UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Tesis de grado

Iluminación global con superficies especulares

Bruno Sena

Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Junio de 2019

UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Tesis de grado

Iluminación global con superficies especulares

Bruno Sena

Tesis de grado presentada en la Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de grado en Ingeniería en Computación.

Directores:

José Aguerre

Eduardo Fernández

Montevideo – Uruguay

Junio de 2019

Sena, Bruno

Tesis de grado / Bruno Sena. - Montevideo: Universidad de la República, Facultad de Ingeniería, 2019.

VII, 32 p.: il.; 29, 7cm.

Directores:

José Aguerre

Eduardo Fernández

Tesis de Grado – Universidad de la República, Ingeniería en Computación, 2019.

Referencias bibliográficas: p. 30 – 32.

1. iluminación global, 2. radiosidad, 3. reflexión especular. I. Aguerre, José, Fernández, Eduardo, . II. Universidad de la República, Ingeniería en Computación. III. Título.

INTEGRANTES DEL TRIBUNAL DE DEFENSA DE TESIS

_____

NombreTribunal1 ApellidoTribunal1

_____

NombreTribunal2 ApellidoTribunal2

_____

NombreTribunal3 ApellidoTribunal3

Montevideo – Uruguay
Junio de 2019

RESUMEN

Aquí va el abstact

Palabras claves:
iluminación global, radiosidad, reflexión especular.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación y problema

1.2. Objetivos

1.3. Resultados esperados

1.4. Estructura del documento

Capítulo 2

Estado del arte

2.1. Modelos de iluminación

El proceso de dibujado de gráficos tridimiensionales por computadora comprende la generación automática de imágenes de cierto nivel de realismo a partir de modelos que componen una *escena* o *mundo* tridimencional, junto a un conjunto de cualidades físicas que rigen las formas en la que la luz interactúa con los objetos.

Trivialmente, esto puede ser reducido al problema de cálculo del valor de intensidad lumínica observada en un punto x y proveniente de otro punto x'. Matemáticamente, este problema fue planteado por Kajiya en 1986, comúnmente denominado «la ecuación del rendering»:

$$I(x, x') = g(x, x')\left[\epsilon(x, x') + \int_S \rho(x, x', x'')I(x', x'')\delta x''\right] \tag{2.1}$$

donde:

- $I(x, x')$ describe energía de radiación lumínica observada en el punto x proveniente de x''
- $g(x, x')$ es un término geométrico, toma el valor de 0 si existe oclusión entre x' y x en otro caso su valor es $\frac{1}{r^2}$ donde r es la distancia entre x' y x
- $\epsilon(x, x')$ mide la energía emitida por la superficie en el punto x' a x
- $\int_S \rho(x, x', x'')I(x', x'')\delta x''$ está compuesta por dos términos:
    - $\rho(x, x', x'')$ es el término de dispersión de la luz que llega desde x'' a x desde el punto x'

- $I(x', x'')$ describe energía de radiación lumínica observada en el punto x' proveniente de x''

por lo que este término refiere a la intensidad percibida desde x considerando todos las reflexiones de luz posibles para el espacio S.

Existen distintos métodos de resolución de la ecuación del rendering, la mayoría implican aproximaciones dado el gran costo computacional requerido para calcular el valor exacto de $I(x, x')$. Estos métodos balancean el costo computacional de los algoritmos utilizados y la fidelidad con el valor final de la función. Dependiendo de las decisiones y simplificaciones consideradas existen dos clasificaciones posibles para el modelo: *local* y *global* 2.1.

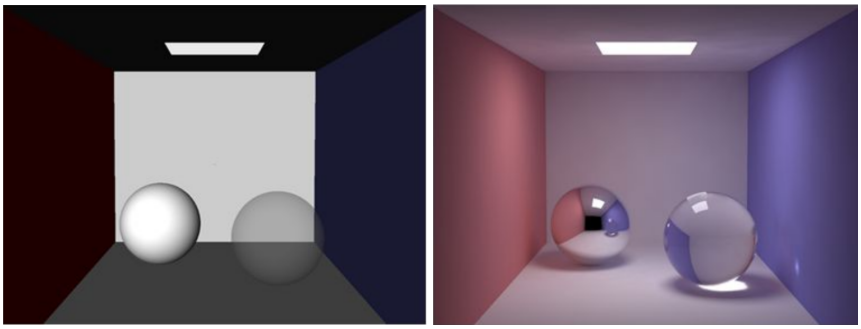

Figura 2.1: Dibujado utilizando iluminación local y global

2.1.1. Iluminación Local

Los modelos de iluminación local como el propuesto por Phong en 1975 tienen en cuenta las propiedades físicas de los materiales y las superficies de cada uno de los objetos de la escena de forma individual. Es decir, al dibujar uno de los objetos no se toman en cuenta las posibles interacciones de los haces de luz con los objetos restantes en la escena.

En referencia a la ecuación del rendering, el término geométrico nunca toma el valor 0 es decir, no se toma en cuenta las colisiones de los haces de luz con otros objetos, $\epsilon(x, x')$ toma un valor constante únicamente dependiente de x y $\int_S \rho(x, x', x'')I(x', x'')\delta x''$ toma el valor constante 1.

2.1.2. Iluminación Global

El término iluminación global refiere a una modelo de computación gráfica en donde se simulan parcialmente o completamente las interacciones de la luz con todos los objetos que se encuentran en la escena. Es decir, en contraposición

a la iluminación local, se consideran los fenómenos de reflexión y refracción de la luz.

Dependiendo de las característica de los modelos y algoritmos empleados, pueden obtenerse resultados más fieles a la realidad en distintos sentidos. El algoritmo de *trazado de caminos de rayos* emula completamente cada haz de luz desde su incepción en una fuente luminosa donde la granularidad impacta directamente en los errores en la imagen final, por otro lado el algoritmo de *mapeo de fotones* simula los efectos producidos por las colisiones de las partículas que componen la luz (fotones).

