Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Lecturas en Ciencias de la Computación
ISSN 1316-6239

Estado del Arte en Iluminación Global de Escenas

Esmitt Ramírez

RT 2012-03

Centro de Computación Gráfica de la UCV
CCG-UCV
Caracas, Junio 2012.

Estado del Arte en Iluminación Global de Escenas

Esmitt Ramírez J.

Junio 2012

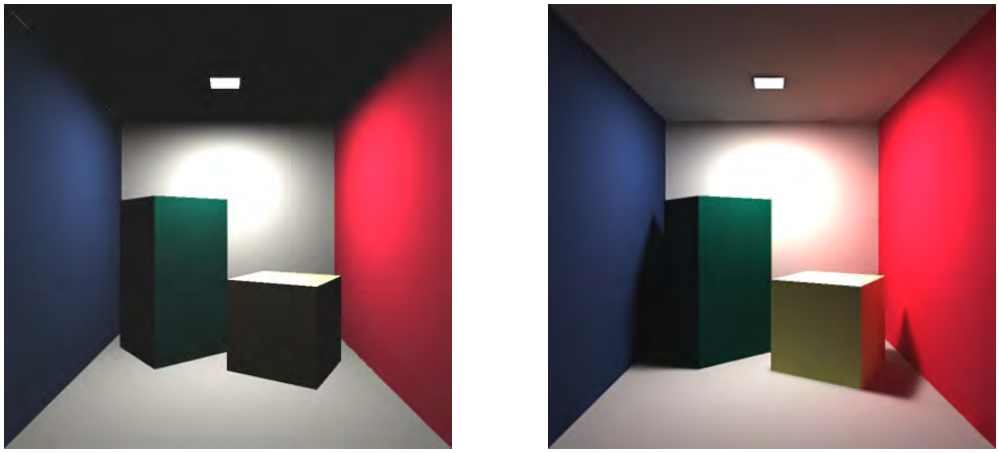
Resumen

En el área de computación gráfica, la iluminación global se define como el efecto causado por la incidencia de la luz sobre una superficie (iluminación directa) y su interacción de ésta con otras superficies (iluminación indirecta). El objetivo es lograr escenas 3D con un alto nivel de realismo basados en la interacción de generan las fuentes de luz en una escena. Durante muchos años se han creado diversas técnicas y algoritmos que permiten simular o representar de forma adecuada este fenómeno. Recientemente, gracias a las capacidades de los procesadores gráficos modernos (GPU), se han realizado diversas investigaciones en desarrollar métodos que ejecuten estos algoritmos en tiempo real. Las técnicas aplicadas en estos métodos tienen muchas variantes para resolver el problema, bien sea modificando el diseño de los algoritmos o creando vertientes que dan resultados aceptables desde un punto el vista visual para una escena. En este documento se muestra una serie de técnicas existentes en la literatura para lograr la iluminación global de una escena así como las principales vertientes creadas a través de los años. Actualmente, el objetivo es lograr aproximar el cálculo de la iluminación global en tiempo real en cualquier escena empleando la GPU para acelerar los cálculos computacionales para el despliegue.

Palabras Claves: Iluminación global, ray tracing, radiosity, GPU.

Uno de los dominios más importantes en el área de la computación gráfica es la creación de imágenes realistas. Generalmente, cuando existe una escena virtual con una cámara virtual, donde se quiere desplegar una imagen que sea lo más parecida a si esta escena realmente existiera. Para ello, es necesario representar todos los fenómenos que involucran los materiales presentes en la escena y la interacción entre estos y la luz. Por lo tanto, es necesario crear modelos que permitan representar estos fenómenos, de forma que sean simulados.

La generación de imágenes realistas tiene diversas utilidades: en la arquitectura (recreando los diseños antes de que se construyan), en el cine con efectos especiales, en los videojuegos, en la publicidad, en simulaciones de carros y vuelos de aviones, recreando objetos que todavía no han sido producidos, entre otros. Sin embargo, debido a las limitaciones de hardware, la generación de estas imágenes no siempre ha sido posible, ya que representar la interacción de la luz en una escena es costoso del punto de vista del cálculo computacional [1]. En un principio, el campo de la iluminación dentro de la computación gráfica se limitaba al trazado de rayos y puntos.



(a) Con iluminación local (b) Con iluminación global

Figura 1: Escena que representa un cuarto iluminado.

Los primeros algoritmos de iluminación consistían en asignar un color determinado dado el ángulo de incidencia de la luz sobre la superficie. Posteriormente Henri Gouraud [2] y Bui Tuong Phong [3] introdujeron sus modelos de iluminación a la computación gráfica. Sin embargo, estos solo consideraban la información local de la superficie, sin tomar en cuenta las demás entidades presentes en la escena. A las técnicas que solo usan información local, se les conoce como técnicas de iluminación local.

Por otra parte, existen las técnicas de iluminación global, las cuales toman en cuenta la interacción de la luz con todas las entidades de la escena. El objetivo de los algoritmos de la iluminación global es el cálculo del estado de la distribución de la energía luminosa en una escena en un momento determinado. Debido al ámbito global de estos algoritmos, son computacionalmente costosos y su implementación para tiempos interactivos es difícil de alcanzar. En la figura 1 se puede apreciar una escena iluminada bajo iluminación local, 1(a), e iluminación global, 1(b). Es importante destacar las diferencias a nivel visual que presentan ambas imágenes. El efecto de color generado sobre las paredes del cuarto así como las sombras, representan una diferencia notable entre ambas técnicas. Para poder conocer los trabajos actuales en este campo de estudio, primero se presentarán algunos conceptos básicos.

En este documento, primeramente se presenta los efectos creados por la interacción de la luz sobre una superficie. Luego, en la sección 2 se muestra la ecuación de despliegue, que sirve como base para todos los algoritmos de iluminación global. La sección 3 de algoritmos de iluminación global, se listan los principales algoritmos existentes de acuerdo a la literatura consultada. En la sección 4, se muestra una clasificación existente en la tendencia de los algoritmos actuales de iluminación global en tiempo real. Luego, la sección 5 muestra algunas tecnologías para el soporte de la iluminación global. Finalmente, la sección 6 presenta las conclusiones de este trabajo.

1. Efectos creados por la interacción de la luz

Dado que la iluminación global tiene como objetivo calcular la distribución de la energía de la luz en una escena, es importante aclarar los diversos efectos que se crean por la interacción de la luz con las superficies presentes en una escena. Los diferentes algoritmos propuestos en la actualidad tratan de recrear la mayoría de estos efectos, pero a mayor cantidad de estos, mayor el procesamiento requerido para calcularlos. A continuación se presentan los efectos que generalmente son buscados cuando se espera lograr un algoritmo de iluminación global:

1.0.1. Luz directa

Se refiere al rayo de luz que viaja directamente desde una fuente de luz hasta una superficie. No se consideran los diferentes caminos que podría tomar la luz para incidir sobre la superficie.

1.0.2. Luz indirecta

Es la luz que ha sido rebotada, dispersada, reflejada o refractada desde otra superficie antes de iluminar otro objeto. El despliegue de imágenes que contengan luz indirecta permite obtener un realismo muy alto, pero muy costoso.

1.0.3. Sombra

Es el efecto ocasionado por un objeto que obstaculiza el paso normal de la luz, resultando el obscurecimiento de los objetos ocultos. En computación gráfica se utilizan principalmente dos tipos de sombras: duras y suaves. En la figura 2(a) se puede observar un ejemplo de sombra dura, y en la figura 2(b) un ejemplo de sombra suave.

(a) Sombra dura (b) Sombra suave

Figura 2: Efecto de sombra creado por un objeto 3D (robot).

- **Sombras duras:** Poseen un borde bien definido, creando una transición abrupta entre el área sombreada y el área iluminada. Estas sombras generalmente son creadas en presencia de una luz puntual e infinitamente pequeña, la cual no existe en la naturaleza, por lo que representa solo una aproximación.

- **Sombras suaves:** Poseen una transición suave entre el área sombreada y el área iluminada. Estas sombras se producen debido a que generalmente un punto en la superficie ve una porción de la fuente de luz. Se puede distinguir entre la penumbra, donde la fuente de luz es parcialmente visible, y la umbra, donde la fuente de luz está totalmente bloqueada.

1.0.4. Refracción y reflexión

La refracción es el fenómeno por el cual un rayo de luz incidente sobre una superficie cambia su dirección. Sólo se produce sobre una superficie que separa dos medios que poseen índices refractarios diferentes, como se observa en la figura 3 los índices n_1 y n_2. Por su parte, la reflexión representa un cambio de dirección del rayo, pero en este caso existe un rayo reflejado que forma el mismo ángulo con el vector normal de la superficie que el rayo incidente.

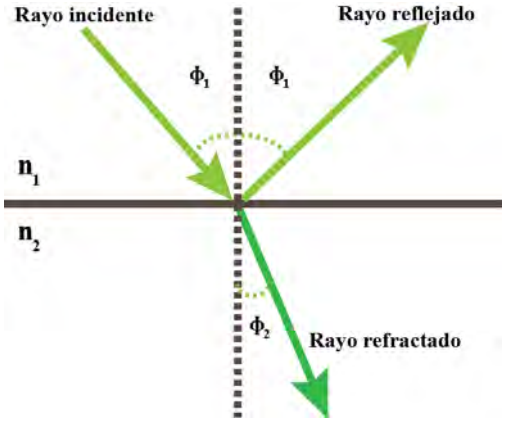


Figura 3: Un rayo incidente sobre una superficie con índices refractarios diferentes es reflejado y refractado.

1.0.5. Transparencia y translucidez

La transparencia es una propiedad óptica de la materia que tiene diversos grados y propiedades, la cual describe la transmisión de la luz a través de objetos sólidos, haciendo posible que se pueda ver a través de ellos. La translucidez es un grado de transparencia que permite la transmisión de la luz difusa a través de un objeto sólido presentando una alta dispersión de la luz incidente, lo que podría ocasionar una distorsión en la forma como se percibe un objeto a través del objeto translúcido.

1.0.6. Medios participantes

Son los diferentes medios que pueden encontrarse en una escena, por los cuales tiene que pasar la luz. Esto causa distorsiones en el rayo luminoso y crea diferentes efectos dependiendo del medio participante. Un ejemplo se puede observar en la figura 4, donde se muestra a la neblina como un medio participante.

