GPU.

1.3.1. Arquitectura de las GPU modernas

Las arquitecturas de GPU modernas siguen una paradigma conocido como SIMD (del inglés, Single Instruction Multiple Data) que consiste en la capacidad de un procesador de ejecutar la misma instrucción en paralelo sobre datos distintos. Es decir que que se ejecuta el mismo proceso en varias unidades de computo pero los datos que trata cada unidad pueden ser distintos.

1.3.2. Shaders

Los fabricantes de GPUs empezaron a explotar esta capacidad de computo en paralelo ofreciendo a los programadores de gráficos la posibilidad de programar ciertos puntos del proceso de renderizado efectuado por las GPU con los llamados shaders. Los shaders son pequeños programas que se ejecutan en la GPU y sirven para programar funcionalidades, tradicionalmente existían dos tipos de shaders: los vertex shaders, que se ejecutaban para cada vértice de cada primitiva a pintar y los pixel o fragment shaders que se ejecutaban para cada pixel rasterizado.

Según el paradigma SIMD, solo puede ejecutarse único shader en cada GPU pero los datos pueden ser distintos: en el caso de los vertex shaders, por ejemplo, cada unidad de procesamiento tendrá acceso a las coordenadas geométricas de un vértice, las coordenadas de textura de ese vértice, etc. Es decir que un vertex shader ejecutara exactamente las mismas operaciones para cada vértice en paralelo.

1.3.3. Capacidad de computo genérico

La principal limitación de los shaders es que solo trabajan con datos relacionados con el renderizado de gráficos (coordenadas de vértices, coordenadas de textura, vectores normales, texturas, etc). Si un programador quería utilizar la capacidad de computo en paralelo de las GPU para problemas distintos al renderizado por rasterizado de gráficos, debía buscar la manera de codificar los datos del problema en forma de vértices y texturas, lo cual podía resultar tedioso o complicado.

Debido a esta necesidad los fabricantes de GPU empezaron a buscar una forma de ampliar las capacidades de computo que ofrecían sus dispositivos y desarrollaron plataformas de programación genérica en GPUs. La primera de estas plataformas fue la desarrollada por Nvidia con el nombre CUDA para sus tarjetas gráficas. Posteriormente se desarrollo OpenCL, un estándar de computo en paralelo tanto para GPUs como para CPUs soportado por la mayoría de fabricantes de hardware.

1.3.4. CUDA

CUDA es la plataforma de computo genérico sobre GPU desarrollada por Nvidia para GPUs de Nvidia. EL compilador nvcc (NVidia C Compiler) permite tanto compilar programas con una parte en el host (CPU) y otra parte en el device (la GPU), como compilar kernels que se ejecutaran en la GPU.

1.3.5. OptiX

OptiX es una librería de ray tracing sobre CUDA desarrolada por Nvidia. Optix esta diseñado con la idea de ofrecer un framework para ray tracing lo mas genérico y programable posible, tratando de no limitar las posibilidades del programador.

1.4. Objetivos

Capítulo 2

FUNDAMENTOS TEÓRICOS

2.1. Unidades Radiométricas

Se conoce como radiometría al estudio de las radiaciones electromagneticas. Ya que la luz visible es una onda electromagnetica los algoritmos de renderizado que buscan el realismo se fundamentan sobre conceptos radiométricos. Por ello en esta sección haremos una pequeña introducción sobre algunos conceptos básicos que nos permitiran entender mejor los algoritmos de iluminación global.

2.1.1. Flujo

El flujo radiometrico mide la cantidad de energia radiante por unidad de tiempo. Sus unidades son Watts o Joules/segundo.

$$\Phi = \frac{dQ(t)}{dt} \tag{2.1}$$

2.1.2. Irradiancia

La irradiancia representa el flujo incidente en una superficie y se mide como el flujo radiante por unidad de area y sus unidades son de W/m^2

$$E = \frac{d\Phi}{dA} \tag{2.2}$$

2.1.3. Angulo solido

El angulo solido no es una unidad radiométrica en si mismo pero es un concepto geométrico necesario para poder explicar otros conceptos radiométricos además de otros apartados del presente trabajo.

Podemos entender el concepto de angulo solido como la extension del angulo a las tres dimensiones.

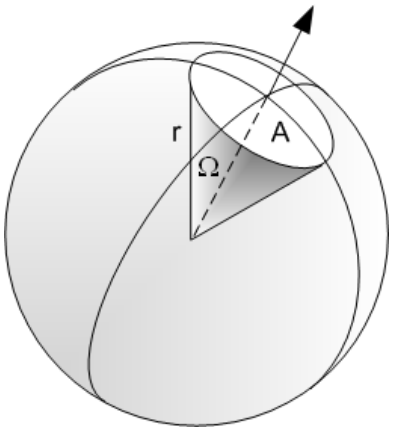

Figura 2.1: Definición de angulo solido [Haade, 2007]

El angulo solido se mide como el área proyectada sobre una esfera de radio unitario. Sus unidades son adimensionales y son llamadas stereorradianes $[sr]$.

$$\Omega = \frac{A}{r^2} \tag{2.3}$$

Usando coordenadas esféricas $\Theta = (\phi, \theta)$ podemos definir el angulo solido diferencial como:

$$d\omega_\Theta = \sin\theta d\theta d\phi \tag{2.4}$$

Informalmente resulta sencillo entender el angulo solido si pensamos en *cuan grande se ve un objeto*. Supongamos una superficie perpendicular a la dirección de visión del observador: si este objeto esta muy cerca, diremos que subtiende un angulo solido mayor que la misma superficie a una mayor distancia en la misma dirección.