2.2. Radiosidad

El método de radiosidad es un método de iluminación global 2.1.2 que emula el transporte de la luz entre superficies regido por la propiedad física conocida como radiosidad, definida como el flujo de energía irradiada por unidad de área.

En este sentido, existen tres métodos posibles de cálculo de la ecuación de radiosidad: la integración basada en elementos finitos, la integración basada en reglas de cuadratura y la integración basada en métodos de Monte Carlo.

Originalmente, este modelo de iluminación global fue propuesto por Goral et al. en 1984, se basa en modelos matemáticos similares a los que resuelven el problema de la transferencia de calor en sistemas cerrados (MEF).

2.2.1. Radiosidad en superficies lambertianas

La solución propuesta por Goral et al. implica que todas las superficies son idealmente difusas, también conocidas como **lambertianas**.

Adicionalmente, se considerará que cada superficie irradia energía lumínica en todas direcciones en un diferencial de área δ_A, para una dirección de vista ω puede ser definida como:

$$i = \frac{\delta P}{\cos\phi\delta\omega} \tag{2.2}$$

donde:

- i es la intensidad de la radiación para un punto de vista particular

- δP es lae energía de la radiación que hemana la superficie en al dirección ϕ con ángulo sólido $\delta\omega$

En superficies perfectamente lambertianas, la energía reflejada puede ser expresada como: $\frac{\delta P}{\delta\omega} = k\cos\phi$. Donde k es una constante. Sustituyendo en (2.2) se obtiene: $\frac{\delta P}{\delta\omega} = \frac{k\cos\phi}{\cos\phi} = k$, esto implica que la energía percibida de un punto x es constante, independientemente del punto de vista.

Es por esto que la energía total que deja una superficie (P) puede ser calculada integrando la energía que deja la superficie en cada dirección posible, esto es, se integra la energía saliente en un hemi-esfera centrada en el punto estudiado:

$$P = \int_{2\pi} \delta P = \int_{2\pi} i\cos\phi\delta\omega = i\int_{2\pi} \cos\phi\delta\omega = i\pi \tag{2.3}$$

Por tanto, dada una superficie S_i, es posible calcular la energía lumínica que deja la superficie utilizando (2.3). Resta definir la *cerradura* de una superficie, definiremos la cerradura de una superficie como los límites que definen los puntos internos y externos de esta, llamaremos parche a cada una de estas superficies cerradas. Esto hace que el problema sea resoluble utilizando métodos de elementos finitos, con esta re-formulación del problema, es fácilmente trasladable a la ecuación (2.1).

$$B_j = E_j + \rho_j \sum_{i=1} NB_iF_{ij} \tag{2.4}$$

donde:

- B_j es la intensidad lumínica (radiosidad) que deja la superficie j.
- E_j es la intensidad lumínica directamente emitida por j.
- ρ_j es la reflectividad del material para la superficie j.
- F_{ij} se denomina *factor de forma*, un término que representa la fracción de energía lumínica que deja la superficie i y llega a j.

Cabe destacar que la naturaleza recursiva de la ecuación anterior, implica que se toman en cuenta todas las reflexiones difusas que existan en la escena. Como puede observarse, resolver el sistema de N ecuaciones lineales bastaría para conocer la energía emitida por cada superficie cerrada.

E, ρ dependen de los materiales que compongan la escena, son parámetros dados. Sin embargo, resta computar la matriz de factores de forma $\mathbf{F}$ para

finalmente obtener el vector de radiosidades B. Para determinar una entrada de la matriz F_{ij} involucrando a las superficies i y j de área $A(i)$, $A(j)$, considerando los diferenciales infinitesimales de área δA_i, δA_j, representados en la figura 2.2, el ángulo sólido visto por δA_i es $\delta\omega = \frac{\cos\phi_j \delta A_j}{r^2}$. Sustituyendo en (2.3) se obtiene:

$$\delta P_i \delta A_i = i_i \cos\phi_i \delta\omega \delta A_i = \frac{P_i \cos\phi_i \cos\phi_j \delta A_i \delta A_j}{\pi r^2} \tag{2.5}$$

Considerando que $P_i A_i$ es la energía que deja i, y que el factor de forma F_{ij} representa el porcentaje de dicha energía que llega a j podemos observar que:

$$F_{\delta A_i - \delta A_j} = \frac{\frac{P_i \cos\phi_i \cos\phi_j \delta A_i \delta A_j}{\pi r^2}}{P_i \delta A_i} = \frac{\cos\phi_i \cos\phi_j \delta A_i}{\pi r^2} \tag{2.6}$$

Integrando, para obtener el factor de forma para el área total:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \cos\phi_j \delta A_i \delta A_j}{\pi r^2} \tag{2.7}$$

De (2.7) se obtienen las siguientes propiedades:

1. $A_i F_{ij} = A_j F ji$
2. $\sum_{j=1}^{N} F_{ij} = 1$
3. $F_{ii} = 0$
4. F_{ij} toma el valor correspondiente a la proyección de j en una hemiesfera unitaria centrada en i, proyectándola a su vez en un disco unitario.

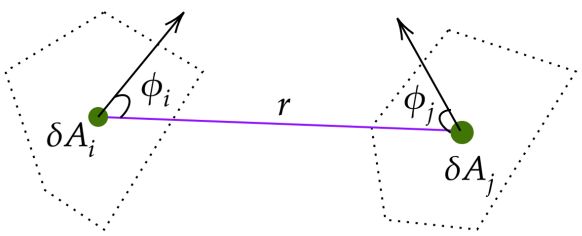

Figura 2.2: El factor de forma entre dos superficies

2.3. Métodos de cálculo de la matriz de Factores de Forma

El cálculo de los factores de forma a través de la ecuación (2.7) analíticamente es inviable en la práctica pues supone la necesidad de calcular la visibilidad entre cada par de parches que componen la escana. Por tanto, es necesario establecer otros métodos que provean aproximaciones lo suficientemente correctas.

Geométricamente, puese establecerse una analogía para la computación de factores de forma conocida como .ªnalogía de Nussel". Se expresará el factor de forma como la proporción de área proyectada de S_j en una hemi-esfera centrada en S_i y luego en un disco centrado en S_i.

