Universidad de las Ciencias Informáticas
Facultad 5 Entornos Virtuales

PROPUESTA DE ANÁLISIS Y DISEÑO DE LA BIBLIOTECA DE EFECTOS DE POST-PROCESAMIENTO PARA ENTORNOS VIRTUALES

Trabajo de Diploma para optar por el título de
Ingeniero Informático

Autor: Yaraíza Reyes Rodríguez.

Tutor: Ing. Frank Puig Placeres.

Ciudad de la Habana

Junio 2008

"...Lo fundamental es hacer algo nuevo cada día

y luego perfeccionar lo que se ha hecho

el día anterior..."

Ché

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al <nombre área> de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Yaraíza Reyes Rodríguez

_____

Firma del autor

Frank Puig Placeres

_____

Firma del tutor

DATOS DE CONTACTO

Nombre y Apellidos: Frank Puig Placeres.
Edad: 26.
Ciudadanía: cubano
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Informática
Categoría Docente: Profesor Instructor
E-mail: fpuig@uci.cu

Graduado de la UCI, con seis años de experiencia en el tema de la Gráfica Computacional, Coautor de los libros ShaderX5, Game Programming Gems 5, Game Programming Gems 6 y AI Wisdoms .

AGRADECIMIENTOS

A Fidel, por hacerme partícipe de este proyecto tan emprendedor.

A la UCI, por ser mi segunda casa, por forjarme como profesional.

A mí, por mi perseverancia y esfuerzo durante estos 5 largos años de estudio.

A mis padres, por estar presente en todo momento, por todo su apoyo y amor, por confiar siempre en mí, los quiero mucho muchooo.

A mi tía Cacha, por ser mi segunda madre, por estar siempre pendiente de mí.

A Ariel el "Virtual", por ser el hermano que la vida me permitió elegir.

A Albe, que aunque no me ayudó en nada, es una persona muy especial en mi vida.

A mi tutor, por insertarme en el mundo de la Realidad Virtual, por sus consejos y ayuda.

A mis amigos, por su apoyo incondicional.

A mis compañeros de laboratorio, en especial a Leonel, por hacer más fáciles estos días de intensa agonía.

A mis compañeros de estudios, mi eterno grupo 5503, por ser el mejor grupo que se pudiera tener, por hacer de la UCI más que una simple universidad.

A mis compañeras de cuarto, por soportarme día a día, por toda su preocupación.

A los que colaboraron en la realización de esta tesis.

A todos ustedes muchas gracias.

DEDICATORIA

A mis padres, que han sido conscientes o no, los principales responsables de mi educación, mi principal fuente de inspiración.

A mí que he sido la principal protagonista de esta aventura.

A la UCI que hace de este sueño una realidad.

RESUMEN

Los Sistemas de Realidad Virtual (SRV) han tenido una elevada connotación económica y social en la vida del hombre durante las últimas dos décadas. Dado el realismo que intentan simular representando escenas virtuales, se hace evidente la utilización de los mismos para solucionar problemas en importantes esferas como la medicina, la educación, la defensa y el entretenimiento.

El realismo en los entornos virtuales (EV) es un factor clave para su aceptación por parte del usuario, creándole al mismo un sentimiento de presencia real en el entorno. Con el fin de aumentar el nivel de realismo en los SRV se recrearon cientos de efectos y técnicas de graficación mediante el uso casi exclusivo del hardware gráfico lo que permite su rápida visualización.

Este trabajo aborda el diseño de una biblioteca de clases para la simulación de los efectos de post-procesamiento en los SRV, teniendo en cuenta el incremento de la calidad visual de los EV con el mínimo coste en tiempo de procesamiento. Para alcanzar esta meta se estudian y describen los efectos de post-procesamiento más usados actualmente; se trabaja en la investigación de las potencialidades que brindan las nuevas tecnologías de hardware gráfico para la representación tridimensional en tiempo real y se generan los artefactos del flujo de trabajo de diseño que guiarán la implementación de una herramienta que permita incluir a los EV un conjunto de efectos especiales y de esta forma mejorar la calidad de visualización de las simulaciones en tiempo real. Además este trabajo tiene el propósito de documentar a los proyectos que corresponden al perfil de “Realidad virtual”.

PALABRAS CLAVE

Diseño, efectos, post-procesamiento, realismo, hardware gráfico, sistemas de realidad virtual.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

INTRODUCCIÓN

Desde el surgimiento y desarrollo de las técnicas de la Realidad Virtual (RV), la Informática gráfica ha elevado la calidad de las aplicaciones que simulan situaciones del mundo real, imprimiéndole a las mismas un mayor realismo.

Una de las principales aplicaciones en este campo se muestra en el área de entrenamiento o capacitación, donde el usuario tiene la oportunidad de interactuar con entornos virtuales(EV) sin tener que ponerse en peligro o pagar un alto costo para adquirir cierta experiencia en un área determinada, además la RV se puede utilizar para elaborar prototipos de productos, en el campo de la medicina (simulación quirúrgica, simulaciones para tratamientos psiquiátricos), en la visualización de datos (como la modelación de moléculas), en el entrenamiento de profesionales (simuladores de tiro y vuelo), en aplicaciones de entretenimiento (Video juegos), entre otras.

Es un hecho que la RV se desarrolla aceleradamente y a cada momento nos enfrentamos con técnicas cada vez más novedosas para la modelación 3d de alta calidad, la confección de complejos modelos matemáticos que ayudan a la visualización de situaciones de la vida real, la elaboración de algoritmos de inteligencia artificial para la creación de objetos dinámicos, entre otros elementos que han propiciado el incremento de la interactividad y el realismo en los entornos virtuales.

Otro de los factores fundamentales en los últimos años en el desarrollo de los mundos virtuales, son los avances tecnológicos ocurridos en el hardware gráfico. Esta mejora técnica, está dada por el reemplazo de los algoritmos y rutinas estándar para la transformación y proyección matemática de vértices 2D y 3D, el procesamiento de imágenes, entre otros; comúnmente conocidos como “Tubería de función fija” (*Fixed Function Pipeline*) por las tuberías de función programable (*Programmable Function Pipeline*) a través de las cuales es posible implementar nuevas rutinas que permiten personalizar la forma en que se manipularán las transformaciones en los vértices y se podrán elegir los colores de cada pixel a través de los *Vertex Shader* y los *Pixel Shader* respectivamente. Estos códigos se ejecutan en la unidad de procesamiento gráfico (*GPU*) que se encuentra en la tarjeta de video, a diferencia de las aplicaciones comunes que se ejecutan en la Unidad Central de Procesamiento (CPU).

El surgimiento de los *Programmable Function Pipelines* ha producido a su vez un cambio en la investigación de las aplicaciones de visualización en tiempo real; buscándose no sólo un mayor

rendimiento en el procesamiento masivo de polígonos, sino también en la calidad de los materiales con que estos se representan. De esta forma, efectos de render como *antialiasing*, desenfoque por movimiento y proyección de sombras comienzan ha ser comunes en las simulaciones con el fin de incrementar el realismo en los entornos virtuales.

En la actualidad uno de los temas que más preocupa a los desarrolladores tanto de video-juegos como de simuladores es la creación de entornos virtuales con un alto grado de realismo debido a la gran cantidad de recursos que consumen las técnicas y algoritmos para lograr una excelente calidad de visualización y en muchas ocasiones no cumplen con las expectativas de los usuarios finales. La modelación y simulación de efectos especiales, como explosiones, humo, lluvia y otros, constituyen una parte esencial en la creación de aplicaciones que simulan con gran veracidad situaciones del mundo real permitiéndole al usuario transportarse dentro de la simulación. Una de las técnicas más interesantes en esta área es la simulación de efectos de post-procesamiento de imágenes, esta práctica es muy usada en el desarrollo de la industria de entretenimiento y se define a sí misma como un área relativamente simple en la Informática gráfica.

Tradicionalmente el procesamiento de imágenes se adueñaba de una cantidad significativa del poder del procesador en el CPU, pero con el surgimiento de los *Programmable Function Pipelines* estos efectos pueden realizarse ahora de forma más eficiente. (Microsoft Corporation, 2005)

La creación de una arquitectura de post-procesamiento de imágenes es una poderosa forma para personalizar y lograr un alto grado de realismo en la apariencia tanto de simuladores, video-juegos o simplemente cualquier imagen que se quiera manipular y de igual manera le adiciona a los entornos virtuales una gran cantidad de características que van desde un simple Bloom para lograr que los brillos de una escena te deslumbren hasta la manipulación del espacio y perturbación de las imágenes. Esta técnica por sí misma consume un valor de tiempo constante ya que sólo depende de la complejidad de la escena. (Microsoft Corporation, 2005)

La Universidad de las ciencias Informáticas (UCI) en conjunto con la empresa de SIMPRO (Simuladores profesionales) es uno de los centros en el país que enfila sus pasos en este mundo de la ilusión y realidad mezclada. La UCI esta dividida en 10 facultades las cuales responden a un perfil determinado, la Facultad 5 tiene como perfil los “Entornos Virtuales” y los proyectos que se desarrollan en esta área ya sea el “Simulador Quirúrgico” o el Grupo de Proyectos de Juegos Virtuales (GPJV), por solo citar algunos, necesitan que sus escenas tengan la suficiente calidad de visualización y de esta forma aumentaría la demanda de sus productos. La implementación de efectos visuales es una de las

líneas fundamentales de trabajo de los proyectos productivos con el propósito de satisfacer las expectativas de los usuarios finales y emerger al mercado mundial, con este fin se ha desarrollado una herramienta de visualización con varios subsistemas incluidos como el de sonido, el de inteligencia artificial y recientemente se diseñó la propuesta de un subsistema para la visualización de efectos especiales profundizando en los efectos de explosiones y fuego aunque no ha sido incluido en dicha herramienta; pero no posee un subsistema que permita la visualización de los efectos de post-procesamiento, así como existe un desconocimiento de las potencialidades que nos brinda la utilización de esta técnica para los EV.

A raíz de lo anterior se identifica como **problema científico:** ¿Cómo las nuevas tecnologías del hardware gráfico, pueden ayudar a lograr un alto nivel de realismo e inmersión en la creación y visualización de entornos virtuales?

Objeto de estudio.

Procesos de generación de efectos para sistemas de realidad virtual.

Campo de acción.

Las técnicas de efectos de post-procesamiento para sistemas de realidad virtual.

Como **objetivo general** se plantea: Diseñar una biblioteca capaz de aplicar los efectos de post procesamiento en los entornos virtuales.

Con este trabajo se defiende la idea de que si se hace un correcto análisis y diseño de una biblioteca de efectos de Post-procesamiento, se podrá implementar una herramienta que incremente el *realismo* de los sistemas de realidad virtual.