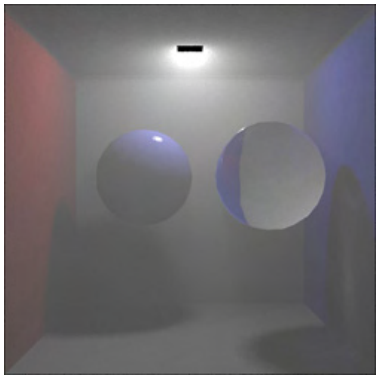


Figura 4: Escena desplegada con neblina como medio participante.

1.0.7. Dispersión

La dispersión (*scattering*) es el efecto causado por la dispersión de la luz cuando interactúa con alguna superficie. En la figura 5(a) se puede apreciar un busto de mármol, el cual presenta dispersión de la luz en su superficie.

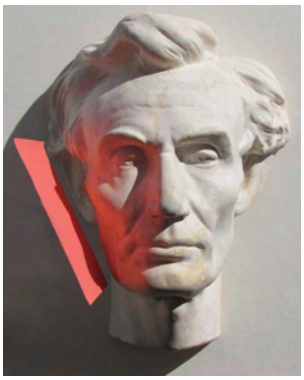
1.0.8. Sangrado de Color

El sangrado de color (*color bleeding*) resulta de la transferencia de color entre objetos cercanos, causado por la reflexión coloreada de la luz indirecta. En la figura 5(b) se puede observar la coloración del busto dado el objeto de color rojo cerca de este.

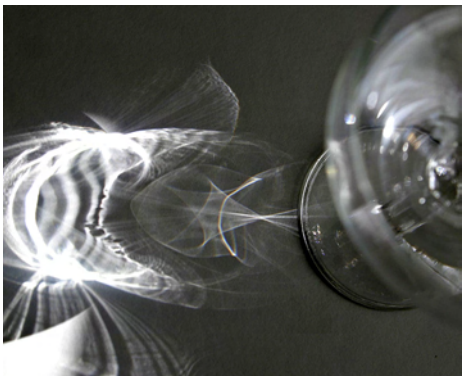
1.0.9. Cáustica

Un efecto de caústica se presencia cuando la luz es reflejada desde una superficie especular o reflectiva, o bien concentrada a través de una superficie refractiva, de tal manera que ilumina indirectamente otras superficies con patrones de luz concentrados. En la figura 5(c) se muestra al efecto de caústica generado por la copa sobre la superficie donde yace.

(a) Dispersión



(b) Sangrado de color



(c) Cáusticas

Figura 5: Diversos efectos de la luz indirecta.

1.0.10. Difracción

La difracción es un fenómeno físico característico de las ondas (lo que incluye las ondas electromagnéticas que definen la luz) que consiste en la inclinación, dispersión e interferencia de la onda al encontrar un obstáculo o apertura que rompe con su trayectoria. Por ejemplo, cuando un haz de luz incide sobre la superficie de un prisma, esta se descompone en fracciones de luz con diferentes longitudes de onda.

1.0.11. Superficies difusas

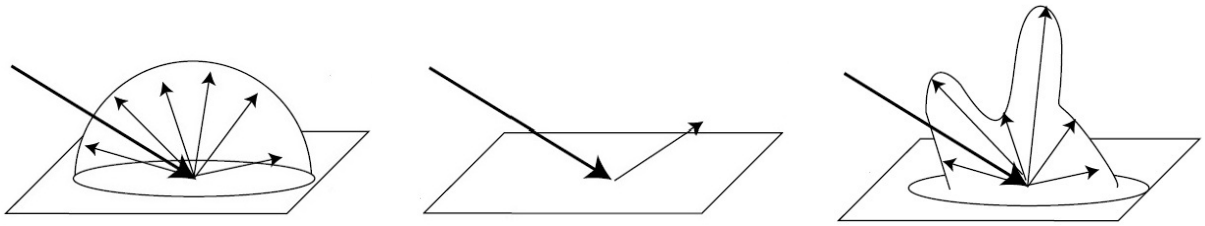
Las superficies difusas reflejan la luz de manera uniforme en todas las direcciones posibles. En la figura 6(a), tomada de [1] se observa este fenómeno. Generalmente en los algoritmos de iluminación, se asume que las superficies son difusas perfectas.

1.0.12. Superficies especulares

Las superficies especulares perfectas son aquellas que reflejan la luz en una sola dirección específica. La figura 6(b) muestra una representación de como se realiza este reflejo sobre una superficie.

1.0.13. Superficies brillosa

Generalmente las superficies en la naturaleza no son puramente especulares ni puramente difusas, sino que exhiben una combinación de los dos comportamientos La figura 6(c) muestra un ejemplo de ello, estas superficies se denominan superficies brillosas, y son difíciles de modelar computacionalmente.



(a) Difusas perfectas (b) Especulares perfectas (c) Brillosas

Figura 6: Tipos de superficies.

2. Ecuación de despliegue

Como se mencionó anteriormente, el objetivo de la iluminación global es calcular la distribución de la energía de la luz en una escena para un momento dado. Debido a la alta complejidad de la interacción de la luz con los diferentes componentes de una escena es necesario tener una ecuación que nos permita representar estas interacciones de tal forma que sea factible su despliegue. La ecuación debe describir el flujo de la radiación a través de un ambiente tridimensional.

Esta ecuación fue introducida por primera vez en el campo de la computación gráfica por Kajiya [4] en el año 1986. Sin embargo, Philip Dutré et al. [1] proponen una ecuación más desarrollada, presentada en términos radiométricos de la siguiente forma:

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{\Omega_x} fr(x, \Psi \to \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) d_{\omega\Psi} \tag{1}$$

Esta ecuación está expresada en términos de radiancia, la cual expresa la cantidad de poder que es recibido en (o emitido desde) cierto punto en la superficie, por unidad de ángulo solido y por unidad de área proyectada (watts/steradian[1] $\times$ m^2). La radiancia saliente de un punto x en la dirección Θ ($L(x \to \Theta)$) es igual a la radiancia emitida en el punto x, saliendo en la dirección Θ ($L_e(x \to \Theta)$), más toda la radiancia proveniente del hemisferio iluminado que es incidente en el punto x y que es reflejada en la dirección Θ, ver figura 7. Esto se modela como una integral sobre todo el hemisferio del punto x, la función fr representa la función de distribución de reflectancia bidireccional (*BRDF* por sus siglas en inglés). La *BRDF* se encarga de definir sobre la esfera visible la relación existente entre la radiación incidente en la dirección Ψ y la radiación reflejada en la dirección Θ. Todas las luces incidentes en el ángulo Θ son sumadas en proporción al *BRDF* y al coseno del ángulo incidente, en [1] se explica con mayor detalle este proceso.

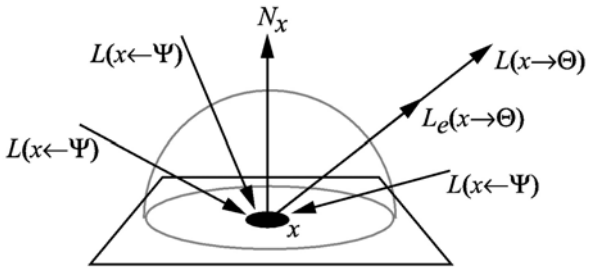


Figura 7: Representación de la radiancia emitida desde el punto x.

La ecuación de despliegue es una integral llamada la ecuación de Fredholm de segundo tipo, debido a su forma: la radiancia, que es el valor desconocido, aparece tanto en el lado derecho como el izquierdo de la ecuación.

Sin embargo, esta función es expresada en términos más simples por Kajiya [4] de la siguiente manera:

$$I(x, x') = g(x, x')[e(x, x') + \int_S p(x, x', x'') I(x', x'') d_{x''}] \tag{2}$$

Donde:
$I(x, x')$ es la intensidad de luz que pasa desde el punto x' al punto x.
$g(x, x')$ es un término geométrico.
$e(x, x')$ es la intensidad de luz emitida desde x' a x.

[1] Un *steradian* es la medida de la unidad para un ángulo sólido.

$p(x, x', x'')$ es la intensidad de luz dispersada desde x'' a x pasando por el punto x'.

Esta ecuación realiza el balance de la energía transmitida de un punto de una superficie a otro punto. En ella, se presenta que la intensidad de luz transportada desde un punto de una superficie a otro es la suma de la luz emitida y el total de luz dispersa hacia de todos los puntos de la superficie. Por lo mismo, pasa de tener una integral sobre el hemisferio a tener una integral sobre $S = \bigcup S_i$, la unión de todas las superficies.

A pesar de lo complejo de la ecuación, la misma no toma en cuenta todos los efectos creados por la luz. Entre otras cosas, asume que las superficies están en el vacío, donde carecen de un medio participante; no toma en cuenta el tiempo que tarda la luz en ser transportada, y no considera efectos como fosforescencia o fluorescencia. Sin embargo, en su trabajo, Kajiya [4] propone extensiones para abarcar estos efectos. A pesar de esto, esta ecuación representa una generalización para la mayor parte de los algoritmos existentes que se usan para el cálculo de la iluminación global.

La resolución de esta ecuación representa un gran reto aún con los computadores actuales, especialmente su resolución para aplicaciones en tiempo real. Por lo tanto, su resolución ha sido un gran campo de estudio en los últimos años y se ha llegado a diversos métodos que tratan de resolverla. Nótese que antes de la introducción de la ecuación ya existían algoritmos para la iluminación global, una gran mayoría de los algoritmos la utilizan como base de sus cálculos. En la siguiente sección se expondrán los algoritmos para el cálculo de la iluminación global.

3. Algoritmos de iluminación global

El objetivo de los algoritmos de iluminación global es calcular todas las posibles interacciones en una escena y así obtener una imagen altamente realista. Estas interacciones deben tomar en cuenta la combinación de las reflexiones especulares y difusas en la escena. A continuación, se describen una serie de algoritmos creados para simular estas interacciones.

3.1. Trazado de rayos

La idea que fundamenta las bases de esta técnica se remonta a la época del renacimiento donde el artista alemán Albrecht Dürer (1471-1528), hacía uso de un equipo que le permitía conseguir una correcta perspectiva de proyección para el desarrollo de sus trabajos en pintura. Su equipo consistía de un paraban que sostenía una red y un marco de madera cubierto con una malla de hilos, junto con un ocular representado por un pequeño obelisco, permitiendo a un artista reproducir la escena en una superficie de dibujo (ver figura 8).