El cálculo de la matriz de factores de forma **F** supone la proyección de los parches, de aquí en más se asumirá que estos parches son poligonales y por tanto es posible utilizar las técnicas de dibujado de objetos tridimensionales tradicionales.

2.3.1. Rasterización

La «tubería de renderizado» es un proceso de dibujado estándar que consiste en un conjunto de etapas bien definidas. Los fabricantes de los dispositivos aceleradores proveen de interfaces de programación (OpenGL, Vulkan, DirectX) que se basan en este modelo para abstraer el uso del hardware.

Si bien la «tubería de renderizado» es sumamente modificable por el programador, cada uno de estas etapas cuentan con pequeñas funciones, también llamadas *kernels* o*shaders* que son ejecutadas en la GPU y transforman los parámetros de la entrada en parámetros que recibirá la siguiente. A continuación, se describe el proceso para OpenGL 4.5 2.3, aunque muchas de estas etapas son trasladables a otras tecnologías.

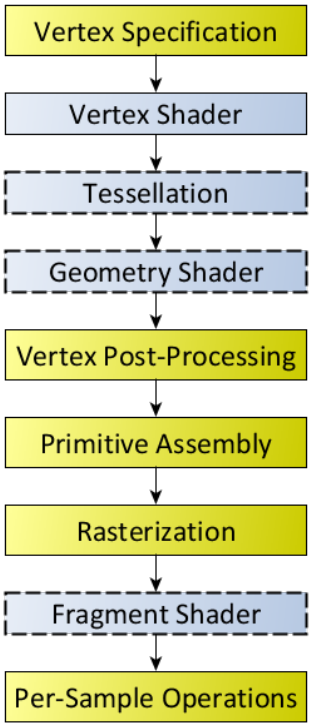

Figura 2.3: La *tubería de renderizado* en OpenGL

1. Procesamiento de primitivas geométricas: Inicialmente, las aplicaciones indican un conjunto de vértices a dibujar, definiendo cierto conjunto de primitivas geométricas como triángulos, cuadriláteros, puntos, líneas u otros. Luego, se procede al procesamiento de estos vértices:

   a) Vertex shader: Esta etapa convierte los vértices de entrada suministrados por la aplicación, generalmente se realizan las transformaciones necesarias para transformar el sistema de coordenadas del objeto a un sistema global. Las coordenadas retornadas deberán corresponderse con coordenadas del espacio de recorte. Es decir, coordenadas correspondientes al frustum de vista.

   b) Geometry shader: En esta etapa se procesan los vértices a nivel de primitiva geométrica, es decir, se recibe como parámetro una primitiva geométrica que se transforma en cero o más dependiendo de los parámetros deseados.

   c) Recortado: Esta etapa es *fija*, es decir, no es programable. Todas las primitivas calculadas anteriormente que residan fuera del frustum serán descartadas en las etapas futuras. Además, se transforma las primitivas a coordenadas de espacio de ventana.

d) Descarte: El proceso de descarte (en inglés *culling*) consiste en la eliminación de primitivas que no cumplan ciertas condiciones, como por ejemplo el descarte de caras cuya normal tiene dirección opuesta a la del observador.

2. Procesamiento de fragmentos (rasterización): El proceso de rasterización genera un conjunto de fragmentos, que se corresponden con los píxeles finales del resultado.

   a) Fragment shader: El procesamiento de cada fragmento se realiza a través del *fragment shader* que calcula uno o más colores, un valor de profundidad, y valores de planilla (del inglés *plantilla*).

   b) Scissor test: Todos los fragmentos fuera de un área rectangular definida por la aplicación son descartados.

   c) Stencil test: Los fragmentos que no pasan la función de planilla definida por la aplicación no son dibujados, por ejemplo, simular el *scissor test* que requieran primitivas más complejas.

   d) Depth test: En esta etapa se ejecuta el algoritmo del Z-Buffer, donde sólo se escribirá el resultado de aquellos fragmentos que tengan la menor profundidad. Es decir, los que se encuentren más cerca del observador.

Esta técnica de dibujado es extremadamente rápida, además la mayoría de dispositivos contienen hardware especializado capaz de acelerar estos cálculos, comúnmente conocidos como Unidades de Procesamiento Gráfico (o GPU en sus siglas en inglés). Con el objetivo de aprovechar este hardware Cohen and Greenberg idearon el método del hemi-cubo para el cálculo de factores de forma.

El método del hemi-cubo

El hardware optimizado para realizar operaciones de rasterización tiene la capacidad de proyectar escenas tridimensionales en imágenes bidimencionales a gran velocidad.

Para utilizar el hardware eficientemente consideraremos que se calculará una fila completa de $\mathbf{F}$, esto implica que dada $S_{i]}$, una superficie, calcularemos simultáneamente los factores de forma para las superficies restantes.

El método original propone la proyección de la escena una hemiesfera centrada en S_i, sin embargo los modelos de proyección utilizados no lo permiten.

Por esto es necesario proyectar la escena a un hemi-cubo centrado en S_i, esto supone el dibujado de cinco superficies bidimensionales, y por tanto puede ser realizada utilizando la rasterización.

Este método aprovecha el buffer de profundidad (Z-buffer), tomando en cuenta los píxeles proyectados para los elementos que se encuentren más cercanos al parche S_i.

El algoritmo, propuesto originalmente por Cohen and Greenberg en 1985, propone rasterizar la escena tridimencional en cinco texturas correspondientes al hemicubo, para cada pixel renderizado se sumará un valor diferencial del factor de forma, que dependerá de la posición del píxel en el hemi-cubo en relación a la hemiesfera que este aproxima. Esta suma genera una fila de la matriz $\mathbf{F}$, específicamente la fila $\mathbf{F}_i$.

Por tanto, podremos definir:

$$\mathbf{F}_{ij} = \sum_{q=1}^{R} \delta F_q \tag{2.8}$$

donde:

- R es la cantidad de píxeles correspondientes a la superficie S_j que cubren el hemi-cubo.
- δF_q el diferencial de factor de forma asociado al píxel del hemi-cubo q.