Para el cumplimiento del objetivo propuesto se trazan las siguientes **tareas de investigación**:

1. Analizar los lenguajes de programación, librerías gráficas y técnicas relacionadas con la simulación de los efectos de post-procesamiento.
2. Seleccionar las herramientas, metodologías y lenguajes para el diseño de la biblioteca de efectos de post-procesamiento.
3. Analizar los efectos de post-procesamiento más usados actualmente.
4. Describir los efectos de post-procesamiento más usados actualmente.
5. Realizar un diagrama de clases lo suficiente extensible que permita la inclusión de nuevos post-procesos.

6. Diseñar una biblioteca que permita la aplicación de los efectos de post-procesamiento de imágenes en los entornos virtuales.

Para el cumplimiento de las tareas de investigación propuestas se aplicaron algunos métodos científicos de investigación:

Métodos teóricos.

Analítico- Sintético: Son dos procesos inherentes al pensamiento que permite buscar la esencia de los fenómenos y los rasgos que lo caracterizan. Su objetivo en una investigación es analizar las teorías, documentos, etc.; permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio.

Histórico- Lógico: Estos procesos se basan en el estudio histórico del fenómeno, donde se analiza la trayectoria completa del mismo, su condicionamiento a los diferentes períodos de la historia. Este método expresa en forma teórica la esencia del objeto, explican la historia de su desarrollo y permiten unir el estudio de la estructura del objeto de la investigación con su concepción histórica.

Sistémico**:** mediante este método se modeló el objeto a través de la determinación de sus componentes, así como las relaciones que existan entre ellos, las que determinan no solo la estructura del objeto sino también su dinámica.

Los **posibles resultados** que se persiguen es obtener una descripción de los efectos de Post-procesamiento básicos en forma de catálogo para documentar a los proyectos que desarrollan entornos virtuales; obtener el diseño de una biblioteca de post-procesos que guíe la implementación de una herramienta que permita incluir a los entornos virtuales un conjunto de efectos visuales y de esta forma mejorar la calidad de visualización de las simulaciones en tiempo real. Además este trabajo sirve como material de estudio para futuras investigaciones y tiene el propósito de documentar a los proyectos que corresponden al perfil de “Realidad virtual”.

La tesis está estructurada en una introducción, cuatro capítulos, conclusiones y recomendaciones:

En el Capítulo 1, "Fundamentación teórica", se realiza un análisis del surgimiento y desarrollo de los efectos de post-procesamiento, así como su repercusión actual en el mundo de la simulación; se abordan los conceptos y técnicas fundamentales que apoyan el trabajo con estos efectos. Además se investigan las tendencias y tecnologías a nivel de hardware que se tendrán en cuenta en la elaboración de la propuesta.

En el Capítulo 2, “Técnicas para la simulación de los efectos de Post-procesamiento”, se realizará una panorámica acerca de las técnicas para la simulación de los efectos de post-procesamiento más usados actualmente; se realiza una descripción de los algoritmos y modelos de los post-procesos básicos, resaltando los diferentes pro y contra.

En el Capítulo 3, “Descripción de la solución propuesta”, se describe a través de diagramas como está conformado el sistema que se desea desarrollar, se detallan los aspectos funcionales y no funcionales con los que cumplirá la biblioteca, así como las características generales y aportes de la misma.

En el Capítulo 4, “Construcción del sistema propuesto”, se exponen los diagramas del flujo de trabajo de análisis, diseño e implementación que guían la construcción de la biblioteca, se detalla el diseño previo a la implementación y se describen los componentes en los que se implementará la biblioteca.

CAPÍTULO 1

FUNDAMENTACIÓN TEÓRICA.

La simulación y modelación de efectos es una de las áreas de la informática gráfica (IG) en que se investiga con más profundidad. Lograr una calidad visual aceptable que provoque al usuario una sensación de inmersión en un entorno virtual (EV) ha sido y será una de las mayores preocupaciones y objetivos de los desarrolladores de simulaciones. Con la rápida evolución de las tarjetas gráficas, simular efectos en tiempo real está al alcance de nuestras manos, quedando atrás los años en que estas técnicas fueran consideradas como prohibitivas. Pero para lograr una buena calidad de visualización las imágenes de los EV no deber ser simuladas en modo de alambre, sino empleando colores y texturas, e imprimiéndole a los escenarios un conjunto de efectos que permitan simular el mundo real.

La reciente generación del hardware gráfico nos ha trasportado a un nuevo nivel en la técnica de programación per-pixel, permitiendo a su vez que las simulaciones de gráficos en tiempo real posean una mayor sensación de realismo. Con la habilidad de realizar complejos cálculos matemáticos per-pixel está a nuestro alcance aplicar una variedad de efectos basados en texturas, antes o después que esta haya sido aplicada, tales como la técnica blur, motion blur, perturbación de texturas, entre otros efectos que son conocidos como Efectos de Post-procesamiento.

Este capítulo realiza un análisis de las principales técnicas y algoritmos relacionados con la simulación de efectos de Post procesamiento y la programación 2D. Se dará a conocer los antecedentes de los efectos de Post-procesamiento y su repercusión actual en la industria del entretenimiento.

1.1 Efectos de Post-Procesamiento. Concepto y antecedentes.

Para entender mejor el concepto de Efectos de post-procesamiento, primero se definirán los significados de los vocablos “Post”, “Procesamiento” y “Efectos”. El diccionario de la Real Academia de Lengua Española define post como “Significa 'detrás de' o 'después de'. Por ejemplo: *Posbélico, posponer, postónico.* A veces conserva la forma latina **post-.** *Postdorsal, postfijo*”; Procesamiento como la “Aplicación sistemática de una serie de operaciones sobre un conjunto de datos, generalmente por medio de máquinas, para explotar la información que estos datos representan” y al vocablo Efectos en su acepción de Efectos Especiales como “Técnica de algunos espectáculos, trucos o artificios para provocar determinadas impresiones que producen ilusión de la realidad”.

Después de analizadas las acepciones de estos vocablo y enmarcarlos en el campo de la IG el autor define a los *Efectos de Post procesamiento* como la *Técnica utilizada en la informática gráfica para agregar un conjunto de efectos visuales después que una escena es renderizada hacia una textura*.

Esta técnica es de gran utilidad para personalizar la apariencia de las simulaciones y permite mejorar la calidad y el realismo de las escenas. “En Direct3D de Microsoft el procesamiento de imágenes usualmente es como un post-procesamiento después que la interpretación de la imagen es completada. Una aplicación primero dibuja la escena hacia textura y como segundo paso procesa la textura junto con un shader para obtener una imagen diferente debido a una mejora o alteración”. (Microsoft Corporation, 2005)

Los antecedentes de la utilización de esta técnica en la IG se remontan al desarrollo de la API (Application Programming Interface - Interfaz de Programación de Aplicaciones) Directx 8.0 de la Microsoft que desarrolló la capacidad de usar los Pixel y Vertex Shader, los cuales podían implementar una gran cantidad de efectos visuales de alto impacto para los usuarios y posteriormente el lanzamiento de tarjetas gráficas para dar soporte a esta tecnología. Entre las compañías desarrolladoras de tarjetas gráficas se destaca la compañía NVIDIA siendo su chipset grafico GeForce 3 de la 3ra Generación (2001), una de las primeras GPU capaz de soportar esta API.

La compañía de Microsoft se enfocó precisamente en la tarea de lograr un mayor realismo en los entornos virtuales creando su API Directx 8.0. “Esta tecnología incluyó una gran variedad de avances significativos, pero el mayor de ellos fue la capacidad de programar el hardware gráfico, esto se puede hacer en dos puntos. La primera trabaja sobre los vértices (geometría) y se le conoce como los *Vertex Shader* (Sombreado por vértices) programables. La segunda opera sobre los pixels y se le conoce

como *Pixel Shader* (Sombreado por pixel), permitiendo aplicar efectos personalizados a cada pixel". (García Guzmán, 2004)

A finales de 1999 y principios de los 2000 la compañía NVIDIA dominó el mercado de las tarjetas gráficas, en este período se enfocó en la creación de nuevos y mejores chips gráficos, denominándolos GPU´s (Unidades de procesamiento grafico), las cuales tienen como función principal desempeñar con una mayor velocidad las operaciones matemáticas necesarias para visualizar el entorno; esta innovación da paso al surgimiento de diversos procesadores gráficos que fueron divididos en 4 generaciones. En el año 2001 aparecen las series GeForce 3, GeForce 4 de la 3ta Generación que incluye la programación a nivel del vértice, la configuración a nivel de pixel y el Directx 8; ya en el año 2002 aparece la 4ta Generación con la serie GeForce FX que incluye la programación per-pixel y el Directx 9; más tarde aparecen las GeForce 6 series hasta las actuales 8 series, esta última serie incluye el Directx 10. (García Guzmán, 2004)

Este significativo avance de la programación per-pixel puso a nuestro alcance la posibilidad de realizar un gran número de operaciones y efectos visuales en tiempo real como son los filtros de texturas, efectos de alto rango dinámicos, desenfoque, mapas de desplazamiento entre otros; los cuales son factibles de aplicar, ya que no sacrifican la precisión ni el rendimiento de los equipos.

Mediante el uso de la API Directx 8.1 de la Microsoft y una tarjeta aceleradora gráfica que dé soporte a esta tecnología, cualquier computadora es capaz de recrear una gran cantidad de impresionantes efectos visuales en tiempo real, lo cual no era posible anteriormente. Estos efectos van desde niebla y efectos de movimiento, ola de calor, entre otros efectos que engrosan la lista de efectos de post-procesamiento.

1.2 Utilización de los efectos de post-procesamiento.

Lograr una apariencia estilizada utilizando la técnica de efectos de post-procesamiento se está convirtiendo en una norma predominante en el desarrollo de simulaciones cada vez más realistas. Estos efectos pueden ser fácilmente incluidos después que la escena es renderizada y aumentan de forma significativa la calidad de la imagen de salida creando sorprendentes efectos visuales.

Un hito importante en la informática grafica fue la inclusión de la programación per-pixel y per-vertex; y el desarrollo de tarjetas gráficas que le dieran soporte a estos códigos que son conocidos mundialmente como Pixel-Shader y Vertex-Shader respectivamente. A partir de ese momento, los shaders comenzaron a ser cada vez más importantes, llegando a estar presentes en, la gran mayoría

de juegos de la actualidad. Técnicas como HDR y Bloom se realizan mediante shaders, incluso el efecto del calor, que produce una distorsión visual de ondulación de la imagen. Los efectos de post procesamiento son ampliamente utilizados en los gráficos generados por ordenador tanto para la industria del entretenimiento como para diversas simulaciones utilizadas para la enseñanza.

