VNIVERSIDAD NACIONAL
AVTNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

MEJOR LUZ AMBIENTAL PARA AMBIENTES VIRTUALES POR MEDIO DEL GPU

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA:
PEDRO XAVIER CONTLA ROMERO

TUTOR
DR. EDGAR GARUÑO ÁNGELES
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACION
IIMAS - UNAM

MÉXICO, D. F. AGOSTO 2015

A mi Ximena que es una de las personas más importantes en mi vida, que me ha hecho ver que los límites se pueden superar.Ella es una gran inspiración para todos.

Agradecimientos

Mis estudios y mi proyecto de maestría no podrían haber ocurrido sin el apoyo de varias personas e instituciones. Primero que nada, me gustaría agradecer al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo económico para la realización de este trabajo con la beca CVU-489062 para estudios de maestría. También me gustaría darle las gracias al Departamento de Ciencias de la Computación del IIMAS-UNAM por otorgarme un espacio para poder llevar a cabo mi trabajo y por permitirme el uso de materiales para realizar la investigación y el desarrollo de esta tesis. Al Posgrado en Ciencias e Ingeniería de la Computación (PCIC) de la UNAM le estoy muy agradecido por haberme permitido especializarme en una área que no se desarrolla mucho en México.

Le agradezco a mis sinodales Edgar Garduño, Jesús Savage, Jorge Márquez, Fernando Arámbula y Ana Luisa Solís por sus comentarios y su tiempo para revisar mi trabajo. En especial a Edgar que hizo un gran esfuerzo por terminar a tiempo.

Fuera del ambiente académico he recibido el apoyo de muchas personas a quienes quisiera expresar mi agradecimiento, empezando por Sharon y José Luis que siempre han estado en las situaciones buenas y en las malas. A Brenda y a Liz que han sido de gran ayuda para seguir adelante. A Elena que me apoyo desde que comencé la maestría, desde un espacio en su casa hasta la confianza de hablar de cualquier tema.

A mi mamá por ser más que un pilar de mi vida y a mi papá por su apoyo incondicional.

A Cinthya y Montserrat que estuvieron junto a mi trabajando, dándome útiles consejos y haciendo que recuerde muy buenos momentos.

A Rina por ser paciente, darme mi tiempo y espacio y apoyarme siempre.

A mis compañeros de maestría Noé, Mike, Selene, Felipe y Andrés por los ratos de estudio y ayuda durante toda la maestría.

Y a ti si es que alguna vez lees esta tesis.

Resumen

Los modelos de iluminación están enfocados principalmente en el cálculo de la luz que se refleja en las superficies de los objetos que forman una escena. El modelo se utiliza para calcular el color de cada píxel en una imagen que ese está generando. El modelo de iluminación de Phong es el más utilizado y está dividido en tres componentes, la componente ambiental, la componente difusa y la componente especular. Este modelo usa una constante para simular la luz ambiental la cual en muchos casos es suficiente para generar una buena simulación, sin embargo no es del todo correcto ya que esta componente debe de obtener la intensidad de luz que se ha reflejado en todas las superficies de la escena y se debe de incorporar en la ecuación del modelo de iluminación. En este trabajo se propone mejorar esa intensidad de luz ambiental para que las escenas sean un poco más realistas pero sin sacrificar tiempo de procesamiento. Utilizando la capacidad del GPU para cambiar la parte constante de la luz ambiental podemos calcular mejor esta componente sabiendo que la luz ambiental principalmente está afectando las superficies que no están siendo iluminadas directamente por las fuentes de luz. Este cálculo depende principalmente de la distancia a la fuente de luz y tomando una vecindad en la escena del punto que se está evaluando podremos saber si afecta más esta componente a dicho punto.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Graficación por computadora, o computación gráfica, (GC) se refiere a la creación de gráficos bidimensionales y tridimensionales por medio de *hardware* y *software*. Este trabajo considera un aspecto del área de graficación por computadora en 3D por medio del cual es posible reproducir varias "pistas" de profundidad. Por lo que en este trabajo usaremos el término graficación por computadora para referirnos a la modalidad que trabaja en 3D.

La graficación por computadora es una disciplina que se basa en campos como la física, las matemáticas, las ciencias de la computación, la percepción humana, la interacción humano-computadora, la ingeniería, el diseño gráfico o el arte, en donde todos ellos juegan papeles importantes [13]. Por ejemplo, la física se usa tanto para modelar la interacción luz-materiales como para hacer simulaciones para animación, las matemáticas permiten describir formas, manipular objetos geométricos, realizar proyecciones, o interpolar datos, la ingeniería se utiliza para optimizar la asignación de ancho de banda, memoria y tiempo de procesador. También es necesario saber sobre el *hardware* en el que se llevaran a cabo las operaciones de graficación, saber de formatos de archivos para almacenar escenas y otros parámetros de operaciones, estructuras de datos e interfaces de programación especializadas en gráficos (API gráfica). La graficación por computadora está evolucionando muy rápido y por esta razón es un ente en movimiento [17]. Las aplicaciones con mayor uso de graficación por computadora se encuentran en los videojuegos, las caricaturas, los efectos visuales de la industria del entretenimiento, en las películas animadas, en CAD/CAM, en

simulaciones de procesos físicos, en la imageneología médica y en la visualización de información.

La graficación por computadora produce imágenes bidimensionales que representan escenas tridimensionales (en otras palabras, las escenas renderizadas). Este proceso se realiza a través de una serie de subprocesos que incluyen la descripción y modelado de objetos tridimensionales y fuentes de iluminación en una escena, la descripción de la interacción de las frecuencias visibles, emitidas por las fuentes de iluminación, con los objetos y sus materiales, y varias transformaciones de los objetos y fuentes de iluminación. Una forma relativamente sencilla de ver el proceso de graficación por computadora es, primero, generar una escena con herramientas geométricas y físicas para luego modelar como una cámara fotográfica, con cierta orientación, obtiene una imagen de dicha escena. De hecho, esta descripción es muy cercana al proceso que se lleva a cabo en un programa que implemente una aplicación de graficación por computadora.

En general existe un consenso en que las siguientes son las principales áreas de la graficación por computadora:

1. Modelado (*Modeling*) - el conjunto de especificaciones matemáticas de forma y apariencia de los objetos; el modelo también incluye una descripción de como se pueden almacenar dichas especificaciones en la computadora.

2. *Rendering* - el proceso de producción de la imagen resultante, creada a partir de modelos por computadora.

3. Animación (*Animation*) - la técnica de crear la ilusión de movimiento por medio de una secuencia de imágenes, usa tanto el modelado de los objetos como el *rendering* para la creación de las imágenes que varían en el tiempo.

En graficación por computadora es común utilizar como modelo de proyección una cámara *pinhole* o estenopéica. Este tipo de cámara no tiene lentes y se compone de una caja, típicamente cúbica, vacía y oscura por dentro con un pequeña apertura en un extremo por la cual la luz del exterior entra al interior de la caja y se proyecta

en su otro extremo. Por medio de este proceso, la luz proyectada en el interior de la caja forma una imagen al revés, la cual es la proyección de la escena enfrente de la caja. Este modelo es el más usado para graficación y visión por computadora porque describe matemáticamente, y de forma relativamente simple, la relación entre las coordenadas de los puntos en el espacio y las coordenadas de los objetos proyectados en el plano de la imagen.

Es claro que modelar la interacción luz-objetos es indispensable para poder calcular una imagen renderizada; a este modelo se le conoce como modelo de iluminación. Debido a que las imágenes generadas por computadora son utilizadas por seres humanos, solo se necesita modelar la luz visible usando el espacio de color que los seres humanos utilizan en la percepción de imágenes. En graficación por computadora existen, básicamente, dos modelos de iluminación: iluminación global e iluminación local o proyectiva. El proceso de renderizado usando la modalidad de iluminación global pone énfasis en generar imágenes "realistas" por medio de la utilización de un modelo más fiel al comportamiento físico de los fotones; en principio, esta modalidad intenta seguir el comportamiento individual de cada fotón por lo que esta modalidad resulta muy costosa computacionalmente. Por otra parte, el modelo de iluminación local prioriza la velocidad de generación de las imágenes por lo que se hacen ciertos compromisos en el modelado de la interacción de los fotones con los objetos. En este caso, el cálculo de la interacción se hace sobre la superficie visible de cada objeto en lugar de realizarlo sobre cada fotón. Típicamente, la aproximación más utilizada en aplicaciones que generan imágenes en tiempo real (por ejemplo, videojuegos o interfaces gráficas) es la iluminación proyectiva mientras que la iluminación global se usa para crear imágenes con más realismo.

Cualquier modelo de iluminación debe considerar la contribución de todas las fuentes de iluminación sobre los objetos de la escena. El modelo de iluminación más simple es aquel que considera que la iluminación final en un punto visible en la imagen es el resultado de la combinación lineal de la reflexión especular (la luz reflejada desde una superficie suave en un ángulo determinado, también conocida como la reflexión espejo), reflexión difusa (la luz que se refleja de superficies rugosas que tiende a

dirigirse en todas las direcciones) y la luz ambiente (la luz proveniente de fuentes indirectas tal como la luz reflejada de otros objetos).

En la modalidad de graficación por computadora que utiliza el modelo de iluminación local la contribución de luz ambiental se modela como una constante, lo cual es una aproximación muy simple de un proceso muy complejo del mundo real. En aplicaciones que requieren generar imágenes en tiempo real realizar un modelado más sofisticado de la componente de luz ambiental implicaría un costo muy alto ya que, como se mencionó antes, se debería calcular en cada punto la contribución de la luz reflejada de todas las superficies. Sin embargo, desde hace muchos años se han delegado las operaciones gráficas a *hardware* especializado (comúnmente conocido como *tarjetas gráficas*) el cual ha tenido un gran avance en años recientes, permitiendo la programación de dicho *hardware*. La programación del hardware gráfico se puede realizar en varias modalidades entre ellas se encuentra la programación de *shaders*. Estos son pequeños programas que intervienen principalmente en la manipulación geométrica de la escena y en la manipulación de los píxeles de la imagen resultante. Cabe mencionar que el desarrollo tecnológico del *hardware* gráfico es tal que ahora permite realizar procesamiento de un gran conjunto de datos que pueden ser ajenos a la generación de imágenes; a este tipo de programación de *hardware* gráfico se le conoce como Programación de Propósito General en las Tarjetas Gráficas o GPGPU por sus siglas en inglés (*General Purpose Computing in Graphic Processing Units*).

El trabajo que se presenta en esta tesis está enfocado en proponer un modelo más complejo de la componente de iluminación ambiental sin impactar demasiado en el rendimiento de generación de imágenes. Para ello se propone cambiar la función constante por una función que se concentra en la luz que llega a las partes sombreadas y en las regiones que no reciben luz directamente; para minimizar el impacto en el rendimiento computacional se utiliza como herramienta principal la programación de la tarjeta gráfica para acelerar los cálculos.

Este documento se encuentra dividido de la siguiente manera: en el Capítulo 2 se desarrollarán los conceptos referentes a las matemáticas aplicadas para la generación de imágenes (*rendering*), en el Capítulo 3 se describen los modelos de iluminación y

técnicas para producir sombras, en el Capítulo 4 se muestra el algoritmo propuesto, los experimentos y resultados obtenidos. Finalmente, en el Capítulo 5 presentamos la conclusiones del proyecto y el trabajo futuro.

Motivación y Objetivos Principales

En el mundo real las fuentes de luz emiten fotones lo que provoca dispersión de luz sobre el mundo y causa la interacción con el objetos. Luz también se refiere a un tipo particular de radiación y los humanos la han descrito, no solamente como fenómeno físico (como energía), sino también en términos de percepción, es decir, la forma en que el sistema nervioso óptico tiene para procesar y percibir la luz.

El objetivo de este proyecto es mostrar resultados más llamativos con el uso de las tarjetas gráficas para la simulación de la luz, en especial este trabajo está enfocado en la componente ambiental, la cual tiene más influencia en las superficies que están siendo ocultas por una sombra.

Lo novedoso de este trabajo es la modificación de un modelo de iluminación local para obtener lugares donde la luz ambiental tiene más interacción dentro de la escena, esto se logra utilizando mapas de sombras y haciendo una comparación con un modelo de iluminación global.

Capítulo 2

Graficación por Computadora Proyectiva

En este capítulo se hace una presentación muy breve de la modalidad de graficación por computadora proyectiva que utiliza un modelo de iluminación local. Como se menciona antes, la idea básica de graficación por computadora en 3D es crear una imagen bidimensional por medio de proyectar una escena del mundo (real o imaginario), creada por medio de una descripción matemática. El uso de las matemáticas es esencial tanto para crear la descripción de la escena del mundo como para todos los subprocesos que permiten generar la imagen bidimensional de dicho mundo. En este capítulo se presenta las herramientas básicas necesarias para producir la descripción matemática y su imagen. El desarrollo de la graficación por computadora se encuentra muy ligada al desarrollo de *hardware* que implementa varias de las herramientas matemáticas y técnicas de graficación por computadora. En la actualidad el *hardware* para graficación por computadora ha alcanzado un nivel de desarrollo que permite realizar miles de operaciones matemáticas por segundo. En esta tesis se aprovecha el *hardware* gráfico para implementar el nuevo modelo para iluminación ambiental y presentamos al final de este capítulo una introducción del *hardware* gráfico.

2.1. Antecedentes Matemáticos

En graficación por computadora, la unidad básica para la descripción de objetos es un punto en el espacio. Por ejemplo, un objeto puede ser representado por medio

de una nube de puntos disconexos o por medio de una nube de puntos conectados entre sí que se representan por medio de una gráfica (V, E), donde V es un conjunto de vértices o puntos en el espacio y E es un conjunto de aristas o conexiones entre puntos. La posición de objetos, las fuentes de iluminación y la cámara se también se representan por medio de puntos en el espacio. Para representar a un punto en el espacio se utiliza un vector $\boldsymbol{x} \in \mathbb{R}^n$, que es una n-tupla $(x_1, x_2, \ldots, x_n)^T$ donde n es la dimensión del espacio. En graficación por computadora se utiliza típicamente el espacio $\mathbb{R}^4$ de números reales al igual que los espacios de números enteros $\mathbb{Z}^2$ y $\mathbb{Z}^3$ debido a la necesidad de trabajar con objetos y espacios discretos. Como se mencionó anteriormente, la modalidad de graficación por computadora que interesa utiliza la geometría proyectiva y por lo tanto usa coordenadas homogéneas, un sistema que utiliza las coordenadas Euclidianas junto con puntos extra para poder representar puntos al infinito; por ejemplo, se tienen dos números reales a y w y la operación $\frac{a}{w}$. Si se deja fijo el valor de a conforme el valor de w se hace menor, el valor $\frac{a}{w}$ se hace mayor y si el valor de w se acerca a cero, el valor de la relación $\frac{a}{w}$ tiende a infinito. Cada punto de la imagen proyectada representa una posible línea de visión de un rayo incidente (cualquier punto a lo largo del rayo se proyecta sobre el mismo punto de la imagen bidimensional) por lo que solo la dirección del rayo es relevante y no la distancia de los puntos a lo largo del rayo. En este sistema, el punto $(x_1, x_2, x_3)^T$ es igual a cualquier punto en la línea $\left(\frac{x_1}{w}, \frac{x_2}{w}, \frac{x_3}{w}\right)^T$ para $w \neq 0$. Por lo tanto, el punto con coordenadas homogéneas $(x_1, x_2, x_3, w)^T$, con $w \neq 0$, corresponde al punto con coordenadas no-homogéneas $\left(\frac{x_1}{w}, \frac{x_2}{w}, \frac{x_3}{w}\right)^T$. Los vectores cuya magnitud es igual a uno también son de gran importancia en graficación por computadora, para un vector $\boldsymbol{x}$ se representará su vector normalizado por medio de $\overrightarrow{\boldsymbol{x}}$.

