Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Computación Gráfica

GENERACIÓN DE ESCENAS USANDO PHOTON MAPPING EN WEBGL Y HTML5

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela por:

Br. Stephanie Piñero
Br. David Rojas
Tutor: Héctor Navarro

Agosto de 2015

AGRADECIMIENTOS

A mis padres y mi hermano por su apoyo incondicional. A mi abuela que aunque no está con nostros, siempre quiso ver a sus nietos graduados. A mi madrina Leonor, sin ti no estaría hoy donde estoy. A mi tutor Héctor Navarro, por brindarnos su ayuda, tiempo y paciencia. A todos mis compañeros y amigos que brindaron sus ideas (algunos sin saberlo), especialmente a Adriana, Javier y Humberto. Y a todos aquellos que ayudaron para que este trabajo de investigación pudiese completarse.

David Rojas

A mi madre, mi ejemplo a seguir, que ha estado para mi toda mi vida y me ha brindado todo su apoyo en mis decisiones. A mi padre, que siempre tiene una respuesta a mis preguntas. A Paola, mi hermana, a la que aunque nunca se lo he dicho admiro, por siempre empujarme a terminar las cosas y a hacerlas mejor. A nuestro tutor Héctor Navarro, por ayudarnos en todo este proceso. A Adri, David, Néstor, Jhonatan y Humberto, los que me ayudaron en muchas cosas en toda la carrera. A Fabi, que me soportó en todas mis crisis y me ha apoyado desde que la conozco. Y a todas las personas que de una u otra forma me han ayudado a estar aquí. Gracias.

Stephanie Piñero

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

AJAX Asynchronous JavaScript And XML

API Application Programming Interface

CGI Common Gateway Interface

CSS Cascading Style Sheets

GLSL OpenGL Shading Language

GPL General Public License

HTML HyperText Markup Language

JS JavaScript

UML Unified Modeling Language

VBO Vertex Buffer Object

RESUMEN

La generación de imágenes en 3D es un área en expansión ampliamente utilizada en los últimos años. La generación de las mismas en el ambiente web no escapa de este hecho, ya que esta área ha ido creciendo exponencialmente.

Actualmente no existen implementaciones en el ambiente web que, utilizando HTML5 y WebGL, apliquen el algoritmo de *Photon Mapping* para la generación de imágenes en 3D.

Motivado a esto la siguiente investigación tiene como fin desarrollar una solucion tecnológica para la generación de imágenes en 3D utilizando un algoritmo de iluminación global. La aplicación cuenta con una interfaz de usuario la cual permite aplicar *Photon Mapping*, *Ray Tracer* y efectos visuales tales como la refracción, reflexión y fresnel.

Palabras Claves: HTML5, WebGL, Algoritmo de iluminación global, *Photon Mapping*, *Ray Tracer*.

INTRODUCCIÓN

El continuo avance de la tecnología ha hecho posible la generación computacional de escenas complejas con altos niveles de detalle o fotorealismo. Estos niveles de detalle son dependientes de varios factores, como lo son los algoritmos de relieve, sombreado e iluminación. Claro está, que entre mayor sea el nivel de fotorrealismo que se desee alcanzar, mayor es la necesidad de hardware y software especializado.

El desarrollo de las tecnologías web y específicamente el desarrollo de WebGL para la generación de escenas 3D en exploradores web, ha impulsado el planteamiento de una aplicación desarrollada bajo esta, donde se puedan aplicar algoritmos de iluminación global, con el fin de generar una escena 3D fotorrealista.

En éste trabajo se propone el desarrollo de una aplicación la cual genere escenas 3D, con el algoritmo de *Photon Mapping* sobre WebGL. Además al desarrollar esta aplicación bajo esta tecnología se desea aprovechar una de las ventajas de WebGL, que es su portabilidad a una mayor gama de dispositivos de despliegue, como teléfonos, tablets y computadoras.

Este documento se encuentra dividido en seis (6) secciones principales. El capítulo 1, ”Planteamiento del problema” contendrá la descripción del problema, la justificación, objetivo general, objetivos específicos, alcance y ambiente de desarrollo de la solución. El capítulo 2, ”Marco conceptual”, explicará brevemente los conceptos básicos necesarios para el desarrollo de la aplicación. En el capítulo 3, ”Método de desarrollo”, se explica el método de desarrollo seleccionado para la realización de la solución. En el capítulo 4, ”Desarrollo de la solución”, se contemplará el desarrollo de la aplicación. En el capítulo 5, ”Pruebas”, se explicarán las pruebas realizadas a la aplicación y en el capítulo 6, ”Conclusiones”, se muestran las conclusiones, recomendaciones y trabajos futuros que pueden ser realizados utilizando las herramientas generadas en esta investigación.

1 PLANTEAMIENTO DEL PROBLEMA

Capítulo 1

1.1 PLANTEAMIENTO DEL PROBLEMA

En la actualidad la cantidad de dispositivos que son capaces de soportar WebGL es muy alta. Unido a esto, el nivel de realismo y la calidad son factores importantes en el área de generación de imágenes 3D. Esta se encuentra constantemente impulsada a crear contenido con cada vez mayor fotorealismo y a su vez, tratando de que esa calidad sea posible apreciarla sin necesidad de tener equipos altamente especializados, hacerla accesible a la mayor cantidad de usuarios posible.

Dado que la calidad de las imágenes está fuertemente atada a algoritmos especializados de iluminación, sombreado, relieve, y todos éstos requieren mayor poder de cómputo a medida que se aumenta la calidad exigida de ellos. Se plantea el desarrollo de una aplicación en WebGL que provea la capacidad de generar escenas con un nivel de fotorealismo moderado a alto usando específicamente el algoritmo de *Photon Mapping*.

1.2 JUSTIFICACIÓN

Photon Mapping es una técnica que permite lograr efectos visuales de forma natural, que no ha sido tan explotado como otros métodos. Se espera que creando una aplicación web basada en WebGL se pueda contar con una herramienta que permita al usuario interactuar con este método y comprender mejor su funcionamiento.

1.3 OBJETIVO GENERAL

Desarrollar una solución tecnológica que utilizando *Photon Mapping* en WebGL y HTML5 permita el despliegue y la visualización de escenas 3D con Iluminación Global.

1.4 OBJETIVOS ESPECÍFICOS

- Proveer al usuario una interfaz para la creación y modificación de escenas 3D.

- Permitir efectos visuales como reflexión, refracción y fresnel.
- Comparar los resultados obtenidos cualitativa y cuantitativamente durante las pruebas.

1.5 ALCANCE

Se plantea que este trabajo de investigación abarque el desarrollo de una escena 3D que utilice *Photon Mapping*, sea navegable y que permita la modificación de la misma por el usuario, usando WebGL.

1.6 AMBIENTE DE DESARROLLO

- Sublime Text.
- Navegador web: Google Chrome

2 MARCO CONCEPTUAL

Capítulo 2

2.1 TECNOLOGÍAS DE DESARROLLO

2.1.1 *HTML5*

Es el acrónimo de Lenguaje de Marcado de Hipertexto versión 5, fue creada con la intención de mejorar la web y el desarrollo de aplicaciones web, online y offline. Con la creación de HTML5 se implementaron nuevas etiquetas tales como video, audio, geolocalización, entre otras.

En la actualidad muy pocos sitios web pueden basarse únicamente en código HTML. En cualquier desarrollo web, se hace uso al menos de CSS (*Cascading Style Sheets*) para definir el aspecto visual de la página, y de JavaScript para hacer una página dinámica.

El objetivo principal de HTML5 es hacer que el proceso de codificación sea más fácil y lógico para el desarrollador, además de proporcionar una plataforma que permita desarrollar aplicaciones web con mayor similitud a las aplicaciones de escritorio, donde su ejecución dentro de un navegador no implique falta de recursos. Para lograr esto se están creando APIs que permitan trabajar con cualquiera de los elementos de la página y realizar acciones que hasta hoy era necesario realizar por medio de otras tecnologías. Estas APIs que tendrán que ser implementadas por los distintos navegadores, se están documentando con mucho cuidado, para que todos los navegadores les provean soporte tal cual se han diseñado.

HTML5 incluye novedades significativas en diversos ámbitos del desarrollo de aplicaciones web. No solo se trata de incorporar o eliminar etiquetas, sino que se intenta mejorar áreas que hasta ahora quedaban fuera del alcance del lenguaje y para las que se necesitaba utilizar otras tecnologías.

Algunas de las mejoras más significativas de HTML5 [**?**] son:

1. Estructura de la aplicación web: HTML5 permite agrupar todas las partes de una página tales como cabecera y pie de página, en nuevas etiquetas que representan cada una de las partes comunes de una página tales como <header>, <footer>,

<article>, <section>, <aside> entre otras. Estas nuevas etiquetas tienen como objetivo dividir el documento HTML en partes lógicas, en las cuales el nombre de cada etiqueta es descriptivo del tipo de contenido que debería tener.

2. Etiquetas de contenido específico: en HTML anteriormente se utilizaba una única etiqueta para incorporar diversos tipos de contenido enriquecido, como animaciones Flash o videos. Ahora con HTML5 se utilizarán etiquetas específicas para cada tipo de contenido en particular, como lo son las etiquetas de audio y video.

3. Canvas: este nuevo elemento permitirá la renderización dinámica de formas 2D e imágenes, por medio de las funciones API, que podrán ser animadas y responder a la interacción con el usuario. Con esta etiqueta se pueden realizar acciones como las que provee Flash, pero dentro de la especificación de HTML y sin necesidad de instalar previamente ningún plugin.

4. Fin de las etiquetas de presentación: todas las etiquetas que tienen que ver con la presentación del documento, es decir, que modifican estilos de página, tales como , <font>, <basefont> entre otras serán eliminadas.

Al tratarse de una tecnología completamente nueva, progresivamente se han ido desarrollando las respectivas adaptaciones en los navegadores web para hacerlos compatibles con el lenguaje. Actualmente los navegadores Opera y Google Chrome son los que poseen mayor soporte del lenguaje, seguidos por Mozilla Firefox y Safari. En la **Tabla 1** se observa para distintos navegadores y distintas versiones de los mismos, el porcentaje de características de HTML5 que soportan [**?**].

Chrome	Safari	Firefox	Opera	Internet Explorer
36: 87.5%	5.1: 45%	32: 80%	22: 83.2%	8.0: 5.9%
37: 89%	6.0: 58.7%	33: 80%	23: 85.2%	9.0: 20.3%
38: 89.9%	7.0: 63.4%	34: 80.9%	24: 87.3%	10.0: 53.5%
39: 90.2%	8.0: 71.3%	35: 80.9%	26: 89.5%	11.0: 60.5%

Tabla 1 – Porcentajes de soporte de HTML5 en los Navegadores (abril 2015)

2.1.2 *JavaScript*

JavaScript es un lenguaje de script del lado del cliente, por lo que el código fuente es procesado por el navegador web del cliente en lugar de ser procesado en el servidor web.

Se trata de un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas e interactivas, las cuales pueden incorporar texto con efectos variados, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

De manera técnica se describe como un lenguaje de programación interpretado por lo que no es necesario compilar los programas para ejecutarlos. Una aplicación que utiliza JavaScript se puede probar directamente en un navegador sin necesidad de ningún proceso intermedio[**?**].

La integración de JavaScript con HTML es muy flexible, ya que existen 3 formas para incluir este código en las páginas web. Una de ellas consiste en incluirlo en el mismo documento HTML, se encierra entre las etiquetas <script> y se incluye en cualquier parte del documento. Aunque es correcto incluir cualquier bloque de código en cualquier zona de la página, se recomienda definir el código JavaScript dentro de la cabecera del documento.

Las instrucciones JavaScript también se pueden incluir en un archivo externo de tipo JavaScript que los documentos HTML se encargan de enlazar. Los archivos de este tipo son documentos normales de texto con extensión .js, que se pueden crear con cualquier editor de texto. Las principales ventajas de enlazar un archivo JavaScript es que se simplifica el código de la página, que se puede reutilizar el mismo código en todas las páginas web de la aplicación y que cualquier modificación realizada en el archivo se ve reflejada inmediatamente en todas las páginas HTML que lo enlazan.

Un último método consiste en incluir trozos de JavaScript dentro del código HTML de la página. El mayor inconveniente es que recarga innecesariamente el código y complica el mantenimiento del mismo.

Una de las características principales de JavaScript es que es orientado a eventos, esto quiere decir que la interacción con el usuario se consigue mediante la captura de los eventos que este produce. Un evento es una acción del usuario ante la cual puede realizarse algún proceso (por ejemplo, el cambio de valor de un formulario).