Figura 8: Ilustración donde se observa a Albrecht Dürer en 1525 realizando un dibujo.

El trazado de rayos *(ray tracing)* fue uno de los primeros esfuerzos para lograr la iluminación global, introducida por Whitted en 1979 [5]. En su trabajo se describe una extensión al algoritmo de lanzado de rayos (*ray casting*) para poder determinar la visibilidad de las superficies e incluir el efecto de refracción y reflexión de superficies especulares perfectas.

El algoritmo consiste en calcular la radiación de luz que recibe un plano de imagen colocado en frente del punto de visión. Para ello, se traza un rayo que parte desde el punto de vista, a través de uno de los píxeles del plano de imagen, hacia la escena. Posteriormente, se calcula la intersección del rayo con el objeto más cercano de la escena para determinar su visibilidad. Sin embargo, en el punto de intersección se calcula el rayo refractado y el reflejado, y se lanzan nuevos rayos en esas direcciones, así como un rayo para cada una de las fuentes de luz. Estos últimos rayos permiten conocer la influencia de las diferentes fuentes de luz sobre la superficie, calculando así la luz directa. Los rayos refractados y reflejados son trazados con el mismo concepto, contribuyendo así al cálculo de la luz indirecta.

Debido a la alta recursividad del algoritmo y a la gran cantidad de rayos que se trazan, el algoritmo de trazado de rayos termina siendo ineficiente. También hay que tomar en cuenta que este método es dependiente del punto de vista, por lo que un movimiento de la cámara o de algún objeto de la escena requerirá el re-calcular la iluminación. Sin embargo, se producen imágenes altamente realistas, por lo que se han realizado grandes esfuerzos para acelerar el proceso. Además, se debe tomar en cuenta que este algoritmo no captura la mayoría de

los efectos de la iluminación global. Generalmente solo se encarga de simular las sombras y los rayos reflejados y refractados de superficies especulares perfectas. Por lo tanto, no puede simular refracciones y reflexiones difusas, cáusticas, etc. En la figura 9(a) se observa el esquema clásico, donde los rayos son trazados desde el ojo a través de un plano de imagen hacia la escena.

Posteriormente al trabajo de Whitted [5], diversas variantes han sido desarrolladas. A continuación se presentan algunos de los trabajos principales en dicha área.

3.2. Trazado de rayos distribuidos y trazado de caminos

Uno de los defectos del *ray tracing* original en cuanto a calidad, es que solo se trazaba un rayo de sombra, un rayo refractado y un rayo reflejado. Esto origina efectos que no son realistas. Por ello, Cook et al. [6] en 1984, proponen un enfoque denominado trazado de rayos distribuidos o trazado de rayos estocásticos *(distributed ray tracing)*. Aquí, se eliminan estas restricciones para promediar el valor obtenido al trazar diversos rayos sobre el intervalo. Así, por ejemplo, las sombras suaves pueden ser creadas a partir del trazado de varios rayos distribuidos sobre el área de las fuentes de luz. Siguiendo el mismo principio se pueden crear otros efectos avanzados que mejoran el realismo de la imagen creada.

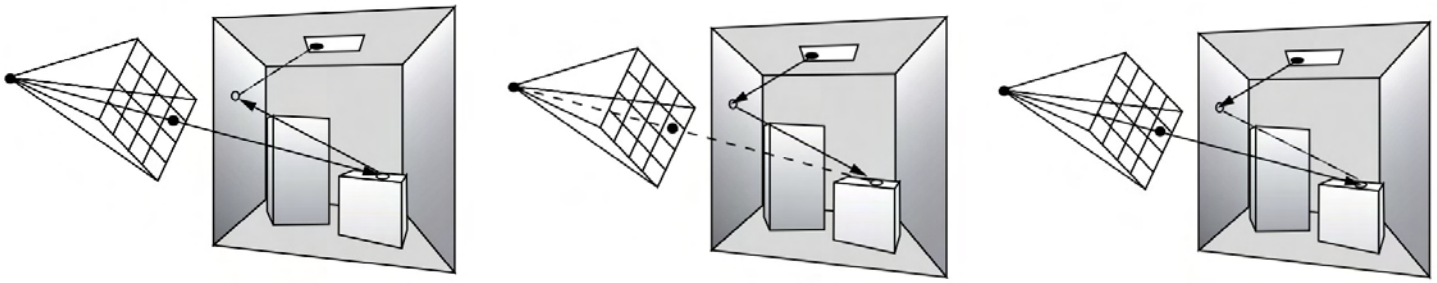
Con la publicación de la ecuación de despliegue de Kajiya [4], se observa que el algoritmo de *ray tracing* trata de resolver esta ecuación. Kajiya propone utilizar el método de integración de Monte Carlo para la resolución de la misma y la adapta al *ray tracing* para crear el método conocido como el trazado de caminos *(path tracing)*.

En este enfoque, se lanzan muchos rayos por píxel en vez de uno solo, y en lugar de separar los rayos en el punto de intersección, simplemente se sigue un solo camino. De esta manera, se evita la creación recursiva de rayos, que puede llegar a ser excesiva y que, en muchos casos, la mayoría de los rayos aporta poco a la imagen final. Este método se asemeja al *distributed ray tracing* debido a que en ambos se utiliza integración de Monte Carlo. Sin embargo, *distributed ray tracing* no presenta una solución completa al problema de la iluminación global, ya que solo estima la integral para interacciones con superficies especulares. Aunque estos esfuerzos representaron una mejora significativa en la calidad de las imágenes creadas, representan un mayor costo computacional que el *ray tracing* original, por lo que su despliegue resulta muy costoso para ser empleado en aplicaciones en tiempo real.

3.3. Trazado de rayos inverso

Aunque en el método original de *ray tracing* los rayos viajan del ojo hacia la luz, naturalmente es la luz la que viaja hacia el ojo. Arvo [7] toma esta idea para proponer el trazado de rayos inversos *(backward ray tracing)*.

En dicho trabajo se proponen dos pasadas. En la primera pasada los rayos son trazados desde la luz hacia la escena, donde se va depositando en las superficies parte de la energía emitida desde la luz. Esta energía es almacenada para la segunda pasada. En la segunda pasada se realiza un trazado de rayos convencional, pero se utilizan los valores almacenados para evitar la alta recursión. En la figura 9(b) se muestra el enfoque del *backwards ray tracing*, donde los rayos son trazados desde la luz hasta llegar al ojo.



(a) *Ray tracing* clásico (b) *Backwards ray tracing* (c) *Bidirectional ray tracing*

Figura 9: Diferentes vertientes de *ray tracing*.

3.4. Trazado de rayos bidireccional

Para ciertos efectos es más conveniente emplear el *ray tracing* convencional y para otros, el algoritmo de *backward ray tracing*. Lafortune y Willems [8] junto con Veach y Guibas [9] tomaron lo mejor de los dos enfoques y desarrollaron lo que se conoce como el trazado de rayos bidireccional *(bidirectional ray tracing)*.

Este algoritmo genera caminos de rayos que comienzan en el punto de una superficie y caminos de rayos que comienzan en la fuente de luz y uniéndolos en el medio para calcular la contribución de la luz en ese punto. Si tenemos un camino desde el ojo de cualquier longitud, este camino puede ser extendido, de tal manera que este camino y cualquier sub-camino pueden ser unidos a un camino de luz,

creando diversas combinaciones y posibilidades. En la figura 9(c) muestra el funcionamiento del *bidirectional ray tracing*, en donde rayos son trazados simultáneamente desde la luz y desde el ojo, y son intersectados en un punto medio.

De esta manera, se pueden obtener tanto efectos que eran mejor creados a través de rayos desde el ojo y efectos que eran mejor creados a través de rayos desde la luz, permitiendo así el despliegue de imágenes de calidad.

3.5. Transporte de luz de metrópolis

El transporte de luz de metrópolis *(Metropolis light transport, MLT)* es una técnica introducida por Veach y Guibas [10], para recrear caminos que son difíciles de capturar. Lleva su nombre debido a que hace uso del muestreo de Metrópolis [11]. El muestreo de Metrópolis es un técnica utilizada para generar secuencias de muestras de una función no negativa f, tal que las muestras están distribuidas acorde a f. El *MLT* utiliza esta técnica de muestreo al espacio infinito-dimensional de los caminos.

El *MLT* consiste en trazar un camino utilizando el *bidirectional ray tracing* y luego mutarlo de manera aleatoria, para conseguir caminos "cercanos" al primero encontrado. Cuando un camino es mutado debe ser aceptado o rechazado dado una probabilidad. Teóricamente, las mutaciones tenderán a caminos que aportan de manera significativa a la imagen final, y las mutaciones producidas a partir de estos también tenderán a aportar significativamente. Es por ello que en comparación con otras técnicas como el *path tracing*, *MLT* tiende a converger más rápido. Además, Veach y Guibas proponen una serie de mutaciones optimizadas a encontrar caminos de luz específicos. Por ejemplo, define mutaciones optimizadas para encontrar cáusticas.

En la figura 10, extraída de [10], se observa una escena desplegada con un *path tracing* en 10(a) y con un *MLT* en 10(b). La única fuente de luz de la escena se encuentra en el exterior de la habitación e ilumina a los objetos a través de la luz indirecta. Dado que un camino que aporte significativamente a la imagen final es difícil de encontrar, el *path tracing* presenta diversos artefactos que no se observan con el *MLT*.

(a) (b)

Figura 10: Escena desplegada en (a) con *path tracing* y con (b) *Metropolis light transport*.

3.5.1. Ray tracing en la GPU

Los cálculos de trazado de trayectoria se realizan de manera independiente por cada rayo, así que el ray tracing es una aplicación de naturaleza paralela. Las GPU's son procesadores paralelos que están basadas en un modelo computacional de flujo de datos llamado modelo *streaming*. De acuerdo a Purcell [12], el cómputo ocurre en respuesta a los datos que fluyen a través de una secuencia de pasos de procesamiento. Una función es ejecutada en un conjunto de registros de entrada (como los vértices o fragmentos en un *vertex* o *fragment program* de la GPU) y da como salida otro conjunto de registros. Este conjunto de registros finales corresponde a la información de color asignada a cada porción de un color *buffer*. En general, en la literatura se refiere a la función que realiza procesamiento sobre los registros como un *kernel* y a los conjuntos de datos como *streams*.