En graficación por computadora interesa transformar tanto los objetos (por ejemplo, cambiar la orientación o posición de un objeto) como las posiciones de la cámara y de las fuentes de iluminación. Las transformaciones más comunes en graficación por computadora son las transformaciones o mapeos lineales $T : \mathbb{R}^n \to \mathbb{R}^m$ donde $m \leq n$. Debido a que los espacios que se utilizan en graficación por computadora son finitos es posible representar las transformaciones lineales por medio de la multiplicación

de matrices y vectores, por ejemplo, para los vectores $\boldsymbol{y} \in \mathbb{R}^m$ y $\boldsymbol{x} \in \mathbb{R}^n$ el mapeo $\boldsymbol{y} = T(\boldsymbol{x})$ se puede representar por medio de la multiplicación $\boldsymbol{y} = \mathbf{A}\boldsymbol{x}$ donde $\mathbf{A}$ es una matriz de dimensiones $m \times n$ cuyos elementos $a_{i,j}$ son números reales, para $1 \leq i \leq m$ y $1 \leq j \leq n$. Cuando m y n son iguales, es posible que la transformación lineal T sea invertible y por lo tanto la matriz correspondiente $\mathbf{T}$ también es invertible; a esta matriz la denotamos $\mathbf{T}^{-1}$ y cumple con $\mathbf{T}\mathbf{T}^{-1} = \mathbf{I}$, donde $\mathbf{I}$ es la matriz identidad [13, 17, 18].

2.1.1. Transformaciones Típicas

Algunas de las transformaciones más usadas en graficación por computadora son las siguientes.

Traslación Para trasladar un punto $\boldsymbol{x}$ por cierta cantidad α, β y γ en dirección de los vectores coordenados $\vec{\boldsymbol{x}}$, $\vec{\boldsymbol{y}}$ y $\vec{\boldsymbol{z}}$, respectivamente, se usa la siguiente matriz:

$$\mathbf{T}_{\alpha,\beta,\gamma} = \begin{bmatrix} 1 & 0 & 0 & \alpha \\ 0 & 1 & 0 & \beta \\ 0 & 0 & 1 & \gamma \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Escalamiento Para escalar un punto $\boldsymbol{x}$ cierta cantidad α, β y γ en dirección de los vectores coordenados $\vec{\boldsymbol{x}}$, $\vec{\boldsymbol{y}}$ y $\vec{\boldsymbol{z}}$, respectivamente, se utiliza la siguiente matriz:

$$\mathbf{S}_{\alpha,\beta,\gamma} = \begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación El cambio de orientación de un punto, con cierto ángulo, alrededor de cualquier punto sobre la línea formada por los vectores coordenados $\vec{\boldsymbol{x}}$, $\vec{\boldsymbol{y}}$ y $\vec{\boldsymbol{z}}$ se representa por medio de

Rotación sobre el eje $\overrightarrow{x}$

$$\mathbf{R}_{\overrightarrow{x},\theta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación sobre el eje $\overrightarrow{y}$

$$\mathbf{R}_{\overrightarrow{y},\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación sobre el eje $\overrightarrow{z}$

$$\mathbf{R}_{\overrightarrow{z},\theta} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.1.2. Otras Transformaciones

En graficación por computadora es importante utilizar los puntos en otro sistema de coordenadas en especial en el sistema de coordenadas de la cámara y de la pantalla en la que se va a desplegar la imagen renderizada.

2.1.2.1. Vista o Cámara

Esta transformación hace un cambio de sistema de coordenadas, donde los puntos que se representaron en un espacio llamado mundo (la representación de los objetos en su espacio de origen) ahora se encuentran representados por otros ejes coordenados que representan el espacio de la cámara. El espacio de la cámara se encuentra

determinado por los ejes coordenados $\overrightarrow{\boldsymbol{u}^c}$, $\overrightarrow{\boldsymbol{v}^c}$ y $\overrightarrow{\boldsymbol{w}^c}$, cuyo origen está determinado por el vector $\boldsymbol{p}^c$. Estos vectores se pueden obtener por medio de la descripción de la orientación de la cámara utilizando los vectores $\boldsymbol{l}^c$y $\boldsymbol{t}^c$, los cuales representan la dirección a la cual observa la cámara y la dirección donde se encuentra la parte superior de la cámara, respectivamente. La obtención de los vectores del sistema coordenado se hace como sigue:

$$\overrightarrow{\boldsymbol{w}^c} = \frac{\boldsymbol{l}^c}{||\boldsymbol{l}^c||},$$

$$\overrightarrow{\boldsymbol{v}^c} = \frac{\boldsymbol{t}^c - (\boldsymbol{t}^c \cdot \overrightarrow{\boldsymbol{w}^c})\overrightarrow{\boldsymbol{w}^c}}{||\boldsymbol{t}^c - (\boldsymbol{t}^c \cdot \overrightarrow{\boldsymbol{w}^c})\overrightarrow{\boldsymbol{w}^c}||}$$

y

$$\overrightarrow{\boldsymbol{u}^c} = \overrightarrow{\boldsymbol{v}^c} \times \overrightarrow{\boldsymbol{w}^c},$$

donde $||\bullet||$ representa la norma ℓ_2 de un vector. Para dos vectores $\boldsymbol{a}$ y $\boldsymbol{b}$, la operación $\boldsymbol{a} \cdot \boldsymbol{b}$ representa el producto interno entre ellos.

Claramente, los vectores $\overrightarrow{\boldsymbol{u}^c}$, $\overrightarrow{\boldsymbol{v}^c}$ y $\overrightarrow{\boldsymbol{w}^c}$ crean una base ortonormal. La representación de un punto $\boldsymbol{x}^m$, localizado en el espacio del mundo, se representa en el espacio de la cámara por medio de la proyección sobre los vectores $\overrightarrow{\boldsymbol{u}^c}$, $\overrightarrow{\boldsymbol{v}^c}$ y $\overrightarrow{\boldsymbol{w}^c}$:

$$x_1^c = \boldsymbol{x}^m \cdot \overrightarrow{\boldsymbol{u}^c},$$

$$x_2^c = \boldsymbol{x}^m \cdot \overrightarrow{\boldsymbol{v}^c}$$

y

$$x_3^c = \boldsymbol{x}^m \cdot \overrightarrow{\boldsymbol{w}^c}.$$

Estas proyecciones se pueden representar en forma matricial como sigue:

$$\boldsymbol{x}^c = \left[\overrightarrow{\boldsymbol{u}^c}\overrightarrow{\boldsymbol{v}^c}\overrightarrow{\boldsymbol{w}^c}\right]^T \boldsymbol{x}^m.$$

Tomando en cuenta el desplazamiento determinado por el vector $\boldsymbol{p}^c$ se tiene la siguiente matriz:

$$\boldsymbol{C}_{\boldsymbol{l}^c,\boldsymbol{t}^c,\boldsymbol{p}^c} = \begin{bmatrix} \overrightarrow{\boldsymbol{u}^c} & \overrightarrow{\boldsymbol{v}^c} & \overrightarrow{\boldsymbol{w}^c} & \boldsymbol{p}^c \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}.$$

2.1.2.2. Proyección

Una vez que los puntos se encuentran en el espacio de la cámara, éstos son proyectados sobre un plano conocido como *plano de la imagen*. El plano de la imagen es la última etapa antes de que los puntos sean convertidos a un espacio $\mathbb{Z}^n$ en el proceso de generación final de la imagen. El proceso de la proyección sobre el plano de la imagen desecha en la proyección final aquellos puntos que no se encuentran dentro de un subespacio del espacio de la cámara, el cual es seleccionado por el observador; este subespacio se le conoce como el volumen de visión o *frustrum*. En la práctica se usan dos métodos de proyección sobre el plano de la imagen: la proyección en perspectiva y ortogonal. La forma del volumen de visión es una pirámide truncada con base rectangular cuya base es paralela al plano definido por los vectores $\overrightarrow{\boldsymbol{u}^c}$y $\overrightarrow{\boldsymbol{v}^c}$ cuando se trata de una proyección en perspectiva, mientras que cuando se trata de una proyección ortogonal el volumen es un cubo rectangular cuyas paredes son perpendiculares a los vectores base $\overrightarrow{\boldsymbol{u}^c}$, $\overrightarrow{\boldsymbol{v}^c}$y $\overrightarrow{\boldsymbol{w}^c}$. El volumen de visión proyecta al plano de la imagen solo los puntos dentro de dicho volumen, a este proceso se reconoce como *Clipping*. Los planos que delimitan el volumen de visión son llamados *clipping planes* y son usados para reducir el tiempo de generación (*rendering*) de una imagen para una escena.

Los puntos que son proyectados sobre el plano de la imagen son los puntos que serán mapeados a las coordenadas de un dispositivo de memoria para que se pueda desplegar en dispositivos tales como pantallas o impresoras.

Ortogonal La proyección ortogonal simula el fenómeno que ocurre cuando se observan objetos a una distancia muy corta por lo que no se presenta perspectiva. Por lo tanto, no se toma en cuenta la distancia sobre el eje $\overrightarrow{\boldsymbol{w}^c}$ que hay entre los puntos. La

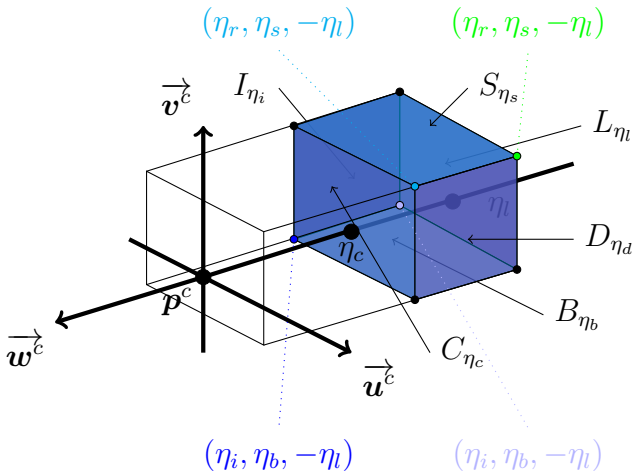


Figura 2.1: Planos que forman el volumen de visión de la proyección ortogonal

definición de volumen de visión es muy sencilla ya que los planos son perpendiculares a los vectores base $\overrightarrow{\boldsymbol{u}^c}$, $\overrightarrow{\boldsymbol{v}^c}$ y $\overrightarrow{\boldsymbol{w}^c}$ y forman los siguientes conjuntos

$C_{\eta_c} = \{\boldsymbol{x} | x_3 = \eta_c w_3^c\}$,

$L_{\eta_l} = \{\boldsymbol{x} | x_3 = \eta_l w_3^c\}$,

$I_{\eta_i} = \{\boldsymbol{x} | x_1 = \eta_i u_1^c\}$,

$D_{\eta_d} = \{\boldsymbol{x} | x_1 = \eta_d u_1^c\}$,

$S_{\eta_s} = \{\boldsymbol{x} | x_2 = \eta_s v_2^c\}$

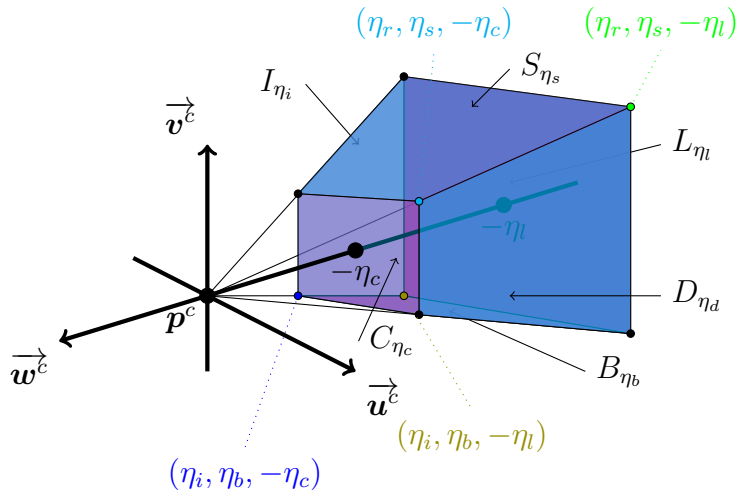
y

$B_{\eta_b} = \{\boldsymbol{x} | x_2 = \eta_b v_2^c\}$;

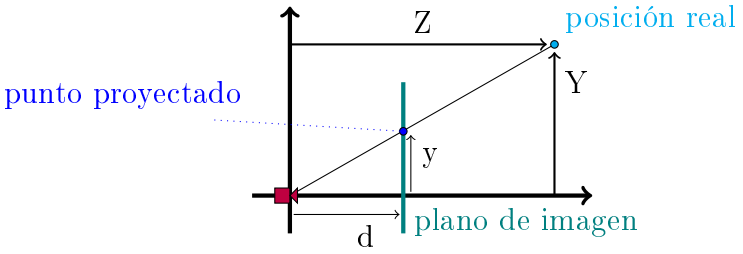
Se refiere a estos planos como *cercano*, *lejano*, *izquierdo*, *derecho*, *superior* e *inferior*, respectivamente (ver Figura 2.1). Basado en estos planos, la proyección ortogonal también se puede llevar a cabo por medio de una transformación lineal representada por la siguiente matriz

$$\mathbf{O} = \begin{bmatrix} \frac{2}{\eta_d - \eta_i} & 0 & 0 & \frac{\eta_d + \eta_i}{\eta_d - \eta_i} \\ 0 & \frac{2}{\eta_s - \eta_b} & 0 & \frac{\eta_s + \eta_s}{\eta_s - \eta_b} \\ 0 & 0 & \frac{-2}{\eta_l - \eta_c} & \frac{\eta_l + \eta_c}{\eta_l - \eta_c} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Perspectiva En el caso de la proyección en perspectiva se desea simular la observación de objetos lejanos que entre más alejados se encuentran estos parecen converger en un punto lejano. En realidad, el volumen de visión en la proyección en perspectiva también es un cubo pero los planos derecho, izquierdo, inferior y superior cambian su orientación debido al fenómeno de perspectiva como se muestra en la Figura 2.2a. Los planos cercano y lejano son los más fáciles de definir ya que se usa la misma interpretación que en el caso de proyección ortogonal. Una representación matricial



(a)



(b)

Figura 2.2: Volumen de visión para el caso de proyección en perspectiva. En (a) se muestra los planos de restricción y en (b) la relación que existe entre el plano proyectado y su posición original.

de esta proyección es la siguiente:

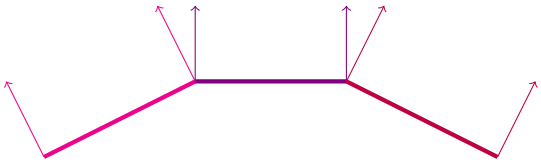
$$\mathbf{P} = \begin{bmatrix} \frac{2\eta_c}{\eta_d - \eta_i} & 0 & \frac{\eta_d + \eta_i}{\eta_d - \eta_i} & 0 \\ 0 & \frac{2\eta_c}{\eta_s - \eta_s} & \frac{\eta_s + \eta_s}{\eta_s - \eta_s} & 0 \\ 0 & 0 & \frac{-(\eta_l + \eta_c)}{\eta_l - \eta_c} & \frac{-2\eta_l}{\eta_l - \eta_c} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

2.2. Coloración (*Shading*)

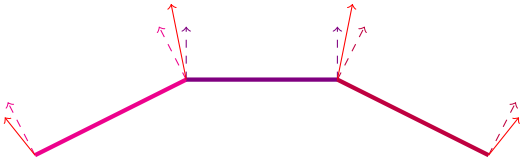
Coloración o *shading* es usado en dibujo para representar los niveles de oscuridad en el papel mediante tonos más obscuros para las áreas más obscuras y tonos claros para las áreas claras que hay en las escenas. En graficación por computadora se le llama *shading* al proceso que altera los colores de los objetos, superficies y polígonos en la escena 3D, con base en la distancia y la orientación con respecto a las fuentes de luz. Este proceso se desarrolla después de haber obtenido los puntos proyectados al plano de la imagen, al final de hacer el proceso de *raster* (en otras palabras, el proceso de barrido). La coloración aplica un modelo de iluminación junto a un modelo de sombreado. Existen varios modelos de sombreados: *Flat Shading* (véase la Figura 2.3a) sombrea cada polígono del objeto basado en el ángulo entre la normal del polígono de la superficie y la dirección de la fuente de luz, el color del objeto y la intensidad de la luz. Se usa principalmente para hacer *render* rápido. El resultado de este *shading* es que cada vértice del polígono es coloreado de un solo color permitiendo ver la diferencia entre los polígonos adyacentes, aquí la normal es igual para todos los vértices. *Smooth Shading* en comparación al *Flat Shading* difiere en que el color cambia de píxel a píxel. Asume que las superficies son curvas y utiliza interpolación para calcular los valores de los píxeles intermedios entre vértices.