2.1.4. Radiancia

La radiancia, también llamada intensidad por algunos autores [Kajiya, 1986, Immel et al., 1986], es probablemente la unidad radiometrica mas importante en lo que concierne al presente trabajo y a muchos de los algoritmos de iluminación global ya que su valor es invariante a lo largo de la longitud de un rayo.

Esta unidad mide la irradiancia por unidad de angulo solido.

$$L = \frac{dE}{d\omega} = \frac{d^2\Phi}{d\omega dA \cos\theta} \tag{2.5}$$

2.2. BRDF

La función de distribución de reflectancia bidireccional (de ahora en adelante BRDF, por sus siglas en inglés), definida por primera vez por [Nicodemus, 1965] Nicodemus (1965), es un función que define la respuesta a la luz de una superficie opaca, tomando como parámetros dos vectores unitarios que definen las direcciones de entrada y salida de la luz. Más formalmente, la BRDF mide la relación entre la radiancia diferencial reflejada en la dirección de salida y la irradiancia diferencial entrante en el ángulo sólido diferencial alrededor del vector de entrada

$$f(x, l, v) = \frac{dL(x \rightarrow v)}{dE(x \leftarrow l)} = \frac{dL(x \rightarrow v)}{L(x \leftarrow l) \cos\theta d\omega_i} \tag{2.6}$$

donde l es el vector unitario que apunta en la dirección opuesta a la de entrada de la luz y v es el vector unitario que apunta en la dirección de salida de la luz.

La BRDF solo esta definida para vectores l y v tales que $n \cdot v > 0, n \cdot l > 0$, siendo n la normal de la superficie.

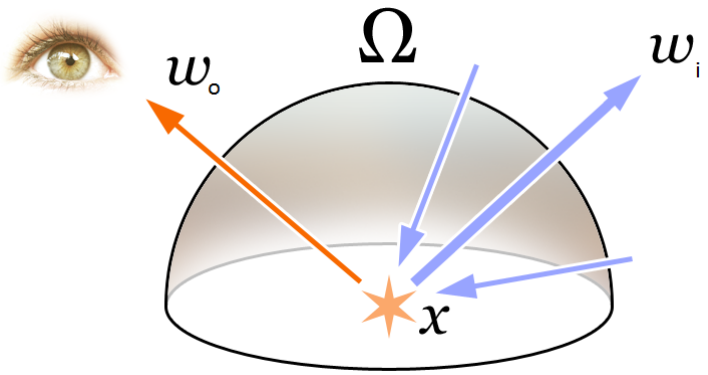

Figura 2.2: BRDF $l = \omega_i, v = \omega_o$

Para obtener la radiancia total reflejada en un punto x en la dirección saliente v es necesario integrar sobre el angulo solido en el dominio de la hemiesfera centrada en x.

$$L_o = \int_{\Omega_x} f(x, l, v) L_i(l)(l \cdot n) d\omega_i \tag{2.7}$$

2.2.1. Propiedades de la BRDF

Una BRDF debe cumplir ciertas propiedades para que sea físicamente plausible. En primer lugar debe cumplir la ley de conservación de la energía. En el caso que nos ocupa esto significa que una superficie puede absorber luz, transformándola en calor, o puede reflejarla pero en ningún caso puede reflejar mas energía lumínica que la que recibe.

$$\forall l, \int_{\Omega_x} f(x,l,v)(n \cdot v) d\omega_o \leq 1 \tag{2.8}$$

En términos informales esta ecuación significa que la integral de toda la luz reflejada debido a un rayo de luz entrante nunca podrá ser superior a la luz entrante por ese rayo.

Además también debe cumplir el *principio de reciprocidad de Helmholtz*, esto significa que si intercambiamos los vectores l y v su valor se mantiene. Este hecho cobra sentido si pensamos que la BRDF es una característica intrínseca de cada material y que al intercambiar los vectores v y l el angulo entre ellos sigue siendo el mismo.

$$f(x,l,v) = f(x,v,l) \tag{2.9}$$

2.2.2. Isotropía y anisotropía de la BRDF

2.2.3. Modelo de Phong modificado

2.2.4. Modelo de Blinn-Phong

2.2.5. Modelo de Cook-Torrance

2.2.6. Modelo de Ward

2.2.7. Modelo de Beckmann

2.3. Ecuación de renderizado

La ecuación de renderizado fue desarrollada en los años 80 simultaneamente y de forma independiente por distintos autores [Kajiya, 1986, Immel et al., 1986]. Se trata de una ecuación integral que unifica y formaliza los distintos algoritmos de renderizado, ya que hasta ese momento no existía un marco de trabajo teórico común.