2.1.3 *JQuery*

Es una librería de JavaScript que simplifica el código de HTML y ofrece una infraestructura que facilita la programación de

aplicaciones complejas, permitiendo el manejo de eventos, manejo del árbol DOM, animaciones e interacciones con AJAX para un rápido desarrollo web en cualquier tipo de plataforma [**?**].

Al ser una librería, JQuery contiene un conjunto de funciones básicas y muy utilizadas que facilitan la tarea al desarrollador, esto permite desarrollar código JavaScript de manera muy rápida y de forma muy intuitiva. Su instalación no es complicada ya que solo consiste en incluir una línea de código en la sección *head* del documento HTML.

JQuery se ha convertido en uno de los complementos más esenciales a la hora de desarrollar una aplicación web, esto se debe a que facilita el desarrollo de aplicaciones enriquecidas del lado del cliente, en JavaScript, que son compatibles con todos los navegadores.

La principal ventaja que ofrece JQuery es que ya no se requiere programar un código JavaScript para cada navegador en el que se va a visualizar la página, sino que la propia librería se encarga de que el código sea compatible con el software con el cual el cliente está accediendo a nuestra web. Además tiene una gran cantidad de *plugins*que facilitan el trabajo del desarrollador.

Una de las características más resaltantes de JQuery es que permite cambiar el contenido de una página web sin necesidad de recargarla, esto se logra mediante la manipulación del árbol DOM y peticiones AJAX.

2.1.4 *three.js*

Es una librería de alto nivel escrita en JavaScript ideada por Ricardo Cabello en abril de 2010. Originalmente se desarrolló en ActionScript y posteriormente se portó a JavaScript. Permite la creación de animaciones 3D utilizando recursos de GPU sin la necesidad de plugins adicionales. En la **Figura 1** se puede observar una aplicación desarrollada con *three.js*.

A continuación se presentan algunas de las características más resaltantes de *three.js* [**?**].

- Escenas: agregar o eliminar objetos en tiempo de ejecución, niebla.
- Cámara: perspectiva, ortográfica y navegable.
- Luces: ambientales, dirigidas, puntuales y *spot lights*.
- Sombras: proyección y recepción.

- Materiales: *Lambert*, *Phong*, *smooth shading*, texturas, entre otras.
- *Shaders*: acceso total a las capacidades de GLSL.
- Objetos y geometría.

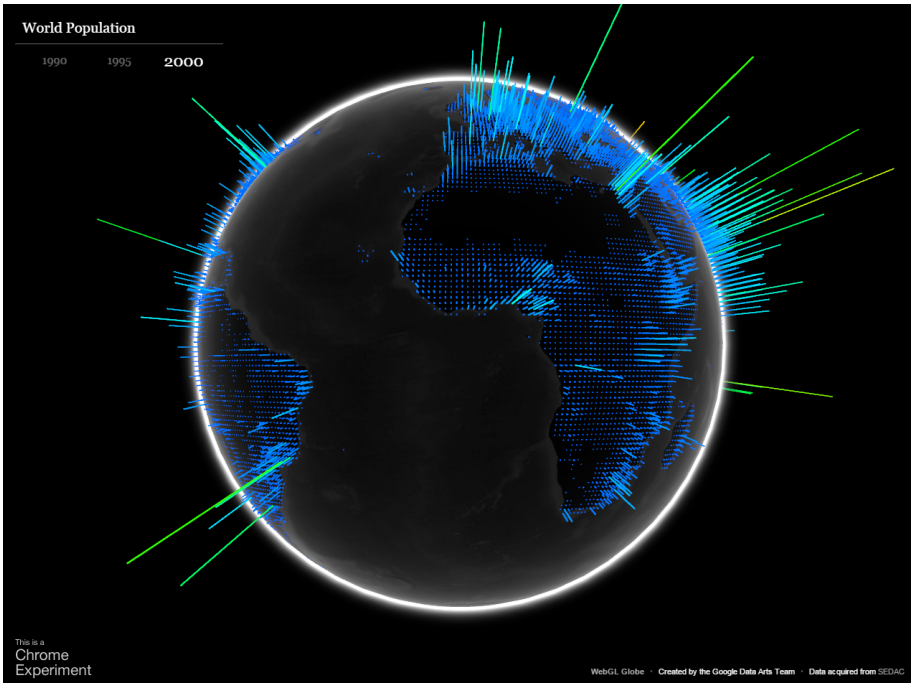

Figura 1 – Aplicación creada con la utilización de *three.js*

2.1.5 *WebGL*

WebGL es lanzado oficialmente en el año 2011, aunque sus origenes datan de hace 20 años, cuando la versión 1.0 de OpenGL fue liberada como una alternativa no propietaria a GL de Silicon Graphics. Hasta el 2004, OpenGL utilizó un *pipeline* fijo. La versión 2.0 de OpenGL fue liberada ese año e introdujo el lenguaje de shaders GLSL (GL *shading language*) que permitía programar las porciones de vértice y fragmento del *pipeline*. La versión actual de OpenGL es la 4.3, sin embargo, WebGL está basado en OpenGL ES (sistemas embebidos) 2.0, el cual se liberó en el 2007 como una versión más liviana de OpenGL 2.0.

OpenGL ES está construido para usarse en dispositivos embebidos como teléfonos móviles, los cuales tienen bajo poder de procesamiento en contraste con computadoras de escritorio. Por ello es más restrictivo y tiene una API más pequeña que la de OpenGL [**?**]. Por ejemplo, OpenGL permite dibujar vértices con las instrucciones glBegin...glEnd, VBO (objeto buffer de vértices), *display*

lists... en cambio OpenGL ES solo permite usar VBO que es la opción más eficiente.

En el año 2006, Vladimar Vukicevic trabajó en un prototipo de canvas 3D que usaba OpenGL para la web. En el 2009, Khronos creó el grupo de trabajo WebGL y desarrollaron la especificación central que ayuda a asegurar que implementaciones entre diferentes exploradores sean muy cercanos unos de otros. El contexto 3D fue modificado para WebGL y una versión 1.0 de la especificación se completó a principios del 2011. Como se puede observar en el *pipeline* de WebGL en la **Figura 2**, este está compuesto por varios elementos que definiremos a continuación.

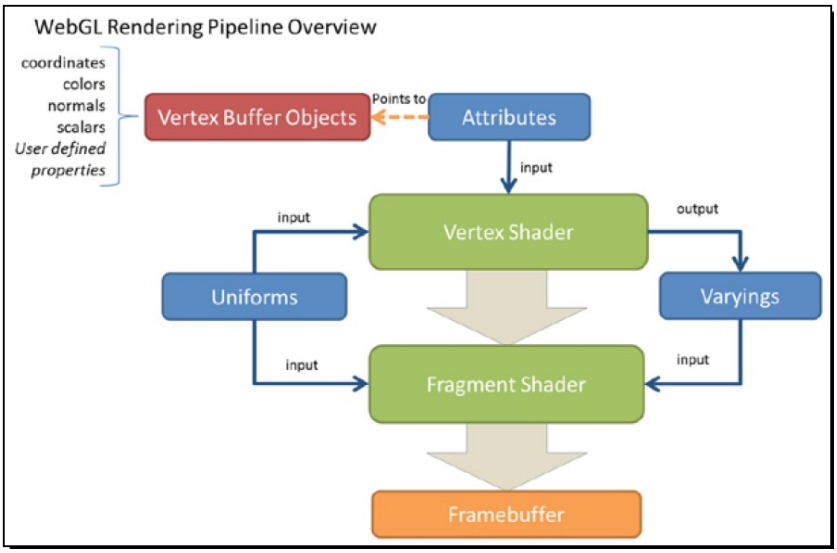

Figura 2 – *Pipeline* de WebGL.

- *Attributes*, *uniforms* y *varyings*: son los tipos de variables que se pueden encontrar cuando se programan *shaders*. Los *attributes* son variables utilizadas para introducir datos a los *shaders*, en cada llamado del mismo los valores de los *attributes* son diferentes. Las *uniforms* son variables utilizadas para introducir datos tanto en el *fragment shader* como en el *vertex shader*. A diferencia de los *attributes*, las *uniforms* mantienen su valor durante un ciclo de renderizado. Las *varying* son variables utilizadas para intercambiar datos entre el *vertex shader* y el *fragment shader*.

- *Vertex Shader*: se llama en cada vértice y manipula datos relacionados a cada uno. Los datos se representa como atributos dentro del *shader* y apunta al VBO correspondiente.

- *Fragment Shader*: cada conjunto de tres vértices definen un triángulo y cada elemento de este debe tener un color asignado,

de lo contrario las superficies, denominadas fragmentos, serían transparentes. El *fragment shader* es el encargado de calcular el color de los pixeles individuales.

- *Vertex Buffer Object*: estos contienen los datos que WebGL requiere para describir la geometría que será renderizada, como lo son las normales de los vértices, colores, coordenadas de textura, entre otras.

- *Framebuffer*: es un buffer bidimensional que contiene fragmentos los cuales han sido procesados por el *fragment shader*. Una vez que todos los fragmentos han sido procesados, se forma una imagen 2D y se procede a su despliegue.

2.1.5.1 *Ventajas de WebGL*

1. Programación con JavaScript: es un lenguaje natural tanto para programadores como para exploradores web. Lo que permite tener acceso a todas las partes del DOM y a la vez permite una comunicación sencilla entre elementos, cosa que las *applets* no permiten. Dado que WebGL se programa en JS es mucho más sencillo su integración con otras librerías de JS como JQuery y otras tecnologías de HTML.

2. Manejo automático de memoria: al utilizarse JS se tiene la ventaja en cuanto al manejo de memoria de las variables que el realiza. Si una variable no se necesita más, automáticamente JS libera la memoria asociada a ella en contraste con *C++* donde el manejo de memoria de estas debe hacerse manualmente.

3. Presencia: debido que en la actualidad tanto computadoras como *smartphones* y *tablets* tienen la posibilidad de utilizar exploradores web con la capacidad de interpretar JavaScript.

4. Desempeño: con algunas excepciones el desempeño de aplicaciones que utilicen WebGL hoy en día es equivalente a aquellas desarrolladas sobre plataformas *standalone*. Esto viene dado por la habilidad de WebGL de acceder al hardware gráfico local, cosa que muchas aplicaciones web de renderizado 3D no tenían la capacidad de hacer siendo necesario el renderizado por software.

5. No se compila: dado que WebGL está escrito en JavaScript, no existe la necesidad de compilar el código antes de ser ejecutado en un explorador web, permitiendo hacer cambios en tiempo de diseño y ver los resultados rapidamente. Es necesario acotar que a la hora de usar *shaders* estos si son compilados directamente en el hardware gráfico.

2.1.5.2 *Estructura Básica de llamadas de una aplicación con WebGL*

Para cualquier aplicación 3D que se vaya a realizar se necesita que ciertos componentes básicos estén presentes a la hora de crear una escena 3D. Estos son:

1. Canvas: estructura de HTML5 que puede ser modificado a través del modelo de objeto del documento.
2. Objetos: son estructuras 3D que forman parte de la escena. Estos se dibujan usando triángulos los cuales se forman entre los vértices que componen al objeto.
3. Luces: todo lo que se dibuje en la escena necesita una fuente de luz para poder ser visto.
4. Cámara: el canvas actúa como la ventana a través de la cual se puede ver la escena que se está renderizando. Por ello es importante apuntar la ventana hacia donde el contenido de la escena se encuentra o será renderizado.

Hay dos maneras de hacer renderizado, una es en el cliente y otra es en el servidor. Generalmente las imágenes que son muy complejas son renderizadas en máquinas especializadas con muy altas capacidades, como en el caso de películas animadas 3D. WebGL tiene un enfoque de renderizado en el cliente, dado que tiene la posibilidad de acceder a los recursos de hardware de la máquina donde se está ejecutando. En la **Figura 3** se puede observar la estructura básica de llamadas de una aplicación con WebGL, y como interactúan HTML, JavaScript y GLSL [**? ? ?**].

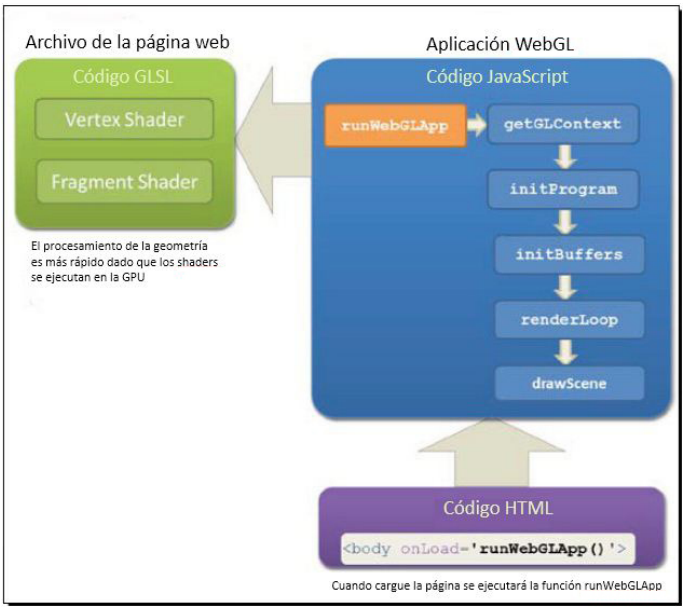

Figura 3 – Estructura de una aplicación de WebGL.