Los principales modelos son *Gouraud Shading* [7] y *Phong Shading* [19]. Gouraud (véase Figura 2.3b) determina la normal en cada vértice del polígono, se aplica el modelo de iluminación en cada vértice para obtener el color correspondiente y finalmente

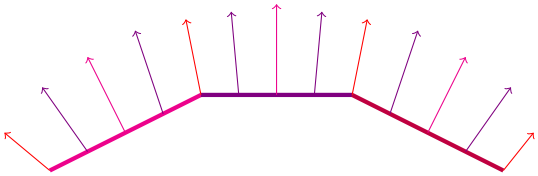
interpola el color correspondiente a cada píxel dentro del polígono. *Phong Shading* (véase Figura 2.3c) es muy similar a Gouraud pero aquí se interpolan las normales sobre el polígono de la superficie, después se interpolan las normales sobre los píxeles dentro del polígono que se van a calcular y con ellos se hace el cálculo del color sobre esos píxeles. El modelo de iluminación describe la física de la interacción de la luz con la materia y se hablará más de él en el siguiente capítulo.



(a) *Flat Shading*, las normales están sobre los vértices, un vértice puede tener diferente normal dependiendo del polígono que se está evaluando.



(b) Gouraud *Shading*, las normales están sobre los vértices, son resultado de la interpolación de las normales que comparten un mismo vértice.



(c) Phong *Shading*, aquí se interpola las normales durante toda la superficie del polígono.

Figura 2.3: En las figuras se muestran las diferencias principales entre los modelos de sombreado.

2.3. Flujo Gráfico

La secuencia de operaciones que se realizan para producir una imagen a partir de la descripción de la escena se le conoce como flujo gráfico o *graphic pipeline*. La implementación de estas operaciones originalmente se llevaba a cabo en el CPU a través

de *software*. El desarrollo de nuevas técnicas de graficación y de *hardware* han permitido que las operaciones del flujo gráfico progresivamente se hayan implementando en *hardware* especializado para generación de imágenes por técnicas de graficación por computadora; a este *hardware* típicamente se le conoce como *graphical processing units* (GPUs) o tarjetas gráficas.

Tradicionalmente, el desarrollo de hardware gráfico ha permitido que el costo computacional para la generación de imágenes se le cargue a este *hardware* y se libere a los CPUs para tareas de cómputo de otro tipo.

Los principales estados de un pipeline se muestran en la Figura 2.4.

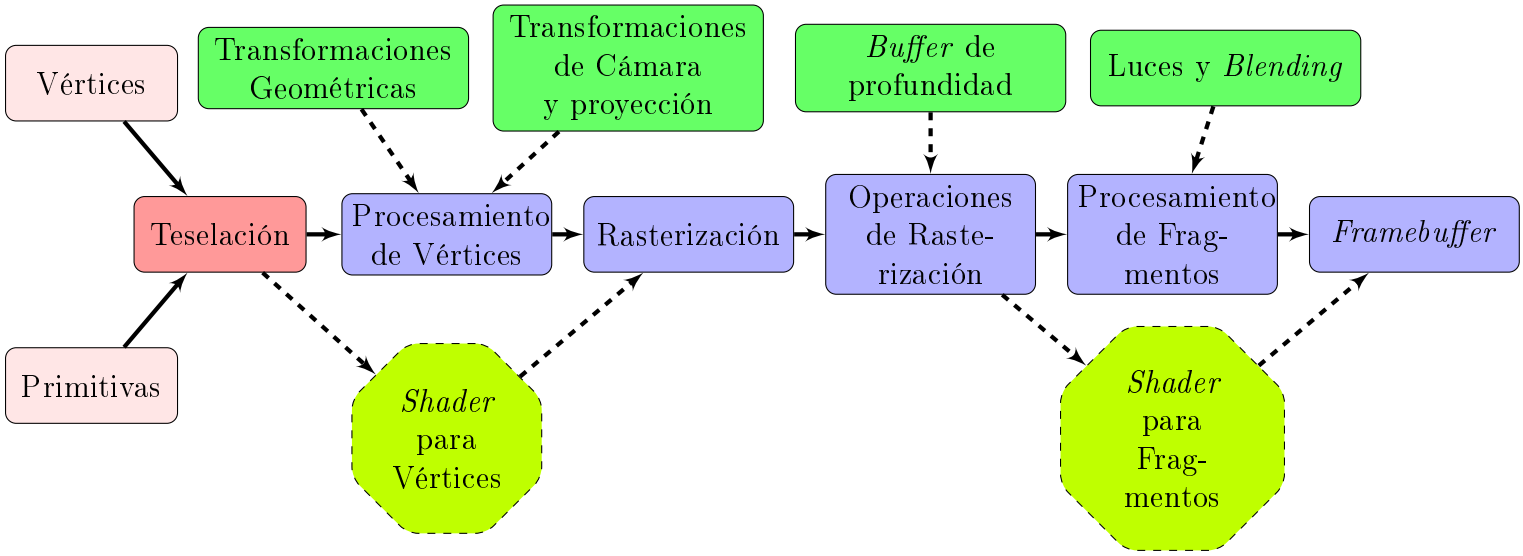


Figura 2.4: Esquema que muestra, simplificadamente, en bloques las operaciones que se realizan en el flujo o *pipeline* gráfico.

Sin embargo, el desarrollo de los GPUs ha sido tal que ahora las tarjetas gráficas contienen miles de procesadores especializados en operaciones necesarias en graficación por computadora tales como las transformaciones lineales. Este desarrollo ha permitido que el paradigma de usar CPUs para cómputo no gráfico y los GPUs para renderizar ya no sea necesariamente así y en la actualidad se utiliza *hardware* gráfico para realizar cómputo no gráfico tal como cómputo científico. Esto se debe a que la implementación del flujo gráfico se ha hecho de forma cada vez más flexible a tal punto que las nuevas tarjetas gráficas pueden ser programadas y sus nuevas interfaces de programación (*application programming interfaces*) proporcionan una herramienta

para poder hacerlo de una forma cada vez más flexible. Ejemplos de estas APIs son OpenGL® y DirectX® (este último solamente para Windows®) las cuales son de las más utilizadas para la programación de gráficos por computadora. La empresa Nvidia®, que produce tarjetas de gráficas, creó su lenguaje de programación con la colaboración de Microsoft® para sus productos llamado Cg (*C for graphics*) , el cual está basado en el lenguaje C, para la programación de gráficos por computadora. Por otro lado, CUDA® de Nvidia® y OpenCL® son APIs diseñadas para programación de propósito general en tarjetas gráficas.

Para el propósito de esta tesis se utiliza OpenGL® ya que da herramientas para desarrollar el *software* del trabajo. El lenguaje de programación de OpenGL® se llama *OpenGL Shading Language* o GLSL. GLSL es un lenguaje de programación de alto nivel procedural para las tarjetas gráficas y los programas que se generan son llamados *shaders*. GLSL se basa en el lenguaje C/C++ tanto en la sintaxis como en el flujo de control. Este lenguaje contiene funciones y variables específicas para las operaciones geométricas y de procesamiento de imágenes. Por último, GLSL es portable para todos los sistemas operativos que cuenten con tarjetas gráficas que soportan OpenGL®.

2.3.1. *Shaders*

Como se menciona anteriormente, los *shaders* son programas en GLSL y existen cinco tipos diferentes [16, 21, 23], los cuales se diferencian porque se encargan de diferentes estados del flujo gráfico, ver Figura 2.4. Algunas ventajas de utilizar los *shaders*, son:

- Incremento en el realismo de materiales como metales, piedras o pintura.
- Incremento en el realismo en luces para los efectos de iluminación en áreas y sombras.
- Modelado de fenómenos naturales como humo, agua, nubes.
- Generación de efectos avanzados en el *render*, iluminación global o *ray-tracing*.

- Simulación de materiales no fotorrealistas: efectos pictóricos, dibujos en tinta y pluma o simulación de técnicas de ilustración.
- Nuevos usos en almacenamiento de normales en memoria de textura, valores de brillo o coeficientes polinomiales.
- Capacidad de generar texturas procedurales dinamicamente en 2D y 3D.

De los cinco tipos de *shaders,* los principales son el de vértices y el de fragmentos porque son la base del *pipeline* de OpenGL®. La implementación de los *shaders* de vértices y de fragmentos permite cambiar las funciones fijas de OpenGL® con que se pueden realizar nuevas operaciones, situación que hasta hace poco tiempo no era posible hacer. Utilizando los *shaders* de vértices y fragmentos se puede manipular toda la escena que ha sido generada por medio de vértices, geometría, coordenadas de texturas, luces, *render* o animaciones. Por otra parte, los *shaders* de geometría y de teselación, como sus nombres lo indican manipulan la geometría para poder generar más vértices dentro de la escena los cuales no estaban definidos en el modelo original de la misma. Por último, el *shader* más reciente es de cómputo, este *shader* se usa para la programación de propósito general.

Para el trabajo que se realiza en esta tesis solo se utilizará los dos *shaders* principales: vértices y fragmentos.

2.3.1.1. *Shader* de Vértices

Este *shader* está definido dentro del flujo gráfico en el estado de procesamiento de vértices, ver Figura 2.4. El procesamiento de vértices se encarga de realizar operaciones sobre la información asociada a los vértices de los modelos donde principalmente se llevan acabo transformaciones lineales en vértices, normalizaciones y transformaciones de normales, generación y asignación de coordenadas de textura y sus transformaciones, aplicación de transformaciones típicas a las luces y asignación del material.

La manipulación de los vértices del objeto se hace de manera individual, teniendo un mejor manejo para detalles que se quieran modificar. Una de las principales

operaciones que se realiza en este tipo de *shaders* es la aplicación de las transformaciones lineales; gracias a que GLSL tiene implementado matrices y vectores como tipo de datos además de las operaciones entre ellos, se hace muy sencillo poder aplicar estas transformaciones a los vértices. En tarjetas gráficas que son compatibles con OpenGL® (en otras palabras, implementan toda o parcialmente el *pipeline* gráfico en *hardware*) las operaciones se llevan a cabo en *hardware* y en las tarjetas actuales estas operaciones se realizan en paralelo sobre todos los vértices de la escena.

Este *shader* principalmente implementa las transformación de modelo, cámara y proyección a todos los vértices, de esta forma OpenGL® puede hacer la rasterización para formar los fragmentos y pasarlos al siguiente *shader*.

Shader de Fragmentos

Los fragmentos son estructuras de datos por píxel que se obtienen durante la rasterización. El *shader* de fragmentos se encarga de la operaciones sobre estos fragmentos. Se considera que los fragmentos son los píxeles antes de que sean convertidos en píxeles. El *shader* de fragmentos está definido dentro del procesamiento de fragmentos, ver Figura 2.4, el cual se realiza después de la rasterización y antes del despliegue en algún dispositivo o almacenado en una textura. Este *shader* permite principalmente hacer operaciones en valores interpolados, acceder a texturas y aplicación de las mismas. También se puede llevar a cabo procesamiento de imágenes y, en casos muy avanzados, *raycasting* y *raytracing*. Un *shader* de este tipo produce valores, típicamente de color, que son el resultado de valores iniciales modificados a través de la interpolación de vértices, normales y texturas. El resultado de este *shader* se guarda en el *framebuffer* o en una textura, dependiendo del uso que se quiera dar.

En este trabajo se ocupa el *shader* de fragmentos para poder aplicar el modelo de iluminación de Phong a la escena, el cual es la base de este trabajo.

Capítulo 3

Modelo de Iluminación de Phong

En graficación por computadora la luz es muy importante ya que es uno de los elementos que da una de las pistas más importantes de tridimensionalidad. La forma en que la luz interactúa con las propiedades de los objetos, tales como su forma, color o material, determina en buena medida la aparición y forma de sombras, la forma e intensidad de los reflejos, y la degradación de los colores. La analogía con una cámara fotográfica sirve para ver la importancia de la iluminación en el proceso de generación de imágenes: la localización, la intensidad, o el color de la iluminación afectará la imagen final. Por lo tanto, es crucial incorporar tanto fuentes de iluminación y materiales para los objetos como un modelo de la luz emitida por la fuentes de iluminación el cual interactúa con los materiales de los objetos de la escena. A este modelo, se le conoce como modelo de iluminación y será el tema del cual hablaremos en este capítulo.

3.1. Naturaleza de la Luz

Vivimos en un mundo donde vemos la radiación electromagnética en todos lados, constantemente nos estamos dando un baño de calor y luz que proviene del sol, televisión, radio y teléfonos celulares. La luz es particularmente una radiación electromagnética en un rango de frecuencias que puede ser detectada por el ojo humano y que desde hace mucho tiempo se ha descrito tanto en términos físicos, por ejemplo, basándose en su energía, al igual que en términos perceptibles como el sistema humano

(a) Luz Puntual

(b) *Spotlight*

Figura 3.1: En (a) no importa donde se encuentren las superficies ya que emite la luz para todos los lados de igual forma. (b) es como la luz puntal emite la luz a cualquier parte, pero dentro del volumen del cono el cual tiene un ángulo ω de apertura.

visual que puede percibir y procesar la luz. A nivel microscópico, la luz se comporta como partículas (fotones) al mismo tiempo que es una onda que está caracterizada por su frecuencia.