El efecto Bloom se realiza durante aquella situación que requiera usar entornos sumamente iluminados. Esta técnica permite que la luz de un objeto sobre iluminado se desborde sobre las partículas que lo rodean, logrando aumentar el brillo de los blancos y la oscuridad de los negros. La sensación final que se produce es casi la necesidad de cerrar los ojos al verlo. Video-juegos como *Oblivion* (2006) cuarta entrega de la saga *The Elder Scroll* y *Half Life 2* (lanzado mundialmente en el 2004) permiten activar este efecto, como se muestra en la Figura 1.

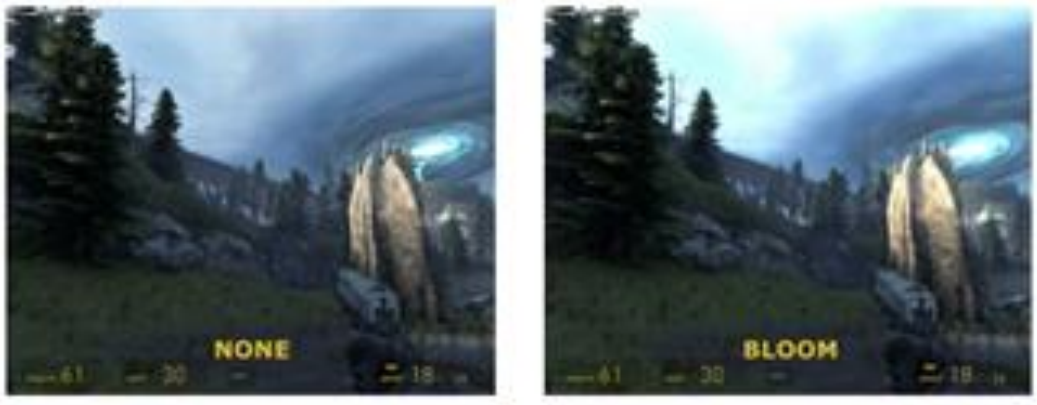

Figura 1: Efecto Bloom habilitado en el video-juego Half Life 2.

Una de las mejores formas para simular velocidad en un video-juego es el post-proceso Motion Blur. Esta técnica se puede apreciar en la mayoría de los video-juegos actuales, especialmente en los video-juegos de carrera, debido a que incrementa el realismo de escenas a través del efecto que se observa en las fotografías de escenarios donde los objetos se mueven a altas velocidades. Entre los video-juegos que permiten activar este efecto está el *Half Life 2*, *Lost Planet Parte II* que se lanzará en este año 2008 y en el motor grafico *Cry Engine 2* de los creadores del Far Cry que fue lanzado en el año 2007, ver Figura 2.

Figura 2: Efecto Motion Blur tomado del motor gráfico Cry Engine 2.

Otro post-proceso ampliamente utilizado es el Depth of Field o profundidad de campo; en pocas palabras este efecto es lo que ocurre cuando una lente enfoca un plano dejando lo demás borroso. La profundidad de campo es un importante componente visual que permite que una imagen o un entorno virtual parezcan reales. Esta técnica se encuentra implementada en motores gráficos *Source* (Aunque no se utiliza en el video-juego Half Life 2) lanzado en el 2004 con *Counter-Strike: Source y Half Life 2*; también se encuentra implementada en motor gráfico *Unreal Engine 3 (Unreal Tournament 2007)* y en el *Cry Engine 2,* ver Figura 3.

Figura 3: Efecto de Campo de profundidad tomado del motor gráfico Cry Engine 2.

El efecto Cell Shading es la técnica que se utiliza para crear escenas con una apariencia de dibujo animado o Cartoon. El primer videojuego en utilizar esta técnica fue el *"Jet Set Radio" del equipo* Smilebit *de* Sega lanzado en Japón en junio del 2000; en el 2002 salió una nueva secuela de este videojuego para Xbox, en esta versión se pulió el efecto de dibujos animados imprimiéndole al mismo

efectos gráficos sorprendentes. Otros títulos figuran en la lista de los video-juegos que tienen una apariencia parecida a los clásicos de Disney, tales como: *"The Legend of Zelda: The Wind Maker" creado para Nintendo* por Shigeru Miyamoto y producida actualmente en conjunto con Eiji Aonuma; "*Dragon Ball Z: Budokai 3*" lanzado en el 2007 para PlayStation 2 en Japón, Europa y América del norte; "*Megaman X: Command Mission*" para PlayStation 2 y Nintendo Game Cube.

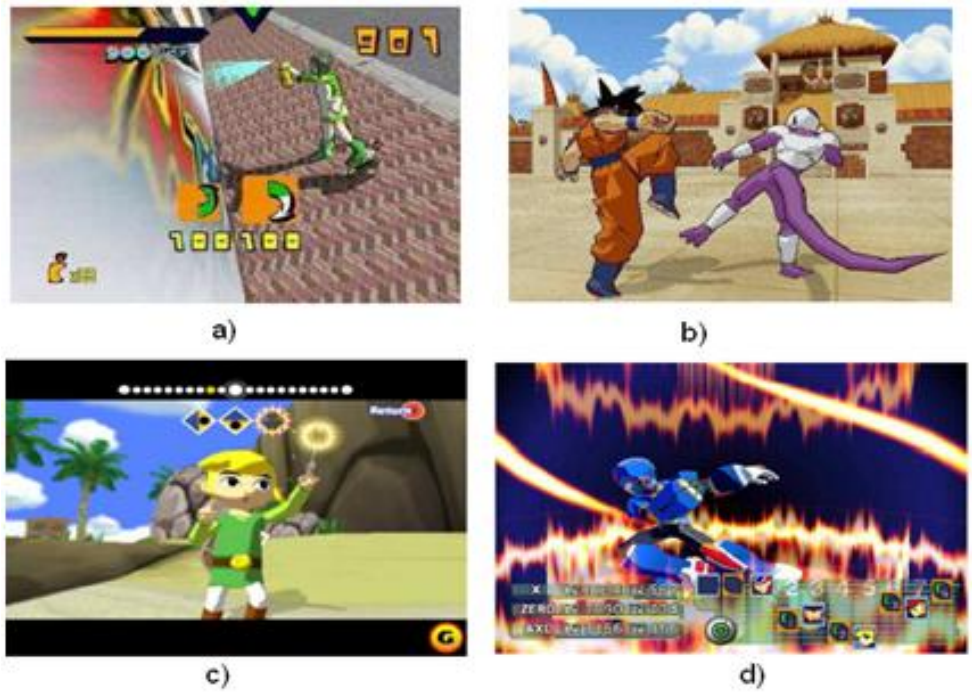

Figura 4: Efecto Cell Shading implementado en los videojuegos: a) Jet Grind Radio, b) Dragon Ball Z Budokai 3, c) The legend of Zelda: The WindMaker, d) MegaMan X: Command Mission.

1.3 Shader.

Los gráficos por computador y el hardware (HW) gráfico han evolucionado hasta convertirse en el centro de muchas aplicaciones modernas. Debido al uso masivo de las tecnologías asociadas a este campo por parte de la industria del entretenimiento, la evolución del HW gráfico ha crecido siguiendo una ley aún mayor que la de Moore para los procesadores generales, doblando el número de transistores aproximadamente cada seis meses, tres veces más rápido que lo que predice la citada Ley de Moore. (Wolfgang, 2002)

El avance en el HW gráfico ha posibilitado un notable incremento en el rendimiento de las aplicaciones de visualización; permitiendo la representación de escenas virtuales de mayor tamaño y complejidad. Sin embargo, aunque este aumento ha permitido ejecutar las secuencias de visualización en un tiempo

cada vez menor, se puede argumentar que al mismo tiempo estos sistemas no se han desarrollado en cuanto a las tecnologías de visualización.

La limitante fundamental del hardware de aceleración gráfica es que su estructura no se puede cambiar. Cuando se crean estos hardwares, los ingenieros prefijan un conjunto de instrucciones y algoritmos en el chip de video. Cada uno de estos algoritmos es acelerado por el HW gráfico. Pero no se brinda la posibilidad de ejecutar en estos chips otras instrucciones que no sean las codificadas en el momento de la creación del hardware. A esta estructura estática se le denomina sistema de funciones fijas (*Fixed-Function*). (Wikipedia, 2007)

En los últimos 5 años se ha obtenido una alternativa al *Fixed-Function* para aumentar la flexibilidad y capacidad gráfica del HW de video. En dos momentos claves del *pipeline* gráfico se ha posibilitado introducir códigos que permitan ejecutar algoritmos no diseñados inicialmente en el hardware. A estos códigos se le llaman *shader*, y según su funcionamiento y lugar de ejecución se dividen en *Vertex Shader* y *Píxel Shader*.

Los *Vertex Shaders* son los encargados de transformar todos los vértices de la escena. En estos se ejecutan las transformaciones de espacio objeto a espacio de mundo, de cámara, y finalmente se obtiene la posición en la pantalla. Además, se incluyen en estos todas las demás operaciones a nivel de vértices como son cálculos procedurales de coordenadas de texturas, iluminación per-vertex, entre otras. Los *Pixel Shader* manipulan y procesan todo lo necesario con los pixels, creando grandes efectos como interacción con agua, permitiendo la alteración de la luz y de la superficie, creando efectos visuales muy reales. Para este tipo de shader, se necesita información que maneja los vertex shader como por ejemplo la orientación en el espacio y los vectores de vista.

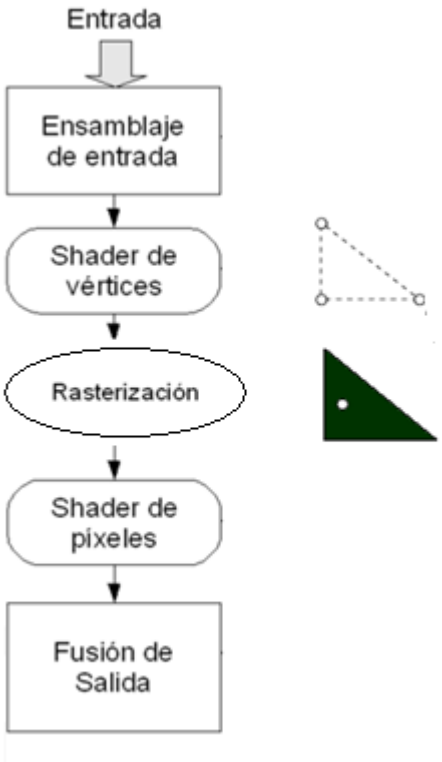

Figura 5: Pipeline Gráfico Conceptual usando shader.

La Figura 5 muestra cómo funciona el pipeline gráfico, este proceso comienza cuando la etapa de ensamblaje de entrada pasa los datos de los vértices al driver; luego que la GPU recibe estos datos el vertex shader es el encargado de transformar los vértices recibidos y realizar otras operaciones a nivel de vértices como las transformaciones de vista y modelo, mundo, proyección, cálculo de luces; el interpolador (Rasterizador) recibe la posición en la pantalla de cada uno de los vértices y genera los triángulos correspondientes. Al dibujar estos triángulos se calcula la posición de cada uno de los píxeles que conforman el triángulo.