2.2 RAY TRACING

Técnica utilizada para la generación de imágenes fotorealistas, en el ámbito de la computación gráfica. Esta técnica se basa en el trazado de los rayos de luz a través de los pixeles en el plano de visualización. Es una solución que ofrece un alto grado de realismo pero tiene un alto costo computacional, por lo que generalmente se reserva para despliegues que no son en tiempo real como efectos especiales de películas. A través de este método se puede simular una gran variedad de efectos como:

1. Reflexión: cambio de dirección de la luz al entrar en contacto con otro medio, devolviendo la onda de luz al medio original donde viaja la misma.
2. Refracción: cambio de dirección de la luz al cambiar de un medio a otro.
3. Fresnel: es un efecto que se calcula a través de la cantidad de luz que se refleja y que se refracta cuando un rayo de luz golpea una superficie.

En la **Figura 4** y **Figura 5** se pueden observar los diferentes comportamientos de la luz antes mencionados y en la **Figura 6** un ejemplo de Ray Tracing sobre la Caja de Cornell [**?**].

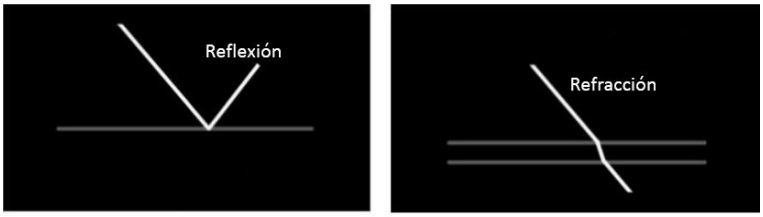

Figura 4 – Comportamientos de la luz, reflexión y refracción

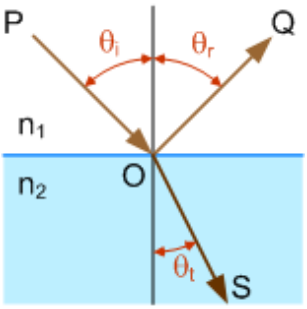

Figura 5 – Comportamientos de la luz, fresnel

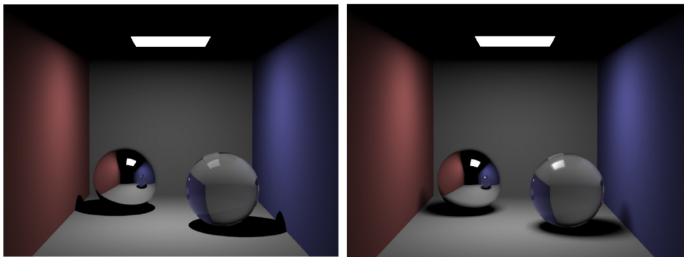

Figura 6 – Caja de Cornell utilizando Ray Tracing
(Sombras solidas y suaves respectivamente)

2.3 PHOTON MAPPING

Photon Mapping es un algoritmo de iluminación global en dos etapas, el mismo fue desarrollado por Henrik Jensen como una alternativa a técnicas de Monte Carlo de *ray tracing*.

Photon Mapping desacopla la iluminación de la geometría usando una estructura de datos espacial denominada *photon map*. Esta separación ha probado ser bastante poderosa dado que permite el cálculo y almacenamiento de los componentes de la ecuación de renderizado por separado en el *photon map*, logrando en escenas de gran tamaño mejor rendimiento aún con la necesidad de una mayor cantidad de memoria en relación a algunos otros algoritmos de iluminación global como trazado de caminos, trazado bidireccional de caminos y *Metropolis Light Transport*, los cuales pueden simular iluminación global en escenas complejas usando muy poca memoria. Además esta separación brinda la capacidad de utilizar diferentes técnicas para los cálculos de estos componentes, dándole a este método una gran flexibilidad. Los *photon maps* pueden ser definidos como estructuras en las cuales se almacenará la dirección del fotón y el punto de intersección del mismo con una superficie de la escena como se puede ver en la **Figura 7 (c)**. Esto con el fin de poder ser consultado en las respectivas etapas del algoritmo y calcular los valores de brillo para dicha superficie como se explicará en éste capítulo.

En la **Figura 7 (b)** se puede observar la Caja de Cornell iluminada utilizando el algoritmo de *Photon Mapping*[**? ?**].

Como se mencionó anteriormente *Photon Mapping* es un algoritmo en dos etapas:

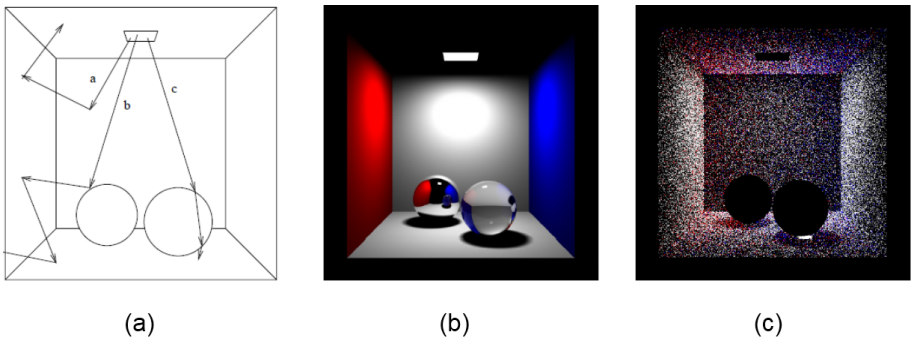

Figura 7 – Caja de Cornell
(a) Luz emitida por la fuente en la habitación.
(b) Habitación iluminada usando Photon Mapping.
(c) Mapa de fotones correspondiente a (b).

2.3.1 *Primera Etapa - Photon Tracing*

Es el proceso a través del cual se emite una cantidad finita de fotones desde las fuentes de luz y se trazan a través de la escena, como se puede observar en la **Figura 7 (a)**. El objetivo principal es poder llenar el *photon map* con el cual se calculará el brillo reflejado de las superficies y la dispersión del brillo en los objetos participantes de la escena en la segunda etapa.

- *Photon Emission*: la vida de un fotón comienza en la fuente de luz y termina cuando este se queda sin energía. Para cada fuente de luz en la escena se crea un conjunto de fotones y se divide la energía de la fuente de luz entre ellos. Fuentes de luz más brillantes emiten mayor cantidad de fotones que las fuentes de luz más tenues. Cualquier tipo de fuente de luz puede ser usado y los modelos de emisión para ellas pueden variar, como se puede ver en la **Figura 8**.

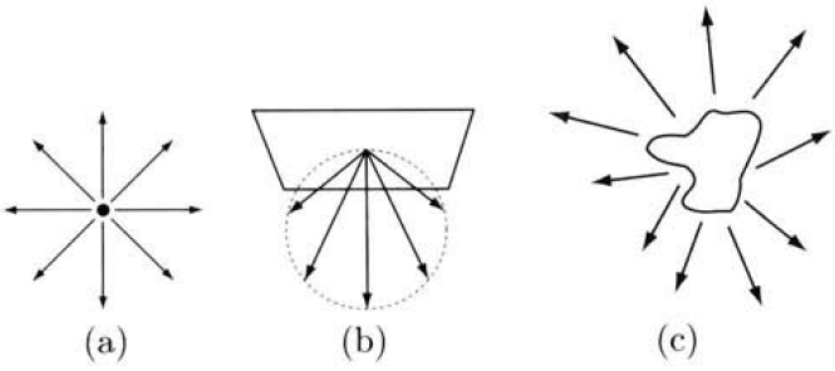

Figura 8 – Tipos de Fuentes de Luz
(a) Luz Puntual.
(b) Luz Cuadrada.
(c) Luz Compleja.

- *Photon Scattering*: una vez emitidos los fotones desde la fuente de luz estos serán absorbidos o se perderán en la misma. Cuando un fotón choca con una superficie es necesario decidir en base a las propiedades de dicha superficie qué ocurre con la energía que este fotón carga, es decir, si es absorbida, reflejada o refractada, como se muestra en la **Figura 9**.

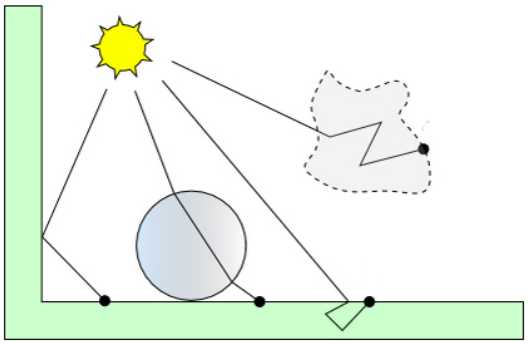

Figura 9 – *Photon Scattering*

Dado que el costo computacional y de almacenamiento aumenta a medida que la cantidad de fotones también lo hace, se propone el uso de la técnica de Ruleta Rusa para la decisión de si un fotón debe ser reflejado, refractado o es absorbido por una superficie. Con algunas modificaciones y consideraciones se puede alcanzar el mismo resultado reduciendo considerablemente los recursos necesarios. Por ejemplo, al tenerse una superficie con 50% de probabilidad de reflexión y emitir 1000 fotones hacia la misma, normalmente esta superficie debería reflejar 1000 fotones con la mitad de la energía original, usando la técnica de Ruleta Rusa es posible que de estos 1000 fotones solo 500 sean reflejados con el 100% de su energía y 500 sean absorbidos por la superficie. De esta manera se consiguen los mismos resultados con el cálculo y almacenamiento de solo 500 fotones.

- *Photon Storing*: una escena puede emitir desde algunos miles hasta millones de fotones desde una fuente de luz, por lo tanto, es conveniente que el *photon map* sea lo más compacto posible. Adicionalmente, es necesario que pueda soportar rápidas búsquedas espaciales en tres dimensiones, dado que será necesario hacer una gran cantidad de peticiones a esta estructura durante el renderizado de la escena.

  Jensen recomienda el uso de un *kd-tree* como estructura para almacenar el *photon map*. El *kd-tree* es un árbol binario para almacenar una cantidad finita de puntos de un espacio k-dimensional [**?**].

Siendo cada nodo un punto y teniendo k coordenadas correspondientes al k de las dimensiones del espacio, se procede a seleccionar el primer eje en torno al cual se particionará dicho espacio. Luego se coloca el primer nodo en donde corresponda, y se genera un plano que corta el espacio, el cual es perpendicular al eje de partición actual. A continuación, se cambia el eje de partición y se insertan los nodos hijos. El nodo hijo izquierdo se insertará en la partición izquierda del espacio anterior y el derecho en la partición derecha respectivamente. De esta manera se continúan cambiando los ejes de partición y asignando los nodos en las nuevas particiones, como se puede ver en la **Figura 10**.

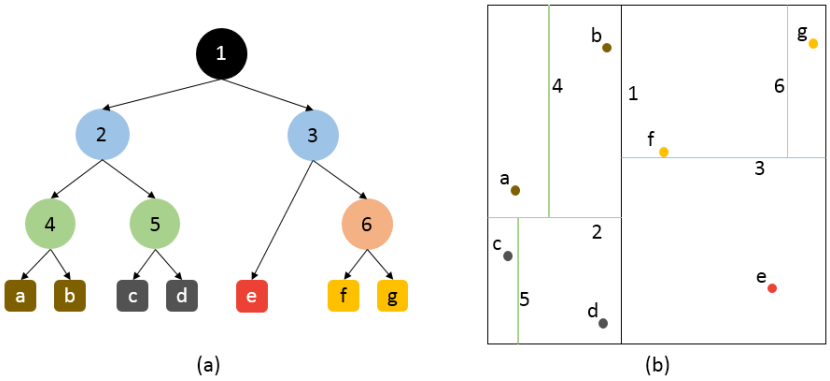

Figura 10 – *kd-tree Bi-dimensional*
(a) Representación Estructurada.
(b) Representación Espacial.

Las búsquedas en esta estructura en el peor caso son O(n) y al estar el *photon map* lleno se balancea logrando que las búsquedas tengan una complejidad O(log n).

2.3.2 *Segunda Etapa - Renderizado*

- Estimación de Brillo: se realiza un trazado de rayos tradicional el cual emite rayos desde la cámara. Cuando estos rayos chocan con un punto de una superficie la información de iluminación que se almacenó de los fotones vecinos en la primera etapa se añadirá a la información obtenida por el trazado de rayos en ese punto para realizar los cálculos necesarios.