Los diferenciales de factores de forma deben corregir la deformación introducida con el cambio de proyección desde una hemiesfera a un hemi-cubo, para ello, para cada píxel que compone el hemi-cubo es necesario calcular la proporición de área que este término ocupa en la hemiesfera unitaria.

Para la cara superior, los diferenciales se calculan como:

$$\delta F_q = \frac{\cos\phi_i \cos\phi_j}{\pi r^2}\delta A = \frac{\delta A}{\pi(x^2+y^2+1)} \tag{2.9}$$

Para las caras laterales, la fórmula dada es:

$$\delta F_q = \frac{\cos\phi_i \cos\phi_j}{\pi r^2}\delta A = \frac{z\delta A}{\pi(x^2+z^2+1)} \tag{2.10}$$

2.3.2. Trazado de rayos

Otra de las técnicas de emulación de iluminación existente es el trazado de rayos, consiste en la computación de los puntos de intersección de una semi-

recta (a la que denominaremos rayo) con la geometría de la escena, cada uno de estos rayos emulará el transporte de los haces de luz emitidos.

Para cada uno de los rayos emitidos, se determinará el punto de intersección más cercano, luego es posible establecer qué primitiva geométrica fue interceptada y por tanto es posible deducir distintos valores dependiendo del problema en cuestión.

El trazado de rayos es una técnica efectiva [1] para computar la ecuación del rendering, utilizando la técnica de trazado de camino donde el haz de luz absorbe las propiedades de los materiales con los que interacciona. Por tanto, cada uno de los rayos observados computa uno de los integrandos de la ecuación.

El método de la hemi-esfera

En el caso de la radiosidad, es posible utilizar esta técnica para calcular los factores de forma, es decir, para resolver la ecuación (2.7).

Recordando, un factor de forma $\mathbf{F}_{ij}$ representa la energía que llega a la superficie S_i desde S_j, es posible re-imaginar el problema original colocando una hemi-esfera unitaria en el centro de S_i orientada en la direccción de la normal de la superficie.

El algoritmo propuesto por Malley consiste realizar un muestreo de la cantidad de rayos que parten desde el centro de S_i e intersecan S_j, las direcciones de los rayos serán determinadas a partir de la *distribución del coseno* cuya función de densidad es $f(x) = \frac{1}{2}[1 + \cos((x-1)\pi)]$.

$$\mathbf{F}_{ij} = \sum_{k=1}^{nMuestras} \frac{\beta(r, S_j)}{nMuestras} \tag{2.11}$$

donde:

$\beta(r, S_x)$ toma el valor 1 si el rayo r interseca a S_x o 0 en otro caso.

2.4. Superficies especulares

Originalmente, el método de cálculo de la radiosidad asume que todas las superficies son lambertianas, lo que supone que solo existirán reflexiones difusas cuando la luz interactúa con ellas. Sin embargo, es necesario simular reflexiones especulares correctamente para obtener resultados que se asemejen

a la realidad.

Por ello existe la extensión del método para superficies especulares, propuesto por Sillion and Puech en 1989. Los autores proponen extender el significado del término *factor de forma* a más que una mera relación geométrica entre parches. Sino que un factor de forma $\mathbf{F}_{ij}$ será la proporción de energía que deja la superficie i y llega la superficie j luego de un número de reflexiones y refracciones.

El algoritmo de cálculo consiste en el trazado de rayos desde S_i en una dirección arbitraria d, se distribuirá el valor final del factor de forma dependiendo en la cantidad de superficies con las que interaccione el rayo, la estructura recursiva formada por los distintos rebotes y refracciones del rayo original se conoce como *árbol de rayo* y dicta los valores finales del factor de forma final.

2.5. Cálculo del vector de radiosidades

Luego de computar la matriz $\mathbf{F}$ y dado los vectores de emisiones E y reflexiones ρ, resta computar el vector de radiosidades correspondiente para cada parche, denominado B.

Recordando (2.4), es posible deducir el problema al sistema de ecuaciones dado por:

$$E = (\mathbf{I} - \mathbf{RF})B \tag{2.12}$$

Los estudios de álgebra lineal modernos permiten la resolución de sistemas de ecuaciones de forma optimizada, dependiendo de las propiedades observadas.

Recordando las propiedades en 2.2.1, podemos observar que:

- $\sum_{j=1}^{N} \mathbf{F}_{ij} \leq 1 \forall i \in [1, N]$
- $\rho_i \leq 1 \rightarrow \sum_{j=1}^{N} \mathbf{R}_{ij} \leq 1 \forall i \in [1, N]$

Esto implica que las entradas de $\mathbf{RF}$ son siempre menores a 1, por tanto, $(\mathbf{I} - \mathbf{RF}) = M$ es diagonal dominante ya que $\sum_{j=1}^{N} |R_{ij}F_{ij}| \leq 1 \forall i \in [1, N]$ y $R_{ii}F_{ii} = 0 \forall i \in [1, N]$. Esto garantiza la convergencia del uso de métodos de resolución iterativos, como el algoritmo de Gauss-Seidel.

Si bien existen maneras alternativas de resolución del sistema planteado con consideraciones específicas del problema a efectos de esta investigación se

considerarán los métodos de resolución de ecuaciones lineales que requieran a lo sumo esta propiedad.

Cabe aclarar, que el método planteado hasta el momento resuelve la radiosidad en un único canal. Es decir, no se toma en cuenta todo el espectro electromagnético de la luz, es por ello que puede establecerse una extensión del método. Esta extensión implica la existencia de tres vectores de reflexión, uno para cada canal *RGB*. Por tanto es necesario que se resuelvan tres y no un único sistema de ecuaciones, aunque cabe destacar que la matriz $\mathbf{F}$ permanece constante, ya que las oclusiones geométricas no dependen de la longitud de onda.

Capítulo 3

Solución propuesta

3.1. Alcance y objetivos

Este proyecto se centra en la implementación completa de una aplicación capaz de calcular tanto la matriz de factores de forma utilizando los distintos métodos en 2, en escenas compuestas por triángulos y cuadriláteros con el objetivo de comparar el rendimiento entre los distintos algoritmos propuestos.