Empleando lenguajes de *shader*, como GLSL [13], Cg [14] y HLSL [15], una secuencia de *kernels* correspondientes a la funcionalidad del *ray tracing* se implementan usando una secuencia de programas de fragmentos. Este modelo de implementación de *kernels* secuenciales también resulta aplicable en lenguajes de programación de propósito general para la GPU, como OpenCL [16], CUDA [17] y DirectCompute [18]. En el año 2005, Christen [19] realiza un estudio más profundo del trabajo presentado en [12] por Purcell. En dicho trabajo se presenta una implementación de 5 *kernels*: generador de rayos, pre-cálculo de los vóxeles, recorrido del rayo, intersección del rayo y sombreado.

Un aspecto importante radica en las estructuras de datos empleadas en un algoritmo de *ray tracing* en la GPU. Entre las estructuras espaciales de datos más usadas en el *ray tracing* en la GPU se encuentran las jerarquías de volúmenes contenedores, los *kd-trees* y los *uniform grids*. Existe un debate abierto acerca de cual de estas estructuras de datos espaciales (y de cual implementación particular) resulta más apropiada para diversas aplicaciones de ray tracing y es posible apreciar variaciones de rendimiento de acuerdo a una escena.

En términos generales, se requiere poder calcular los efectos producidos por el *ray tracing* bien sea de forma separada (y luego mezclar los resultados) o de forma conjunta (buenas aproximaciones).

3.6. Radiosidad

La técnica de radiosidad *(radiosity)* fue propuesta por Goral et al. [20] para resolver la interacción de la luz para superficies difusas. Por lo tanto, solo resuelve parcialmente el problema de la iluminación global. Su idea principal es calcular el promedio de radiosidad B_i en cada elemento de superficie o parche i para una escena tridimensional.

Goral et al. [20] definen dos conceptos básicos para entender este método. El primero es el recinto, que son el conjunto de superficies que definen completamente el ambiente a ser iluminado. Todas las superficies deben ser subdivididas en estructuras denominadas *elementos* que luego pasarán a formar estructuras denominadas *parches*. Para cada elemento del recinto es necesario especificar la radiosidad emitida por el mismo elemento B^e por la superficie i, B_i^e, y su factor de reflectancia ρ_i. El segundo concepto son los factores de formas, los cuales se definen como la fracción de radiación de energía de luz, que parte desde una superficie hasta otra. La energía irradiada puede ser dependiente de un ángulo, pero este método fue desarrollado para solamente tomar en cuenta emisores y reflectores difusos ideales, los cuales no tienen dependencia de un ángulo.

Debido a que solo se toman en cuenta superficies difusas ideales, es posible transformar la ecuación de despliegue de Kajiya [4] de la siguiente forma (para más detalles revisar [1]):

$$B_i' = B_i^e + \rho_i \sum F_{ij} B_j' \tag{3}$$

Donde F_{ij} representa el factor de forma que va de la superficie-i a la superficie-j y pueden expresarse como se muestra en la ecuación 11. La figura 11 muestra de forma gráfica el principio matemático de los factores de forma.

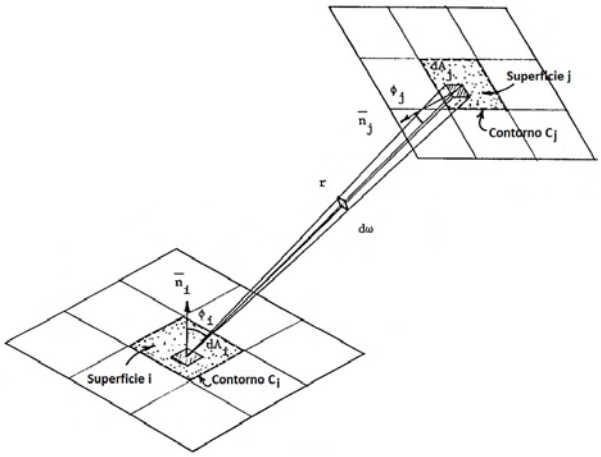


Figura 11: Representación del principio matemático de los factores de forma.

$$F_{A_i - A_j} = F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{cos\phi_i cos\phi_j dA_i dA_j}{\pi r^2} \tag{4}$$

Sin embargo, utilizando el teorema de Stokes [21, 22] es posible representar la ecuación de manera eficiente de la siguiente forma:

$$F_{ij} = \frac{1}{2\pi A_i} \oint_{C_j} \oint_{C_i} [\ln(r) dx_i dx_j + \ln(r) dy_i dy_j + \ln(r) dz_i dz_j] \tag{5}$$

A partir de estas ecuaciones, los pasos para realizar el algoritmo de *radiosity* son los siguientes:

- Discretización de la geometría en parches.
- Cálculo los factores de forma.
- Resolución del sistema de ecuaciones lineales propuesto en la ecuación 3.
- Despliegue de la solución, lo que incluye la transformación de los valores de radiosidad en color.

Debido al buen comportamiento de la matriz generada por el cálculo de los factores de forma, la resolución del sistema de ecuaciones es realmente rápido. También, debido al hardware actual, el despliegue de la solución se realiza en tiempos despreciables. El proceso que requiere mayor procesamiento es el cálculo de los factores de forma.

El primer problema lo representa el almacenamiento de la información. Esto se debe a que una escena puede ser discretizada en muchos parches y los factores de forma se tienen que calcular entre cada par de parches. Además, el cálculo de los mismos representa un gran reto. La primera solución factible para esto, es resolver numéricamente la ecuación utilizando técnicas de cuadratura e integración. Esta manera presenta una serie de inconvenientes reconocidos por los mismos desarrolladores [20]. De esta forma, se genera un sistema de ecuacíones de la forma $Ax = b$ donde el tamaño de la matriz depende del número de parches de la escena. Por lo tanto, se han desarrollado diferentes formas para realizar este cálculo, donde una de las más conocidas son los semicubos propuestos por Cohen et al. [23]. A raíz de estos inconvenientes, el cálculo de los factores de forma de manera eficiente ha sido de gran interés. Diversas técnicas y optimizaciones han sido desarrolladas.

Radiosity presenta la ventaja de que una vez realizado el cálculo de la iluminación es posible el despliegue de manera interactiva, ya que no depende del punto de visión. Sin embargo, se limita a escenas y luces estáticas, ya que un movimiento de cualquiera de estos dos implicaría volver a calcular los factores de formas, proceso que no se realiza de manera interactiva. Además, solo representa la interacción con superficies difusas y no da el aporte especular, restando así realismo a la escena.

Existen diversos tipos de algoritmos para realizar la radiosidad. Básicamente estos tipos se pueden dividir en:

- Clásico: Basado en el concepto de recolección de energía o *gathering*. Consiste en comprobar iterativamente, parche a parche, que cantidad de energía proveniente desde los elementos de la escena llega al parche que está siendo analizado. Es importante, encontrar un equilibrio entre velocidad y calidad, ya que a mayor número de subdivisiones, el cómputo gráfico es mayor; sin embargo, la escena generada será de mayor calidad en cuanto a la iluminación.
- Progresivo: Consiste es una aproximación al cálculo de solución de radiosidad. Se realizan diversas iteraciones, donde para cada una existe una serie de elementos que emiten energía hacia el resto de la escena. Esto es equivalente a calcular una columna de la ecuación matricial de radiosidad en cada pasada del algoritmo. La ventaja de este método es que permite obtener resultados intermedios en cada iteración. Gracias a una componente ambiental inicial, que se decrementa en cada pasada, hasta hacerse nula al terminar, se puede obtener pre-visualizaciones cada vez más fiables de la imagen final. La condición de parada se determina de acuerdo a la calidad, o al llegar a un límite prefijado de iteraciones.
- Re-mallado adaptativo: Dada una subdivisión inicial de la escena, se recorren los parches existentes comparando su valor de radiosidad con el valor de sus vecinos. Si existe una diferencia tal entre dichos valores que supere un umbral prefijado, se procede a subdividir los parches en elementos de menor tamaño, y así aumentar la resolución de la malla y modelar los cambios de iluminación de forma más precisa.
- Jeráquica: Este tipo crea una jerarquía de elementos en cada parche, y permite conectarlos con otros parches en un nivel de subdivisión en donde la interacción sea óptima. Primero, se construye la jerarquía sobre cada parche inicial, y se establecen todos los enlaces con los demás parches. Posteriormente, en un proceso iterativo, se distribuye la radiosidad a lo largo de los enlaces ya calculados.

3.6.1. Refinamiento progresivo en la GPU

En el 2004, Coombe et al. [24] presentan una implementación realizada completamente en la GPU, alcanzando una tasa de refrescamiento adecuada para ser considerada en tiempo real. Este enfoque adapta el algoritmo clásico de *radiosity* progresivo a la GPU, utilizando como estructuras de almacenamiento básico texturas en punto flotante y realiza una subdivisión uniforme de la escena en elementos, los cuales en este caso estarán representados por los texel de las texturas.

La razón por la cual implementan el enfoque progresivo es debido a que no se debe realizar ningún almacenamiento significativo en memoria a diferencia del algoritmo de clásico en forma matricial (matriz completa). Además la forma como el algoritmo de radiosidad progresivo calcula los valores de radiosidad permite explotar la arquitectura paralela de la GPU, ya que el cálculo de un valor de radiosidad es independiente del otro y no presenta ciclos. Además, este enfoque permite desplegar soluciones intermedias de manera que se adapta a aplicaciones interactivas y permite que el usuario controle, si desea, la calidad de la imagen o rendimiento según sus necesidades.

En cuanto al almacenamiento, se guardaran los valores de radiosidad por polígono en una textura, y en otra textura se guardan los valores de energía residual. En la visibilidad, utilizan un *vertex shader* que deforma los vértices mediante una proyección estereográfica, con esta proyección se efectúa una única pasada sobre la geometría a diferencia de cinco pasadas que son requeridas por el método del semicubo. Este tipo de proyección realiza una representación más amplia de la escena al abarcar una cantidad de geometría adecuada para un punto de vista.

Posteriormente, Wallner [25] propone una extensión del trabajo de Coombe et al. [24] al agregar el soporte de normal mapping y resolver varios detalles presentes en el trabajo original. Wallner solventa el problema de los T-vértices presente en el trabajo de Coombe et al. [24], creando restricciones en la construcción del quadtree generado.