  Se considerarán todos los fotones que se encuentren dentro de una pequeña esfera alrededor del punto como se puede observar en la **Figura 11**. Con esto se calculará un área pequeña alrededor del punto de la superficie dentro de la cual se buscarán los fotones que contribuyan al brillo de ese punto y se

descartarán aquellos que no contribuyan, este valor se añadirá al valor que el trazado de rayos calculó y tras normalizarlo se tendrá el valor final para ese punto de esa superficie.

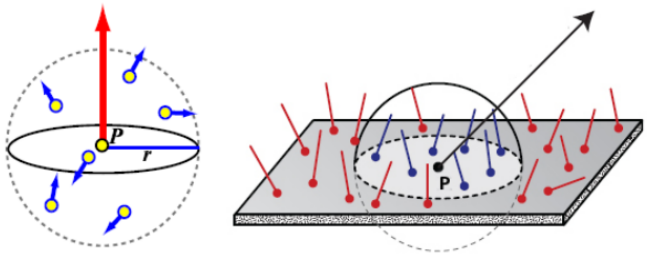

Figura 11 – Recolección de Fotones Alrededor de un Punto

En la **Figura 12**, **Figura 13**, **Figura 14** y **Figura 15**, se pueden observar distintos ejemplos de escenas renderizadas con *Photon Mapping*.

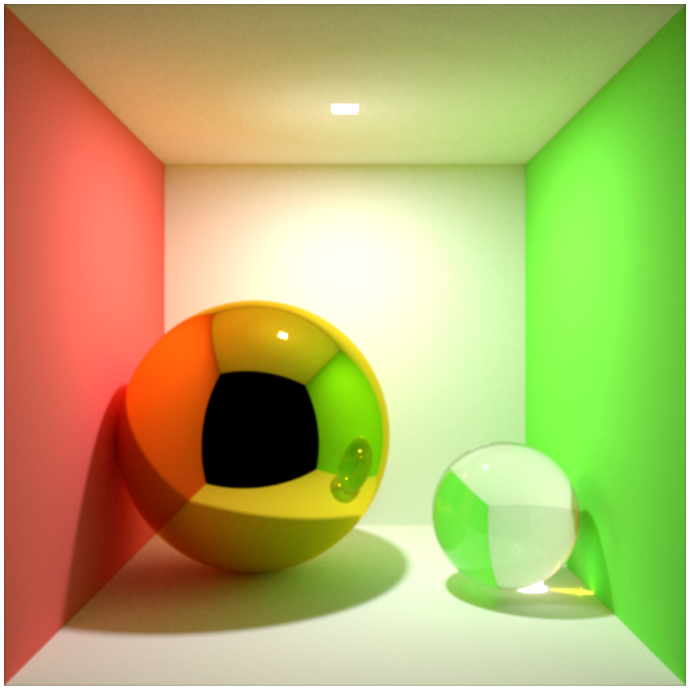

Figura 12 – Ejemplo de *Photon Mapping*

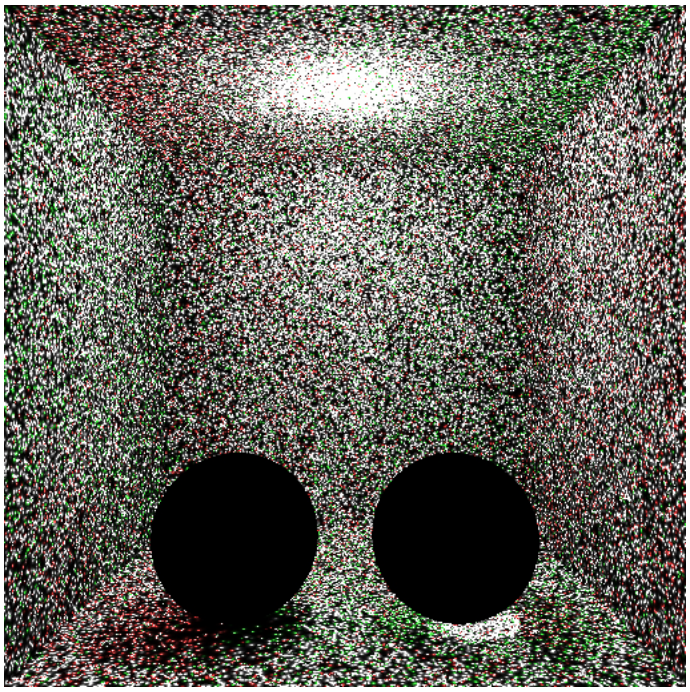

Figura 13 – *Photon Map* correspondiente a la escena de la **Figura 14**

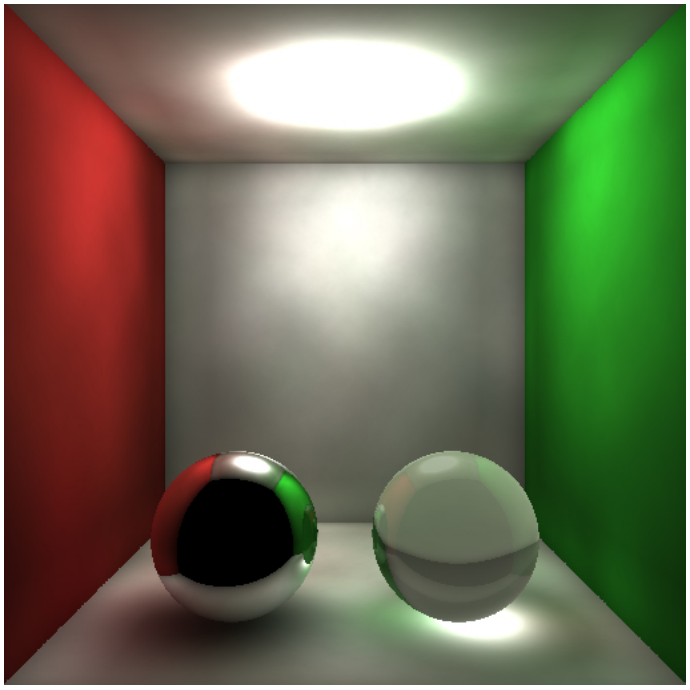

Figura 14 – Ejemplo de *Photon Mapping*
Escena en la cual se puede apreciar el efecto de cáusticos.

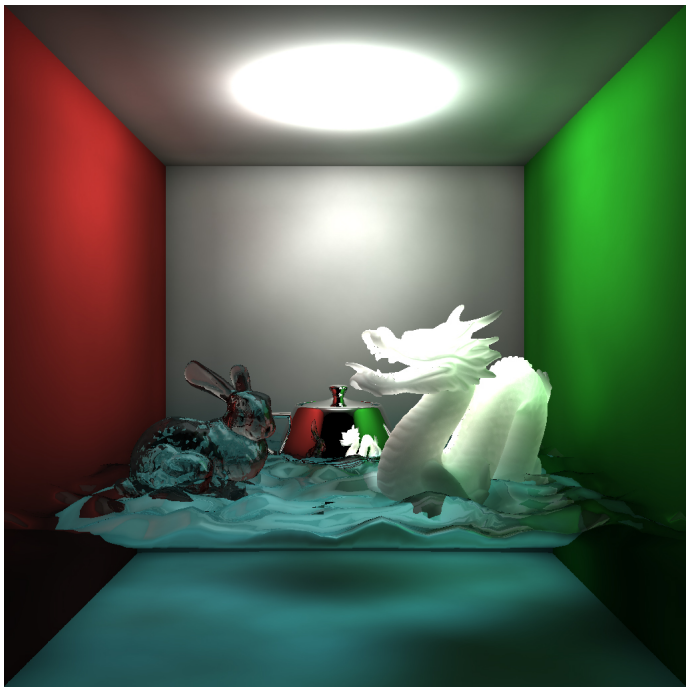

Figura 15 – Ejemplo de *Photon Mapping*
Escena con la tetera de Utah, el conejo y dragón de Stanford en la Caja de Cornell con agua.

2.3.3 *Ventajas y Desventajas*

Ventajas:

- No requiere de un mallado para funcionar.
- Por lo general tiene un rendimiento mayor que otros métodos de Iluminación Global, destacándose en escenas complejas.
- No es dependiente de la geometría de la escena.
- Es capaz de simular cáusticos e interreflexiones difusas.

Desventajas:

- Requiere de mayores cantidades de memoria para funcionar de manera correcta comparado con los métodos de trazado de rayos, trazado bidireccional de caminos y *Metropolis Light Transport*. Esto como consecuencia de que cada punto de impacto de los fotones debe ser almacenado en el *photon map*.
- En algunos casos los objetos pequeños pueden no recibir suficientes impactos de los fotones emitidos por las fuentes de luz.
- Las búsquedas de fotones cercanos en el espacio pueden ser ineficientes.

2.4 ILUMINACIÓN GLOBAL EN WEBGL

A continuación se presentan algunos trabajos de iluminación global usando WebGL:

- Implementacón de Evan Wallace 2010: esta implementación usando *Path Tracing* soporta superficies difusas, brillantes y reflectantes como se pueden ver en la **Figura 16** [**?**]. Se puede encontrar el demo en: `http://madebyevan.com/webgl-path-tracing/`

- Implementación de Florian Boesch 2012: aplica iluminación global usando *Deferred Shading* y *Deferred Lighting*. *Deferred Shading* es una técnica que consiste en desacoplar la geometría y la iluminación de la escena. A esta técnica se le define como diferida dado que en la primera ejecución del *vertex* y el *pixel shader* solo se calcula la información necesaria para el *shading* y no es hasta la segunda ejecución que este se realiza. *Deferred Lighting* es una modificación de *deferred shading*, la cual usa tres ejecuciones en vez de dos. En la primera ejecución se calcula el valor de brillo por *pixel*, el cual da como resultado la data correspondiente a la iluminación especular y difusa, la segunda ejecución es para leer dichos valores y calcular el valor final de brillo correspondiente a cada *pixel*, tras lo cual, se procede a la tercera ejecución donde se realiza el *shading* [**?**]. Se pueden apreciar algunas capturas de la escena en la **Figura 17**. El demo se encuentra en: `http://codeflow.org/webgl/deferred-irradiance-volumes/www/`

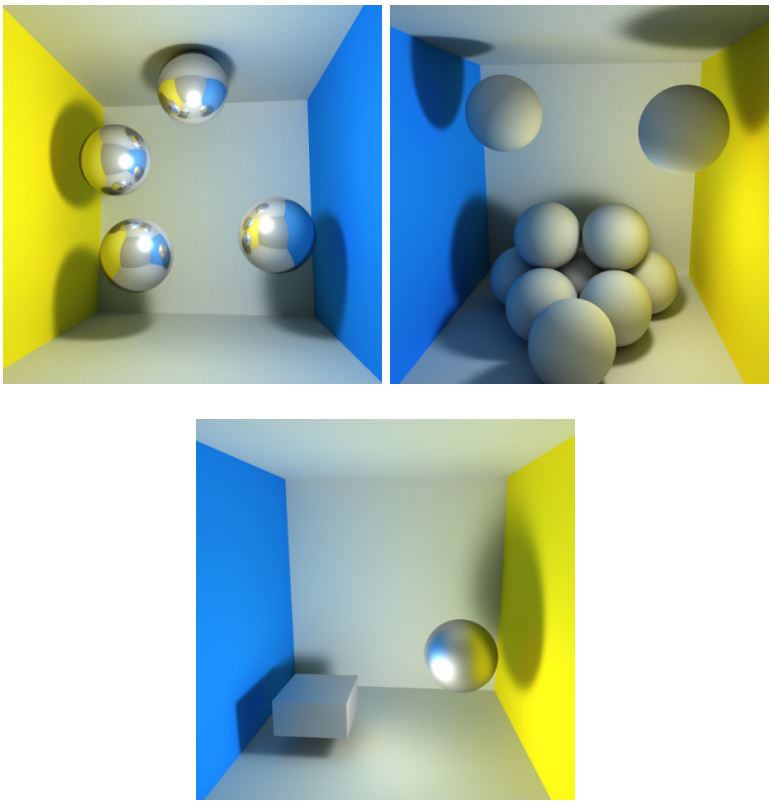

Figura 16 – Capturas de la implementación de Evan Wallace

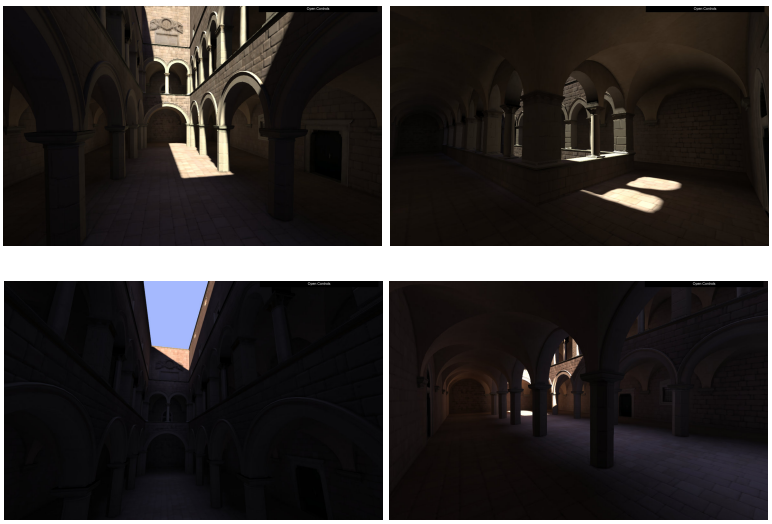

Figura 17 – Capturas de la implementación de Florian Boesch

3

MÉTODO DE DESARROLLO

Capítulo 3

3.1 MODELO DE PROCESOS DE INGENIERÍA DE SOFTWARE

Los modelos de procesos de ingeniería del software no imponen un modelo concreto de cómo desarrollar, ni como realizar las diferentes actividades incluidas en cada proceso, por lo que cada lector debe implementar sus propios métodos, técnicas y herramientas.