Para simular la luz en graficación se han desarrollado varios modelos que calculan el color de los píxeles. Estos modelos son llamados modelos de iluminación y se dedican principalmente a calcular el color de las superficies de los objetos con la interacción de la luz en la escena.

En graficación por computadora existen distintos tipos de fuentes de luz que se usan para simular la luz. Los siguientes son los más comunes:

Luz Puntual Es una fuente que irradia luz en todas direcciones de forma uniforme como se muestra en la Figura 3.1a. Por sus características, esta fuente de luz se modela como un punto en el espacio al cual denominaremos como $\boldsymbol{l}_p$ y el cual tiene asociado a él las propiedades de la fuente de iluminación (por ejemplo, el color de la luz). La presencia de una luz puntual puede introducir mucha variación en la escena con su conjunto infinito de valores para $\boldsymbol{l}_p$, lo que asegura que cada punto en una superficie que mira hacia la luz recibe su energía de una única dirección $\boldsymbol{l}_p$. Los focos son un ejemplo de esta luz.

Spotlight Esta es una fuente de luz similar a la luz puntual pero en este caso la fuente está delimitada por un volumen de forma cónica como se muestra en la

Figura 3.1b. En este caso, la fuente de iluminación se modela con dos vectores, el de posición y otro que indica la dirección hacia donde crece la base del cono.

Luz Direccional Esta fuente de luz es una luz puntual su posición está situada muy lejos de la escena, haciendo que sus rayos sean percibidos como paralelos entre ellos, ver Figura 3.2. Esta fuente de luz se modela con un vector normalizado $\overrightarrow{l_d}$ que señala una dirección determinada. El Sol es un buen ejemplo de esta luz.

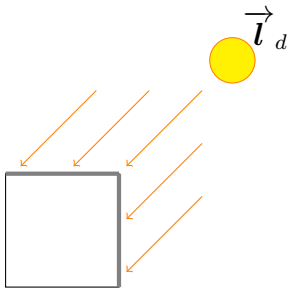


Figura 3.2: Luz Direccional, dentro de graficación por computadora para diferenciar una luz puntual de la luz direccional se hace mediante la cuarta componente de las coordenadas homogéneas que es igual a cero.

Luz Ambiental

La luz ambiental es aquella luz que no proviene de una fuente específica, sino es la luz que proviene de la reflexión de la luz producida por diferentes fuentes de luz sobre las superficies de los objetos que conforman la escena, véase Figura 3.3. La graficación por computadora realista y no realista se diferencian, básicamente, en la forma en que modelan este tipo de fuente de iluminación con las superficies de los objetos.

3.2. Reflexión sobre Superficies

La interacción entre la luz y una superficie se modela básicamente usando la ecuación de Fresnel y las leyes de Snell, las cuales nos dicen como se refleja y refracta un rayo de luz al hacer contacto con la superficie que tiene un material determinado. En este modelo un rayo de luz viaja en un medio, o material, con un índice refractivo

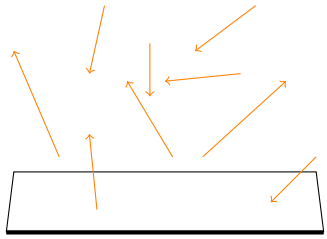


Figura 3.3: Luz Ambiental, la luz proviene de muchos lados de la escena, por lo que es difícil simular.

n_1 en dirección a otro medio con índice refractivo n_2. El rayo de luz que viaja en el medio n_1 se conoce como incidente y se modela con un vector $\overrightarrow{\boldsymbol{l}_i}$. El rayo incidente $\overrightarrow{\boldsymbol{l}_i}$ llega al punto $\boldsymbol{p}$ de la interfaz del medio n_i con el medio n_2 en este punto se puede obtener el vector normal $\overrightarrow{\boldsymbol{n_p}}$ con dirección al medio n_1. El rayo de luz $\overrightarrow{\boldsymbol{l}_i}$ forma un ángulo θ_i con el vector normal $\overrightarrow{\boldsymbol{n_p}}$. Como resultado de la interacción de la luz incidente $\overrightarrow{\boldsymbol{l}_i}$ sobre la interfaz n_1- n_2 se producirán un rayo reflejado $\overrightarrow{\boldsymbol{l}_r}$ y un rayo transmitido $\overrightarrow{\boldsymbol{l}_t}$. Los rayos $\overrightarrow{\boldsymbol{l}_r}$ y $\overrightarrow{\boldsymbol{l}_t}$ forman los ángulos θ_r y θ_t con el vector normal $\overrightarrow{\boldsymbol{n_p}}$, respectivamente como se muestra en la Figura 3.4. En graficación por computadora, se asume que los ángulos θ_i y θ_r son iguales y que los ángulos θ_i y θ_t están relacionados de la siguiente manera por la ley de Snell:

$$n_1 \sin\theta_i = n_2 \sin\theta_t. \tag{3.1}$$

Por lo tanto, es posible calcular $\overrightarrow{\boldsymbol{l}_r}$ y $\overrightarrow{\boldsymbol{l}_t}$ a partir de $\overrightarrow{\boldsymbol{l}_i}$, $\overrightarrow{\boldsymbol{n_p}}$, n_1 y n_2.

La tasa de luz reflejada se conoce como reflectancia y la tasa de luz transmitida se conoce como trasmitancia. La reflectancia se puede calcular por medio de

$$\mathcal{R}\left(\theta_i, \theta_t, n_1, n_2\right) = r_p\left(\theta_i, \theta_t, n_1, n_2\right) + r_s\left(\theta_i, \theta_t, n_1, n_2\right), \tag{3.2}$$

donde

$$r_p\left(\theta_i, \theta_t, n_1, n_2\right) = \frac{n_2 \cos\theta_i - n_1 \cos\theta_t}{n_2 \cos\theta_i + n_1 \cos\theta_t} \tag{3.3}$$

y

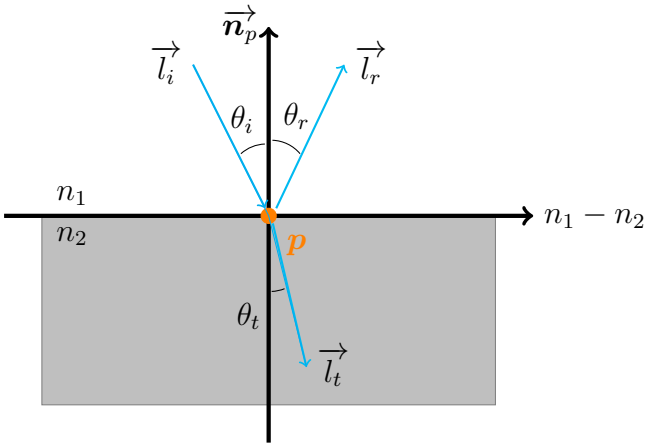


Figura 3.4: Un rayo es reflejado y se transmite dentro de la superficie con $\overrightarrow{n_p}$ normal de la superficie y las superficies con índice de refracción n_1 y n_2 .

$$r_s\left(\theta_i, \theta_t, n_1, n_2\right) = \frac{n_1 \cos\theta_i - n_2 \cos\theta_t}{n_1 \cos\theta_i + n_2 \cos\theta_t}. \tag{3.4}$$

Las funciones r_p y r_s se conocen como la polarización paralela (*p-polarized*) y perpendicular (*s-polarized*), respectivamente. La transmitancia se calcula con base en la reflectancia de la siguiente forma

$$\mathcal{T}\left(\theta_i, \theta_t, n_1, n_2\right) = 1 - \mathcal{R}\left(\theta_i, \theta_t, n_1, n_2\right). \tag{3.5}$$

La luz se dispersa después de interactuar con la superficie y una forma de caracterizar las propiedades reflectivas de una superficie es por medio de una función de distribución de la reflectancia bidireccional o BRDF, una función que define las características espectrales y espaciales de reflexión de una superficie. En una forma general, una BRDF calcula la relación entre la cantidad de luz reflejada de la superficie en la dirección $\overrightarrow{l_r}$, o radiancia (L_s), y la cantidad de luz que llega en la dirección $\overrightarrow{l_i}$, o irradiancia (H). Para calcular una BRDF se utiliza la función ρ, la cual considera una fracción de la luz incidente con dirección $\overrightarrow{l_i}$ (llamada reflectancia semiesférica direccional o *directional hemispherical reflectance*), se integra sobre todas las direcciones $\overrightarrow{l_r}$ salientes.

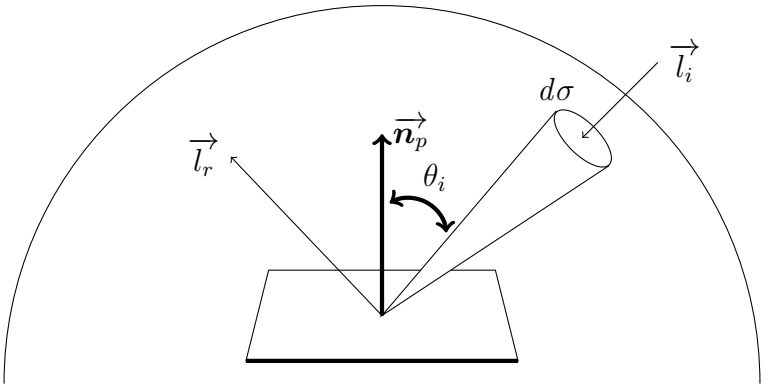


Figura 3.5: Se muestra un diagrama del modelo BRDF.

$$s(\overrightarrow{l_i}) = \int_{\forall \overrightarrow{l_r}} \rho(\overrightarrow{l_i}, \overrightarrow{l_r}) \cos\theta_r \, d\sigma_r. \tag{3.6}$$

En el caso de una superficie lambertiana, la cual tiene un brillo constante, la función ρ es igual a una constante lo que significa que el reflejo es igual para todos los ángulos y el brillo va a ser proporcional a la irradiancia.

Con el modelo de BRDF se puede describir el resplandor de una superficie en términos del resplandor o brillo que llega de diferentes direcciones. En graficación por computadora se puede reescribir la función BRDF exclusivamente en términos del resplandor si se toma una pequeña parte de la fuente de luz $\overrightarrow{l_i}$ con un ángulo sólido $\Delta\sigma$ con intensidad L_i y medimos el brillo reflejado en dirección $\overrightarrow{l_r}$ debido a la pequeña porción de la luz, ver Figura 3.5.

La irradiancia es $H = L_i \cos\theta_i \Delta\sigma_i$ debido a la pequeña porción de luz por lo que la BRDF queda como $\rho = \frac{L_o}{L_i \cos\theta_i \Delta\sigma_i}$. Con esto podemos saber el brillo que proviene de una dirección $\overrightarrow{l_i}$, acomodando los términos se tiene:

$$\Delta L_o = \rho(\overrightarrow{l_i}, \overrightarrow{l_r}) L_i \cos\theta_i \Delta\sigma_i. \tag{3.7}$$

Integrando todas las luces que provienen de todas las direcciones $L_i(\overrightarrow{l_i})$ entonces se obtiene:

$$L_s(\overrightarrow{l_r}) = \int_{\forall \overrightarrow{l_i}} \rho(\overrightarrow{l_i}, \overrightarrow{l_r}) L_f(\overrightarrow{l_i}) \cos\theta_i \, d\sigma_i. \tag{3.8}$$

Esta ecuación se conoce como la ecuación de *rendering* (*rendering equation*) [17], donde L_f representa un campo de radiación.

3.2.1. Modelo de Reflexión de Phong

El modelo (3.8) se considera una aproximación que permite representar comportamientos de la interacción luz-materia bastante sofisticados, como lo demuestran las imágenes de las Figuras 3.6 y 3.8; a las técnicas de graficación por computadora que utilizan este modelo se les conoce como fotorrealismo. Sin embargo, la generación de imágenes usando el modelo (3.8) requiere mucho tiempo computacional y es por ello que en muchas aplicaciones utilizan un modelo más simple propuesto por Biu Tuong Phong [19, 20]. Este modelo es una aproximación muy simple al de (3.8) sobre superficies lambertianas y se basa en la combinación lineal de tres componentes: ambiental, difusa y especular. En este modelo la componente ambiental modela la luz que ha sido reflejada muchas veces y aparenta ser emitida uniformemente en todas direcciones en todos los puntos de la escena. La componente difusa representa la reflexión omnidireccional. Finalmente, la componente especular modela el brillo de la superficie y representa la reflexión alrededor de una dirección preferente. En este modelo la intensidad luminosa en un punto $\boldsymbol{p}$ sobre la superficie iluminada está dada por

$$\boldsymbol{i}(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}) = \boldsymbol{i_a}\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right) + \boldsymbol{i_d}\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right) + \boldsymbol{i_e}\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right), \tag{3.9}$$

donde $\boldsymbol{i_a}$ es la componente ambiental, $\boldsymbol{i_d}$ es la componente difusa y $\boldsymbol{i_e}$ la componente especular. Para calcular las componentes difusa y especular se utiliza el modelo de reflexión discutido al principio de esta sección.

La componente difusa modela una superficie dura que dispersa la luz en todas direcciones. La intensidad de la luz que se refleja depende del ángulo entre la normal de la superficie y el vector que representa la luz incidente sin importar la posición del observador y se calcula como sigue

Figura 3.6: Ejemplo de fotorrealismo: Imagen de la animación "*The Light of Mies van der Rohe*" que demuestra como el *photon mapping* puede usarse para el cálculo de la iluminación global en un escena compleja. El sol es la única fuente de luz y casi toda la luz es iluminación indirecta [10].

$$\boldsymbol{i_d}\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right) = \ell_d k_d \left(\overrightarrow{\boldsymbol{l}_i} \cdot \overrightarrow{\boldsymbol{n}_p}\right), \tag{3.10}$$

donde ℓ_d representa la intensidad de la luz emitida, k_d es la propiedad reflectiva difusa del material.

Por otra parte, la componente especular es usada para modelar el brillo de la superficie. Cuando una superficie tiene brillo, la luz es reflejada de la superficie como un espejo. En un espejo, la luz reflejada es más fuerte en la dirección del reflejo perfecto.

La física de esta situación nos dice que para un reflejo perfecto, el ángulo incidente es igual al ángulo de reflexión y el vector es coplanar con la normal de la superficie. Para modelar la reflexión especular, se usa la siguiente ecuación

$$i_s\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right) = \ell_s k_s \left(\overrightarrow{\boldsymbol{l}_r} \cdot \boldsymbol{v}\right)^s, \tag{3.11}$$

donde $\boldsymbol{v}$ es un vector que representa al observador desde el punto $\boldsymbol{p}$, ℓ_s representa la intensidad de la luz emitida por una fuente de luz, k_s es la propiedad reflectiva

especular del material y s es la intensidad de la luz reflejada, ver Figura 3.7. El vector de reflexión $\overrightarrow{\boldsymbol{l}_r}$ puede ser calculado con base a los vectores $\overrightarrow{\boldsymbol{l}_i}$ y $\overrightarrow{\boldsymbol{n}_p}$ con la siguiente ecuación:

$$\overrightarrow{\boldsymbol{l}_r} = -\overrightarrow{\boldsymbol{l}_i} + 2\left(\overrightarrow{\boldsymbol{l}_i} \cdot \overrightarrow{\boldsymbol{n}_p}\right)\overrightarrow{\boldsymbol{n}_p}. \tag{3.12}$$

En la ecuación (3.11) el brillo máximo se obtiene cuando el observador está alineado con el vector $\overrightarrow{\boldsymbol{l}_r}$.