Cada uno de estos píxels se introduce en el *Píxel Shader* para que éste calcule el color del píxel en la pantalla. De esta forma en el *Píxel Shader* se realizan todas las operaciones a nivel de píxel como es el cálculo de la iluminación *per-píxel*, mapeo de texturas, uso de los mapas de normales para la determinación de la normal del píxel, etc.

De esta forma, el uso de los *shaders* permite introducir nuevos algoritmos en el HW gráfico. Dando la posibilidad de realizar avances en la visualización virtual sin la necesidad de esperar a que estos algoritmos sean codificados en el chip de video para obtener aceleración por HW.

1.3.1 Unidad de Procesamiento Gráfico.

La primera Unidad de Procesamiento Gráfico (GPU) fue diseñada por una de las más famosas compañías productoras de hardware para gráficos, NVIDEA. La primera tarjeta aceleradora de gráficos en el mercado que poseía un GPU fue la GeForce 256. Ésta primera tarjeta de video tenía la capacidad de calcular miles de millones de operaciones por segundo, procesar 10 millones de polígonos por segundo y contaba con 22 millones de transistores.

La GPU es muy parecida a la CPU de una computadora, solamente que se dedica al procesamiento de gráficos para un mejor rendimiento y desempeño en aplicaciones como los video-juegos. Básicamente, la GPU es una CPU dedicada exclusivamente al procesamiento de gráficos, para aligerar la carga de trabajo del procesador del ordenador en aplicaciones como los videojuegos, RV, etc. De esta forma, todo lo relacionado con los gráficos se procesa en la GPU y por tanto la CPU puede dedicarse a realizar otras operaciones.

Después de la llegada de la API Directx versión 8.0, los GPU´s agregaron la tecnología de los shader, en donde cada pixel y vértice pueden ser programados con pequeños archivos. La utilización de estos códigos se requiere muchas operaciones con matrices y vectores en el GPU, por lo que los ingenieros y científicos han estudiado la manera de poder combatir esta situación para mejorar el desempeño de las tarjetas gráficas. Una de las soluciones propuestas es la idea de aprovechar al máximo las capacidades de un ordenador personal al utilizar al mismo tiempo la GPU y la CPU, distribuyendo el trabajo entre ellas sabiendo que tarea asignar a cada una.

1.4 Formatos de textura.

Comúnmente las texturas son representadas mediante arreglos de colores unidimensionales, bidimensionales o tridimensionales donde cada elemento contiene el valor de una pequeña región de la imagen, denominada texel. Cada texel consta de tres o cuatro componentes en dependencia de si la imagen es de 24 ó 32 bits, es decir, si la imagen posee transparencia o no. Estos componentes son: Red, Green, Blue (RGB) los colores primarios en la computadora y el componente Alfa que es la transparencia.

Cuando se aplica una textura a una superficie 3D, en el proceso conocido como mapeado, se asignan los texels a los píxeles correspondientes que aparecerán en la imagen final. El mapeado consiste en envolver al objeto con la textura, asignando a cada vértice del objeto una posición en la textura. Los niveles en la imagen final son definidos por los números de pixels necesarios para representar una

imagen, y el número de bits necesarios para el espectro de color de una imagen. Todas las superficies tienen un tamaño específico y almacenan un número específico de bits que representan el color. Estos bits son separados dentro de elementos individuales de color: rojo, verde y azul.

Cuando creamos una superficie es necesario especificar el formato en que se almacena dicho recurso en memoria, de esta forma determinamos como los datos para cada pixel son interpretados en la memoria de la textura. Este trabajo centra su atención en los tipos de formatos de textura entera *A8R8G8B8* (Microsoft Corporation, 2005) y los formatos de textura de punto flotante *R32F* (Microsoft Corporation, 2005) y *R16G16F* (Microsoft Corporation, 2005).

Antes del lanzamiento de Directx 9.0 solamente los formatos de textura entera eran soportados por la API Direct3D de la Microsoft. Estos formatos tienen un rango de valores limitados que va desde 0.0f hasta 1.0f en los pixel shader. De esta manera si usamos un valor de color mayor de 0.1, es necesario ajustar este valor a 0.1 con el objetivo de utilizar un valor de textura entera de 8 bit. A8R8G8B8 es un formato estándar de 32- bit ARGB (alfa, rojo, verde, azul) que utiliza 8 bit por canal. (Microsoft Corporation, 2005)

Los formatos de punto flotante son lanzados junto con la API Directx 9.0, estos formatos a diferencia de las tradicionales texturas en formato entero son capaces de almacenar un extenso rango de valores de colores, incluyendo valores más allá del rango [0.0, 0.1] utilizado en los formatos de textura entera. R32F es formato de superficie de punto flotante que usa 32 bits para el canal rojo. El formato de punto flotante de 32 bits es también conocido como un formato s23e8, donde se reserva 23 bits mantissa y 8 bits para el exponente. G16R16F es un formato de textura de punto flotante que utiliza 16 bits por canal, que utiliza 16 bits para el canal rojo y 16 bit para el canal verde.

1.5 Render a textura.

La **renderización** es el proceso de generar una imagen desde un modelo. En términos de visualizaciones en ordenador, más específicamente en 3D, la "renderización" es un proceso de cálculo complejo desarrollado por un ordenador destinado a generar una imagen 2D a partir de una escena 3D. La técnica de render a textura es una modalidad del proceso tradicional de renderizado.

Esta técnica es fundamental para la implementación de algoritmos que utilizan las potenciales que brinda el GPU, ya que es la única forma de mantener los resultados de un programa en la memoria de video sin tener que transferir los datos nuevamente desde la memoria principal. Esta operación es como un interfaz de memoria de sólo escritura. También es posible realizar una copia de los datos

desde el Frame Buffer hacia la textura, aunque esto implique un coste adicional. “La razón de realizar la escritura de los datos en la GPU sobre un buffer es que el procesador de fragmentos puede leer de cualquier forma desde la textura; pero puede escribir sólo una vez por cada fragmento en el buffer de destinación al final del shader. La salida del procesador de fragmentos es un flujo de pixels, por lo cual la lectura y la escritura de datos utilizando la GPU deben hacerse de la forma descrita”. (Ogayar Anguita, 2006)

En un principio se empezó a denominar render a textura a todo proceso de render cuyo resultado era un buffer que no sería visualizado sino utilizado posteriormente como datos de entrada. Esto se debe a que los primeros algoritmos que necesitaban de esta técnica eran los de generación dinámica de sombras, proyección del entorno en texturas cúbicas, etc. Además todos estos buffer se reutilizaban como texturas en la primeras GPU’s programables. Últimamente ha cambiado un poco el concepto, ya que los lenguajes de sombreados de alto nivel como Cg consideran la entrada desde la memoria como sampler (orígenes de datos para el muestreo) en lugar de simples texturas. Estas texturas pueden disponer de formatos de puntos flotantes con datos no visualizables, lo cual rompe el concepto clásico de textura y las presenta como buffers o matrices convencionales. De esta forma sería más apropiado en la mayoría de las situaciones hablar de render sobre buffer. (Ogayar Anguita, 2006)

El algoritmo de implementación de esta técnica propone tres pasos fundamentales:

I. Crear una textura como el render target.
II. Renderizar la escena hacia la textura creada, copiando los pixel renderizados hacia la textura.
III. La textura es utilizada para renderizar la escena final.

1.6 Técnicas de apoyo a los efectos de post-procesamiento.

1.6.1 Alpha Blending.

En los gráficos por computadoras las imágenes se forman mediantes el uso de los tres canales tradicionales Rojo-Verde-Azul y un cuarto canal denominado “canal alpha” que es el encargado de manejar las transparencias, de esta forma en los gráficos de 32 bits, tendremos 8 bits para cada uno de los colores y 8 bits más para el canal Alpha, lo que nos dará 256 “intensidades” distintas de transparencia que van desde el valor 0 (totalmente transparente) hasta 255(totalmente opaco).

El nombre de “Alpha Blending” procede del hecho que generalmente los factores de mezcla usados son los valores Alpha de la superficie. Alpha Blending es un método para combinar dos imágenes que usan los valores de los cuatro canales de color (ARGB) de los pixel para determinar el valor del color

resultante de los pixel. Este documento analiza la variante del “Alpha Blending” que combina el color de la primitiva dibujada con el color previamente almacenado en el pixel sobre frame buffer, este pixel fue previamente almacenado utilizando la modalidad de render a textura analizada anteriormente.

Esta técnica permite que una imagen pueda ser renderizada en el tope de otra imagen, con una mezcla de ambas. Cuando mezclamos dos pixels, las componentes de color de los mismos primeramente son escaladas por sus valores Alpha, luego el último pixel es escalado por la inversa del valor Alpha del pixel superior y adicionado al pixel superior para formar el color final mezclado.

La ecuación que rige el comportamiento del método de mezcla (Walsh, 2003) es definida por:

$$final\ color = source \text{ x sourceBlendFactor} + destination \text{ x destinationBlendFactor}$$

Ecuación 1: Color final del pixel mezclado.

El valor de la variable “*final color*” es el color que pasa al frame buffer después de la operación de mezclado. El “*source*” es el pixel que se está intentando dibujar en el frame buffer. La variable “*destination*” se refiere al pixel que ya existía en el frame buffer antes de intentar dibujar un nuevo pixel. La fuente y el destino de los factores de mezcla están representados por las variables “*sourceBlendFactor*” y “*DestinationBlendFactor*” son utilizadas para modificar como los colores son combinados juntos. (Walsh, 2003)

1.6.2 Filtrado bilineal de textura.

Cuando renderizamos una primitiva, primero mapeamos la primitiva 3D hacia una escena 2D. Si la primitiva tiene una textura, se debería usar dicha textura para producir un color para cada pixel en la imagen 2D renderizada. De igual forma para cada pixel en la imagen de la primitiva dibujada se debería obtener un valor de color desde la textura. Este proceso que especifica como interpolar los texel a pixel es conocido como “*Filtrado de textura*” (Microsoft Corporation, 2005) (McCurskey, 2002).

La operación de filtrado es la forma en la cual se obtienen los texels de los mapas de textura determinando un par de coordenadas u, v y constituye una técnica de interpolación basada en el cálculo del promedio de los cuatro pixels más próximos en una imagen digital. Los problemas de filtrado están divididos en 2 cuestiones opuestas: magnificación y reducción.

La magnificación ocurre cuando tratamos de mapear un pixel individual en un mapa de textura hacia muchos pixels en el frame buffer. El proceso de reducción es el proceso opuesto al de magnificación, donde un número de texel podrían ser mapeados a un pixel individual.

La mayoría de los más nuevos tipos de hardware comerciales podrían usar cuatro tipos de filtrado entre los que se encuentra: filtro linear, filtro anisitropico y el filtro mipmap.