Estos se pueden definir como una simplificación o abstracción de un proceso real, es decir, consiste en una presentación abstracta de alto nivel de un proceso de software.

Dentro del conjunto de métodos de desarrollo del software, existe una familia de patrones para el desarrollo ágil de estos, tiene como objetivo fundamental minimizar las actividades que no se consideren relevantes, aumentar la productividad del equipo de desarrollo y elevar la adaptabilidad del resultado. Por otro lado, hay métodos para el desarrollo de software orientada a prototipos, estos presentan un conjunto de ventajas a los desarrolladores los cuales se definirán más adelante.

3.1.1 *Modelo de Desarrollo Ágil*

Los métodos de desarrollo ágil consisten en elementos individuales llamados prácticas. Las prácticas incluyen el uso de control de versiones y estándares de codificación, haciendo entrega de versiones del producto cada semana. Los métodos ágiles combinan de forma única aquellas prácticas que apoyan la filosofía ágil, descartando las demás y mezclándolas con ideas nuevas.

Los métodos de desarrollo ágil surgen a raíz de las fuertes desventajas de la ingeniería del software, como la dificultad para aceptar cambios debido a altos costos, demanda de tiempo, documentación exhaustiva y la reestructuración de proyectos.

3.1.2 *Modelo basado en Prototipos*

Los modelos basados en prototipos están enfocados en especificar los requerimientos, presentando al cliente versiones experimentales

de un sistema que tiene los suficientes elementos como para permitir su utilización, denominados prototipos. Luego de presentar un prototipo se capturan requerimientos nuevamente, con el objetivo de aclarar los requerimientos que el cliente necesita y tener una mejor percepción del sistema desde el punto de vista del cliente.

Inicialmente no se conocen todos los requerimientos del sistema, se generan prototipos, los clientes prueban y añaden requerimientos, se realiza una implantación parcial del sistema y se prueba.

Las etapas del ciclo de vida del modelo basado en prototipos son:

- Análisis de requisitos del sistema.
- Análisis de requisitos de software.
- Diseño, desarrollo e implementación del prototipo.
- Prueba del prototipo.
- Refinamiento interactivo del prototipo.
- Refinamiento de las especificaciones del prototipo.
- Diseño e implementación del sistema final.
- Mantenimiento.

Dentro de las ventajas se encuentran, la alta comunicación entre el desarrollador y el cliente, estadísticamente se estima que ello aumenta la posibilidad de aceptación del sistema y es útil cuando el cliente conoce el objetivo general del sistema pero ignora los detalles del mismo.

La construcción o desarrollo basado en prototipos, se aplica como una técnica implementada dentro del contexto de otro modelo de ingeniería del software. Esto motivado a las ventajas que presenta tanto para el desarrollador del sistema como para el cliente.

3.2 MÉTODO DE DESARROLLO SELECCIONADO

Para el desarrollo de esta aplicación fue seleccionado un método de desarrollo ágil basado en prototipos. Debido a que se busca desarrollar un sistema con las ventajas en tiempo y esfuerzo que permiten un desarrollo ágil, mientras se mantiene una fuerte comunicación y relación con el cliente, construyendo la solución y recolectando requerimientos utilizando la técnica del modelo de desarrollo orientado a prototipo.

Las fases del ciclo de vida, como muestra la **Figura 18**, del método de desarrollo seleccionado son:

- Análisis y diseño.
- Desarrollo.
- Pruebas.
- Implantación.

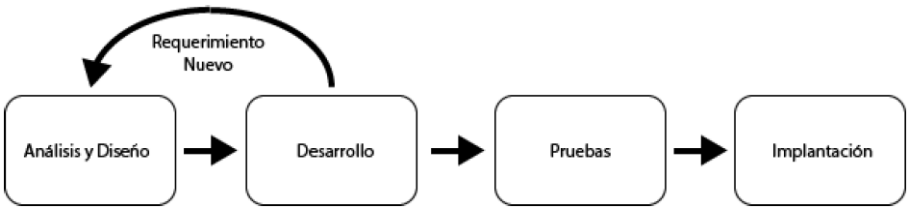

Figura 18 – Fases del método de desarrollo

Para la fase de análisis y diseño del sistema se realizaron diagramas de casos de uso, utilizando el estándar UML. En el caso de la fase de desarrollo, se seleccionó el modelo orientado a prototipo, en el cual se capturan requerimientos y se obtiene retroalimentación del cliente mostrándole un prototipo del sistema. Esto se realiza adaptando los instrumentos de análisis y diseño del mismo, mientras evolucionan los prototipos.

Para la fase de pruebas del sistema, se deben realizar pruebas de aceptación del sistema junto con el cliente en el cual se somete el sistema a una situación, esperando una respuesta acorde, si el sistema no responde de la manera esperada deben realizarse los cambios necesarios.

Finalmente, si el cliente está de acuerdo con todas las pruebas de aceptación que le fueron realizadas al sistema, se procede a implantar el sistema y verificar su correcto funcionamiento.

4

DESARROLLO DE LA SOLUCIÓN

Capítulo 4

El desarrollo de la solución se realizó siguiendo las fases del método de desarrollo seleccionado, las cuales son: análisis y diseño, desarrollo, pruebas e implantación. A continuación se describirán en detalle estas fases, sus procesos y resultados dentro del desarrollo de la solución.

4.1 ARQUITECTURA DE LA SOLUCIÓN

La aplicación web consiste en una interfaz que le permite al usuario generar diferentes escenas con diferentes objetos y luego aplicar el algoritmo de *Photon Mapping* a dicha escena.

4.2 ANÁLISIS Y DISEÑO DEL SISTEMA

Se realizó un diagrama de casos de uso para registrar los requerimientos de la aplicación, en la **Figura 19** se muestra el nivel 1 del diagrama de casos de uso de la aplicación.

A continuación se muestran las descripciones del diagrama de casos de uso en las **Tablas 2,3,4,5,6,7,8**. Cada uno de estos indica la pre-condición que se debe cumplir para ser ejecutado, el nombre, los actores que interactúan con el sistema, la descripción del caso de uso y el flujo de funcionamiento básico.

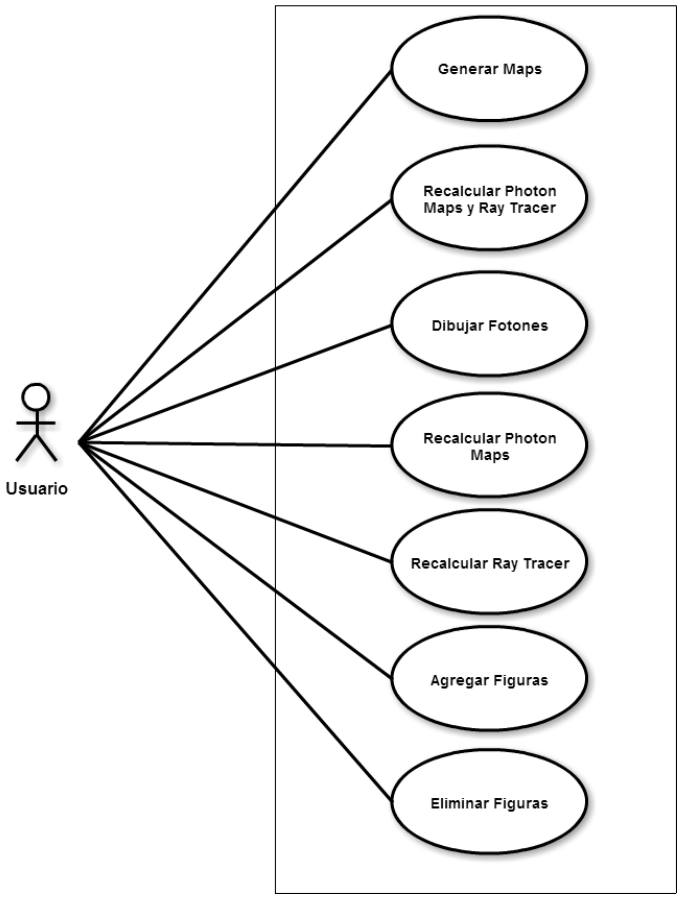

Figura 19 – Diagrama de Casos de Uso nivel 1

Pre-condición	Haber iniciado la aplicación
Nombre	Generar Maps
Actores	Usuario
Descripción	Permite generar imágenes usando *Photon Mapping* y *Ray Tracing*
Flujo Básico	1. Se muestra la escena al usuario, 2. El usuario habilita los controles de la aplicación, 3. El usuario presiona el boton "Maps", 4. Se generan las imágenes

Tabla 2 – Descripción caso de uso Generar Maps

Pre-condición	Haber iniciado la aplicación
Nombre	Recalcular *Photon Map* y *Ray Tracer*
Actores	Usuario
Descripción	Permite calcular el *Photon Map* y aplicar el *Ray Tracer* con los valores definidos en los controles y para la escena actual
Flujo Básico	1. Se muestra la escena al usuario, 2. El usuario habilita los controles de la aplicación, 3. El usuario presiona el botón "PM_RM_Recalculate", 4. La aplicación realiza los cálculos correspondientes

Tabla 3 – Descripción caso de uso Recalcular *Photon Map* y *Ray Tracer*

Pre-condición	Haber iniciado la aplicación
Nombre	Recalcular *Photon Maps*
Actores	Usuario
Descripción	Permite recalcular los valores del *Photon Map* con los valores definidos en los controles del mismo
Flujo Básico	1. Se muestra la escena al usuario, 2. El usuario habilita los controles de la aplicación, 3. El usuario presiona el botón "Photon Maps", 4. El usuario coloca los valores deseados, 5. El usuario presiona el botón "Recalculate"

Tabla 4 – Descripción caso de uso Recalcular *Photon Map*

Pre-condición	Haber iniciado la aplicación
Nombre	Recalcular *Ray Tracer*
Actores	Usuario
Descripción	Permite recalcular los valores del *Ray Tracer* con los valores definidos en los controles del mismo
Flujo Básico	1. El usuario habilita los controles de la aplicación, 2. El usuario presiona el botón "Ray Tracer", 3. El usuario coloca los valores deseados, 4. El usuario presiona el botón "Recalculate"

Tabla 5 – Descripción caso de uso Recalcular *Ray Tracer*

Pre-condición	Haber iniciado la aplicación
Nombre	Dibujar Fotones
Actores	Usuario
Descripción	Dibuja todos los fotones que se encuentran en el *Photon Map*
Flujo Básico	1. Se muestra la escena al usuario, 2. El usuario habilita los controles de la aplicación, 3. El usuario presiona el botón "Phots_3D", 4. La aplicación dibuja los fotones en la escena

Tabla 6 – Descripción caso de uso Dibujar Fotones

Pre-condición	Haber habilitado los controles de la aplicación
Nombre	Agregar Figura
Actores	Usuario
Descripción	Permite agregar distintas figuras a la escena
Flujo Básico	1. Se muestra la escena al usuario, 2. El usuario habilita los controles de la escena, 3. El usuario presiona el botón "Add Figures", 4. El usuario selecciona la figura que quiere agregar, 5. El usuario coloca los valores de la figura a agregar, 6. El usuario presiona el botón "Create"

Tabla 7 – Descripción caso de uso Agregar Figura

Pre-condición	Haber habilitado los controles de la aplicación
Nombre	Eliminar Figura
Actores	Usuario
Descripción	Permite eliminar figuras de la escena
Flujo Básico	1. Se muestra la escena al usuario, 2. El usuario habilita los controles de la escena, 3. El usuario selecciona la figura a eliminar, 4. El usuario selecciona el botón "Delete_Selected"

Tabla 8 – Descripción caso de uso Eliminar Figura

4.3 DESARROLLO DEL PROTOTIPO INICIAL

Para el prototipo inicial, se implementaron las opciones de agregar figuras a la escena y modificar la luz puntual en intensidad, alcance, color y posición con respecto a la escena. Al momento de agregar una figura a la escena, esta se crea en una posición aleatoria de la escena, con tamaño y color aleatorios. Una vez creada una figura esta puede ser trasladada y rotada.