3.7. Caché de irradiancia

Debido a que el *ray tracing* basado en Monte Carlo es realmente lento, Ward et al. [26] presentan una mejora para el cálculo de la iluminación difusa indirecta de las escenas, conocida como caché de irradiancia *(irradiance caching)*. La irradiancia es el total de energía incidente en una superficie, por unidad de área de la superficie. La idea principal consiste en mantener almacenada la irradiación de diferentes puntos de la escena calculados anteriormente, basándose en que la radiación en superficies difusas varía de manera muy suave.

Por lo tanto, al tratar de calcular la irradiancia en un punto cualquier de la escena, el algoritmo primero verifica si el valor ya se encuentra calculado o, si existen varios puntos cercanos entre los cuales pueda ser interpolado. Entonces, si el valor no puede ser interpolado, se calcular de manera normal, para posteriormente ser almacenado.

3.8. Mapeado de fotones

El mapeado de fotones *(photon mapping)* fue propuesto por Jensen [27, 28, 29] en 1994, como un algoritmo de dos pasadas, que se comporta de forma similar al *bidirectional ray tracing*, ya que traza rayos desde la luz y desde el punto de vista. Sin embargo, esta técnica hace uso del almacenamiento de la radiación calculada a través de los rayos emitidos desde la luz. En la primera pasada se disparan fotones desde la luz, los cuales poseen *flux*[2] y son almacenados en una estructura de datos llamada mapa de fotones [30]. En la segunda pasada se despliega la imagen utilizando la información almacenada.

Dependiendo del tipo de trazado de fotones que se realice, es posible crear mapas de fotones especializados. Por ejemplo, si se trazan y almacenan los rayos de fotones que causan cáusticas, es posible recrearlas guardándolas en un mapa de cáusticas. Sin embargo, para realizar el despliegue de estos efectos es necesario tomar ciertas consideraciones.

En la figura 12 se observa el uso del mapeado de fotones para el cálculo de la iluminación. El muestreo en el mapa de fotones se realiza con el segundo rebote del rayo. Sin embargo, el muestreo en el mapa de cáusticas se realiza con el primer rebote y no con el segundo. Diferentes mapas conllevan diferentes formas de muestrear.

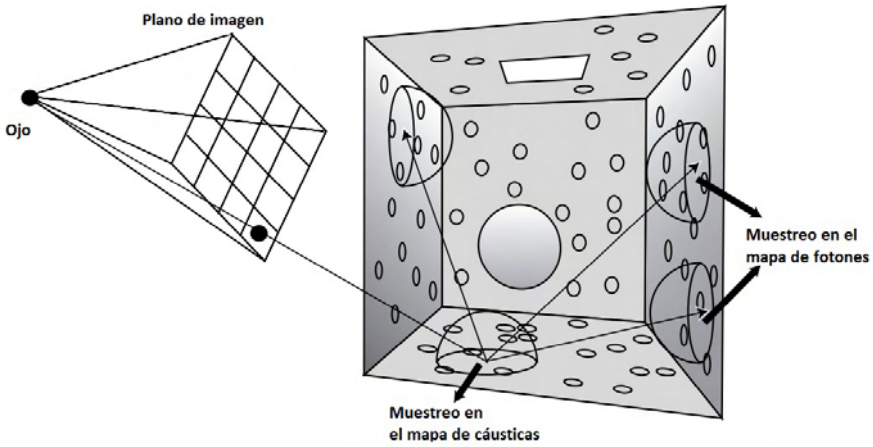


Figura 12: Cálculo de la iluminación usando el *photon mapping*.

Para el uso del mapa de fotones en la segunda pasada se tienen que tomar las siguientes consideraciones:

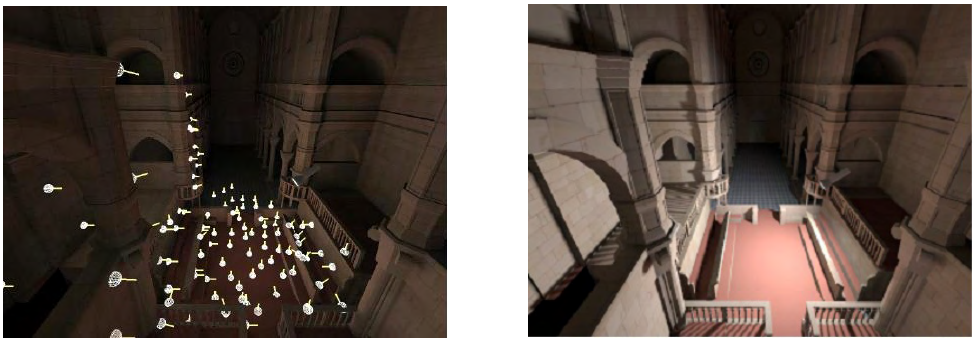
- Por cada píxel se calcula el punto de intersección con la superficie más cercana.
- La luz directa en este punto se puede calcular utilizando muestreo de Monte Carlo.
- Los rayos reflejados y refractados son trazados.
- Para calcular la iluminación indirecta se procede a realizar muestreos sobre el hemisferio y se calcula la radiación usando el mapa de fotones.

3.9. Radiosity instantáneo

En 1997, Keller [31] propone *radiosity* instantáneo *(instant radiosity)*, como un algoritmo de dos pasadas. La idea consiste en reemplazar la luz difusa indirecta por luz difusa directa proveniente desde un punto de luz virtual *(virtual light point, VPL)*. Para ello, se trazan rayos

[2]Un *flux* se define como el flujo por unidad de área, donde del flujo es el movimiento de alguna cantidad por unidad de tiempo.

desde la fuente de luz y se colocan luces virtuales en el punto de intersección. Eventualmente, se puede permitir la dispersión especular para colocar más luces virtuales a partir de la primera luz virtual. La figura 13 muestra el despliegue de la escena modelada Marko Dabrovic de Catedral de Sibenik [32] donde se observa un ejemplo de luces virtuales y el resultado final del despliegue.



(a) (b)

Figura 13: Despliegue de una escena donde (a) es inyectada con un conjunto de luces virtuales e, (b) iluminada con *instant radiosity*.

Este algoritmo es comparable al *bidirectional ray tracing*, donde las luces virtuales son creadas a partir de caminos provenientes de la luz. Sin embargo, como estos caminos son siempre los mismos para todos los píxeles, las imágenes producidas por el *instant radiosity* lucen más suaves y con menos ruido que con el algoritmo de *bidirectional ray tracing*.

Con el uso de un software eficiente para el cálculo del *ray tracing*, se pueden obtener resultados de alta calidad en pocos segundos. Sin embargo, debido a lo complejo del algoritmo de *ray tracing*, Keller introdujo un método para calcular la iluminación usando la GPU. Debido a que generalmente se crean alrededor de unos cientos de luces, el hardware actual no es capaz de desplegarlas todas en una simple pasada. Por lo tanto, se hace uso del *buffer* de acumulación para combinar los resultados de diferentes despliegues. Usando esta técnica es posible conseguir tiempos interactivos en escenas poco complejas.

Por otro lado, Laine et al. [33] plantean una optimización al algoritmo de *instant radiosity*, que se refiere a re-calcular los puntos de luz virtuales que pierden validez al moverse la fuente de luz principal. Un punto de luz virtual se considera válido cuando no hay obstrucción entre este y la fuente de luz de donde es emitido, como se muestra en la figura 14 donde solamente se calculan los puntos de luz virtuales que han perdido validez con respecto a la iteración anterior (imágenes de la izquierda y centro). El movimiento de la cámara no afecta en nada la validez de los puntos virtuales (imagen de la derecha).

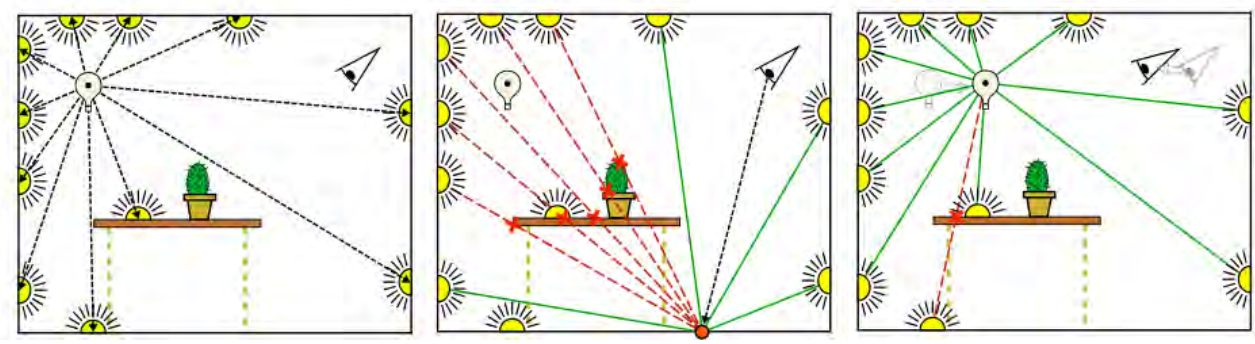


Figura 14: Cálculo de los puntos de luz virtuales.

Se efectuará una prueba de oclusión entre cada punto de luz virtual y su correspondiente fuente de emisión; los resultados de estas pruebas nos dirán cuales puntos de luz virtuales siguen siendo válidos con respecto a la iteración anterior. Solamente se volverán a calcular aquellos puntos que hayan perdido validez. Las pruebas de oclusión pueden realizarse, ya sea con *oclussion query* o con pruebas de *ray casting*.

3.10. Uso de texturas para resolver el sistema de ecuaciones en la GPU

Carr et al. [34] presentaron una técnica para aprovechar la naturaleza paralela de los procesadores del GPU, y así resolver el sistema de ecuaciones que se genera en el problema de *radiosity*. Esta técnica necesita de una tarjeta de video que soporte texturas en punto flotante. Se asume que la geometría no varía a lo largo de la ejecución del algoritmo; sin embargo, si se permite el uso de luces que cambian de forma dinámica.

El sistema de ecuaciones de *radiosity*, que puede ser derivado de la ecuación 3, es un sistema de ecuaciones lineales que puede resolverse al ser planteado como: $M\vec{B} = \vec{E}$ donde M es la matriz de factores de forma multiplicado por la reflectancia de cada elemento, $\vec{B}$ es el vector incógnita cuyos componentes son los valores de radiosidad de todos los elementos de la escena y $\vec{E}$ es el vector independiente cuyos componentes son los valores de emisión de todos los elementos de la escena.