En este documento sólo se hace referencia a una de las formas del filtro linear conocida como filtro bilinear, debido a su bajo coste computacional; además esta técnica es aplicada sobre imágenes 2D las cuales son comúnmente utilizadas para almacenar los pixels renderizados a través de la modalidad de render a textura. Este método a gran escala lo que realiza es un promedio con los cuatro texel más cercanos teniendo en cuenta la distancia relativa desde el punto muestreado.

El algoritmo de implementación de esta técnica propone los siguientes pasos. Primeramente se calcula una dirección de texel, la cual usualmente no tiene direcciones enteras; entonces busca los texel cuya dirección entera están más próximas a las direcciones calculadas y finalmente se calcula un average con los pesos de los texel que están inmediatamente por encima, por debajo, a la derecha y a la izquierda de los puntos cercanos a la muestra. (Microsoft Corporation, 2005)

1.7 Fundamentación de las tecnologías.

En este epígrafe se analizan dos de las bibliotecas gráficas que compiten en el mundo del desarrollo de gráficos 3D, y los lenguajes a utilizar.

1.7.1 C++.

C++ se trata de un lenguaje de programación basado en el lenguaje C, estandarizado (ISO/IEC 14882:1998), ampliamente difundido, y con una biblioteca estándar C++ que lo ha convertido en un lenguaje universal, de propósito general, y ampliamente utilizado tanto en el ámbito profesional como en el educativo.

Las principales características del C++ son el soporte para la programación orientada a objetos y el soporte de plantillas o programación genérica (*templates*). Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. *C++* está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatizaciones trae (obliga a hacerlo casi todo manualmente al igual que C) lo que "dificulta" mucho su aprendizaje.

Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores)
- Identificación de tipos en tiempo de ejecución (*RTTI*)

Una de las ventajas del C++ es el gran número de compiladores que soportan el leguaje. Algunos de estos son:

- Borland C++
- Cygwin (GNU C++)
- IBM C++
- Intel C++
- Microsoft C++
- Microsoft Visual C++ Toolkit 2003
- Paradigm C++

1.7.2 Librería grafica OpenGL y Directx.

Para la creación de aplicaciones 3D en la plataforma Windows hay dos corrientes fundamentales. Una es usar OpenGL, que es una API abierta y regida por un comité. La otra es usar Directx que es una API creada y propiedad de la Microsoft. Ambas APIs constituyen excelentes alternativas para el desarrollo de aplicaciones gráficas.

"OpenGL es una librería gráfica que provee a los programadores de una interfaz de acceso al *hardware* (HW) gráfico. Es poderoso, con *rendering* a bajo nivel y una librería de *software* de modelamiento, disponible en la mayoría de las plataformas, con un amplio soporte de HW. Es diseñado para ser usado en cualquier aplicación gráfica, desde juegos y simuladores hasta modelaciones CAD (*Computer Aided Design*)" (Chover, 2004) .Esta API está regida por un comité compuesto por empresas desarrolladoras de software y HW que son las que deciden cuales son las nuevas funcionalidades a implementar, esto posibilita su utilización en diferentes plataformas pero al mismo tiempo retarda las salidas de las actualizaciones para la comunidad de desarrolladores de gráficos 3D.

En cambio en Directx sólo Microsoft tiene el control sobre las nuevas funcionalidades agregadas, esto posibilita que pueda adaptarse con gran facilidad a los cambios, pero a su vez provoca que los desarrolladores gráficos desconozcan que funcionalidades serán agregadas en la siguiente versión. Directx a diferencia de OpenGL solo funciona en Windows, no hay una implementación para otras

plataformas y muy difícilmente la habrá. La idea detrás de esta API es dar acceso directo al HW de una PC, manteniendo independencia de la implementación que el HW realice de las funcionalidades.

Ambas APIs usan la tradicional *graphics pipeline* (tubería gráfica). Es el mismo *pipeline* que se diseñó desde las primeras computadoras gráficas, que se ha ido modificando de acuerdo a los avances de HW aunque sin cambiar la idea básica. (Chover, 2004)

Ambas APIs describen los vértices como un grupo de datos consistente en coordenadas en el espacio que definen la localización del vértice. Las primitivas gráficas (puntos, líneas, y triángulos) están definidos como un grupo ordenado de vértices. Sin embargo, la diferencia entre las APIs está en cómo los vértices son combinados para formar las primitivas: cada uno lo maneja diferente. (Chover, 2004)

1.7.3 High Level Shading Language.

High Level Shading Language (HLSL) no es más que una capa adicional que reside conceptualmente sobre la API existente. Los shaders facilitan realmente la tarea de programación gráfica, ofrecen un aumento en la velocidad de proceso gráfico o flexibilidad en la programación. El lenguaje de programación HLSL de Microsoft para la GPU en DirectX 9.0 trabaja solamente en Windows y consigue altas marcas de calidad. (Santos., 2006)

HLSL es el lenguaje de programación de alto nivel, que aunque puede crear animaciones ultra realistas, hasta efectos visuales sorprendentes, sin tener que preocuparse por el tipo específico de hardware, está basado en DirectX 3D y esta librería a pesar de ser muy popular por lo programadores gráficos no es multiplataforma. (Santos., 2006)

HLSL soporta instrucciones para escribir expresiones matemáticas. Como otros lenguajes gráficos, las expresiones matemáticas son más eficientes con vectores y matrices. Este lenguaje sigue muchas reglas e instrucciones de C, así como otras propias para hacer la programación gráfica más intuitiva y compacta.

1.7.4 OpenGL Shading Language.

OpenGL Shading Language (GLSL) fue el primer lenguaje de alto nivel comercial para la programación de *shaders* en 2000 creado por Marc Olano de Sillicon Graphics Inc.

GLSL es un lenguaje de alto nivel diseñado específicamente por el entorno *OpenGL*. Este lenguaje permite aplicaciones para hardware gráfico. Contiene funciones que permiten expresiones abreviadas de algoritmos gráficos de manera que sea natural para programadores con experiencia en C y C++.

Ofrece opciones avanzadas en el diseño de gráficos al proporcionar acceso de alto nivel a las características programables de los procesadores de gráficos modernos, lo que representa un gran paso adelante en la creación de gráficos 3D foto-realistas en tiempo real.

Posee implementaciones en UNIX, Windows, Linux y otros sistemas operativos. Esta amplia compatibilidad permite a los desarrolladores mover fácilmente sus trabajos entre los principales sistemas operativos comerciales y las plataformas hardware.

1.7.5 Cg.

Cg es un lenguaje de alto nivel que hace posible la creación de diferentes efectos especiales. Diseñado para que el programador pueda controlar una serie de atributos gráficos usando un hardware gráfico programable, controlando estos atributos con una rapidez increíble.

Este leguaje provee al programador de una completa plataforma de programación de fácil uso y soporta diferentes plataformas; para sistemas operativos como Windows, Linux, Macintosh OS X y plataformas de consolas como Xbox; y es compatible con OpenGL y DirectX. A diferencia de otros lenguajes como MOD o HLSL que fueros diseñados para soportar OpenGL y DirectX respectivamente.

Cg establece una jerarquía de comportamientos dependientes de las capacidades programables de cada GPU llamados *perfiles* del lenguaje. El perfil especifica qué características del lenguaje son soportadas por la GPU. El perfil se fija en tiempo de compilación del código fuente Cg y depende fundamentalmente de la API a utilizar (Direct3D u OpenGL) y de la compatibilidad de la tarjeta en cada caso. (Sanz Montemayor, y otros, 2004)

1.7.6 Comparación entre los lenguajes de programación de shader.

Las virtudes de estos lenguajes frente a la programación a bajo nivel son las mismas que hicieron de los lenguajes de alto nivel de propósito general Fortran, Basic o C lo que son hoy en día. La abstracción, la reducción de los tiempos de desarrollo, la portabilidad, la optimización automática y la legibilidad del código hacen de estos lenguajes potentes herramientas de desarrollo de *shaders*. En cuanto a las diferencias entre los citados lenguajes se puede comentar que en esencia, *Cg* y *HLSL* son

lenguajes realmente parecidos, tanto en sintaxis como en filosofía. Ambos, junto con el GLSL, están muy orientados al empleo masivo por parte de los desarrolladores de juegos para ordenador y videoconsolas. (Sanz Montemayor, y otros, 2004)

1.7.7 Lenguaje unificado de modelado.

Para modelar el análisis y diseño de la biblioteca se escogió el lenguaje *UML* (Lenguaje Unificado de Modelado), debido a las potencialidades descriptivas que posee.

UML es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; aún cuando todavía no es un estándar oficial, está apoyado en gran manera por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

1.8 Metodologías y herramientas de desarrollo.

Para la realización de la tesis se hizo un estudio de cada una de las posibles metodologías y herramientas a utilizar. Se tuvo en cuenta la tendencia actual y cada una de ellas y se realizo la selección descrita a continuación.

1.8.1 RUP.

Se escogió RUP (Rational Unified Process, Proceso Unificado de Desarrollo de Software) como una guía para realizar el análisis y diseño de la aplicación debido a que es una metodología que ha probado su efectividad durante muchos años, ya que numerosos proyectos la han utilizado para desarrollar su software. Además esta metodología tiene un gran número de documentos publicados que es posible consultarlos para esclarecer dudas. También porque siguiendo los pasos que propone se obtiene una amplia documentación.

RUP es una metodología que se modela con UML (Unified Modeling Language). La misma constituye una guía de cómo usar UML de forma más efectiva. Ambos están estrechamente relacionados entre sí pues, mientras RUP establece las actividades y los criterios para conducir un sistema desde su máximo nivel de abstracción (la idea en la cabeza del cliente), hasta su nivel más concreto (un

programa ejecutándose en las instalaciones del cliente), el segundo ofrece la notación gráfica necesaria para representar los sucesivos modelos que se obtienen en el proceso de refinamiento.

1.8.2 Rational Rose.

Rational es la herramienta UML desarrollada por Rose y es posiblemente la más utilizada actualmente en el mundo. Como todos sus demás productos, Rational Rose proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad más rápidamente.

Existen varias razones por las que utilizar Rational Rose a la hora de desarrollar un sistema, algunas de las cuales se citan a continuación:

- Comunicación: Todos los miembros del equipo de trabajo, desde los analistas hasta los programadores, trabajarán con un lenguaje común de Modelado Visual. Esto asegura que todos entenderán los requerimientos del proyecto.
- Administración de la complejidad: El modelado visual de Rational Rose permite a los diseñadores de software navegar fácilmente a través de proyectos complejos logrando, en base a sus necesidades, visualizar desde cuadros de imágenes que muestran el proceso genérico del proyecto, hasta observar diagramas que plasman detalles de bajo nivel en la arquitectura de la aplicación. Al poder ver un anteproyecto desde el inicio surgirán pocos errores en el proyecto.