Estos incluyen:

1. Cálculo de factores de forma utilizando el hemi-cubo
2. Cálculo de factores de forma utilizando trazado de rayos
3. Cálculo de factores de forma extendidos utilizando dibujado de portales
4. Cálculo de factores de forma extendidos utilizando trazado de rayos

Además, será necesario implementar una interfaz de usuario que facilite la edición y visualización de los materiales que componen la escena.

3.2. Proceso de desarrollo

Dada la naturaleza del proyecto, fue deseable establecer una metodología de desarrollo para facilitar el proceso de seguimiento del progreso incluso cuando el equipo de desarrollo fue individual.

Para ello, internamente, se utilizó una metodología ágil de desarrollo similar a la conocida como *Kanban*.

Los principios claves del método aplicado a este proyecto fueron:

- La visualización sencilla del curso de trabajo (una lista de tareas a realizar conocida como *Backlog*)
- La limitación de las tareas en progreso.
- Dirigir y gestionar el flujo de trabajo implica la priorización de tareas a realizar dada una cantidad finita de recursos.

La gestión de las tareas a relizar se llevó a cabo en el respositiorio del proyecto, con tareas como las vistas en 3.1. Donde se consideran un conjunto de tareas:

- Backlog: Las tareas a realizar, en orden de importancia.
- In progress: Las tareas actualmente en desarrollo.
- Done: Las tareas cuya funcionalidad fue completamente desarrollada y probada.

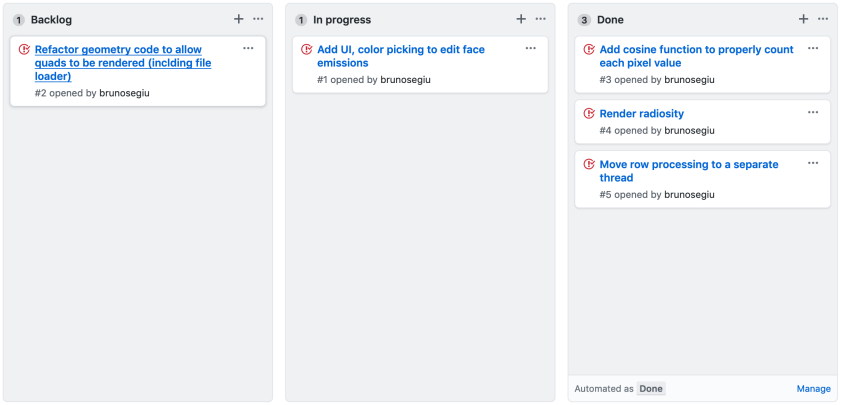

Figura 3.1: Tabla de Kanban utilizada en el proyecto

3.3. Diseño

Con la finalidad de evitar el alto acoplamiento, facilitar la extensión y reducir la cantidad de errores de integración se tomó la decisión de utilizar distintos módulos y sub-módulos que ofrezcan un conjunto de funcionalidades bien definido utilizando programación orientada a objetos. Esta decisión permite el añadido de nuevas características y la optimización de ciertas funcionalidades independientemente de los demás módulos contruidos.

El diseño de la solución comprende dos componentes principales, la interfaz gráfica de usuario (GUI, en inglés) y el motor de renderizado.

3.3.1. Motor de renderizado

El paquete del motor de renderizado se compone de un conjunto de sub-módulos, el primero de ellos que maneja el pre-procesado de una escena, es decir, el cálculo de l matriz de factores de forma y la radiosidad. El siguiente conjunto de sub-módulos se ocupan del renderizado en tiempo real de la escena, así como la carga de modelos desde el disco duro, la modificación de materiales, entre otras funcionalidades detalladas en 3.3.2.

Módulo de geometría

El módulo de geometría encapsula la información de las escenas leídas desde el disco duro, además de adaptar y optimizar los formatos de las primitivas geométricas para ser utilizados en las APIs de dibujado de terceros. La clase `Scene` 3.2 cargará distintos objetos desde el disco duro que serán manejados como una instancia de `Mesh`.

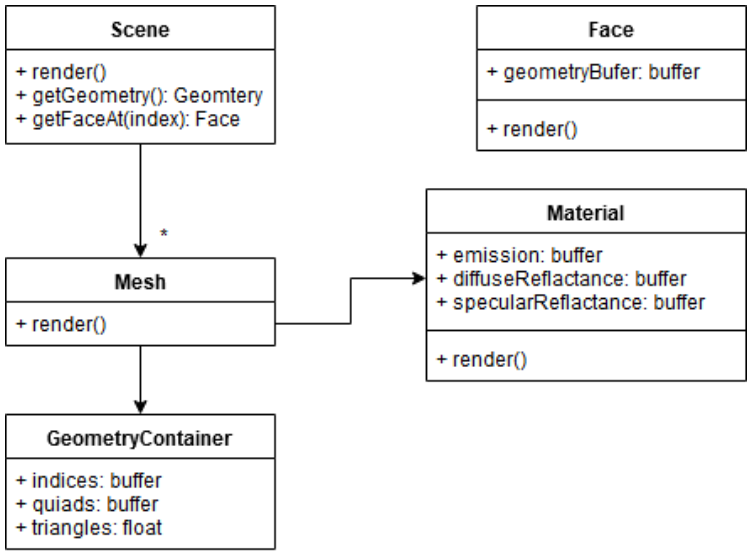

Figura 3.2: Módulo de manejo de geometría

Módulo de pre-procesado

El módulo de preprocesado se compone de un controlador principal (`PreprocessController` en la figura 3.3), que maneja el estado y la ejecución de los comandos de los distintos *pipelines* implementados que resuelven el cálculo de la radiosidad.