En la **Figura 20** podemos visualizar la interfaz en donde se encuentra la escena y los controles mencionados.

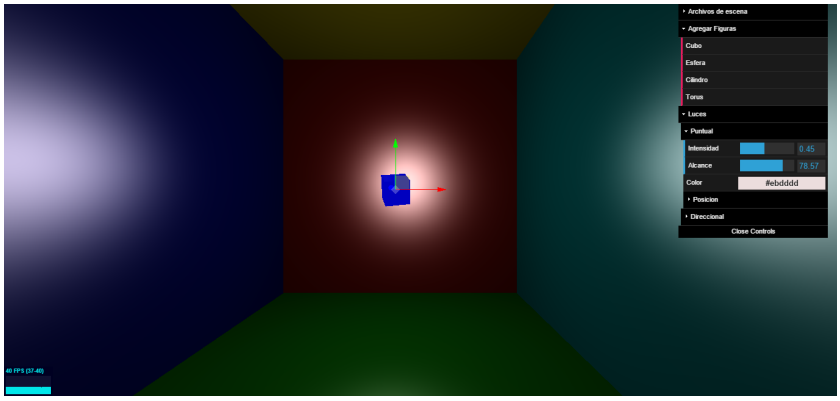

Figura 20 – Primer Prototipo

4.4 DESARROLLO DEL SEGUNDO PROTOTIPO

Para el desarrollo de este prototipo se agregaron las funcionalidades para aplicar y recalcular tanto el *Photon Map* como el *Ray Tracer*, además de poder dibujar los fotones que se encuentran almacenados en el *Photon Map* en la escena. Al momento de agregar una figura se modificó para que dicha figura se creara dentro de la caja en la escena.

Para la implementación del algoritmo de *Photon Mapping*, el primer paso fue la generación de los fotones, tanto su origen como su dirección se distribuyeron uniformemente a través de la superficie de la fuente de luz y la escena respectivamente. En la **Figura 21** se puede observar un pequeño extracto de pseudocódigo referente a la lógica tras la creación de un fotón y su trazado por la escena.

```
InitPhotonRays(){
    For( i=0; i<NumPhotones; i++){
        foton= new Photon();
        foton.origen= origenRand();
        foton.direccion=direccionRand();
        fotonMap.store(foton);
        tracePhoton(foton,0);
    }
    kdtree.build(photonMap);
}

tracePhoton(photon,profundidad){
    If profundidad<MaxProfundidad
        //intersecta los photones contra las supeficies de la escena
        //almacena el foton que haya golpeado una superficie
        //crea un nuevo foton y le calcula su nueva dirección o lo absorbe
        If valorRandom < limite //se traza nuevamente
            tracePhoton(nuevoFoton,profundidad+1);
        Else //es absorbido
            return;
}
```

Figura 21 – Pseudocódigo algoritmo de *Photon Mapping*
(Creación y trazado de fotones por la escena)

Tras la creación del *Photon Map* y su almacenamiento en el *kd-tree*, se procedió al desarrollo del *Ray Tracer*, el cual emitiría rayos desde la cámara hacia la escena y junto con la información guardada en el *Photon Map* se realizaron los cálculos para determinar la irradiancia que cada punto visible en la escena tendría. En la **Figura 22** se puede ver el pseudocódigo correspondiente a la lógica tras la generación y dirección de los rayos a ser emitido por el *Ray Tracer*.

```
InitCamRays(){
    For i=0;i<widthRes;i++){
        For j=0;j<heightRes;j++){
            rayo=new Ray();
            rayo.origen= calcOrigen();
            rayo.direccion= direction();
            traceRay(rayo); /*traza el rayo por la escena, calcula los valores
            de irradiancia para cada punto de la imagen y genera la imagen final*/
        }
    }
}
```

Figura 22 – Pseudocódigo del algoritmo de *Photon Mapping*
(Creación y trazado de rayos desde la cámara)

En la **Figura 23** se pueden observar algunos de los cambios realizados sobre los controles de la aplicación.

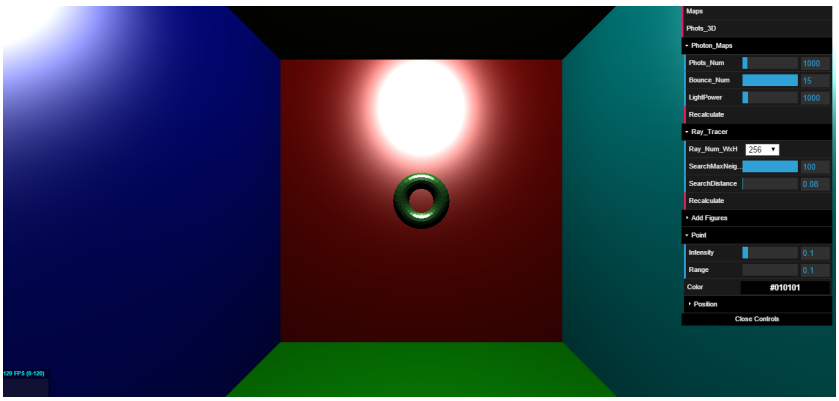

Figura 23 – Segundo Prototipo

4.5 DESARROLLO DEL PROTOTIPO FINAL

En el prototipo final se agregaron los efectos de reflexión, refracción y fresnel. Se agregó también la capacidad de definir las dimensiones, posición, color, opacidad, coeficiente de refracción y reflexión para cada figura de la escena, asi como también la funcionalidad de borrar una figura previamente seleccionada. Para este prototipo se mejoró la distribución de los fotones en la escena y se arreglaron errores en la emisión de los rayos del *Ray Tracer*. Por último se agregó un botón, el cual recalcula tanto el *Photon Map* como el *Ray Tracer* con los valores definidos en los controles correspondientes.

En la **Figura 24** se muestra el prototipo final con todos sus controles.

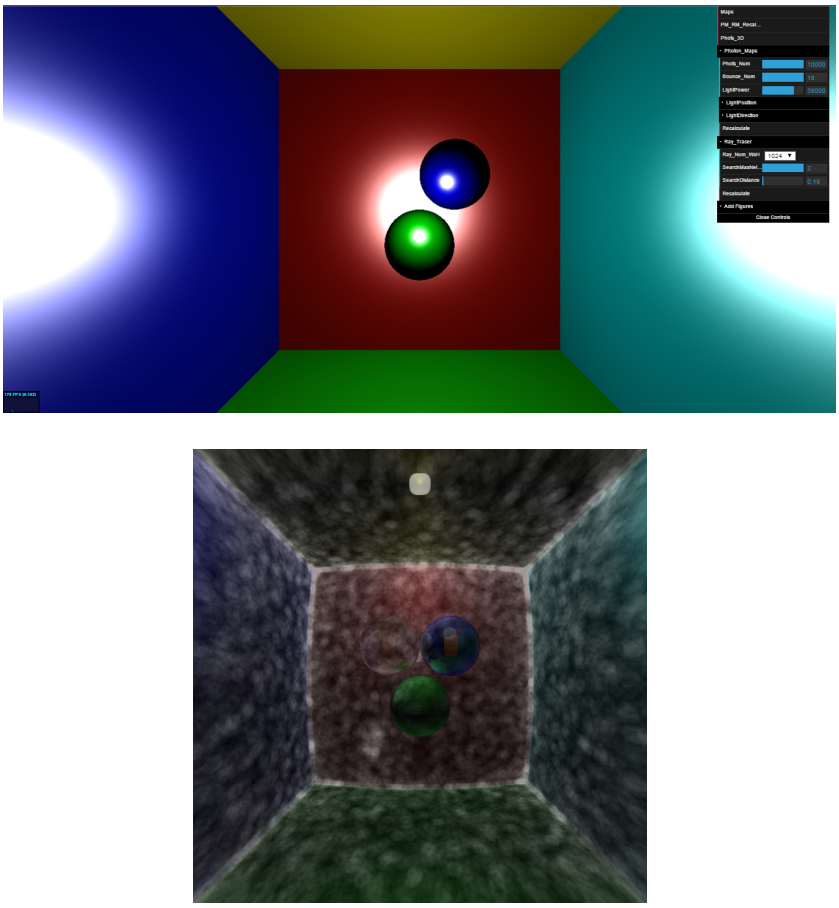

Figura 24 – Prototipo Final
(Interfaz e imagen resultado)

5 PRUEBAS

Capítulo 5

Se realizaron tanto pruebas cuantitativas como cualitativas a la aplicación, haciendo variaciones entre los parámetros de la escena y la cantidad de esferas en la misma.

5.1 PRUEBAS CUANTITATIVAS

En las primeras pruebas cuantitativas realizadas se variarion todos los parámetros para determinar como estos afectan en tiempo cuando están en su máximo valor. Estas pruebas fueron realizadas con tres esferas.

En la **Tabla 9** podemos observar las mediciones de las pruebas.

Nro. Prueba	1	2	3	4	5	6
Total Fotones	10.000	5.000	5.000	5.000	5.000	5.000
Número de rebotes	9	15	9	9	9	9
Potencia de la luz	30.000	30.000	50.000	30.000	30.000	30.000
Resolución	512x512	512x512	512x512	1024x1024	512x512	512x512
Cantidad máxima de vecinos	60	60	60	60	100	60
Distancia	0.6	0.6	0.6	0.6	0.6	1
Tiempo en minutos	2:00	2:32	2:17	8:50	2:31	2:56
Nro. Figura	25	26	27	28	29	30

Tabla 9 – Pruebas Cuantitativas variando todas las variables

Se puede notar que la variable que más afecta en tiempo es la resolución, debido a que esta es la cantidad de rayos que se emiten hacia la escena.

A continuación se muestran las respectivas imágenes de las pruebas.