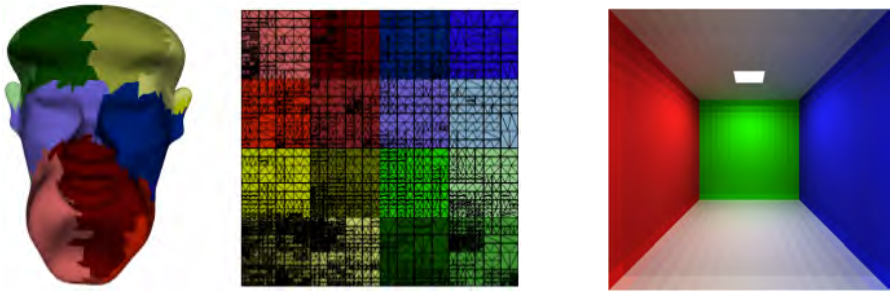
En [34] se plantea el uso de texturas 2D punto flotante RGB para almacenar en cada uno de los canales las matrices de *radiosity* calculadas para cada banda de color. Adicionalmente se utiliza una textura 1D, igualmente en formato RGB, para almacenar de manera similar los valores de emisión del vector $\vec{E}$. Alternativamente se puede utilizar una textura del tipo *luminance* para almacenar la matriz de *radiosity* junto al vector $\vec{E}$ como una columna adicional, en caso de que no se necesite calcular el color de la escena sino solamente su iluminación.

Para resolver el sistema, dado que las texturas están en el *fragment shader*, se utiliza el método iterativo de Jacobi en su forma de producto matriz-vector:

$$\vec{B}^{k+1} = \vec{B}^{(k)} + \vec{E} - diag(M)^{-1} M\vec{B}^{(k)} \tag{6}$$

En la ecuación 6, $diag(M)^{-1}$ es la inversa de la diagonal de la matriz M, que en la mayoría de los casos es la matriz identidad. Se utiliza esta forma de resolver el sistema de ecuaciones porque la fórmula general de Jacobi tiene una condición que es difícil de evaluar en un programa de fragmentos.

Esta técnica tiene un inconveniente, no puede aplicar interpolación bilineal entre las diferentes radiosidades debido a que están almacenadas en una textura 1D. Esto elimina la posibilidad de obtener un sombreado suave entre los diferentes vértices de la escena, ver figura 15(b). Para solventar este problema, se plantea el uso de un tipo de textura especial llamada textura atlas[3] (figura 15(a)). En una textura atlas es posible proyectar todos los vértices de una escena con sus valores de radiosidad de forma que es posible hacer interpolación bilineal entre todos los vértices adyacentes.



(a) La textura es un atlas asociado a la malla (b) Resultado del despliegue

Figura 15: Uso de texturas para el cálculo de la radiosidad presentada por Carr et al. [34].

Esta técnica acelera el cálculo de *radiosity* en comparación con el cálculo completo en la CPU para escenas grandes, debido a que la CPU está limitado por el ancho de banda de la memoria de video. Es decir, el pasar escenas grandes calculadas en la CPU a la tarjeta de video, es más lento que calcular la escena utilizando la GPU. Sin embargo, la GPU está limitada por el tamaño máximo de texturas que pueda soportar, lo que limita efectivamente el tamaño de las escenas que pueden calcularse en esta.

3.11. Mapas de índices

Nielsen et al. [35] presentaron un método para optimizar los cálculos de *radiosity* eliminando la necesidad de subdividir geométricamente la escena en parches y elementos. El método utiliza una textura especial llamada *mapa de índices* (figura 16) que codifica en cada uno de sus elementos un color distinto que puede utilizarse como índice a una tabla de búsqueda en la cual se calculan y almacenan los factores de forma. Se utiliza un solo mapa de índices para toda la escena con la intención de ahorrar el espacio en memoria que consumirían varios mapas de índices diferentes.

A cada superficie de la escena se le asigna un desplazamiento p que es añadido al mapa de índices al momento de texturizar dicha superficie. El desplazamiento debe ser único en la tabla de factores de forma, como se observa en la figura 17.

Al momento de calcular los factores de forma, las superficies texturizadas con el mapa de índices desplazado apropiadamente son proyectadas sobre los semicubos. Luego, al discretizar el semicubo, se determina qué elementos caen sobre cada una de sus celdas para obtener su índice correspondiente en la tabla de factores de forma, en la que hay almacenada la información relevante sobre ese elemento, como su valor de emisión y la superficie a la que pertenece.

[3] Una textura atlas es una imagen que contiene muchas imágenes más pequeñas, cada una de las cuales es una textura de una parte de un modelo, o bien texturas de modelos diferentes.

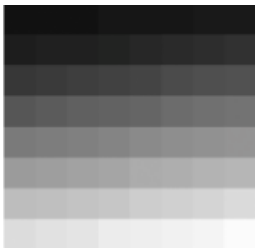


Figura 16: Ejemplo de un mapa de índices con 64 texeles.

Figura 17: De izq. a der.: mapa de índices aplicado directamente; desplazamientos a aplicar por superficie y; escena texturizada final.

El objetivo principal de este método es eliminar la subdivisión geométrica de la escena, de forma que se aceleren las proyecciones que se realizan sobre los semicubos. Al no haber subdivisión, la cantidad de polígonos que deben desplegarse se reduce considerablemente, lo que a su vez reduce la complejidad añadida por el cuello de botella que suele representar el bus del sistema al tener que pasar menos superficies de la memoria principal a la memoria de video.

Sin embargo, la tabla de factores de forma que se utiliza para almacenar los valores calculados requiere de un costo de almacenamiento sumamente grande, mucho mayor que lo que es necesitado por la matriz generada por la ecuación 3. Adicionalmente, una vez que la tabla de factores de forma está llena hay que crear las matrices necesarias para resolver el sistema de ecuaciones.

3.12. Radiosity Normal Mapping

En el año 2007, Green [36] con el propósito el optimizar el motor Source, los desarrolladores de Valve [37], diseñaron una técnica que denominaron *radiosity normal mapping*. Esta técnica es una mejora a un sistema de *normal mapping* combinado con *light mapping* que utilizaban en las primeras versiones de su sistema. La figura 18 muestra unas escenas donde es aplicado dicha técnica.

(a) Escena de una cueva con poca iluminación (b) Escena donde se muestra una entrada de luz exterior

Figura 18: La técnica de *radiosity normal mapping* se aplicó por primera vez en el juego *Half-Life 2: Episode 2.*

En *radiosity normal mapping* se agrega un coeficiente de auto-sombreado a la técnica de *normal mapping* tradicional, el cual combinado con los mapas de luz (*lightmaps*), se produce una solución heurística de iluminación global que varía en tiempo real según el movimiento del jugador y el movimiento de las fuentes de luz.

El primer paso del método es cambiar la base de los mapas de relieve utilizados por la siguiente base:

$$\vec{B}_{0.,2} = \{\frac{-1}{\sqrt{6}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{3}}\}, \{\frac{-1}{\sqrt{6}}, \frac{-1}{\sqrt{2}}\}, \frac{1}{\sqrt{3}}, \{\frac{\sqrt{2}}{\sqrt{3}}, 0, \frac{1}{\sqrt{3}}\}$$

aplicando las siguientes ecuaciones:

$$\vec{N} = 2\vec{T} - 1$$

$$\vec{D}_i = \frac{clamp(\vec{N}.\vec{B}_i, 0, 1)^2}{\sum_{i=0}^{2} \vec{D}_i}$$

donde $\vec{T}$ es un texel en espacio tangente obtenido del mapa de relieve.

En principio el nuevo mapa de relieves va a tener como texeles a los vectores $\vec{D}_i$. Aún falta añadir un término adicional que es llamado coeficiente de oclusión ambiental direccional. El cálculo de este coeficiente, se realiza extendiendo la técnica para calcular un coeficiente de oclusión ambiental unidireccional, que sirve para conocer si un texel se encuentra a la luz o a la sombra con respecto a una fuente de luz no puntual directa.

El coeficiente de oclusión ambiental unidireccional se calcula trazando un gran conjunto de rayos desde cada texel del mapa de relieve hacia una semiesfera imaginaria ubicada sobre cada uno de estos, probando intersecciones con la misma superficie a la que pertenecen los texeles. Basados en las intersecciones que se producen después de probar estos rayos, se determina que proporción de la semiesfera es visible desde cada texel. Sin embargo, este coeficiente es igual para cada uno de los vectores de la base canónica $\vec{B}_{0.,2}$. Esto tiene como resultado que al efecto visual producido por el normal mapping se le agregue el efecto producido por el método de *ambient oclussion* [38].

El principal aporte de Green [36] radica en que la contribución de cada prueba de rayo al coeficiente de oclusión ambiental mencionado, es ponderada por el producto interno entre dicha prueba de rayo con el vector $\vec{D}_i$ del texel a procesar. Esto produce un efecto de auto-sombreado difuso adicional a los efectos de oclusión ambiental y de relieve que se obtenían anteriormente (ver figura 18).

3.13. Cortes-de-luz

Walter et. al [39, 40] proponen la técnica de cortes-de-luz *(Lightcuts)*. Esta es una técnica para el despliegue de efectos complejos como los son: una gran cantidad de luces, desenfoque por movimiento, desenfoque por distancia y presencia de medios participantes. En [39] se propone representar las luces a través de un árbol, de tal manera que estas puedan ser unidas en *clusters*. De esta forma, quedarán las fuentes de luz en las hojas y en los nodos internos quedarán representaciones promediadas de las mismas. Utilizando este árbol, no se necesita evaluar todas las luces, sino un corte del árbol tal que no se exceda de un error establecido previamente.

Esa idea es extendida en [40] para poder modelar efectos como el desenfoque por movimiento, desenfoque por distancia y presencia de medios participantes, recreando varios árboles y haciendo los cortes utilizando producto de grafos.

4. Tendencias actuales en Iluminación

A pesar de los grandes avances en computación en los últimos años, el cálculo de la iluminación global en tiempo real continua siendo un gran reto. Sin embargo, debido a su gran importancia, se han realizado diversas investigaciones que tratan de aproximarla de la mejor manera, consumiendo el menor tiempo posible. Esto se debe, a que actualmente se reconoce que no es necesario poseer una solución exacta en la mayoría de los casos para crear un efecto convincente. Además, la mayoría de las investigaciones actuales tienden al uso de la GPU.