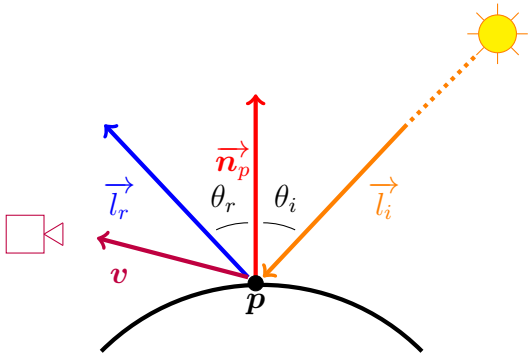


Figura 3.7: Modelo de Phong

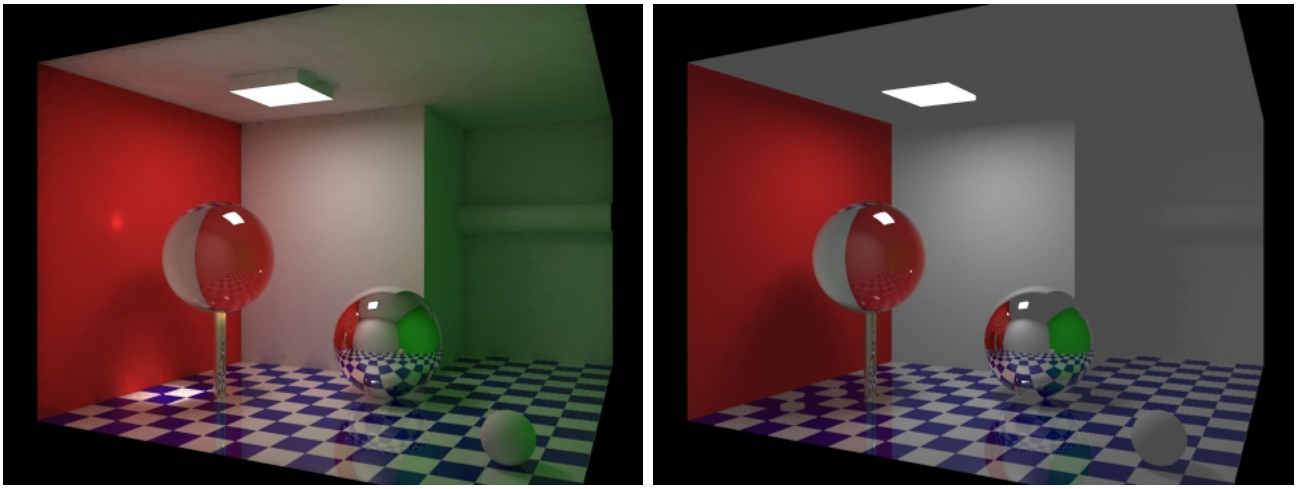
Por último, la componente ambiental representa la luz que ilumina todas las superficies por igual y refleja uniformemente a todas direcciones. Se utiliza principalmente para darle más brillo a las áreas obscuras de la escena. El modelo de Phong la calcula simplemente multiplicando la intensidad de la fuente de luz ℓ_a por la propiedad reflectiva ambiental k_a de la superficie como se muestra en la siguiente ecuación:

$$\boldsymbol{i}_a\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right) = \ell_a k_a \tag{3.13}$$

Combinando las ecuaciones (3.13), (3.10) y (3.11) el modelo de iluminación de Phong queda como sigue

$$\boldsymbol{i}\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right) = \ell_a k_a + \ell_d k_d \left(\overrightarrow{\boldsymbol{n}_p} \cdot \overrightarrow{\boldsymbol{l}_i}\right) + \ell_s k_s \left(\overrightarrow{\boldsymbol{l}_r} \cdot \boldsymbol{v}\right)^s. \tag{3.14}$$

La luz ambiental en el modelo de Phong es una aproximación burda a lo que sucede en la física del mundo porque la luz no se comporta de una forma constante.



(a) (b)

Figura 3.8: Diferencias entre (a) iluminación global [26] e (b) iluminación local [8].

La luz ambiental tiene más influencia en los lugares sombreados o donde las fuentes de luz no inciden directamente en las superficies. Esto no se ve muy realista y hace que la escena se vea muy plana, ver Figura 3.8 (a) , en donde se puede observar que la pared blanca del fondo se "contamina" con el color de las paredes laterales por la luz indirecta que a su vez nos da información de como es el cuarto, también se puede ver la refracción que hace la luz al atravesar la esfera más a la izquierda y que se proyecta sobre el piso y la pared. La esfera del centro proyecta una suave sombra la cual tiene la contribución de la pared verde para su iluminación. La esfera pequeña del lado derecho tiene un color de la luz reflejada en la pared verde. En (b) no se puede percibir la forma del cuarto completo, las esfera izquierda, a diferencia de (a), carece de refracción y la luz indirecta es nula en la escena.En el mundo real la cantidad de luz ambiental que alcanza un punto en su superficie depende del ambiente.

3.3. Propuesta para Iluminación Ambiental

Existen varios algoritmos que mejoran la falta de realismo en las imágenes producidas con el modelo de reflexión de Phong; la mayoría de ellos utilizan métodos que incorporan mejores modelos para la componente ambiental tales como *ray tracing* y

path tracing [25, 15], pero la mayoría de estos modelos se consideran como parte de la graficación fotorrealista y su costo de procesamiento es muy grande. En este trabajo se busca una aproximación diferente en la que consideramos los avances tanto en hardware como en estándares de graficación por computadora como OpenGL® para mejorar la componente ambiental sin un uso computacional excesivo.

3.3.1. Antecedentes y trabajo relacionado

La preocupación por mejorar la componente ambiental no es reciente y desde hace muchos años se han realizado esfuerzos para mejorar el modelo para dicha componente. Uno de esos primeros modelos es ***An ambient light illumination model*** [22] el cual buscaba más precisión en la luz ambiental que la que da Phong, pero sin utilizar algún modelo de iluminación global. La **Oclusión Ambiental** (*Ambient Occlusion* o AO) es una técnica que aproxima el efecto de la luz ambiental por medio de calcular la accesibilidad de cada punto de la superficie y asignar la atenuación del modelo de sombreado con base en dicha accesibilidad. En este contexto, la *accesibilidad* es una medida de la cantidad de luz ambiental que puede alcanzar un punto de la superficie sin que sea opacada por las superficies cercanas [1]. Otra técnica que es similar a la oclusión ambiental es la oclusión ambiental en el espacio de la imagen (***Screen-space Ambient Occlusion***) cuya idea básica es utilizar la misma técnica que usa AO pero compara la accesibilidad con los valores que se han almacenado en el *buffer* de profundidad o haciendo la comparación con las profundidades cercanas al punto que se evalúa [28]. Los autores de [3] propusieron otra técnica conocida como ***Ambient Aperture Lighting*** la cual trabaja con un modelo de sombreado que aproxima la luz incidental proveniente de áreas de las fuentes de luz dinámicas. Por otro lado, la técnica de ***Ambient Occlussion Fields*** [9] hace un precálculo de los campos (*fields*) en los espacios que un objeto puede ocasionar oclusión a otras regiones de la escena. Modelando terrenos en 3D uno de los algoritmos usados es el de ***Shadow Graphs and 3D Texture Reconstruction*** que obtiene superficies usando mapa de alturas e incorporando sombras *[29]*.

Una técnica más reciente es la ***Volumetric Obscurance***, propuesta en [**?**], la cual modela la luz ambiental como una integral volumétrica 3D, calculada usando *hardware* gráfico, en escenas que son dinámicas.

Otras aportaciones dentro del área de fotorrealismo han sido ***Importance driven path tracing using photon map*** [12] y ***Photon maps in bidirectional Monte Carlo ray tracing of complex objects*** [11], respectivamente.

3.3.2. Propuesta

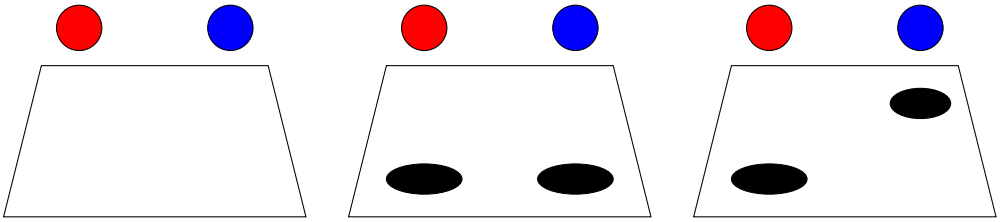
Este trabajo se enfoca en las superficies que no reciben luz ambiental directamente; en otras palabras, las superficies que tienen sombras proyectadas. Dentro la graficación proyectiva hay ciertas técnicas para crear sombras en la escena las cuales incrementan el realismo y nos dan más pistas de tridimensionalidad. Para la generación de las sombras se utiliza el *buffer* de profundidad [24] y se toma en cuenta la luz ambiental en la formación de esas sombras.

3.3.2.1. Sombras

Como se mencionó, una de las formas para aumentar el realismo en las escenas es el uso de sombras: sin sombras se puede malinterpretar la posición relativa de los objetos, como se muestra en la Figura 3.9, y producir la percepción de iluminación irreal. Por lo tanto, las sombras son importantes claves visuales para las escenas reales y generarlas de manera eficiente en programas que requieren ejecutarse en tiempo real puede ser desafiante.

Una de las técnicas más populares en graficación por computadora proyectiva para crear sombras en tiempo real es el algoritmo llamado "*shadow mapping*" o mapa de sombras [27] y este será la base para este trabajo.

Shadow mapping El concepto del mapa de sombras es muy similar al de mapa de distancias o *buffer* de distancias, el cual es un arreglo que almacena las distancias más cercanas de las regiones de los objetos de la escena al plano de la cámara por lo que permite decidir las porciones visibles durante el proceso de *rendering*. En el caso de



(a) Las posiciones de los objetos se desconoce.

(b) Las posiciones de los objetos se conocen y se infiere que los objetos son del mismo tamaño.

(c) Las posiciones de los objetos se conoce pero se infiere que la pelota azul está más atrás de la roja y es de mayor tamaño

Figura 3.9: Al incorporar pistas de tridimensionalidad la percepción de las posiciones de los objetos mejora.

un mapa de sombras la información se genera desde el punto de vista de una fuente de iluminación por lo que es posible decidir que partes serán iluminadas y cuales estarán en una sombra.

El algoritmo para generar sombras con base en mapas de sombras [27, 6, 24] está dividido en dos partes, la primera es obtener una imagen de la escena desde la fuente de luz donde se van a guardar los valores de profundidad. La segunda parte es hacer el *render* normal de la escena, comparando cada fragmento con un mapa de sombras para saber que parte está en la sombra.

Para crear un mapa de sombras hay que preparar la matriz de vista, ver la subsección 2.1.2.1, como si la cámara estuviera en la posición de una fuente de iluminación y se encontrase orientada hacia los objetos a los cuales se les desea generar sombra. También se prepara la matriz de proyección, ver la subsección 2.1.2.2, de forma tal que el *frustrum* incluya tanto a los objetos que puedan hacer sombras como a las regiones sobre las que se proyectarán las mismas. Con esta información se realiza el primer *render* de la escena y la información que se almacena en el *buffer* de profundidad se utiliza como mapa de sombras para la fuente de iluminación, típicamente en una textura. Este mapa de distancias se puede interpretar como la distancia que hay desde la fuente de iluminación hacia las diferentes ubicaciones de las superficies.

Una vez que se ha generado el mapa de sombras se prosigue a restablecer la matriz de vista con los parámetros de la cámara y se realiza el *render* final de la

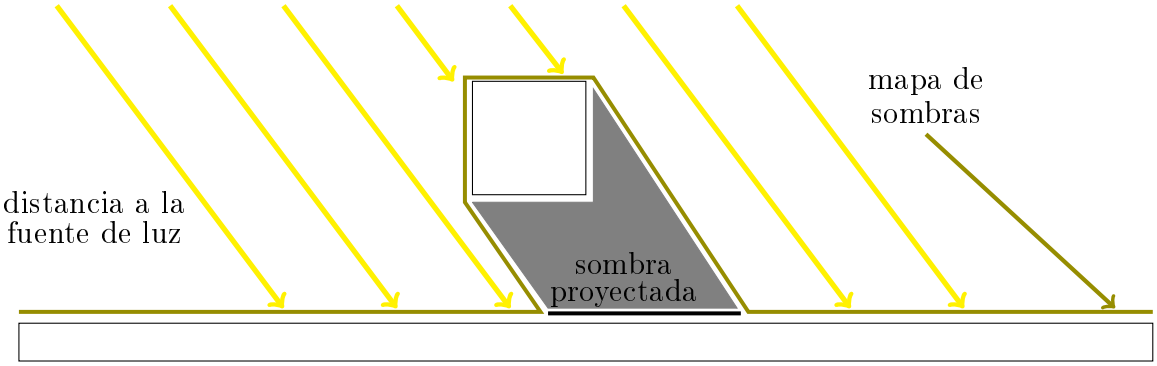


Figura 3.10: Mapa de sombras, en la imagen se muestra como hay luz que incide sobre la escena, este valor es lo que se toma como distancia a la fuente de luz de la superficie y nos ayuda a saber si un punto está en la sombra de un objeto o no.

escena. En este proceso es que se usa el *shader* de fragmentos para asignar color a los fragmentos de la imagen en formación pero tomando en cuenta el mapa de sombras. La posición del fragmento visto desde la cámara es convertida en coordenadas en la proyección de la fuente de luz. El resultado es interpolado con el fin de obtener coordenadas válidas para el mapa de sombras y compararlas con la profundidad del fragmento. Si la profundidad del fragmento es mayor a la que se encuentra en el mapa de sombras entonces hay una superficie que se encuentra entre el fragmento y la fuente luz como se muestra en al Figura 3.10. En otro caso, el fragmento tiene una vista clara a la fuente de luz por lo tanto se hace el sombreado normalmente. Uno de los aspectos importantes es la conversión de las coordenadas 3D de los fragmentos a las coordenadas apropiadas para la búsqueda en el mapa de sombras. Como el mapa de sombras es una textura en 2D, entonces necesitamos coordenadas en el rango de cero a uno para puntos que están dentro del *frustrum* de la luz. La matriz de vista de la fuente de luz toma puntos con coordenadas en el mundo y los convierte al sistema de coordenadas de la luz. La matriz de proyección de luz transforma los puntos que están en el *frustrum* de la luz a coordenadas homogéneas de recorte *(homogeneous clip coordinates)* donde estas coordenadas son mapeadas para que el mapa de sombras pueda acceder a las profundidades guardadas[16, 5, 28].

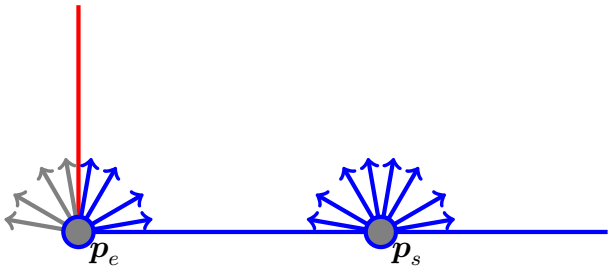


Figura 3.11: Oclusión Ambiental

3.3.3. Metodología

Como se menciona anteriormente, se propone un cambio en la ecuación de iluminación de Phong (3.9). Este cambio es particular porque propone mejorar la luz indirecta en las partes de la escena que están siendo ocultadas por una sombra.