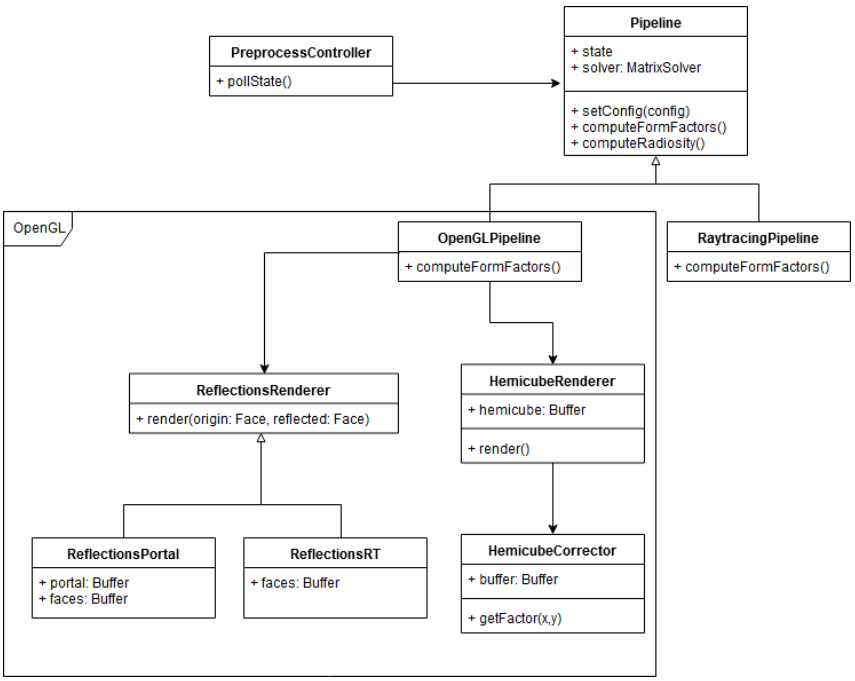

Figura 3.3: Arquitectura del módulo de pre-procesado

Un pipeline es definido a partir de un conjunto de funciones ejecutadas en el siguiente orden:

1. `setConfig(scene, interpolator, reflections, n_channels, solver)`
2. `computeFormFactors()`
3. `computeRadiosity()`

Donde `computeFormFactors()` variará dependiendo del método de cálculo elegido (veáse 3.3), donde puede utilizarse el método del hemi-cubo o el de la hemi-esfera. El primero de ellos utilizará un *pipeline* configurado utilizando una API de rasterización, mientras que el segundo utilizará una API capaz de calcular intersecciones utilizando *trazado de rayos*.

La ejecución de `computeRadiosity()` dependerá directamente de `solver` seleccionará el algoritmo que calculará el vector de radiosidades para la escena.

Módulo de visualización

El módulo de visualización se encarga de renderizar la escena actual desde el punto de vista seleccionado por el usuario. Además, debe tener la capacidad de mostrar las distintas propiedades de los materiales como valor de emisión inicial, valor de reflexión especular, visualización de geometría para facilitar la

edición de las propiedades de los objetos o sus caras. El proceso de dibujado comienza con el dibujante de la escena que dibujará un conjunto de imágenes correspondiente a las propiedades de los materiales `SceneRenderer` en la figura 3.4, luego un dibujante de texturas `TextureRenderer` seleccionará y convertirá correctamente el resultado anterior a valores tres canales (RGB). El módulo de visualización siempre tendrá una textura bidimensional como parámetro de salida.

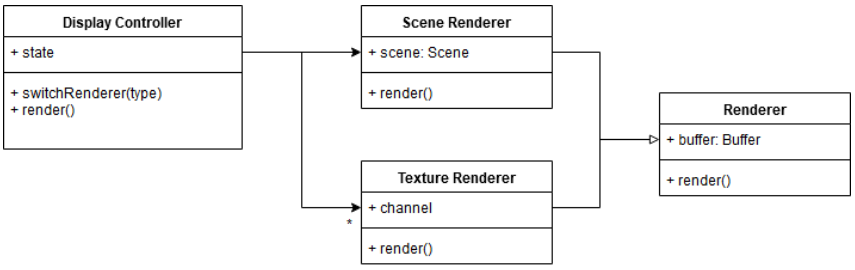

Figura 3.4: Arquitectura del módulo de visualización

3.3.2. Interfaz gráfica

El módulo de visualización (UI) utiliza una arquitectura basada en el paradigma del *bucle de eventos* 3.5, consiste en un bucle que detecta y maneja los distintos eventos recibidos por el sistema. Este método es útil para el manejo sencillo de la concurrencia en sistemas con múltiples hilos en ejecución y es de fácil implementación pues procesa cada uno de los eventos completamente antes de procesar el siguiente

En alto nivel, el *bucle de eventos* se compone de la siguiente forma:

while $queue.waitForEvent()$ **do**
  $queue.processEvent()$
end while

El *bucle de eventos* procesará todos los eventos de la aplicacación, lo que desencadena un conjunto de acciones que modificarán su **estado** de la interfaz de usuario. Este estado será dibujado por un conjunto de **componentes**, que no son más que presentadores del estado actual. Es decir, a partir de un conjunto de valores los presentarán en un formato gráfico adecuado y sencillo de comprender.

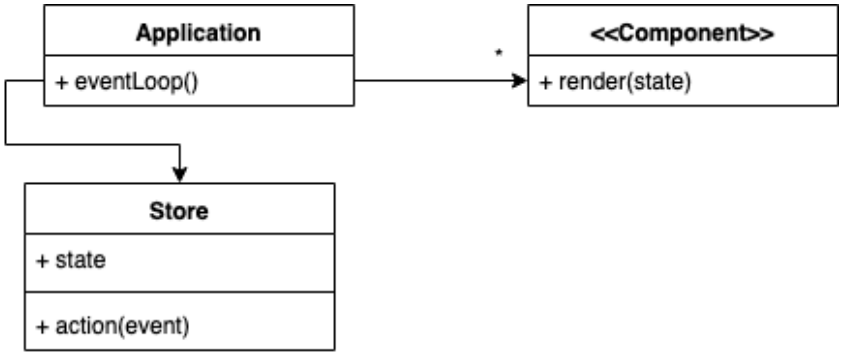

Figura 3.5: Arquitectura general del módulo de interfaz de usuario

Capítulo 4

Implementación

El siguiente capítulo encapsula los detalles de implementación y optimización de los algoritmos que se han desarrollado.