Según Kaplayan y Dachsbacher [41] las técnicas actuales de iluminación global en tiempo real pueden clasificarse de la siguiente manera:

4.0.1. Métodos clásicos

Dadas las mejoras actuales de hardware, métodos clásicos como *raytracing* [42] y *radiosity* [43], han sido mejorados para poder alcanzar así desempeños interactivos. Sin embargo, estas mejoras representan grandes restricciones o el uso de grandes cantidades de poder de computación, por lo que no son la mejor opción para aplicaciones en tiempo real.

4.0.2. Iluminación precalculada

Esta es una de las técnicas más usadas especialmente en videojuegos, como *Halo 3* [44] o *Dragon Age II* [45]. Esta consiste en realizar un cálculo previo de la iluminación, incluyendo información de visibilidad, lo que conlleva la restricción del uso de escenas estáticas o semi-estáticas [46] [47].

4.0.3. Métodos en espacio de imagen

Representan las técnicas que hacen uso de la GPU para el cálculo de la iluminación en el espacio de imagen [48] [49]. Estas técnicas solo trabajan en el espacio de imagen e ignoran objetos o luces que se encuentran fuera de este. Así crean una gran cantidad de artefactos y es necesario un post-procesamiento para disminuirlos.

Dachsbacher y Stamminger [50] proponen una técnica eficiente basada en el uso de *reflective shadow maps (RSM)*. Los *RSM* utilizan la idea de los mapas de sombras para capturar puntos iluminados de forma directa, y así convertir cada píxel en pequeños puntos de luz. Papaioannou [51] combina el uso de los RSM con técnicas basadas en grids de radiancia para obtener una solución basada en la GPU que soporta múltiples rebotes de la luz. Otra de las técnicas con más auge ha sido el cálculo de oclusión ambiental en espacio de imagen [52] [53].

4.0.4. Métodos basados en *instant radiosity*

Haciendo uso del *instant radiosity* [31], es posible calcular la iluminación de una escena a través de *VPL*. Con el uso del *RSM* [50], se pueden calcular *VPL* para el primer rebote de luz indirecta, por lo que esta técnica ha cobrado gran interés en el cálculo de la iluminación global durante los últimos años [54] [55] [56] [57]. Una de las principales desventajas de estas técnicas es que es necesario un gran número de *VPL* para solventar los artefactos en la imagen final.

4.0.5. Métodos basado en modelos discretos

Consiste en la discretización de los términos de radiación de una escena, de tal manera que estos puedan ser guardados en una malla tridimensional que represente la misma. La luz interactúa con elementos vecinos del volumen, reduciendo el cálculo por interacción a un ámbito local [58] [41] [59] [60]. Son principalmente usados para poder modelar medios participantes.

4.0.6. Métodos de aproximaciones geométricas

Esta vertiente es expuesta por Kaplayan [58], y se basa en la idea de calcular la iluminación global en objetos atómicos pre-calculados, usualmente empleando discos, esferas [61] o *surfels* [62] [63]. Debido al incremento del poder de procesamiento de la GPU, estas técnicas han cobrado importancia. Sin embargo, la representación de las escenas en estos elementos atómicos puede llegar a ser costosa y difícil de calcular. Una de las vertientes actuales es el uso de la discretización de escenas empleando vóxeles para el cálculo de la iluminación global, ya sea con el uso de vóxeles para la representación de la geometría de la escena [64] o de *grids* para la dispersión de la luz en el ambiente [58] [41] [65].

Uno de los mayores aspectos a tomar en cuenta cuando se quiere calcular la iluminación global, es la complejidad y el dinamismo de la escena. El uso de una descripción poligonal de una escena generalmente consume mucho tiempo. Por lo tanto, su representación en elementos más simples puede agilizar el cálculo del transporte de la luz. Una de las vertientes actuales es el uso de vóxeles para discretizar una escena [64] [66] [65] [67]. Uno de los trabajos más recientes en esta área es el realizado por Thiedemann et al. [64], donde se introduce una nueva técnica basada en el uso de texturas atlas para obtener la voxelización de la escena, además de proponerse pruebas de intersección entre rayo/vóxel. Estas pruebas permiten el cálculo de la iluminación de un solo rebote de la luz en un área cercana en tiempo real. Además, con ciertas extensiones, se puede lograr el cálculo de la iluminación con múltiples rebotes de la luz con el uso de *path tracing* o *instant radiosity*.

Recientemente, Ritschel et al. [68] presentaron un amplio y detallado estado del arte en los algoritmos de iluminación global en tiempo real. En dicho trabajo, los autores presentan una clasificación de acuerdo a factores influyentes en el desarrollo del algoritmo como: velocidad, calidad, dinamismo, escalabilidad, implementación y soporte en la GPU.

5. Tecnologías para el soporte de la iluminación global

5.1. Motor OptiX

Para las tarjetas gráficas Nvidia [69], existe el motor de ray tracing OptiX [70], que consiste es un un motor programable de *ray tracing* para desarrolladores de software. La idea principal es obtener un trazador de rayos extremadamente veloz como resultado del uso de las GPU's de Nvidia bajo un lenguaje tradicional como C. Mientras que el potencial del *ray tracing* casi instantáneo está siendo reconocido para servir a diseñadores automovilísticos, diseños de visualizaciones, y efectos visuales; el motor OptiX también está siendo utilizado para disciplinas donde el renderizado no es nativo como el diseño de óptica y acústica, estudios de radiación, calculo de volúmenes, y análisis de colisiones, etc.

(a) Bugatti (b) Julia

Figura 19: Ejemplo de imágenes creadas con el motor OptiX.

El motor OptiX de ray tracing de NVIDIA es un sistema programable diseñado para las GPU's de NVIDIA y otras arquitecturas muy similares. Se basa en la observación clave de que la mayoría de los algoritmos de *ray tracing*, que pueden ser implementados utilizando un pequeño conjunto de operaciones programables. En consecuencia, el núcleo OptiX es un compilador *just-in-time* de dominio específico, que genera núcleos personalizados de *ray tracing* mediante la combinación de los programas suministrados por el usuario para la generación de rayos, el sombreado de materiales, intersección de objetos y recorrido de la escena. Esto permite la implementación de un conjunto muy diverso de aplicaciones y algoritmos basados en *ray tracing*. OptiX logra un alto rendimiento a través de un modelo de objeto compacto y de la aplicación de varias optimizaciones específicas del compilador de *ray tracing*. Para facilitar su uso, este expone un modelo de programación de rayo sencillo con soporte completo para la recursividad y un mecanismo de expedición dinámico.

5.2. Proyecto OpenRT

La meta del proyecto OpenRT (*Real-Time Ray-Tracing*) es desarrollar algoritmos de *ray tracing* tal que se pueda ofrecer una alternativa muy similar al proceso de rasterización actual, basado en gráficos interactivos en 3D. Es por ello que el proyecto está compuesto por diversas partes: un núcleo de *ray tracing* altamente optimizado, el OpenRT-API el cual es similar a OpenGL y una amplia gama de aplicaciones, desde modelos masivos animados dinámicamente hasta iluminación global, usando prototipos de visualización de alta calidad para desarrollo de juegos de computadora.

5.3. RPU

La RPU (*Ray Tracing Processing Unit*) [71] es un prototipo de una unidad de procesamiento dedicada al trazado de rayos. Su objetivo principal es acelerar las operaciones del trazador de rayos empleando hardware especializado que sea programable. Este procesador se basó en un trabajo previo desarrollado por los mismos autores llamado SaarCOR (*Saarbrücken's Coherence Optimized Ray Tracer*) [72].

6. Conclusiones

Recordando que la iluminación global es una técnica que trata de emular el comportamiento de la luz para designar la simulación de todos los posibles fenómenos asociados a los trazos de luz que componen la escena, se puede ver que hoy en día estas técnicas siguen avanzando, aprovechando al máximo los recursos computacionales de última generación con el objetivo de llevar un mundo virtual a lo más cercano de la realidad.

Se recomienda que al momento de hacer alguna aplicación 3D, se deba evaluar las ventajas y desventajas de cada método de iluminación global, y seleccionar aquel que se ajuste más a los requerimientos que desea cubrir con la aplicación. Las limitaciones del hardware, capacidad de almacenamiento, pérdida de información por precisión en los formatos de las texturas, entre otros, son un conjunto de factores que en muchas ocasiones limitan la efectividad total de alguno de los algoritmos aquí planteados.

Gracias a todos estos métodos de iluminación global, como *radiosity*, *ray tracing*, *photon mapping*, etc., basados en la ecuación de Kajiya y el crecimiento del hardware gráfico en cuanto a sus capacidades, se estima que será posible realizar estas operaciones en tiempo real. Del mismo modo, realizar efectos que simulen un ambiente con interacciones indirectas en tiempo real es una meta que actualmente está siendo desarrollada en diversos centros de investigación a nivel mundial.

Referencias

[1] P. Dutré, K. Bala, P. Bekaert, and P. Shirley, *Advance global illumination*. A K Peters, Ltd., 2006.

[2] H. Gouraud, "Continuous shading of curved surfaces," *IEEE Transactions on Computers*, vol. 20, pp. 623–629, Jun 1971.

[3] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, pp. 311–317, Jun 1975.

[4] J. Kajiya, "The rendering equation," *ACM SIGGRAPH Computer Graphics*, vol. 20, pp. 143–150, 1986.

[5] T. Whitted, "An improved illumination model for shaded display," *ACM SIGGRAPH Computer Graphics*, vol. 13, pp. 343–349, 1979.

[6] R. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," *ACM SIGGRAPH Computer Graphics*, vol. 18, pp. 137–145, Jul 1984.

[7] J. Arvo, "Backward ray tracing," in *ACM SIGGRAPH '86 Course Notes - Developments in Ray Tracing*, pp. 259–263, 1986.

[8] E. Lafortune and Y. Willems, "A theoretical framework for physically based rendering," *Computer Graphics Forum*, vol. 13, pp. 97–107, May 1994.

[9] E. Veach and L. Guibas, "Bidirectional estimators for light transport," in *Proceeding of the 5th Eurographics Workshop on Rendering*, pp. 147–162, Darmstadt, 1994.

[10] E. Veach and L. Guibas, "Metropolis light transport," in *SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 65–76, ACM, 1997.