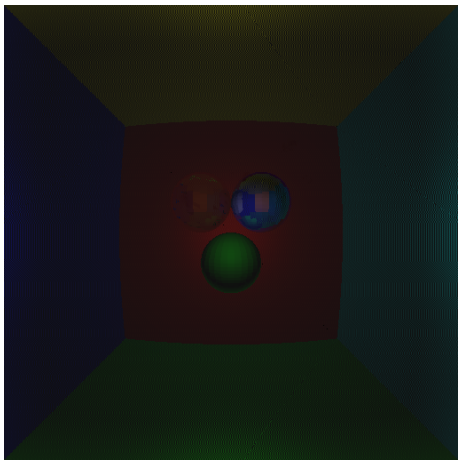

Figura 25 – Prueba Cuantitativa valor máximo de fotones

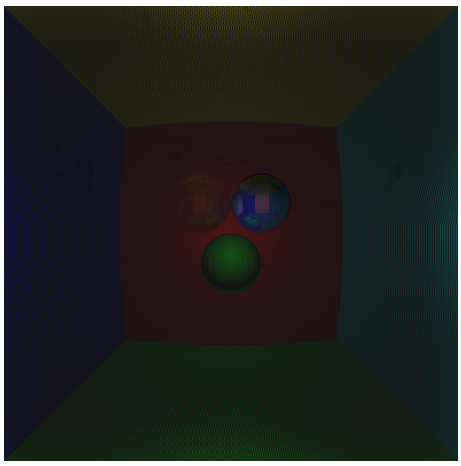

Figura 26 – Prueba Cuantitativa valor máximo de rebotes

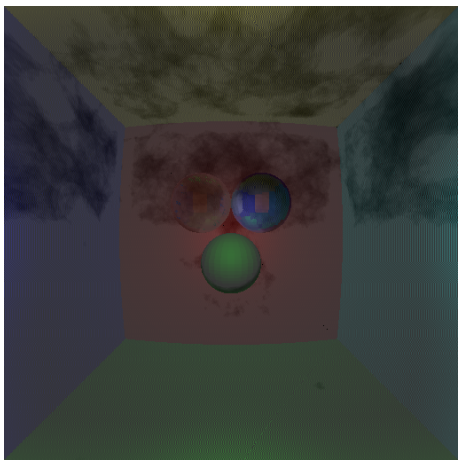

Figura 27 – Prueba Cuantitativa valor máximo de potencia de la luz

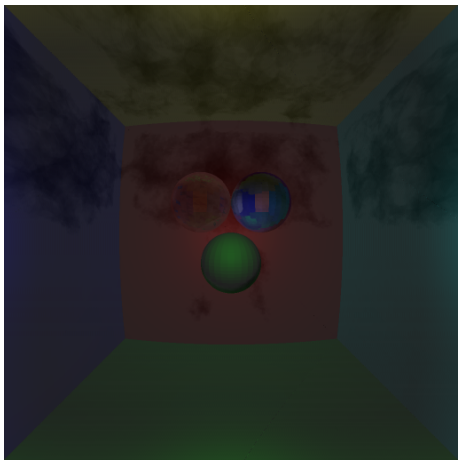

Figura 28 – Prueba Cuantitativa valor máximo de resolución

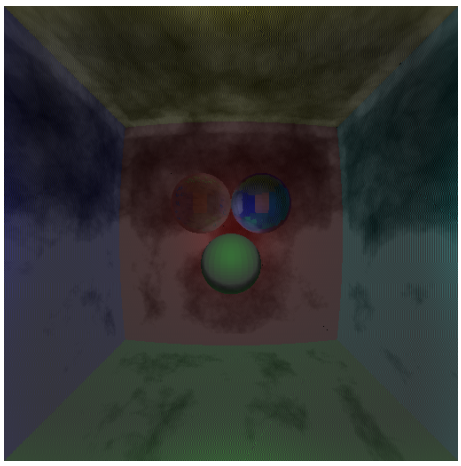

Figura 29 – Prueba Cuantitativa valor máximo de vecinos

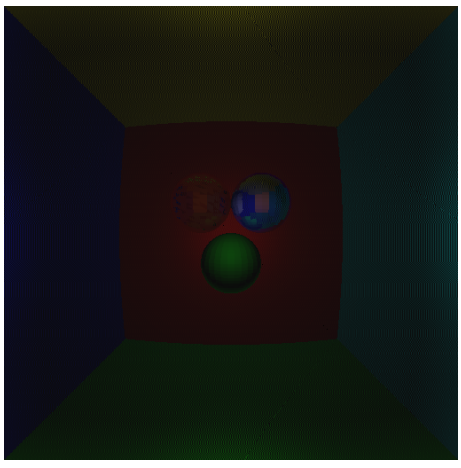

Figura 30 – Prueba Cuantitativa valor máximo de distancia

En las pruebas en donde se colocaron los valores máximos de la potencia de la luz, resolución y valor máximo de vecinos fueron en las que se obtuvieron mayores cambios con respecto a las imágenes resultantes.

En la **Figura 31** se muestra la gráfica de la tabla anterior mostrando cada prueba contra el tiempo de ejecución de la misma. Nuevamente se puede observar como la prueba 4, en la cual se colocó la resolución máxima, es la que más tardó en ejecutarse.

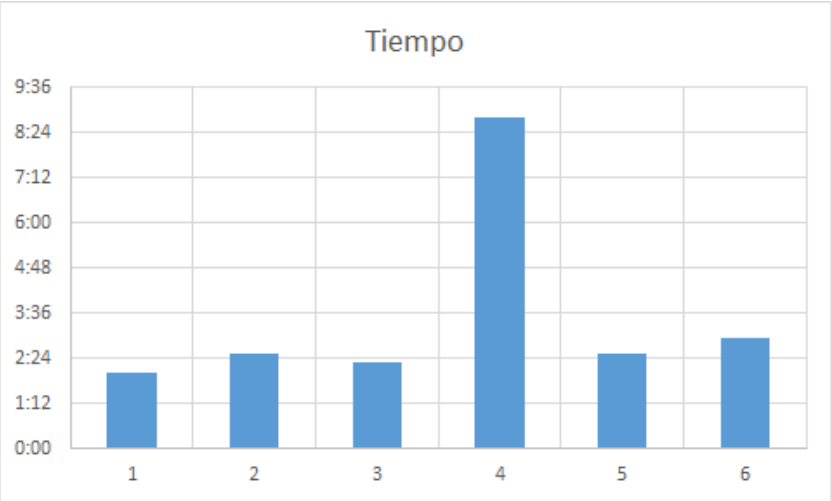

Figura 31 – Gráfica de las pruebas en relación al tiempo de ejecución

Luego de determinar cuáles son los valores que afectan en tiempo la ejecución de la aplicación, se procedió a realizar pruebas variando solo los párametros de la cantidad total de fotones y el número de rebotes de la luz, dejando los otros párametros fijos en los siguientes valores: potencia de la luz 30000, resolución 1024, distancia 0.2 y cantidad máxima de vecinos 100. Estas pruebas se realizaron con una, dos y tres esferas. En la **Tabla 10** se muestran los resultados obtenidos en las mediciones.

Se puede notar que al incrementar la cantidad total de fotones y la cantidad de rebotes de la luz el tiempo de ejecución va aumentando, siendo así las pruebas en donde se tienen 10000 fotones y 15 rebotes las que más tardan en ejecutar.

Las siguientes figuras son las imágenes resultantes de las pruebas realizadas. Están agrupadas según la cantidad de fotones en la escena y la cantidad de esferas respectivamente.

Nro. Prueba	1	2	3	4	5	6	7	8	9
Total Fotones	8.000	8.000	8.000	9.000	9.000	9.000	10.000	10.000	10.000
Número de rebotes	9	12	15	9	12	15	9	12	15
Tiempo en minutos, 1 esfera	2:15	3:37	4:57	4:09	4:47	5:37	4:2	5:03	5:53
Tiempo en minutos, 2 esferas	2:26	3:45	5:11	4:23	4:57	5:38	4:26	5:17	6:07
Tiempo en minutos, 3 esferas	2:26	3:51	5:25	4:10	5:05	5:51	4:40	5:41	6:26
Nro. Figura	32	33	34	35	36	37	38	39	40

Tabla 10 – Pruebas Cuantitativas variando la cantidad de esferas

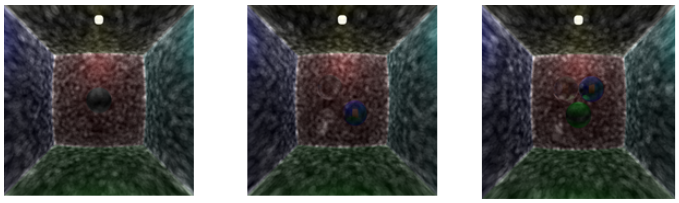

Figura 32 – Prueba Cuantitativa con 8.000 fotones y 9 rebotes
(a) Prueba con 1 esfera
(b) Prueba con 2 esferas
(b) Prueba con 3 esferas

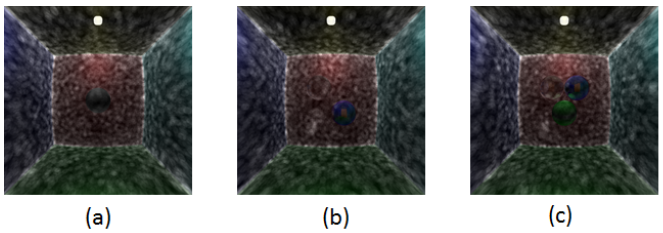

Figura 33 – Prueba Cuantitativa con 8.000 fotones y 12 rebotes
(a) Prueba con 1 esfera
(b) Prueba con 2 esferas
(b) Prueba con 3 esferas

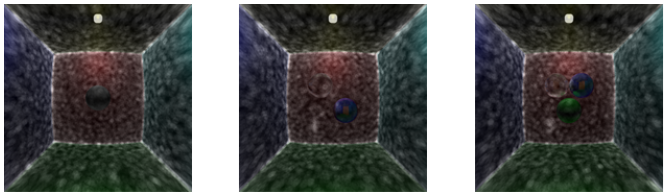

Figura 34 – Prueba Cuantitativa con 8.000 fotones y 15 rebotes
(a) Prueba con 1 esfera
(b) Prueba con 2 esferas
(b) Prueba con 3 esferas

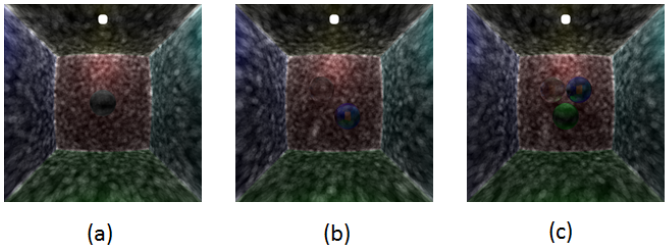

Figura 35 – Prueba Cuantitativa con 9.000 fotones y 9 rebotes
(a) Prueba con 1 esfera
(b) Prueba con 2 esferas
(b) Prueba con 3 esferas

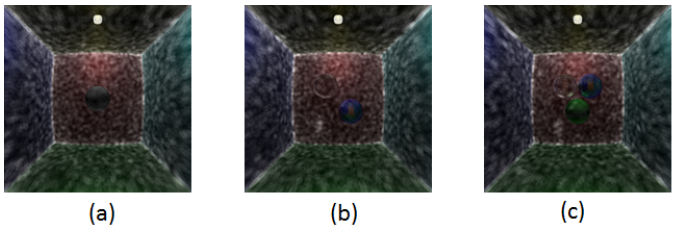

Figura 36 – Prueba Cuantitativa con 9.000 fotones y 12 rebotes
(a) Prueba con 1 esfera
(b) Prueba con 2 esferas
(b) Prueba con 3 esferas

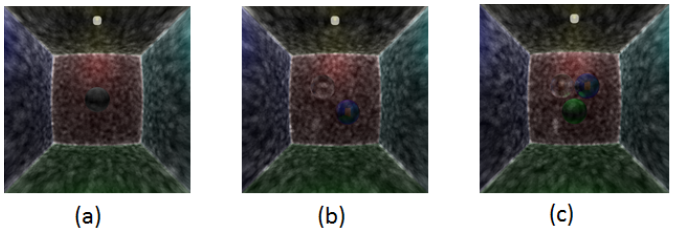

Figura 37 – Prueba Cuantitativa con 9.000 fotones y 15 rebotes
(a) Prueba con 1 esfera
(b) Prueba con 2 esferas
(b) Prueba con 3 esferas

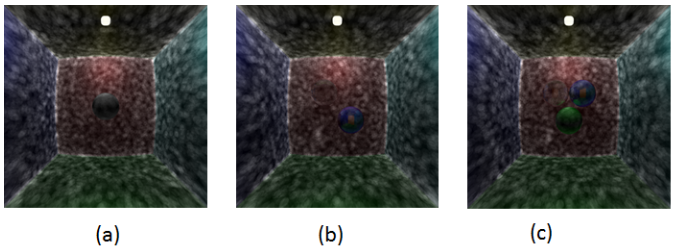

Figura 38 – Prueba Cuantitativa con 10.000 fotones y 9 rebotes
(a) Prueba con 1 esfera
(b) Prueba con 2 esferas
(b) Prueba con 3 esferas

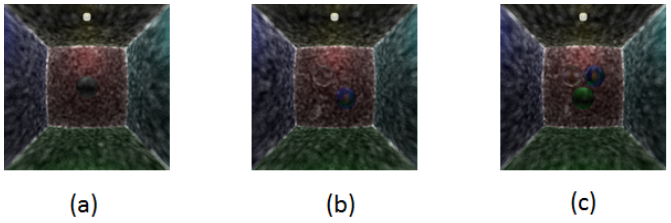

Figura 39 – Prueba Cuantitativa con 10.000 fotones y 12 rebotes
(a) Prueba con 1 esfera
(b) Prueba con 2 esferas
(b) Prueba con 3 esferas

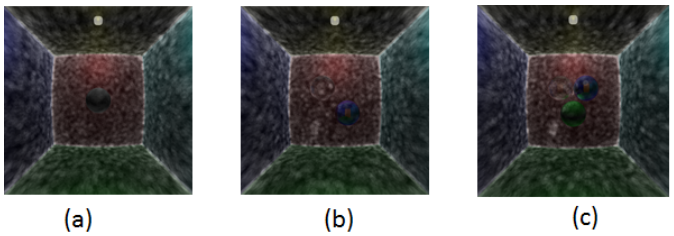

Figura 40 – Prueba Cuantitativa con 10.000 fotones y 15 rebotes
(a) Prueba con 1 esfera
(b) Prueba con 2 esferas
(b) Prueba con 3 esferas

5.2 PRUEBAS CUALITATIVAS

Para la realización de las pruebas cualitativas se variaron tanto la cantidad de fotones, como el número de rebotes en la escena lo cual se puede observar en la **Tabla 11**.