CAPÍTULO 2

TÉCNICAS PARA LA SIMULACIÓN DE LOS EFECTOS DE POST-PROCESAMIENTO.

Tradicionalmente el procesamiento de imágenes utilizaba una cantidad significativa del poder de la CPU y desperdiciaba muchos recursos; debido a que en la CPU se realizaban las operaciones de manipulación de vértices y después se enviaba los datos de los vértices transformados a la tarjeta gráfica a través del bus de la computadora. Pero con la aparición de las nuevas tecnologías del hardware gráfico estos efectos pueden ser realizados de forma más eficiente.

En este capítulo se expondrán algunos efectos visuales que podrían ser usados con el fin de mejorar el realismo en los entornos virtuales.

2.1 Efecto Monocromático.

El término monocromo, traducción latina del vocablo Monochrome, proviene de dos palabras griegas: *mono* (μovo, que significa "solo" o "único") y *chrome* (χρωμα, que significa "color"). Un objeto o imagen monocromática es un rango de colores que consta de shaders de un solo color o tonalidad, una imagen monocromática con colores neutrales es conocida como escala de grises o blanca-y-negra. (Wikipedia, 2008)

Para una imagen este término usualmente es identificado con la modalidad de escala de grises o blanco-y-negro, pero también hace referencia a otras combinaciones de dos colores, tales como verde-y-blanco o verde-y-negro. En este trabajo analizaremos el efecto monocromático en su modalidad de blanco-y-negro. (Wikipedia, 2008)

El efecto Black-y-negro puede realizarse a través de un cálculo simple de luminosidad o brillo (**Luminance**) la cual es leída desde los pixels de color RGB de la imagen original y les devuelve un color RGB donde R=G=B=Luminosidad. Esta operación es realizada mediante un producto vectorial para calcular el valor de la luminosidad de la escena:

$$Luminance = c1 * red + c2 * green + c3 * blue$$

Ecuación 2: Luminosidad de cada pixel.

El cálculo de luminosidad propuesto en este epígrafe está fundamentado por una de las leyes de Grassman, específicamente la "*Cuarta Ley: Ley de la aditividad".* Esta ley parte del hecho que cualquier color puede crearse por síntesis aditiva de los colores primarios rojo, verde y azul, y dado que al mezclar aditivamente estos componentes se suman sus respectivas luminancias, se puede deducir que la luminancia de un color cualquiera equivale a la suma de las luminancias de sus componentes primarias.

Figura 6: Imagen de “Lenna” (a) imagen original (b) imagen monocromática.

2.2 Efecto Sepia.

El tono Sepia es un espacio de color que le imprime a las imágenes un tono pardusco desteñido produciendo a su vez una sensación de antigüedad a la escena. Realmente no existe un color individual conocido como “Sepia”, este término cubre un rango de color producto de la mezcla de los colores amarrillo y marrón. Este efecto realiza una conversión del espacio de color RGB al espacio de color Sepia. Una de las vías utilizadas en esta conversión es el proceso de “transformar una muestra RGB desde la imagen de entrada hacia un espacio YIQ, y luego transformarla nuevamente al espacio RGB”, esta vía es detallada en el libro “*Game Programming Gems 4*”. Otro método analizado en este trabajo es la implementación de los post-proceso sepia, utilizando el concepto de “lectura dependientes”.

Efecto Sepia mediante el término de “Lectura dependiente”.

El concepto “lectura dependiente” hace referencia a la lectura realizada desde un mapa de textura usando una coordenada de textura calculada dentro los pixel shader.

Este método aplica un mapa de textura 1D sepia para calcular la luminosidad de la imagen de entrada. Luego el color desde la imagen original es convertido a un valor de luminosidad y luego es usado como una coordenada de textura para probar el efecto Sepia. (Mitchell, 2002)

Figura 7: Efecto Sepia a) imagen original, b) imagen con tono sepia.

Efecto Sepia utilizando un espacio YIQ.

Usualmente la conversión desde el espacio de color RGB al espacio de color Sepia es realizado mapeando un color RGB hacia una tabla lookup (mapas de texturas), pero aunque esta técnica es efectiva y simple, la conversión del espacio sepia utilizando un espacio YIQ es una vía más rápida y óptima debido a que reduce el número de instrucciones en el pixel shader. (Marwan, y otros, 2004)

Primero se analiza el algoritmo general para este proceso de conversión:

I. Convertir la muestra RGB al espacio YIQ utilizando una matriz de multiplicación (**M**).
II. Remplazar las componentes I y Q con los valores 0.2 y 0.0 respectivamente.
III. Convertir el espacio YIQ al espacio RGV utilizando una matriz de multiplicación inversa (**M'**).

El espacio YIQ es similar al espacio YUV y algunas veces es usado en la transmisión de señales de los programas de televisión. La diferencia entre ellos es que el ancho de banda de la señal IQ del modelo YIQ puede ser reducido en mayor grado que las componentes UV, para un nivel igual de calidad de visión.

En el modelo de color YIQ la componente Y representa la información de luminancia y es el único componente utilizado por los televisores de blanco y negro; mientras que las componentes I y Q representan la información de crominancia, la cual representa la componente de la señal que contiene la información del color.

Este modelo de color fue diseñado teniendo en cuenta las características del sistema visual humano, obteniendo mayor sensibilidad a los cambios en luminancia que a los cambios de matiz o saturación. Así pues, este estándar usa más bits (o ancho de banda) para representar (Y) y menos para I y Q. Una de sus mayores ventajas es que las informaciones de luminancia (Y) y color (I y Q) están separadas, esto permite procesar la luminancia sin afectar al color. (Corral Martín, y otros)

En la Figura 8 se muestra la conversión lineal desde el espacio de color RGB hacia el YIQ, asumiendo un rango de colores monocromáticos [0,1] para ambos modelos.

$$M = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Figura 8: Representación matricial para convertir del espacio RGB al espacio YIQ.

La primera fila de esta matriz de conversión calcula la luminosidad de la muestra RGB de entrada, lo que seria la componente Y del espacio, mientras que la segunda y tercera fila codifica los valores cromáticos dentro de las variables I y Q del espacio. Luego convertimos nuevamente al espacio RGB desde el espacio YIQ utilizando la matriz inversa **M'**. (Marwan, y otros, 2004)

$$M' = \begin{bmatrix} 1.0 & 0.956 & 0.620 \\ 1.0 & -0.272 & -0.647 \\ 1.0 & -1.108 & 1.705 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

Figura 9: Representación matricial para convertir del espacio YIQ al RGB.

La parte interesante de este método es el remplazo de las componentes I y Q con los valores 0.2 y 0.0. Este proceso será ejemplificado mediante la conversión del espacio YIQ al espacio RGB. La operación **M' [YIQ]** es ampliada, ver Figura 10 (Marwan, y otros, 2004):

$$R' = I + 0.956 * I + 0.620 * Q$$
$$G' = I - 0.272 * I - 0.647 * Q$$
$$B' = I - 1.108 * I + 1.705 * Q$$

Figura 10: Cálculos para la conversión del espacio YIQ al espacio RGB.

Luego remplazamos los valores escogidos de las componentes I y Q, la ecuación queda simplificada de esta manera:

$$R' = Y + 0.191$$
$$G' = Y - 0.054$$
$$B' = Y - 0.221$$

Figura 11: Ecuaciones simplificadas de los canales de colores RGB.

Finalmente remplazamos la componente Y con los valores calculados de luminosidad, resultando una imagen con un tono sepia, ver Figura 7.

2.3 Filtros kernels.

Una característica de la imagen digital es el parámetro denominado ***frecuencia espacial***, el cual se define como el número de cambios en valores de brillantez por unidad de distancia para cualquier zona de la imagen. Si los valores de brillantez varían mucho en un área determinada de la imagen, se dice que es un área de alta frecuencia espacial; en caso contrario, se trata de un área de baja frecuencia espacial.

La frecuencia espacial puede ser manipulada mediante el filtrado de la convolución espacial, el cual se fundamenta en el uso de máscaras de convolución; estos filtros son denominados filtros kernels o de convolución y son utilizados para modificar los valores de brillantez de la imagen original en función de un peso promedio creado mediante una combinación lineal.

La convolución es una de las operaciones más usuales que se realizan en el dominio espacial de las imágenes. En ella, una matriz de números (denominada kernel) es multiplicada por cada píxel y sus vecinos en una pequeña región, se suman los valores y este resultado se asigna al píxel de la imagen que coincide con el centro del kernel. Existen diferentes tipos de filtros según la función que realicen, bien sea para destacar algún rasgo o característica de la imagen o para corregir algún defecto. Así pues, tenemos filtros paso bajo, para disminuir contraste, paso alto para aumentarlo, Laplacianos para destacar límites entre distintos valores numéricos de la imagen, Sobel para determinar la orientación en estructuras de la imagen, Gaussianos para eliminar o minimizar el ruido de una imagen.

Los kernels pueden tener tamaños arbitrarios, pero los de 3x3 son los más usados en la mayoría de las situaciones (también debido a que son los más rápidos), ya que sólo toma en consideración el valor del pixel mismo y el de sus 8 vecinos.

En este epígrafe se analiza el efecto Blur utilizando los filtros Gaussianos; y el filtro Canny para la simulación del post-proceso que permite la detección de bordes de los objetos de una imagen.

2.3.1 Efecto Blur.

El *post-proceso Blur o desenfoque* (Microsoft Corporation, 2005) es un sencillo y común efecto basado en imágenes; su objetivo es suavizar la imagen, este efecto es muy útil en aquellas ocasiones en las que se desea eliminar o disminuir el ruido de una imagen. Esta técnica primero aplica un filtro hacia abajo a la textura con el objetivo de reducir el número de pixel dibujados. Luego realiza una pasada del efecto Gaussian Blur en la dirección vertical y horizontal. Finalmente desarrolla un filtro hacia arriba para conseguir que la imagen resultante regrese a su tamaño original.

El efecto *Gaussian Blur horizontal* (Microsoft Corporation, 2005) realiza un muestreo de los texel a la derecha e izquierda de la coordenada de la textura fuente, mientras que el efecto *Gaussian Blur vertical* (Microsoft Corporation, 2005) realiza un muestreo de los texel que están en el tope y debajo de la coordenada de la textura fuente; como último paso promedia los pesos tomados de las muestras de texel analizados por los pixel shader. Estos filtros separables son usados con el objetivo de simular una apariencia borrosa en las imágenes.

El filtro Gaussian Blur toma muestras de una vecindad circular desde la imagen de entrada y promedia los pesos utilizando la Ecuación 3 (Mitchell, y otros, 2004):

$$g_{2D}(x,y) = \frac{1}{\sqrt{2x\sigma^2}} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

Ecuación 3: Filtro Gaussiano.