Para ello se usa la oclusión ambiental [1, 28] calculando un factor de accesibilidad trazando un conjunto de rayos que se originan en el punto de la superficie que se está trabajando. Estos rayos son distribuidos dentro de una semiesfera cuyo origen se encuentra en dicho punto de la superficie. El factor de accesibilidad es proporcional a la porción de la semiesfera que es intersectada por otra superficie. En la Figura 3.11 la superficie roja bloquea la mitad de los rayos que se emiten desde el punto $\boldsymbol{p}_e$ localizado en la esquina, mientras que ninguno de los rayos que se emiten desde un punto $\boldsymbol{p}_s$ sobre la superficie azul es bloqueado.

Utilizando este principio nos enfocamos en los rayos que están dentro de las sombras y no en aquellos que se encuentran sobre la superficie. Las sombras están guardadas en unas texturas de profundidades [24] y con el *shader* de fragmentos se accede a estas texturas para saber si un fragmento $\boldsymbol{p}_f$ (prepíxel) está sombreado o no. Sabiendo qué fragmentos se encuentran en la sombra se procede a calcular la siguiente relación

$$\xi = \frac{1}{N_{S_{\boldsymbol{p},r}}} \sum \text{Shadow}\left(\boldsymbol{s}_{\boldsymbol{p},r}\right), \tag{3.15}$$

donde $\boldsymbol{s}_{\boldsymbol{p},r}$ es un punto sobre la semiesfera con centro en $\boldsymbol{p}$ y radio r, $N_{S_{\boldsymbol{p},r}}$ es el número de puntos muestreados en la semiesfera, véase en la Figura 3.12, y $\text{Shadow}\left(\boldsymbol{s}_{\boldsymbol{p},r}\right)$:

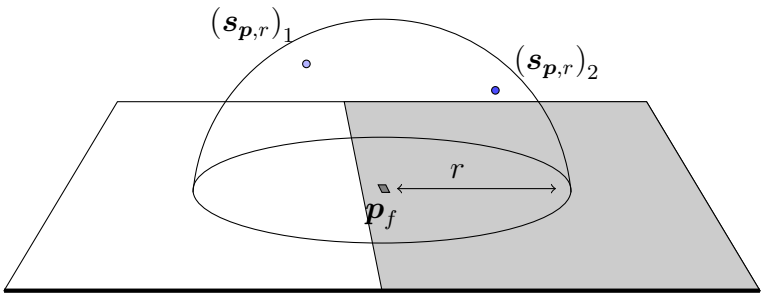


Figura 3.12: La superficie de la semiesfera, con radio r y centro en $\boldsymbol{p}_f$, se muestra para obtener puntos con posición $(\boldsymbol{s}_{\boldsymbol{p},r})_n$, para $1 \leq n \leq N_{S_{\boldsymbol{p},r}}$.

$\mathbb{R}^3 \to \mathbb{R}^+$ es una función que indica si el punto $\boldsymbol{s}_{\boldsymbol{p},r}$ está dentro o fuera de la sombra. Para calcular ξ, el factor de sombra, se utiliza una semiesfera de radio r sobre el fragmento $\boldsymbol{p}_f$ que, debido a como se construyó el programa, se encuentra en la parte sombreada de la textura; vale la pena hacer notar que no toda la esfera puede estar sombreada. Por cuestiones de velocidad se muestrea la superficie de la semiesfera con $N_{S_{\boldsymbol{p},r}}$ puntos, ver Figura 3.12.

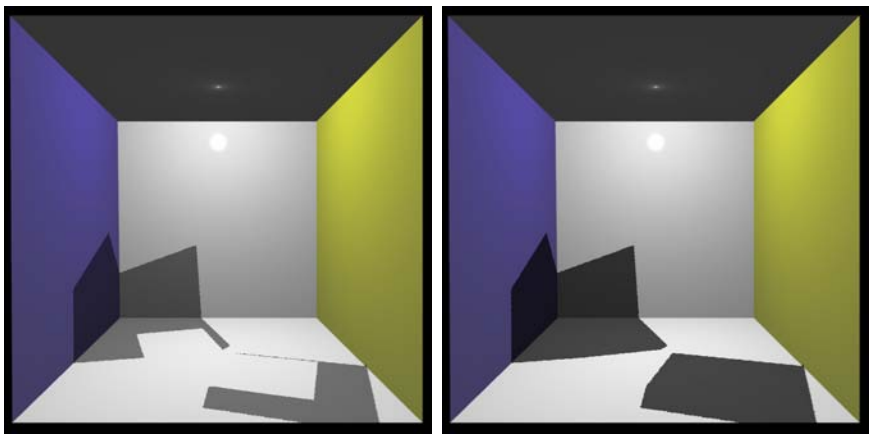
Finalmente, se incorpora el factor de sombra ξ a la ecuación (3.9) de forma que la intensidad queda formulada como

$$\boldsymbol{i}(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}) = \xi \boldsymbol{i_a}\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right) + \xi \boldsymbol{i_d}\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right) + \boldsymbol{i_e}\left(\boldsymbol{p}, \overrightarrow{\boldsymbol{l}_i}\right) \tag{3.16}$$

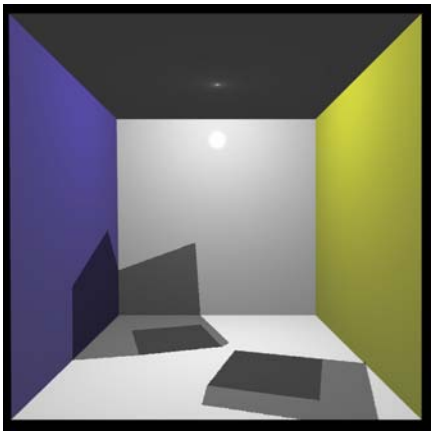
donde el factor de sombra ξ se encuentra en el rango $[0, 1]$.

Al usar el mapa de sombras tenemos una desventaja, la geometría de los objetos que hacen las sombras se desconoce, y una de las formas que se encontró para minimizar esta desventaja fue el uso de dos mapas de sombras pero el *render* de las escenas difieren en que caras del objeto, frontales o posteriores, estamos tomando para hacer el *render*. Para la implementación de los mapas de sombras aprovechamos de la capacidad que posee OpenGL® para decidir si utiliza caras de polígonos frontales o posteriores para renderizar la imagen, un proceso conocido como *Culling*. El *Culling* se usa para saber que caras de un modelo se quieren pintar en la imagen, esto se hace tomando la normal de las superficies del modelo y comparándolas con la posición y dirección que tiene la cámara. Este proceso nos ahorra tiempo de procesamiento al

saber que caras están viendo a la cámara para pintarlas en la imagen. El *Culling* que se maneja en este trabajo consiste en hacer un render con la caras posteriores, Figura 3.13a, de los modelos que proyectan la sombra al igual que un render de caras frontales, Figura 3.13b, finalmente hacer una mezcla de ambos resultados como se muestra en la Figura 3.13 (c) y podemos calcular mejor el factor de sombra ξ, porque con esto podemos saber que fragmentos están más al centro de la sombra o donde pueden intersectar con alguna superficie, haciendo una reducción proporcional en el factor se sombra ξ. En nuestro modelo, el factor de sombra logra el efecto de que la sombra sea más intensa en el centro y se difumine hacia las orillas dependiendo de la forma de los objetos.



(a) Front-culling (b) Back-culling



(c) Combinación

Figura 3.13: Ejemplos de *renderings* de sombras usando (a) *front-culling* (caras frontales se omiten), (b) *back-culling* (caras posteriores se omiten) y (c) la combinación del resultado *(front-* y *back-culling).* Las sombras son generadas por cubos,

Capítulo 4

Experimentos y Resultados

En este capítulo se presentan los experimentos que se realizaron con nuestro algoritmo para mejorar la componente de luz ambiental. Para todos nuestros experimentos utilizamos la escena conocida como Caja de Cornell (*Cornell Box*) [4]. Esta escena se usa comúnmente en la comunidad de graficación por computadora para determinar la exactitud del resultado de algún algoritmo de *rendering* comparándolo contra una foto tomada de la misma escena. La caja de Cornell tiene las siguientes características: la escena se compone de un cuarto que tiene dentro dos cubos de color blanco de diferente tamaño y una fuente de luz en el techo. El cuarto tiene dos paredes de color blanco y dos de diferente color, comúnmente verde y rojo, véase Figura 4.1. El material de las paredes como el de los cubos tiene propiedades lambertianas. Las propiedades de la cámara se muestran en la Tabla 4.1.

4.1. Implementación

El modelo propuesto en el Capítulo 3 se implementó usando el lenguaje de programación C/C++ con una biblioteca que implementa OpenGL® 4.2 con su lenguaje de *shading* GLSL 4.1. El programa genera una escena siguiendo las características de

Posición ($\boldsymbol{p}^c$)	Dirección ($\boldsymbol{l}^c$)	Dirección Arriba ($\boldsymbol{t}^c$)	Distancia Focal (f)	Ancho (w)	Largo (h)
$(278, 273, -800)$	$(0, 0, 1)$	$(0, 1, 0)$	0.035	0.025	0.025

Cuadro 4.1: Propiedades de la Cámara para la escena de la Caja de Cornell.

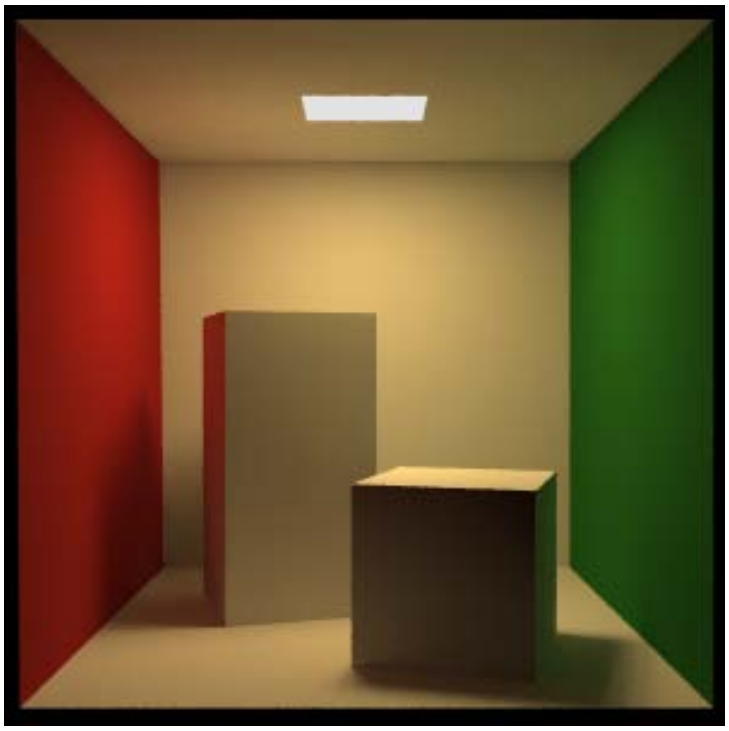


Figura 4.1: Caja de Cornell rendereada utilizando la técnica de *ray-tracing*, la cual produce imágenes de alta calidad, con el programa POV-Ray [14].

la Caja de Cornell, ver Figura 4.2.

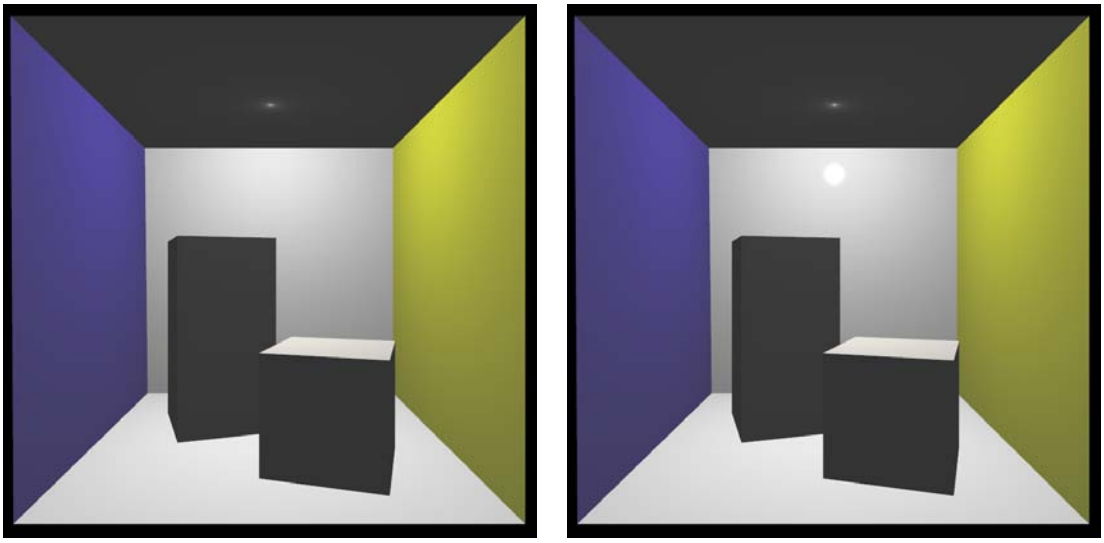
El el programa se ejecutó en una computadora con un procesador Intel® Core i5 a 2.50GHz (con cuatro *cores*), con 4GB de memoria RAM, con una tarjeta gráfica GeForce 630M, bajo el sistema operativo Xubuntu 13.10.

4.1.1. Resultados

Como referencia para nuestros experimentos se utilizaron *rendering* de la Caja de Cornell utilizando el modelo de Iluminación de Phong con y sin componente especular, ver Figura 4.2, y también con sombras, ver Figura 4.3.

En la Figura 4.3 se muestran escenas rendereadas de la Caja de Cornell usando sombras y el modelo de Iluminación de Phong donde 4.3a se hizo con una velocidad entre 172 y 178 fps y para 4.3b el *rendering* fue de entre 172 y 178 fps.