Nro. Prueba	1	2	3	4	5	6	7	8	9
Total Fotones	8.000	8.000	8.000	9.000	9.000	9.000	10.000	10.000	10.000
Número de rebotes	9	12	15	9	12	15	9	12	15
Tiempo en minutos	2.15	3.37	4.57	4.09	4.47	5.37	4.23	5.03	5.53
Nro. Figura	41	42	43	44	45	46	47	48	49

Tabla 11 – Pruebas Cualitativas con 1 esfera

A continuación se muestran las imágenes resultantes de las pruebas anteriormente realizadas.

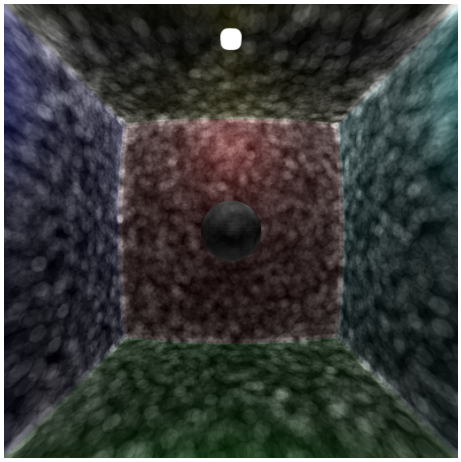

Figura 41 – Prueba Cualitativa con 8.000 fotones y 9 rebotes

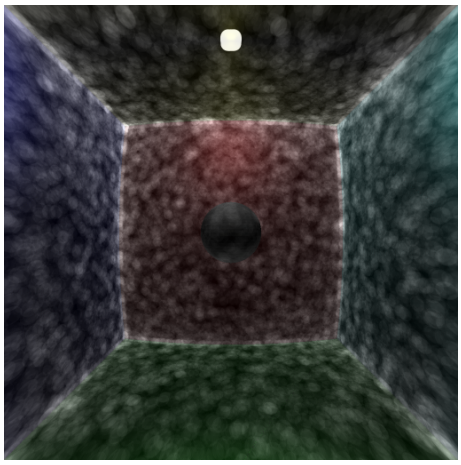

Figura 42 – Prueba Cualitativa con 8.000 fotones y 12 rebotes

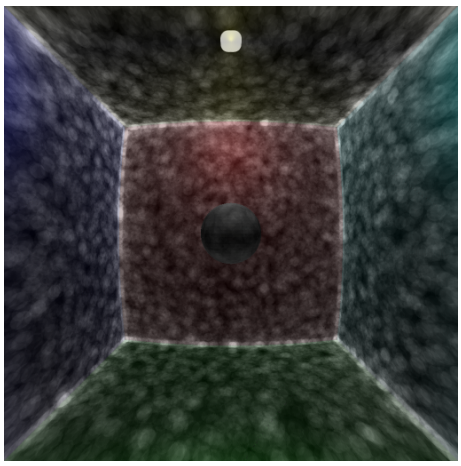

Figura 43 – Prueba Cualitativa con 8.000 fotones y 15 rebotes

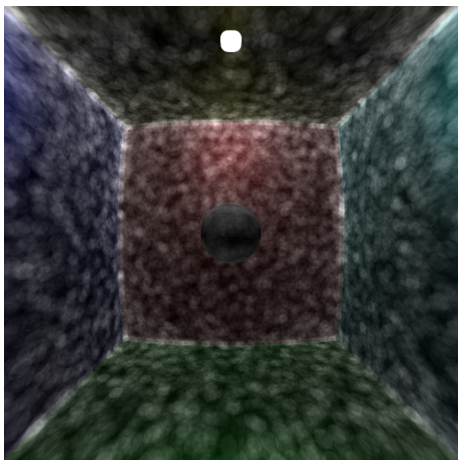

Figura 44 – Prueba Cualitativa con 9.000 fotones y 9 rebotes

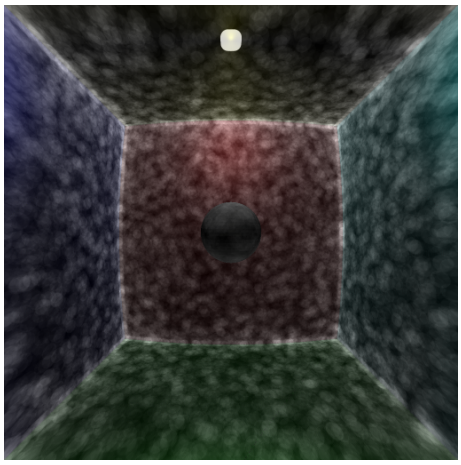

Figura 45 – Prueba Cualitativa con 9.000 fotones y 12 rebotes

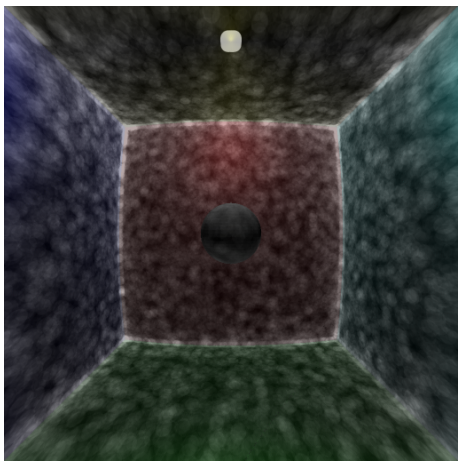

Figura 46 – Prueba Cualitativa con 9.000 fotones y 15 rebotes

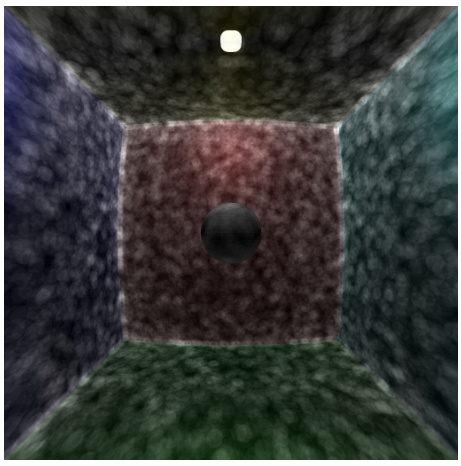

Figura 47 – Prueba Cualitativa con 10.000 fotones y 9 rebotes

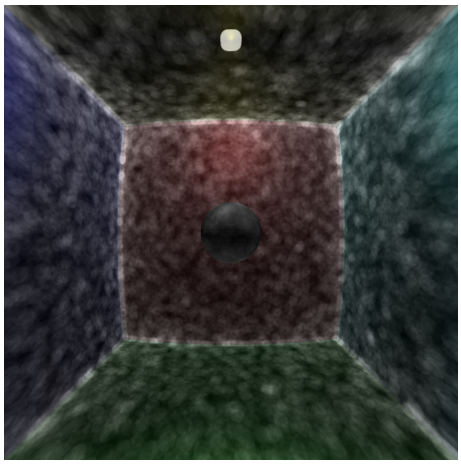

Figura 48 – Prueba Cualitativa con 10.000 fotones y 12 rebotes

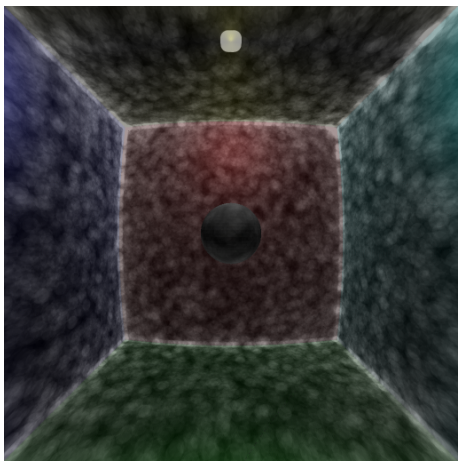

Figura 49 – Prueba Cualitativa con 10.000 fotones y 15 rebotes

Tras observar las imágenes relacionadas con las mediciones se puede apreciar que la variación de la cantidad de fotones y el número de rebotes en sus valores más altos, no afecta de manera importante la calidada de las imágenes.

6 CONCLUSIONES

Se logró desarrollar una aplicación que utilizando *Photon Mapping* permitiera el despliegue y visualización de escenas 3D usando WebGL. Dicha aplicación permite al usuario la configuración de varios parámetros relacionados al cálculo del *Photon Map* y del *Ray Tracer*, con el fin de que el usuario pueda generar imágenes de varias calidades y tener un mayor entendimiento del algoritmo de iluminación global *Photon Mapping*.

Se analizaron las herramientas tecnológicas que permitían desarrollar una solución de software para el problema planteado. Al seleccionar la herramienta acorde, se procedió a diseñar la solución, basada en un proceso de desarrollo ágil. Seguido a esto, se construyó la solución de software basado en un modelo orientado a prototipos. Finalmente, se realizaron pruebas cualitativas y cuantitativas con las cuales se pudo encontrar el párametro que más afecta al tiempo de ejecución de la aplicación, el cual es la resolución del *Ray Tracer*.

En las pruebas cualitativas observamos que variar solo entre los valores más altos de la cantidad de fotones y el número de rebotes no afecta de forma importante la calidad de las imágenes por lo que se recomienda colocar 8000 fotones y 9 rebotes de luz, de esta manera se tendrá una buena calidad de imagen con un menor tiempo de ejecución.

6.1 RECOMENDACIONES

Para el correcto funcionamiento de la aplicación se recomienda un computador que al menos posea un procesador Intel Core i5 de 4 núcleos a 3.2Ghz o su equivalente, 4GB de RAM y un procesador gráfico Nvidia GTX430 o su equivalente.

6.2 TRABAJOS FUTUROS

Tras la culminación de este trabajo de grado se propone para la continuación y mejora del proyecto:

- Realizar una optimización general del código, con el fin de mejorar los tiempos de respuesta y aumentar la cantidad de fotones y rayos que la aplicación pueda manejar. Actualmente, la aplicación puede manejar hasta 10000 fotones, 15 rebotes

y una resolución de 1024x1024 para la imagen resultante, con cantidades superiores no se puede garantizar el correcto funcionamiento de la aplicación

- Hacer uso de una base de datos para la creación de sesiones, permitiendo que un usuario pueda guardar una escena creada para poder consultarla posteriormente
- Modificar la capacidad del *Ray Tracer* con el fin de poder emitir rayos desde el punto de vista de la cámara, sin importar su posición y sentido
- Añadir la posibilidad de agregar geometrías más complejas
- Agregar fuentes de luz dinámicamente

REFERENCIAS

[1] [Fecha de consulta: 29 de Septiembre de 2014]
Disponible en: http://www.w3.org/TR/2011/WD-html5-diff-20110405/

[2] [Fecha de consulta: 04 de Abril de 2015]
Disponible en: http://www.html5test.com/compare/browser/index.html

[3] [Fecha de consulta: 29 de Septiembre de 2014]
Disponible en: http://librosweb.es/javascript/capitulo_1.html

[4] [Fecha de consulta: 22 de Abril de 2015]
Disponible en: http://json.org/json-es.html

[5] [Fecha de consulta: 29 de Septiembre de 2014]
Disponible en: http://www.desarrolloweb.com/manuales/manual-jquery.html

[6] [Fecha de consulta: 13 de Febrero de 2015]
Disponible en: http://threejs.org/

[7] [Fecha de consulta: 29 de Septiembre de 2014]
Disponible en: http://www.desarrolloweb.com/articulos/303.php

[8] [Fecha de consulta: 29 de Septiembre de 2014]
Disponible en: http://laravel.com/docs/4.2/introduction#where-to-start

[9] [Fecha de consulta: 03 de Octubre de 2014]
Disponible en: https://www.khronos.org/

[10] Henrik Wann Jensen, Niels Jorgen Christensen. *A Practical Guide to Global Illumination using Photon Maps.* Siggraph 2000, Course 8, 23 de Julio de 2000.

[11] Diego Cantor, Brandon Jones. *WebGL Beginner's Guide.* Packt Publishing, Junio 2012.

[12] Tony Parisi. *WebGL: Up and Running.* O'Reilly Media, Agosto 2012.

[13] Sumeet Arora. *WebGL Game Development.* Packt Publishing, Abril 2014.

[14] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping.* A K Peters, 2001.

[15] Andrew W. Moore. *An intoductory tutorial on kd-trees* Carnegie Mellon University, 1991.

[16] Greg Ward, Okan Arikan, Henrik Wann Jensen. *Practical Global Illumination with Irradiance Caching.* Siggraph 2007, Course 16.

[17] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, Pat Hanrahan *Photon Mapping on Programmable Graphics Hardware.* Graphics Hardware (2003).

[18] [Fecha de consulta: 12 de Febrero de 2015]
Disponible en: http://madebyevan.com/webgl-path-tracing/

[19] [Fecha de consulta: 12 de Febrero de 2015]
Disponible en: http://codeflow.org/entries/2012/aug/25/webgl-deferred-irradiance-volumes/