Donde σ es el estándar de desviación del efecto Gaussian y determina el tamaño del filtro, ver Figura 12 (Turkowski, 2007). La variable x e y representan las coordenadas de la muestra relativa de la imagen en el centro del filtro.

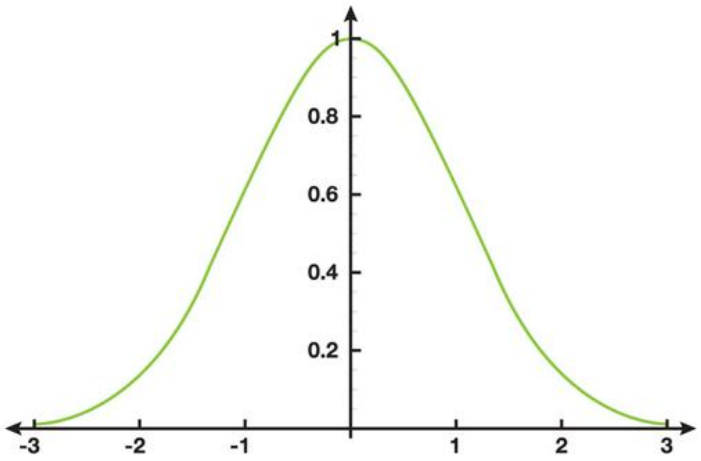

Figura 12: Efecto Gaussian con σ = 1. Esta función es separable y simétrica radialmente.

Una propiedad extremadamente importante de la técnica Gaussian es precisamente la característica de ser un filtro separable. Por lo tanto la ecuación anterior puede ser cambiada a la siguiente formula:

$$g_{2D}(x,y) = \left[\frac{1}{\sqrt{2x\sigma}} e^{\frac{-x^2}{2\sigma^2}}\right] * \left[\frac{1}{\sqrt{2x\sigma}} e^{\frac{y^2}{2\sigma^2}}\right] = g_{1D}(x) + g_{1D}(y)$$

Ecuación 4: Filtro Gaussiano separable.

La Ecuación 4 (Mitchell, y otros, 2004) significa que podemos implementar un filtro Gaussian con una serie de operaciones de filtrado 1D: uno horizontal ($g_{1D}(x)$) y uno vertical ($g_{1D}(y)$). De igual forma nos permite desarrollar un efecto Gaussian con un kernel mucho más largo mientras realizamos la misma cantidad de cálculos que se requieren para implementar un filtro de kernel no separable más pequeño.

2.3.2 Detección de bordes.

Al proyectar una escena hacia una imagen lo que se obtiene una matriz de pixels del tamaño de la resolución de la escena, la cual representa lo que es visible de la escena. Los pixels que componen la imagen sólo contienen la información acerca de que color representan excluyendo de esta forma la información del espacio 3D que representa la escena, como la profundidad, posición de los objetos, etc. Este proceso es el que intenta explotar los algoritmos de detección de bordes.

Estos algoritmos dependen de la resolución y el nivel de detalle de la textura, por lo cual entre más pequeña y comprimida sea la imagen más difícil resulta detectar las siluetas de los objetos. Otra de las limitaciones de este método se muestra en escenas donde los objetos sean del mismo o colores similares debido a que este post-proceso trabaja sobre la discontinuidad de colores de las texturas.

El efecto de Detección de bordes permite resaltar las siluetas de los objetos basándose en el color de la textura. Este post-proceso propone generar líneas blancas a lo largo del borde de los objetos de una imagen, realizando una comparación entre las componentes normales de los cuatro pixels vecinos al pixel analizado, luego suma los valores absolutos obtenidos por la diferencia de color entre los pixels. La imagen resultante de este proceso se observará brillosa en aquellos pixel que tengan una diferencia de color significativa sobre sus pixel vecinos. (Microsoft Corporation, 2005)

A continuación se analiza el algoritmo propuesto por Hertzmann para la detección de silueta en el espacio imagen, el cual fue descrito en el articulo “Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outline”. Además se investiga uno de los filtros kernel más utilizados en la implementación de este post-proceso.

Algoritmo de Aaron Hertzmann.

El algoritmo de Aaron Hertzmann trabaja con dos conceptos fundamentales para el proceso de detección de bordes: la silueta y los pliegues del objeto.

La silueta de un objeto es muy importante porque es lo que nos da las pistas visuales de cómo es que está formado el objeto, donde están sus límites y ayuda a diferenciar a un objeto de su entorno. Los pliegues del objeto o valles y montañas son líneas que aparecen cuando el ángulo entre dos caras contiguas es mayor que cierto ángulo previamente establecido como límite; estas líneas son útiles para describir las características internas de los objetos. En este paso no se encuentran los pliegues de los objetos, pero es muy posible que se encuentren los bordes de los objetos. Esto generalmente ocurre debido a que la diferencia entre los objetos es suficiente para que el detector de discontinuidades los detecte.

Este algoritmo propone obtener los detalles de los objetos de una escena utilizando un *Mapa de profundidad (Depth Map)* y un *Mapa de Normales (Normal Map)*, los cuales son conocidos como Texturas Procedurales de Pre-cálculo. Comúnmente este tipo de textura no almacena el color de un objeto, sino que en los valores de cada componente de color guarda información pre-calculada del objeto. Entre los datos que puede almacenar se encuentran la iluminación del modelo, la radiosidad, las normales, entre otros. Para la obtención de estos datos es necesario hacer cálculos muy costosos, que al ser almacenados previamente se evita realizarlos en el momento de la simulación.

El *Depth Map* es una imagen en la escala de grises en la cual la intensidad de cada pixel es proporcional a la profundidad que ese pixel representa a la escena 3D. Primero se calcula el *Depth Map* de la escena y se le aplica un algoritmo de detección de discontinuidades.

Posteriormente se genera el *Normal Map* de la escena. Un *Normal Mapa* es una imagen que representa las normales de cada punto que componen la imagen. Lo que hace para generar el *Normal Map* es que a cada objeto de la escena se le asigna el color blanco y se traslada al origen. Después se pone una luz roja en el eje X, una verde en el eje Y e una azul en el eje Z, apuntando al objeto. Además se ponen luces con una intensidad negativa en el lado opuesto de cada eje. Se genera la imagen 2D y se buscan las discontinuidades en el objeto, estos últimos dos pasos se repiten para cada objeto de la escena. En esta segunda fase del algoritmo se pueden diferenciar a un objeto de otros que están en la misma profundidad y se detectan los pliegues del objeto. Esto es porque el *Normal Map* muestra las discontinuidades en la orientación de las caras que componen al objeto.

Como último paso, se juntan los detalles que se encontraron el *Depth Map* con los *Normal Map* para generar una sola imagen que contengan los detalles de los dos pasos anteriores. En la Figura 13 se muestran los pasos del algoritmo de Hertzmann, en el extremo superior izquierdo se puede observar el *Depth Map* y debajo de él está el *Normal Map*; en las imágenes del centro se muestran los detalles obtenidos en cada uno de los pasos anteriores; mientras que en la imagen de la derecha se muestra el conjunto de los detalles obtenidos de ambas imágenes.

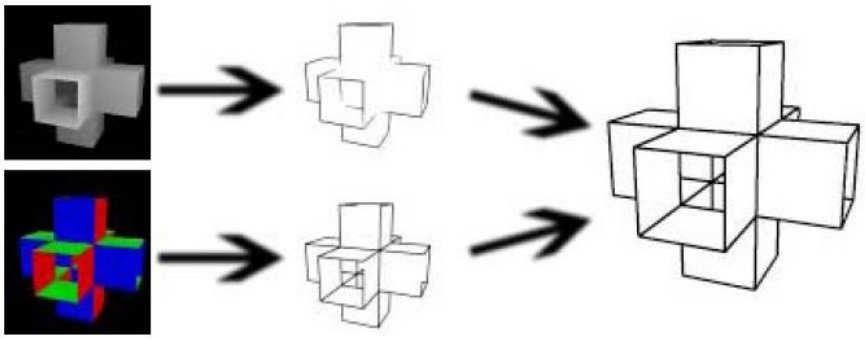

Figura 13: Pasos del algoritmo de Hertzmann para encontrar los detalles de un objeto.

Filtro de detección de bordes.

Una técnica propuesta para la implementación del shader de detección de bordes es la utilización de los filtros del Kernel. La técnica analizada en este epígrafe es el "*Filtro Canny*" (Mitchell, y otros, 2004).

El algoritmo general de implementación del filtro de detección de bordes Canny propone las siguientes operaciones (Mitchell, y otros, 2004):

I. Aplicar el filtro Gaussian Blur.
II. Calcular la derivada parcial en cada texel en la dirección horizontal y vertical.
III. Calcular la magnitud del gradiente y dirección de la línea [tan^ (-1)] en cada punto.
IV. Comparar la vecindad de pixel en la dirección de la línea y realiza una supresión mínima.

El filtro de Gaussian es realizado con el objetivo de eliminar las altas frecuencias de ruido de la imagen de entrada antes de intentar localizar los bordes de los objetos. El resultado final que se obtiene en este paso es una versión de la imagen original ligeramente borrosa. Este efecto es analizado en epígrafes anteriores.

El siguiente paso es el cálculo de las derivadas parciales (*Gx* y *Gy*), en las direcciones *vertical y horizontal* respectivamente. El algoritmo de Canny utiliza cuatro filtros para detectar las derivadas parciales para cada dirección, algunos de estos filtros podrían ser el de "*Roberts*", "*Prewitt"*, "*Sobel"*. Algunos de estos filtros podrían ser consultado en libro "Direct3d ShaderX: vertex and pixel shader tips and tricks" (Mitchell, 2002). En la Figura 14 se muestra la imagen resultante después de utilizado el filtro Sobel.

Después de buscadas las derivadas parciales están las condiciones creadas para hallar la magnitud del gradiente de cada texel de la imagen usando la Ecuación 5 con el objetivo de buscar los bordes de los objetos de la imagen:

$$G = \sqrt{Gx^2 + Gy^2}$$

Ecuación 5: Gradiente de cada texel.

La dirección de los bordes en ese texel es calculada usando la Ecuación 6:

$$\theta = arctan\left(\frac{Gy}{Gx}\right)$$

Ecuación 6: Dirección de cada punto.

El valor de la dirección de los bordes es redondeado de acuerdo al rango de valores que va desde $-\pi$ hasta π.

Figura 14: Magnitud del gradiente utilizando el filtro Sobel. A la izquierda imagen original, a la derecha imagen filtrada.

Una vez que todas las direcciones de los bordes son conocidas, es necesario particionar estos valores (en grados) en sectores de acuerdo a las características de la aplicación. Estos sectores deben ser simétricos y estarán acotados a una colección de pixel tomados de la dirección del gradiente. La búsqueda de la vecindad de cada texel puede ser fácilmente realizado llevando el valor de cada ángulo θ a un rango de valores que va desde 0 a 1, esta conversión se realiza multiplicando θ por π (180 grados); finalmente este valor es multiplicado por 4. Esta técnica es una excelente forma de reducir el número de comparaciones desarrolladas entre los punto de la muestra.