Existen varias versiones de esta ecuación, según el autor que la use, que en general se pueden clasificar en dos tipos: las que integran sobre la hemiesfera, que se corresponde con la ecuación propuesta por Immel y las que integran sobre la unión de las superficies de la escena, que es la version propuesta por Kajiya.

Consideremos la ecuación 2.7 y consideremos que además de dispersar luz una superficie también puede emitir luz, siendo L_e la radiancia de la luz emitida, entonces tenemos la ecuación de renderizado.

$$L_o = L_e + \int_{\Omega_x} f(x, l, v) L_i(l)(l \cdot n) d\omega_i \tag{2.10}$$

Es decir, que la radiancia total L_o que sale de un punto x es igual a la suma de la radiancia emitida por ese punto en la dirección de salida v mas la integral de toda la radiancia que llega a ese punto y es reflejada en la dirección de salida.

Lo significativo de esta ecuación es que resulta muy intuitivo derivar algoritmos de renderizado de la misma: se evalúa para cada punto a pintar y se evalúa L_i recursivamente hasta que se cumpla determinada condición.

El problema es que no parece factible encontrar una solución analítica de esta ecuación y por este motivo se aplican métodos de integración numérica para aproximar una solución.

2.4. El método de montecarlo

El método de montecarlo se trata de un método de integración numérico para integrales definidas sobre un dominio de dimension arbitraria, del tipo:

$$I = \int_D f(x)dx, D \subseteq \mathbb{R}^m \tag{2.11}$$

Sabemos que la esperanza de una función continua se define como la integral de la función por la probabilidad de x. Y que podemos estimar la esperanza calculando la media de los valores que toma la función en puntos aleatorios escogidos independientemente y con la misma distribución.

$$E(f(x)) = \int f(x)p(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{2.12}$$

El método de montecarlo se basa en este hecho para estimar el valor de una integral definida tomando muestras aleatorias sobre el dominio $x_1, x_2, ..., x_n \in D$ y aplicando:

$$I = \int_D f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \tag{2.13}$$

Siendo $p(x_i)$ la probabilidad de tomar una muestra x_i concreta de entre todas las posibles en el dominio D. En el caso de tomar las muestras sobre una distribución uniforme en D:

$$\forall x_i, p(x_i) = \frac{1}{\int_D dx} \tag{2.14}$$

$$I \approx \frac{\int_D dx}{N}\sum_{i=1}^{N} f(x_i) \tag{2.15}$$

El error en una estimación de este tipo se reduce a medida que N crece.

2.4.1. Muestreo de importancia

Otra forma de reducir el error a parte de tomar mas muestras es tomarlas de forma mas inteligente. Anteriormente hemos supuesto que tomamos las muestras de una distribución uniforme sobre el dominio pero el método de montecarlo no impone ninguna limitación en este aspecto. Lo que implica que podemos tomar las muestras de otro tipo de distribuciones que sean mas apropiadas para cada caso. Por ejemplo tomando mas muestras en aquellas partes del dominio de integración que sean mas interesantes o importantes para nuestros propósitos.

Para ello basta con tomar las muestras $x_1, x_2, ..., x_n$ según la distribución usada y substituir $p(x_i)$ en la ecuación 2.13 por la probabilidad correspondiente.

2.4.2. Muestreo estratificado

El muestreo estratificado es otro metodo para reducir la variancia de la estimación. En este caso lo que se hace es dividir el dominio de la integral en regiones y aplicar montecarlo para cada región.

2.5. Aplicaciones del muestreo de importancia

2.5.1. Muestreo del angulo solido subtendido

2.5.2. Muestreo de la BRDF

Capítulo 3

FRAMEWORK

3.1. Cuda

3.2. La libreria OptiX

3.2.1. Host

clase Material

clase Geometry

3.2.2. Device

Intersection programs

Closest hit programs

Any hit programs

Bibliografía

[Goral et al., 1984] Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B. (1984). Modeling the interaction of light between diffuse surfaces.

[Haade, 2007] Haade (2007). Solid_Angle.

[Immel et al., 1986] Immel, D., Cohen, M., and Greenberg, D. (1986). A radiosity method for non-diffuse environments. *ACM SIGGRAPH Computer ...*, 20(4):133–142.

[Kajiya, 1986] Kajiya, J. (1986). The rendering equation. *ACM Siggraph Computer Graphics*, 20(4):143–150.

[Lafortune and Willems, 1993] Lafortune, E. and Willems, Y. (1993). Bi-directional path tracing. *Proceedings of CompuGraphics*.

[Nicodemus, 1965] Nicodemus, F. (1965). Directional reflectance and emissivity of an opaque surface. *Applied Optics*, 4(7):767–773.

[Veach, 1997] Veach, E. (1997). *Robust Monte Carlo methods for light transport simulation*. PhD thesis.