Para nuestros experimentos utilizamos mapas de sombras con diferentes dimensiones al igual que diferentes vecindades en (3.15) para obtener el factor de som-



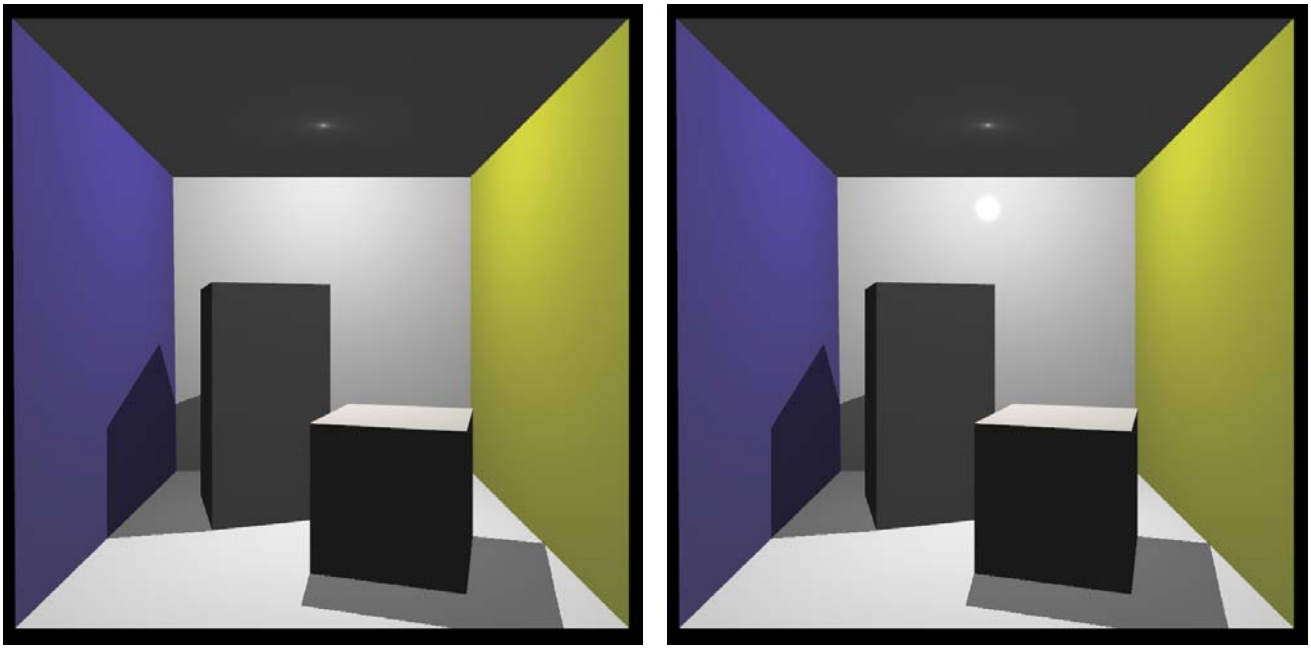
(a) (b)

Figura 4.2: Escena de la Caja de Cornell usando (a) el modelo de iluminación de Phong con únicamente componentes difusa y ambiental donde el *render* tiene una velocidad de 203 a 207 fps(*frames per second*) y (b) el modelo completo de iluminación de Phong. Cada una de las imagen es de 800×800 píxeles y fueron generadas a una velocidad de 109 a 195 fps.

bra. Para describir la posición de un punto $\boldsymbol{s}_{\boldsymbol{p},r}$ localizado sobre la semiesfera se utilizaron coordenadas esféricas con las siguientes identidades $s_{\boldsymbol{p},r,1} = r \sin\varphi \cos\theta$, $s_{\boldsymbol{p},r,2} = r \sin\varphi \sin\theta$ y $s_{\boldsymbol{p},r,3} = r \cos\theta$, ver Figura 4.4.

Número de Muestras $N_{S_{\boldsymbol{p},r}}$	fps			ms		
	512×512	1024×1024	2048×2048	512×512	1024×1024	2048×2048
2592	0.87	0.7	0.5	1000	1000	1000
648	3.38	3	2.2	250	250	333
162	13	12	9	86	76	100
72	30	26	19	33	37	50
36	44	39	29	40	25	33

Cuadro 4.2: Tiempos en cuadros por segundo (fps) y en segundos necesarios para generar imágenes cambiando el número de puntos sobre la superficie y con diferentes tamaños de texturas. Para cada textura se usaron diferentes radios y se reporta el promedio.



(a) (b)

Figura 4.3: (a) Utilizó iluminación de Phong sin componente especular y para las sombras se utilizó un mapa de sombras con dimensiones 2048 × 2048 píxeles, (b) Imagen generada con el modelo de iluminación de Phong completo y con sombras utilizando un mapa de sombras de 2048 × 2048 píxeles, .

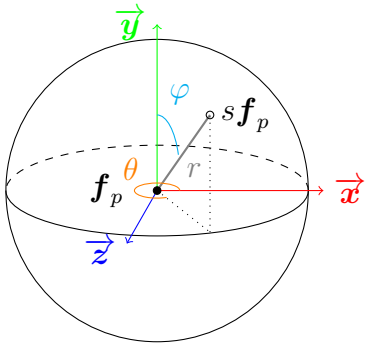


Figura 4.4: Esquema que representa las coordenadas esféricas para un punto $\boldsymbol{s}_{\boldsymbol{p}_f,r}$ sobre una semiesfera con radio r centrada sobre un punto $\boldsymbol{p}_f$.

Las Figuras 4.5, 4.6 y 4.7 muestran los resultados de *renderings* de la Caja de Cornell utilizando el modelo de iluminación (3.16) sin componente especular con diferentes dimensiones para el mapa de sombras y varios radios de la semiesfera al igual que diferente número de puntos muestreados sobre la superficie. En la Tabla 4.2 se presentan los tiempos que fueron necesarios para generar dichas imágenes. También se puede notar un difuminado en las orillas de la sombra hacia el centro, este difuminado simula la penumbra generada por la fuente de luz en la escena.

Para hacer una comparación entre los tiempos que se obtienen del modelo local contra un modelo fotorrealista se realizaron varios renderings utilizando un programa hecho en C/C++ que implementó un método llamado *unbiased Monte Carlo pathtracing* [2]. El programa tiene un *macro* con OpenMP para hacer el *render* más rápido. Los resultados se observan en la Tabla 4.3.

La Figura 4.5 se muestran los resultados de los *renderings* utilizando el modelo modificado de iluminación de Phong (3.16) sin componente especular, con una semiesfera de radio igual a $30u$ y con mapas de sombras de dimensiones 512×512 píxeles. Cada figura tiene diferentes números de muestras. En este caso, el perímetro de la sombra tiene una forma de escalera y se debe al tamaño de la textura.

En la Figura 4.6 se muestran los resultados de los *renderings* utilizando el modelo modificado de iluminación de Phong (3.16) sin componente especular, con mapas de sombras de dimensiones 1024×1024 píxeles. Cada figura tiene diferentes números de muestras haciendo que el difuminado en la sombra sea más suave. Apenas se alcanza ver que el perímetro de las sombras que tienen una forma de escalera, se debe a que el mapa de sombras tiene más resolución.

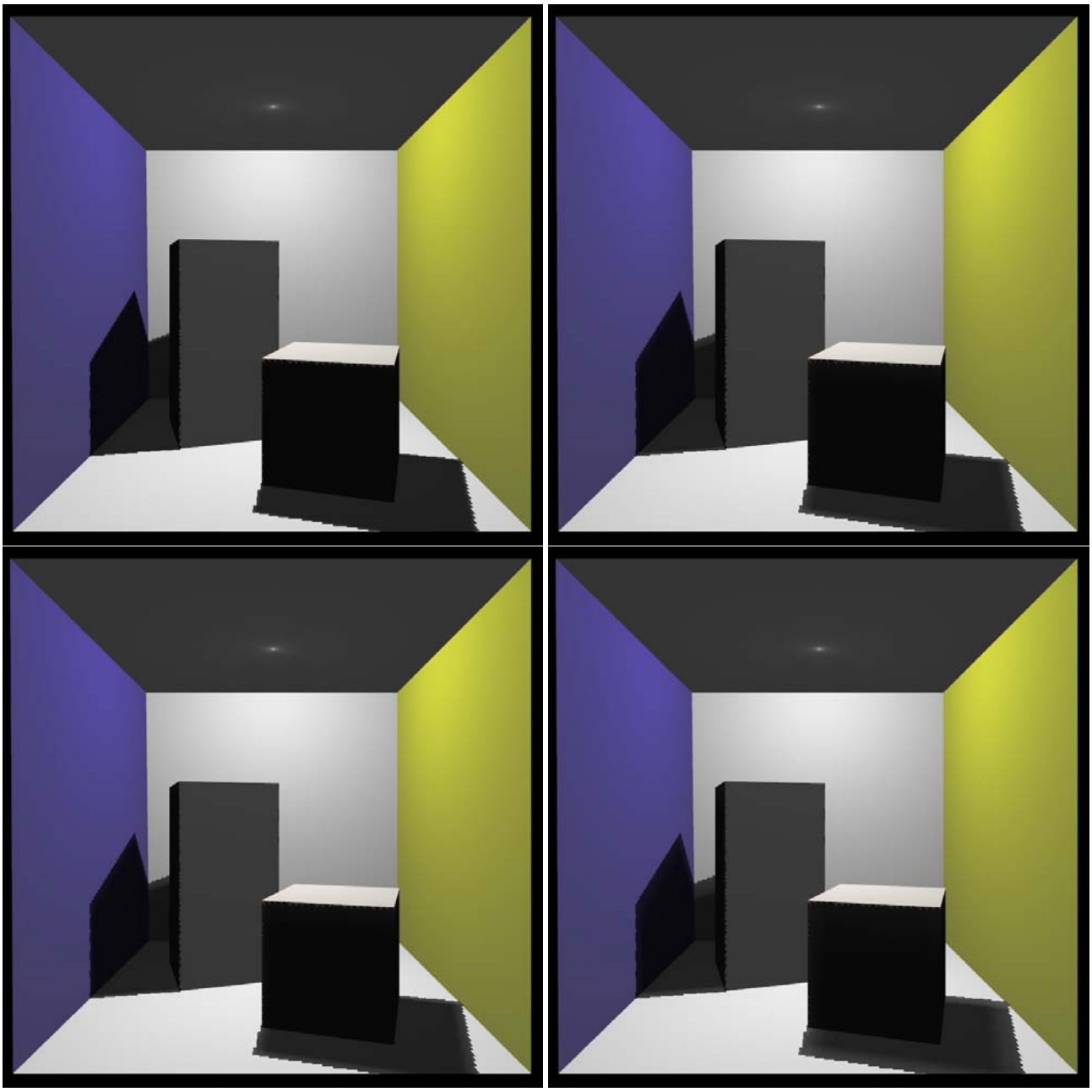


Figura 4.5: Semiesfera de radio igual a $30u$ y mapas de sombra de dimensiones 512×512 píxeles.

En la Figura 4.7 resultados de los *renderings* utilizando el modelo modificado de iluminación de Phong (3.16) sin componente especular, con mapas de sombras de dimensiones 2048×2048 píxeles. Cada figura tiene diferentes números de muestras haciendo que el difuminado sea mucho más suave. Para estas imágenes casi no se distingue los artefactos en el perímetro de la sombra.

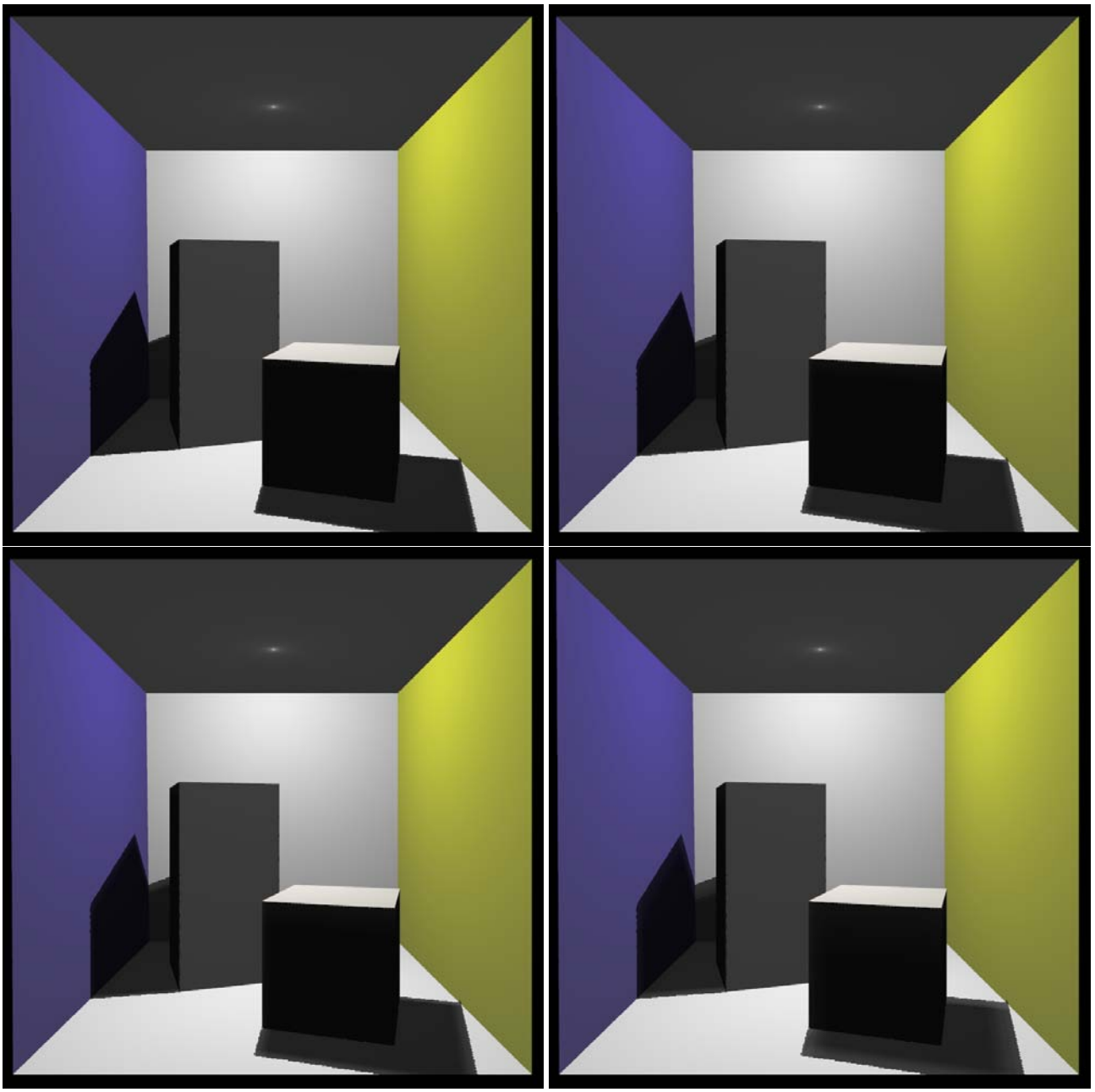


Figura 4.6: Semiesfera de radio igual a $30u$ y con mapas de sombras de dimensiones 1024×1024 píxeles.