[11] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, H. Teller, and E. Teller, "Equations of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[12] T. J. Purcell, *Ray tracing on a stream processor*. PhD thesis, Stanford, CA, USA, 2004.

[13] The Kronos Group, "OpenGL shading language - GLSL," Jun 2012. http://www.opengl.org/documentation/glsl/.

[14] Nvidia Corporation, "C for graphics - Cg," Jun 2012. http://developer.nvidia.com/cg-toolkit.

[15] Microsoft Corporation, "High level shader language - HLSL," May 2006. http://msdn.microsoft.com/en-us/library/bb509561.aspx.

[16] The Kronos Group, "Open computing language - OpenCL," Sep 2008. http://khronos.org/opencl.

[17] Nvidia Corporation, "Nvidia CUDA," Jun 2012. http://developer.nvidia.com/category/zone/cuda-zone.

[18] Microsoft Corporation, "DirectCompute," Jun 2012. http://www.microsoft.com/en-us/download/details.aspx?id=16995.

[19] M. Christen, "Ray tracing on GPU," Master's thesis, University of Applied Sciences Basel (FHBB), Switzerland, 2005.

[20] C. Goral, K. Torrance, D. Greenberg, and B. Battaile, "Modeling the interaction of light between diffuse surfaces," *ACM SIGGRAPH Computer Graphics*, vol. 18, pp. 213–222, Jul 1984.

[21] E. Sparrow, "A new and simpler formulation for radiative angle factors," *Transactions of the ASME, Journal of Heat Transfer*, vol. 85, pp. 81–88, 1963.

[22] E. Sparrow and R. Cess, "Radiation heat transfer," in *Thermal and Fluids Engineering*, Hemisphere Publishing Corporation, 1978.

[23] M. Cohen and D. Greenberg, "The hemi-cube: a radiosity solution for complex environments," *ACM SIGGRAPH Computer Graphics*, vol. 19, pp. 31–40, Jul 1985.

[24] G. Coombe, M. J. Harris, and A. Lastra, "Radiosity on graphics hardware," in *Proceedings of Graphics Interface 2004*, (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada), pp. 161–168, Canadian Human-Computer Communications Society, 2004.

[25] G. Wallner, "GPU radiosity for triangular meshes with support of normal mapping and arbitrary light distributions," in *Proceedings of the Sixteenth International Conference in Central Europe on Computer Graphics Visualization and Computer Vision 2008 WSCG 2008*, University of West Bohemia, 2008.

[26] G. Ward, F. Rubinstein, and R. D. Clear, "A ray tracing solution for diffuse interreflection," *ACM SIGGRAPH Computer Graphics*, vol. 22, pp. 85–92, Jul 1988.

[27] H. W. Jensen and N. J. Christensen, "Photon maps in bidirectional monte carlo ray tracing of complex objects," *Computers and Graphics*, vol. 19, pp. 215–224, 1995.

[28] H. W. Jensen, "Global illumination using photon maps," in *Proceedings of the eurographics workshop on Rendering techniques '96*, pp. 21–30, Springer-Verlag, 1996.

[29] H. W. Jensen, "Rendering caustics on non-lambertian surfaces," in *GI '96 Proceedings of the conference on Graphics interface '96*, pp. 116–121, Canadian Information Processing Society, 1996.

[30] H. W. Jensen, *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., 2009.

[31] A. Keller, "Instant radiosity," in *SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 49–56, ACM Press/Addison-Wesley, 1997.

[32] M. Dabrovic, "Sibenik cathedral," 2001. http://hdri.cgtechniques.com/ sibenik2/.

[33] S. Laine, H. Saransaari, J. Kontkanen, J. Lehtinen, and T. Aila, "Incremental instant radiosity for real-time indirect illumination," in *Proceeding of Eurographics Symposium on Rendering 2007*, pp. 277–286, Eurographics Association, 2007.

[34] N. A. Carr, J. D. Hall, and J. C. Hart, "GPU algorithms for radiosity and subsurface scattering," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '03, (Aire-la-Ville, Switzerland, Switzerland), pp. 51–59, Eurographics Association, 2003.

[35] K. H. Nielsen and N. J. Christensen, "Fast texture-based form factor calculations for radiosity using graphics hardware," *Journal of Graphics Tools*, vol. 6, pp. 1–12, Sept. 2002.

[36] C. Green, "Efficient self-shadowed radiosity normal mapping," in *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, (New York, NY, USA), pp. 1–8, ACM, 2007.

[37] Valve Corporation, "Valve," Jun 2012. http://www.valvesoftware.com/.

[38] M. S. Langer and H. H. Bülthoff, "Depth discrimination from shading under diffuse lighting," *Perception*, vol. 29, pp. 649–660, July 2000.

[39] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg, "Lightcuts: A scalable approach to illumination," *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2005*, vol. 24, pp. 1098–1107, Jul 2005.

[40] B. Walter, A. Arbree, K. Bala, and D. P. Greenberg, "Multidimensional lightcuts," *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2006*, vol. 25, pp. 1081–1088, Jul 2006.

[41] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," in *I3D '10 Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pp. 99–107, ACM, 2010.

[42] R. Wang, R. Wang, K. Zhou, M. Pan, and H. Bao, "An efficient GPU-based approach for interactive global illumination," *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2009*, vol. 28, pp. 1–8, Aug 2009.

[43] Z. Dong, J. Kautz, C. Theobalt, and H.-P. Seidel, "Interactive global illumination using vmplicit visibility," in *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pp. 77–86, IEEE Computer Society, 2007.

[44] Bungie, "Halo 3," 2007. http://halo.xbox.com/en-us/intel/titles/halo3.

[45] BioWare, "Dragon age II," 2011. http://dragonage.bioware.com.

[46] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2002*, vol. 21, pp. 527–536, 2002.

[47] H. Chen and X. Liu, "Lighting and material in Halo 3," in *SIGGRAPH '08 ACM SIGGRAPH 2008 classes*, pp. 1–22, ACM, 2008.

[48] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher, "Micro-rendering for scalable, parallel final gathering," *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2009*, vol. 28, Dec 2009.

[49] M. McGuire and D. Luebke, "Hardware-accelerated global illumination by image space photon mapping," in *Proceedings of the Conference on High Performance Graphics 2009*, pp. 77–89, ACM, 2009.

[50] C. Dachsbacher and M. Stamminger, "Reflective shadow maps," in *I3D '05 Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pp. 203–231, ACM, 2005.

[51] G. Papaioannou, "Real-time diffuse global illumination using radiance hints," in *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, (New York, NY, USA), pp. 15–24, ACM, 2011.

[52] M. Mittring, "Finding next gen: CryENGINE 2," in *SIGGRAPH '07 ACM SIGGRAPH 2007 courses*, pp. 97–121, ACM, 2007.

[53] T. Ritschel, T. Grosch, and H.-P. Seidel, "Approximating dynamic global illumination in image space," in *I3D '09 Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pp. 75–82, ACM, 2009.

[54] C. Dachsbacher and M. Stamminger, "Splatting indirect illumination," in *I3D '06 Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pp. 93–100, ACM, 2006.

[55] G. Nichols and C. Wyman, "Multiresolution splatting for indirect illumination," in *I3D '09 Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pp. 83–90, ACM, 2009.

[56] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz, "Imperfect shadow maps for efficient computation of indirect illumination," *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2008*, vol. 27, Dec 2008.

[57] G. Nichols, J. Shopf, and C. Wyman, "Hierarchical image-space radiosity for interactive global illumination," *Computer Graphics Forum*, vol. 28, pp. 1141–1149, 2009.

[58] A. Kaplanyan, "Light propagation volumes in CryENGINE 3," in *ACM SIGGRAPH 2009 Courses - Advances in Real-Time Rendering in 3D Graphics and Games Course*, 2009.

[59] R. Fattal, "Participating media illumination using light propagation maps," *ACM Transactions on Graphics (TOG)*, vol. 28, January 2009.

[60] R. Geist, K. Rasche, J. Westall, and R. Schalkoff, "Lattice-boltzmann lighting," in *Rendering Techniques 2004 Proceedings of the Eurographics Symposium on Rendering*, pp. 355–362, 2004.

[61] P.-P. Sloan, N. Govindaraju, D. Nowrouzezahrai, and J. Snyder, "Image-based aproxy accumulation for real-time soft global illumination," in *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pp. 97–105, IEEE Computer Society, 2007.

[62] C. Dachsbacher, M. Stamminger, G. Drettakis, and F. Durand, "Implicit visibility and antiradiance for interactive global illumination," *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2007*, vol. 26, Jul 2007.

[63] A. Evans, "Fast approximations for global illumination on dynamic scenes," in *SIGGRAPH '06 ACM SIGGRAPH 2006 Courses*, pp. 153–171, ACM, 2006.

[64] S. Thiedemann, N. Henrich, T. Grosch, and S. Müller, "Voxel-based global illumination," in *I3D '11 Symposium on Interactive 3D Graphics and Games*, pp. 103–110, ACM, 2011.

[65] G. Greger, P. Shirley, P. Hubbard, and D. Greenberg, "The irradiance volume," *IEEE Computer Graphics and Applications*, vol. 18, pp. 32–43, March 1998.

[66] P. Mavridis and G. Papaioannou in *Proceedings of the International Conference on Computer Graphics Theory and Application*.

[67] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, "Interactive indirect illumination using voxel cone tracing," *Pacific Graphics 2011*, vol. 30, 2011.

[68] T. Ritschel, C. Dachsbacher, T. Grosch, and J. Kautz, "The state of the art in interactive global illumination," *Computer Graphics Forum*, vol. 31, no. 1, pp. 160–188, 2012.

[69] Nvidia Corporation, "OptiX Ray Tracing Engine," Jun 2012. http://developer.nvidia.com/optix.

[70] Nvidia Corporation, "Nvidia homepage," Jun 2012. http://www.nvidia.com.

[71] S. Woop, J. Schmittler, and P. Slusallek, "RPU: a programmable ray processing unit for realtime ray tracing," *ACM Transactions on Graphics (TOG)*, vol. 24, pp. 434–444, July 2005.

[72] J. Schmittler, I. Wald, and P. Slusallek, "SaarCOR – a hardware architecture for ray tracing," in *Proceedings of the conference on Graphics Hardware 2002*, pp. 27–36, Saarland University, Eurographics Association, 2002. available at http://www.openrt.de.