4.1. OpenGL

El algoritmo del hemi-cubo fue implementado utilizando la API OpenGL que provee de interfaces de alto nivel para la programación de tarjetas gráficas que facilitan el uso de algoritmos como el del Z-Buffer y facilita el manejo de memoria en la GPU.

4.1.1. Cálculo de factores de forma de la componente difusa

Para implementar el cálculo de factores de forma se implementará la función `computeFormFactors` como se aprecia en la figura 3.3. Recordando la arquitectura diseñada, la función debe computar completamente la matriz de factores de forma. Esta función seguirá un conjunto de etapas:

1. En primera instancia, se configurarán los *buffers* de memoria necesarios para representar el hemi-cubo que se dibujará. Para ello, se crea un *Frame Buffer Object* en la GPU que estará compuesto de 5 texturas, cada una de ellas correspondiente a una de las caras a dibujar. Cabe destacar, que estas texturas se compondrán de dos imágenes, una de ellas contiene enteros sin signo que serán utilizados para representar un `id` de cara

parche de la escena y la restante contiene los valores de profundidad necesarios para el algoritmo del Z-Buffer.

2. En la segunda se establecen las matrices de transformación de vista, es decir, las transformaciones lineales que alinean el volúmen de vista al hemicubo. Esto implica trasladar el origen de vista a el baricentro de la cara en cuestión, y alinearla a su normal como se ve en la figura.
3. En la tercer etapa se procede a dibujar cada objeto en la escena desde el parche considerado en las cinco texturas que componen el hemi-cubo. Con el objetivo de tener el mejor rendimiento posible, se hace uso de los *geometry shaders* para realizar una única llamada de dibujado por objeto. Este método solamente realiza un cambio de *render target*, una de las operaciones más costosas según 4.1. Como puede apreciarse en 4.1.1 solo se realiza una única llamada de *binding* del hemi-cubo. Específicamente, la implementación sigue el siguiente patrón:

    a) El *vertex shader* es simplemente *passthough* lo que signfica que conecta las entradas proveídas por la CPU con su salida.
    b) El *geometry shader* genera cinco primitivas donde cada una estará en las coordenadas correspondientes a los frustums de las caras del hemicubo además de añadir un plano adicional de corte del dibujo necesario debido a la imposibilidad de que las caras laterales posean una resolución menor a la cara superior.
    c) El *fragment shader* corregirá el identificador local `gl_PrimitiveId` a un identificador global y escribirá el identificador de la cara detectada en la textura que le corresponda.

function $processHemicube(face, hemicube)$
    $row \leftarrow [0, ..., 0]$
    loop $pixel \in hemicube$:
        $factor \leftarrow getHemicubeCorrection(pixel)$
        $seenFace \leftarrow getFaceId(pixel)$
        if $isValid(seenFace)$ **then**
            $row[seenFace] \leftarrow +factor$
            $formFactorMatrix[face] \leftarrow row$
        end if
    end loop
end function
function $computeFormFactors$
    $bindHemicube()$
    loop $face \in scene$
        $alignCamera(face)$
        $clearBuffers()$
        $render(scene)$
        $hemicube \leftarrow getHemicube()$
        $startThread(processHemicube, face, hemicube)$
    end loop
end function

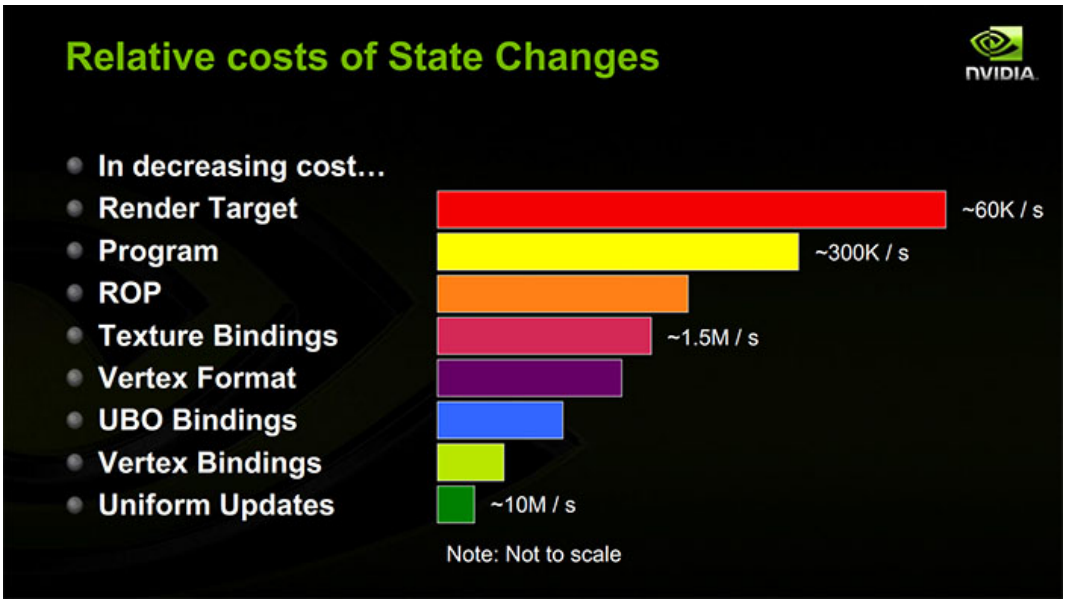

Figura 4.1: Costo de cambios de estado en OpenGL. Fuente: Nvidia

4.1.2. Cálculo de factores de forma de la componente especular

El cálculo de factores de forma extendido en fue implementado en dos variantes para el método del hemi-cubo. Ambas variantes son métodos de "dos pasadas", es decir, se dibujará el hemicubo normalmente y luego de determinar qué caras visibles son reflectivas se computará el conjunto de caras visibles gracias a la reflexión.