Las imágenes que se presentan en la Figura 4.8 fueron rendereadas por el algoritmo mencionado en [2], cada imagen está dividida en el número de muestras que se necesitaron para el cálculo del algoritmo. El programa utiliza esferas en lugar de cajas porque es sencillo calcular las intersecciones de una rayo con una esfera analíticamente que con un plano delimitado. Las paredes como el techo y el piso son esferas

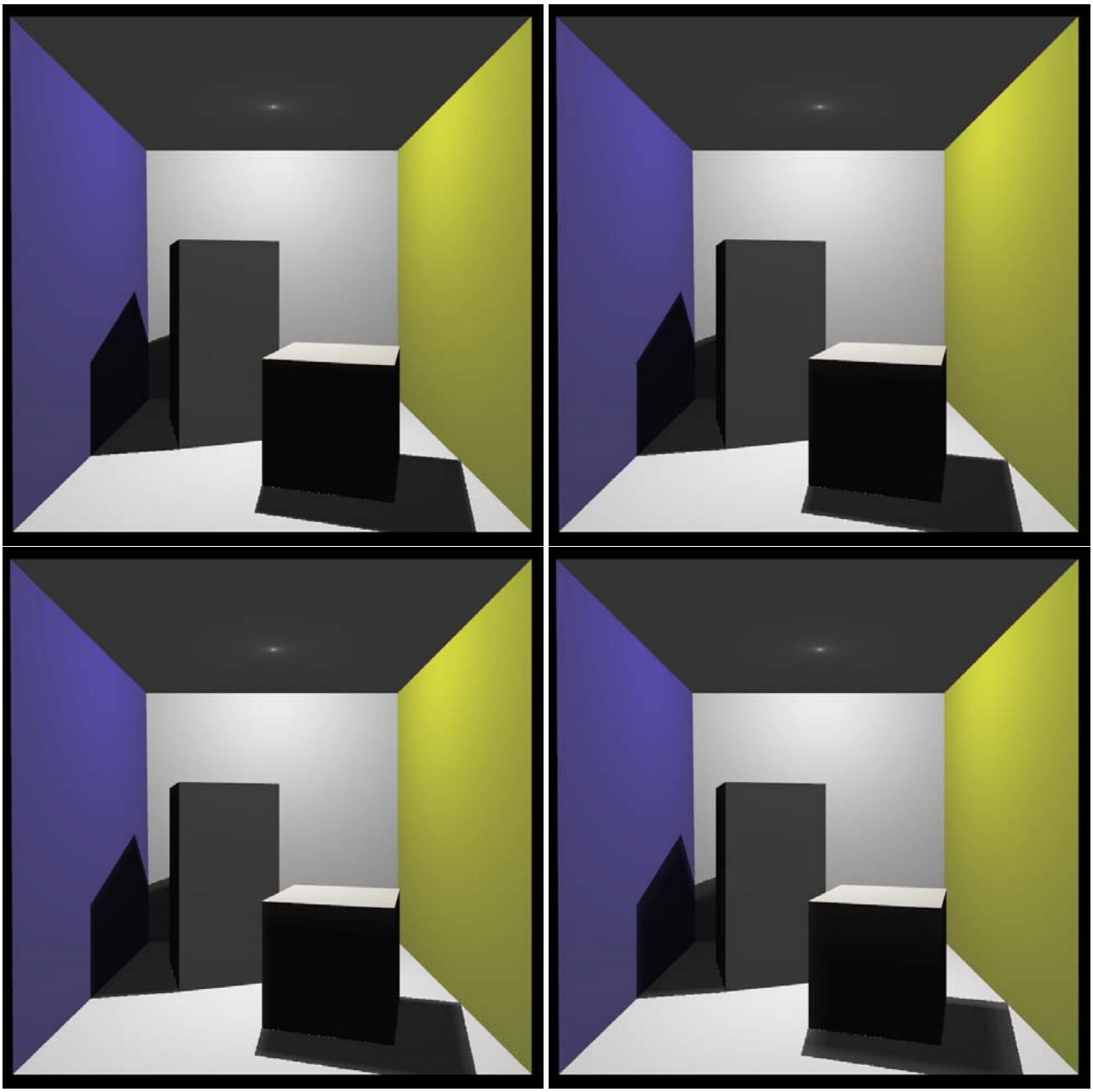
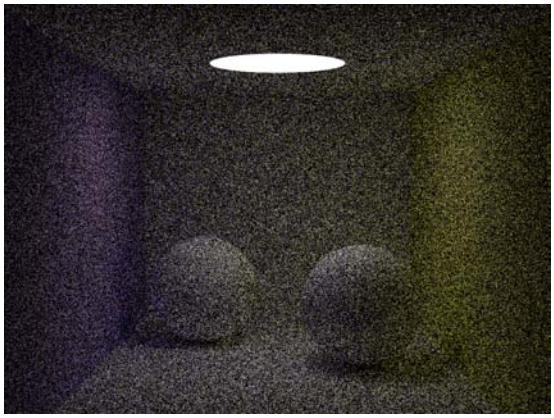
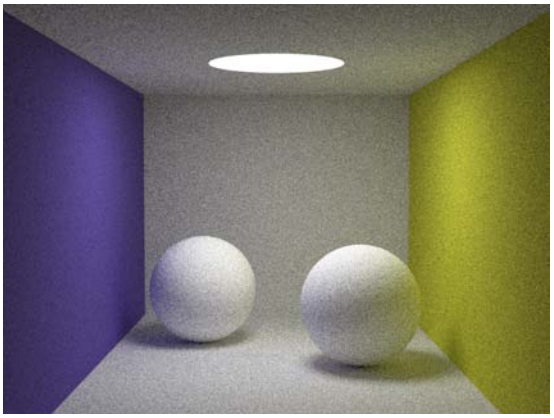


Figura 4.7: Semiesfera de radio igual a $30u$ y con mapas de sombras de dimensiones 2048×2048 píxeles.

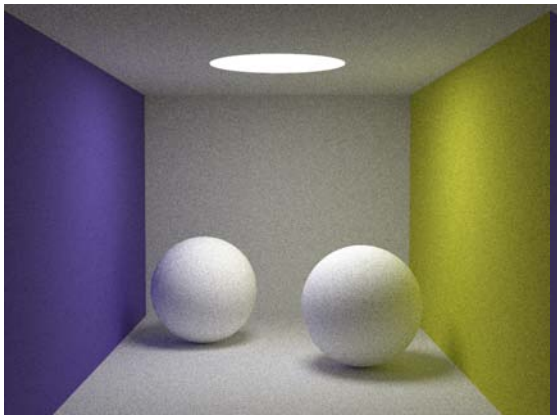
deformadas al igual que la luz.



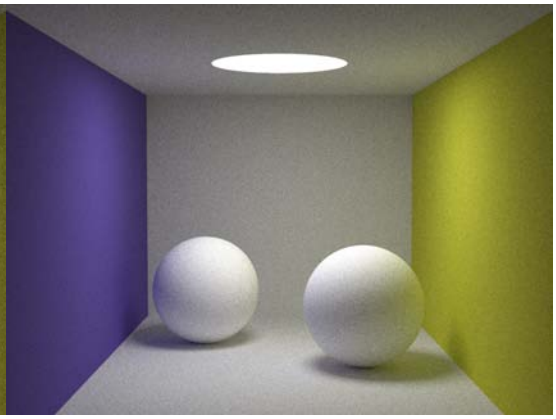
(a) 8 spp



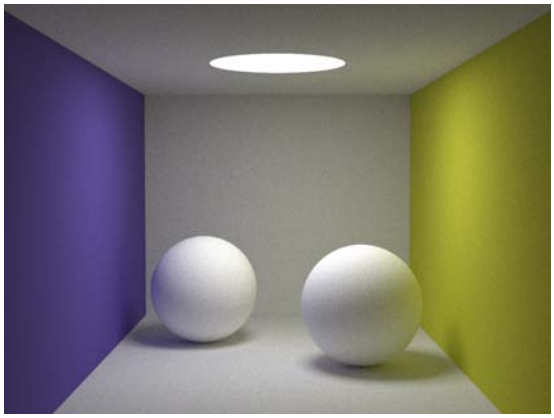
(b) 248 spp



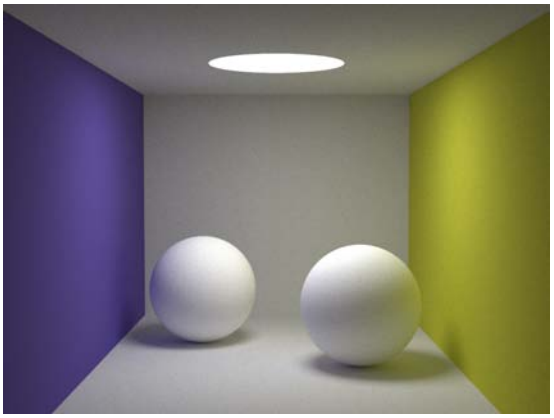
(c) 500spp



(d) 1000 spp



(e) 2000 spp



(f) 4000 spp

Figura 4.8: spp (*samples per pixel*) son las muestras de los fotones que se envían por píxel

Sample per Pixel	Tiempo usando CPU	Tiempo usando OpenMP (4 *cores*)
8	0m55s	0m6.2s
248	29m13s	3m51s
500	54m43s	7m15s
1000	115m44s	14m50s
2000	227m19s	31m13s
4000	449m28s	57m54s

Cuadro 4.3: Cuadro que muestra el tiempo para hacer el render de una imagen con un método fotorrealista. Entre más muestras se hagan, mayor es el tiempo de *render* pero mejores resultados en la calidad de las imágenes se pueden observar. Las imágenes que se generaron con este proceso toman mucho tiempo lo cual hace que el algoritmo no se use para aplicaciones en tiempo real.

Capítulo 5

Conclusiones

Para aumentar el realismo en las escenas es necesario incluir pistas de tridimensionalidad donde las sombras tienen un aspecto normal. Usando la técnica de *shadow mapping* podemos obtener resultados buenos pero no tan vistosos como los métodos usados en iluminación global. Esto sucede así porque la luz ambiental o indirecta en el modelo local incide en los objetos no proviene de la luz que rebotó por la escena, proviene de un cálculo aproximado de cuanta luz ambiental llega a diferentes partes de la escena.

El algoritmo que se propone hace que la luz ambiental tenga un factor de sombra haciendo que partes de la sombra sea más obscura por el hecho de que existe una geometría dentro de la sombra que disminuye la luz indirecta en esa zona. Al tener dos mapas de sombras de la escena, podemos obtener un poco más de información de la geometría de los objetos pero sin llegar a tener completa su geometría. Para ser una escena generada con un modelo de iluminación local hace una buena aproximación en las áreas sombreadas. Si tenemos la geometría de los objetos podemos saber con certeza la posición de los objetos, distancia entre objetos y demás, pero usa mucha memoria. Al usar dos mapas de sombra la memoria utilizada para la generación de la imagen es menor y tenemos más pistas de los objetos en la escena.

Los mejores resultados se dieron cuando las muestras sobre la esfera fueron menores ver Tabla 4.2, esto es algo que se esperaba porque hay menos datos que procesar. Aún así, al tener muchas muestras, más de dos mil, sobre la superficie de la semiesfera, sigue siendo mucho más rápido que un modelo de iluminación global. Entre

más muestras tengamos en la vecindad podemos obtener un mejor difuminado sobre la sombra. Por otra parte el radio de la esfera nos indica la penumbra que tiene la sombra, la cual hay que calcular mejor con otro método así podemos saber que radio de la esfera hay que seleccionar para lograr una mejor exactitud en la escena.

Por último, la velocidad promedio para que un programa se considere que corre en tiempo real es de 30 fps (*frames per second*) y en su mejor caso es en 60 fps (es la velocidad que utilizan los videojuegos). En este trabajo el número de muestras para alcanzar la velocidad de 30 fps fue de 36 muestras en cualquier resolución de los mapas de sombra. Si se quiere llegar a que sean 60 fps entonces el número de muestras sobre la semiesfera debe de ser menor a 20, perdiendo detalle en la penumbra de la sombra.

5.1. Trabajo futuro

Uno de los objetivos es encontrar mejores muestras de la vecindad, pude se utilizando alguna probabilidad o un método pseudoaleatorio. También se puede incluir pistas de reflejo de los objetos para materiales específicos en puntos específicos de la escena. También cambiar los tipos de las fuentes de luz e implementar más fuentes de luz dentro de la escena. Seguir aplicando los métodos de *antialiasing* para quitar artefactos que son provocados por el *shadow mapping*. Por último aplicarlo en escenas diferentes y hacer la comparación por píxel en escenas iguales usando el método que se describe en este trabajo contra un global.

Como se está utilizando OpenGL® se puede hacer una implementación de este algoritmo para web utilizando WebGL. Con esto podemos mostrar trabajos de forma remota que necesiten el *render* en tiempo real que quieran ser un poco más llamativos.

Bibliografía

[1] M. S. Alex Mendez-Feliu, "From obscurances to ambient occlussion: A survey," *Springer-Verlag*, 2008.

[2] K. Beason, "smallpt: Global illumination in 99 lines of c++." http://www.kevinbeason.com/smallpt/.

[3] P. V. S. Chistopher Oat, "Ambient aperture lighting," *ACM Transactions on Graphics*, 2007.

[4] K. E. T. Cindy M. Goral, "Modeling the interaction of light between diffuse surfaces," *Computer Graphics*, vol. 18, no. 3, 1984.

[5] J. D. Florian Kirsch, "Real-time soft shadows using a single light sample," *Journal of WSCG, Vol.11, No.1., ISSN 1213-6972*, 2003.

[6] L. B. Gael Guennebaud y M. Paulin, "Real-time soft shadow mapping by back-projection," *The Eurographics Association 2006.*, 2006.

[7] H. Gouraud, "Continuous shading of curved surfaces," *IEEE Transactions on Computers*, vol. C-20, pp. 87–93, June 1971.

[8] Gtanski, "Local illumination - image," December 2004.

[9] S. L. Janne Kontkanen, "Ambient occlussion fields," *ACM Transactions on Graphics*, 2005.

[10] H. Jensen. http://graphics.ucsd.edu/ henrik/images/global.html.

[11] H. Jensen y N. Christensen, "Photon maps in bidirectional monte carlo ray tracing of complex objects," *Computers Graphics*, vol. 19, p. 215 224, 1995.

[12] H. Jensen *et al.*, "Importance driven path tracing using the photon map," *Rendering Techniques*, vol. 95, p. 326 335, 1995.

[13] A. V. D. e. a. John f. Huges, *Computer Graphics Principles and Practice*. Addison-Wesley, 2014.

[14] K. Kivisalo, "Cornellbox.png," August 2003.

[15] E. Lafortune, "Mathematicals model and monte carlo algorithms for physically based rendering," *Phd Thesis*, 1996.

[16] C. R. Patrick Cozzi, *OpenGL Insights*. CRC Press, 2012.

[17] e. a. Peter Shirley, Steve Marschner, *Fundamentals of Computer Graphics*. CRC Press, 2009.

[18] M. Pharr y G. Humphreys, *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2010.

[19] B. T. Phong, "Computer generated pictures," *Communications of the ACM*, vol. 18, pp. 311–317, June 1975.

[20] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, p. 311 317, June 1975.

[21] R. J. Rost, *OpenGL Shading Language*. Addison-Wesley Professional, 2006.

[22] G. K. S. Zhukiv, A. Iones, *Rendering Techniques '98: Proceedings of the Eurographics Workshop in Vienna*. Springer, 1998.

[23] D. Shreiner, *OpenGL Programming Guide*. Addison-Wesley Professional, 2010.

[24] H.-P. S. Stefan Brabec, "Single sample soft shadows using depth maps," 2001.

[25] P. S. Strauss, "A realisitic lighting model for computer animators," *IEEE Computer Graphics and Applications*, vol. 10, pp. 56–64, November 1990.

[26] G. Tansk, "Global illumination image with kray renderer," December 2004.

[27] L. Williams, "Casting curved shadows on curved surfaces," *In Proceedings of ACM SIGGRAPH*, p. 270 274, 1978.

[28] D. Wolff, *OpenGL 4.0 Shading Language Cookbook*. Packt Publishing, 2011.

[29] J. T. C. Yizhou Yu, "Shadow graphs and 3d texture reconstruction," *International Journal of Computer Vision 62*, p. 35 60, 2005.