Los métodos utilizados proveen **aproximaciones**, ya que al contrario de la técnica de trazado de rayos se desconoce el punto exacto en que los rayos emitidos desde el hemicubo colisionan con los parches del entorno.

La primer variante utiliza el método de dibujado de portales en la GPU, mientras que la segunda fue implementada utilizando *trazado de rayos* en la CPU.

Dibujado de portales

El dibujado de portales es una técnica de dibujado en la rasterización en la que se dibujan únicamente los puntos cubiertos por cierta superficie (como una ventana, una puerta), normalmente se utiliza esta técnica para optimizar el dibujado de escenas donde la oclusión entre objetos es alta o la simulación de espejos planos.

Este método puede ser realizado de forma sencilla utilizando la GPU debido a los `Stencil Buffers`, similares a los buffers de profundidad, pero almacenan información arbitraria que puede ser utilizada para decidir qué *fragmentos* pasan la prueba del `Stencil Test`.

En el caso de la implementación propuesta, el primer paso consiste en dibujar un *stencil buffer* donde se encuentre el parche cuyo coeficiente de reflexión especular es mayor a cero desde la dirección simétrica como se aprecia en **??**. Luego, manteniendo el mismo volúmen de vista, se dibujarán los identificadores de los parches de forma similar al dibujado del hemi-cubo en una textura bi-dimensional. Este proceso se realiza a una resolución muy baja con el objetivo de preservar el rendimiento y minimizar el costo de transferencia de memoria.

Finalmente, se obtienen los identificadores de las caras reflejadas. En caso de que existan caras con valores de reflexión mayor a cero se vuelven a procesar, en otro caso se utilizarán los identificadores obtenidos para distribuir el factor de forma correspondiente al píxel del hemi-cubo entre los parches visualizados.

Trazado de rayos

El algoritmo para el trazado es una «traducción» del algoritmo utilizando dibujado de portales. En este caso, se procederá de forma similar, salvo que en lugar de utilizar la rasterización para dibujar las caras desde el punto de vista simétrico, se trazarán rayos hacia la escena en la dirección de reflexión a partir de un conjunto de puntos distribuidos en el parche reflectivo.

4.2. Embree

El algoritmo de la hemi-esfera fue implementado utilizando la biblioteca de traza de rayos Embree. Esta biblioteca soporta el trazado de rayos en la CPU en múltiples superficies, en particular, triángulos y cuadriláteros utilizando BHV (del inglés *Bounding Volume Hirarchies*). Estas estructuras de datos se basan en árboles que sub-dividen la escena en cada nivel donde cada nodo corresponde a un volúmen que cubre un conjunto de primitivas en el cual el cálculo de las intersección rayo-BV tiene un costo computacional despreciable. Si la intersección rayo-BV no existe, es posible descartar todos los elementos que le preceden.

4.2.1. Cálculo de factores de forma de la componente difusa

De forma similar a 4.1.1 es necesario posicionar el origen de cada rayo a trazar en el baricentro de la superficie. Luego, recordando (**??**), es necesario generar un conjunto de direcciones utilizando la distribución del coseno o similar. Si bien originalmente se utilizó la generación de números pseudoaleatorios para generar rayos correctamente distribuidos, el uso de números aleatorios perjudicó el rendimiento del algoritmo. Es por esto que se decidió utilizar otro algoritmo de generación de direcciones pseudoaleatorias en una hemi-esfera, en este caso la *regla general de particionado de una hemiesfera en celdas de área equitativa* propuesta por **?** .

Luego se procede a la traza de rayos, $\mathbf{F}_{ij}$ se calculará como $\frac{nIntersecciones_{ij}}{nMuestras}$, esto significa que por cada rayo que parta de la superficie S_i impactando S_j se adiciona $\frac{1}{nMuestras}$ al valor de la entrada correspondiente en $\mathbf{F}$.

4.2.2. Cálculo de factores de forma de la componente especular

En el caso del trazado de rayos, la extensión de los factores de forma es prácticamente trivial. Comprende la extensión de la función `traceRay()` a devolver un conjunto de caras intersecadas. Básicamente, si el rayo inicial interseca una cara cuyo coeficiente de reflexión especular es estrictamente positivo se almacenará el total de $k(1-\rho_j)$ (donde $k = \frac{1}{nMuestras}$) como contribuyente del factor de forma $\mathbf{F}_{ij}$ y se calcularán las siguientes intersecciones con el *residuo* de la reflexión que se distribuirá en los factores de forma que correspondan a la reflexión. Es decir, suponiendo que un rayo impacta S_k desde el camino (S_i, S_j) donde $\rho_j \geq 0$ se agregará $k\rho_j * (1-\rho_k)$ y se procederá de forma recursiva hasta que $\rho_z = 0$ para una superficie intersecada S_z o se alcance el máximo límite de recursión.

4.3. Interfaz de usuario

Capítulo 5

Experimental

5.1. Hardware

5.2. Escenas

5.3. Casos de prueba

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

6.2. Trabajo futuro

Lista de figuras

Lista de tablas

APÉNDICES

Apéndice

Referencias bibliográficas

[1] James T Kajiya. The rendering equation. In ACM SIGGRAPH computer graphics, volume 20, pages 143–150. ACM, 1986.

[2] Bui Tuong Phong. Illumination for computer generated pictures. Communications of the ACM, 18(6):311–317, 1975.

[3] Cindy M Goral, Donald P Torrance, Kenneth E'; Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In ACM SIGGRAPH computer graphics, volume 18, pages 213–222. ACM, 1984.

[4] Michael F Cohen and Donald P Greenberg. The hemi-cube: A radiosity solution for complex environments. In ACM SIGGRAPH Computer Graphics, volume 19, pages 31–40. ACM, 1985.

[5] TJ Malley. A shading method for computer generated images. Master's thesis, Dept. of Computer Science, University of Utah, 1988.

[6] Francois Sillion and Claude Puech. A general two-pass method integrating specular and diffuse reflection. In ACM SIGGRAPH Computer Graphics, volume 23, pages 335–344. ACM, 1989.