Conjuntamente con la búsqueda de la vecindad de cada punto se compara la magnitud del texel en proceso con la magnitud de sus texel vecinos. Si la magnitud del texel en proceso es mayor que la de su vecindad, entonces ese punto seria un máximo local y su valor es mantenido; pero si es menor entonces el valor de este punto es fijado a 0. Este proceso es conocido como "*supresión mínima*", su objetivo es principal es retener los texel que son considerados como máximos locales en la dirección del gradiente. En el proceso final, se aplican thresholds o umbrales a la imagen con el objetivo de reducir el número de bordes falsos que podrían aparecer. Los umbrales con frecuencia son ubicados cuando se busca realiza un correcto balance entre los bordes verdaderos y los falsos.

Una vez que el filtro de Canny es desarrollado se obtiene una imagen binaria donde son representados los pixel que pertenecen a la silueta de los objetos, ver Figura 15 (Mitchell, y otros, 2004).

Figura 15: Imagen resultante del filtro Canny.

2.4 Efecto Motion Blur.

El efecto Motion Blur o difuminado de movimiento aparece en muchas fotografías debido a la limitada velocidad de obturación de la cámara. “En una imagen, si cualquier objeto se mueve a gran velocidad con relación a la velocidad de obturación de la cámara este se verá borroso a lo largo de la dirección del movimiento relativo” (Wikipedia, 2008). Este efecto óptico se aprecia en escenas donde los objetos están en movimiento o si la cámara se mueve durante el tiempo de exposición (este tiempo transcurre desde que el obturador de la cámara se abre hasta que se cierra).

En este epígrafe se analiza el método tradicional para la simulación del motion Blur, el efecto que se produce también es conocido como “Temporal antialiasing”; esta técnica tiene la desventaja de producir una imagen fantasma o el efecto de doble visión sobre la escena como se muestra en la Figura 16. El segundo método analizado es una vía más eficiente para la generación del efecto difuminado de movimiento debido a que utiliza los datos de las velocidades de los vértices como entrada para calcular las velocidades de cada pixel de la imagen apoyándose en las nuevas tecnologías del HW gráfico.

Figura 16: Imagen fantasma o efecto temporal antialiasing que se produce en el método tradicional para simular el efecto motion Blur.

Motion Blur o temporal antialiasing.

Para reproducir un movimiento real se toman muestras del mismo a intervalos de tiempo regulares, y este muestreo puede sufrir el denominado "**aliasing temporal**", relacionado con el aliasing geométrico. El aliasing temporal se llama también "efecto estroboscópico" porque un muestreo insuficiente hace que el movimiento se perciba como iluminado con una luz estroboscópica. Para evitarlo, se usa el **difuminado de movimiento** ("motion blur") que recoge en un solo fotograma varias posiciones del objeto en movimiento. (Curso 2004/05)

Esta técnica a modo general consiste en la selección de una tira de frame y superponerlas todas en una sola aplicándole el grado de alfa (medida de opacidad de los pixel) decreciente según la antigüedad de la imagen.

Esta estrategia de programación propone hacer múltiples llamadas al método encargado de dibujar el objeto al cual se quiere aplicar el post-proceso. Antes de cada llamada la posición del objeto es actualizado ligeramente, produciendo una serie de imágenes borrosas sobre la superficie renderizada. Si los objetos tienen una o más texturas la simulación puede mejorar este efecto renderizando la primera imagen del objeto con todas sus texturas casi transparente. En cada tiempo que los objetos son renderizados, la transparencia de la textura de los objetos decrecen. (Microsoft Corporation, 2005)

Ahora analizaremos dos vías para la implementación de este algoritmo general:

La **primera vía** consiste en mantener una colección de pequeños buffer auxiliares para almacenar los frames previamente renderizados, entonces se mezcla cada frame sobre la cima del que se esta utilizando en ese momento disminuyendo el peso de un frame a otro como se muestra en la Figura 17 (Kylmamaa, 2006).

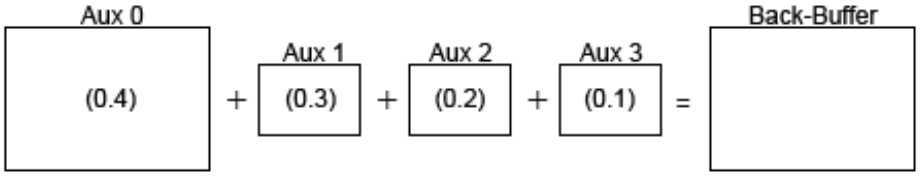

Figura 17: Método de mezcla aditiva, con ejemplos de pesos mezclados mostrados dentro de los buffer auxiliares.

Este método tiene la desventaja de representar un efecto blur bastante limitado debido a que es poco factible para almacenar un elevado número de frames previamente renderizados. El frame anterior también es almacenado generalmente como 1/4 o 1/16 del tamaño del buffer teniendo en cuenta el consumo de la memoria lo cual reduce su resolución. Esto es por regla general aceptable puesto que los frames mezclados están destinados para que la escena parezca un poco borrosa.

La **segunda vía** analizada consiste en almacenar el frame actual para mezclarlo recursivamente con el frame anterior como se muestra en la Figura 18 (Kylmamaa, 2006), en cuyo caso ningún buffer de trabajo extra es necesitado para el efecto, y ninguna solución se pierde en el frame anterior.

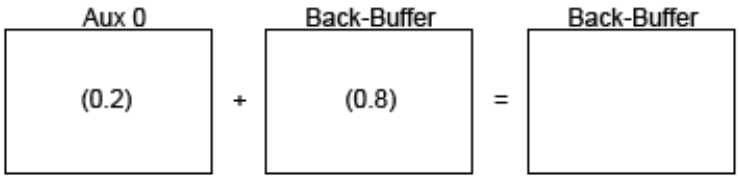

Figura 18 Mezcla recursiva, con ejemplo de pesos mezclados dentro de los buffer.

Este método tiene una particular desventaja, porque aunque es muy fácil producir un intenso efecto blur, hay que destacar que si un frame previo tiene un peso elevado puede producir una imagen fantasma que permanece sobre la escena. Esto método tiene que hacerse con una precisión exacta del tamaño de los buffer, los cuales generalmente tienen 8 bits por cada cannel o canal.

Para analizar cómo podría repercutir esta desventaja en la calidad de las simulaciones vamos a tomar como ejemplo hipotético un buffer de 8 bits, quedando distribuido de la siguiente forma: un buffer auxiliar que contiene un valor de 0/255 para el actual buffer renderizado, y el back buffer contiene un valor de 1/255 para el frame anterior, la operación de mezcla entonces podría resultar en: 0*0.2 + 1*0.8

= 0.8. El valor resultante de 0.8 entonces podría ser redondeado a 1. Esto significa que incluso si el frame actual es totalmente oscuro, una imagen fantasma desde el frame anterior podría permanecer en la escena.

Motion Blur usando mapas de velocidad per-pixel. (Microsoft Corporation, 2005) (Rosado, 2007)

Otra de las estrategias utilizada para la simulación del post-proceso Motion Blur es la generación de los mapas de velocidad usando múltiples objetivos de render (puede ser una textura) e imprime la información de la velocidad hacia uno de los objetivos de render. “La principal desventaja de esta técnica es que requiere modificar todos los shader de la escena para adicionar el código que permite calcular la velocidad per-pixel de salida” (Rosado, 2007), pero a su vez incrementa la sensación de velocidad en cualquier entorno virtual.

Este método utiliza los buffer de profundidad como una textura de entrada a los programa de pixel shader para generar mapas de velocidad de la escena. Los pixel shader calculan las posiciones del espacio mundo para cada pixel usando los valores de profundidad –estos valores son almacenados en el buffer de profundidad- junto con la matriz vista-proyección del frame actual. Una vez determinada la posición espacio-mundo para un determinado pixel, transformamos estas posiciones usando la matriz vista-proyección del frame anterior. Finalmente calculamos la diferencia de la posición del puerto de visión (Superficie física de la pantalla, o dispositivo de salida) entre el frame actual y el frame anterior con el objetivo de generar los valores de posición per-pixel.

El primer paso de esta técnica es la extracción de las posiciones de los objetos desde el buffer de profundidad. Cuando un objeto es renderizado y sus valores de profundidad son escritos en los buffers de profundidad. Los valores almacenados en este buffer tienen la coordenada z interpolada de un triángulo dividido por la coordenada w interpolada de un triángulo después que los tres vértices de los triángulos son transformados por las matrices vista-proyección. Usando el buffer de profundidad como una textura, podemos extraer las posiciones del espacio-mundo de los objetos que fueron renderizados hacia este buffer para transformar la posición del puerto de visión en los pixel por la inversa de la actual matriz vista-proyección y entonces multiplicamos el resultado por la componente w. Es importante definir la posición viewport como la posición de los pixel es el espacio viewport, las componentes x e y están en el rango de -1 hasta 1 con el origen (0,0) en el centro de la escena; la profundidad almacenada en el buffer de profundidad para que los pixel transformen las componente z, y la componente w es ajustada a 1.

Una vez escogidos los valores de velocidad per-pixel, es necesario realizar una pasada a lo largo de la dirección de esta velocidad en el buffer de color, a la misma vez acumulamos los valores de los colores y dividimos por el numero de muestras para obtener un average de color y escribimos estos valores como el color final en esos pixel. (Microsoft Corporation, 2005)

2.5 Efecto Bloom.

El efecto de Post-procesamiento *Bloom* (Microsoft Corporation, 2005) (Kylmamaa, 2006) simula la sensación de deslumbramiento que se produce en el ojo humano al estar expuesto a entornos con un alto grado de iluminación, como es el caso de los exteriores soleados, los interiores con tubos fluorescentes o la acción de entrar de un día soleado a una habitación oscura y viceversa. Esta técnica permite que la luz de un objeto sobre iluminado se desborde sobre las partículas que lo rodean, logrando aumentar el brillo de los blancos y la oscuridad de los negros. La sensación final que se logra es casi la necesidad de cerrar los ojos al verlo.

Note que al ser el Bloom un efecto de Post procesamiento, el tiempo de ejecución requerido por él, solamente depende de la resolución de la escena, sin tener en cuenta la complejidad de iluminación de la misma.

Este efecto es complejo de realizar porque debe someterse a varias pasadas antes de conseguir el acabado final. Lo primero que se hace es aplicarle al buffer de render un efecto de saturación (un filtro), para quedarse con las zonas brillosas de la escena, las cuales se pondrán de color blanco, las demás áreas se dejan de color negro. El segundo paso es aplicarle un Blur horizontal a la imagen saturada. El tercer paso es aplicar otro Blur pero esta vez en la dirección vertical, y por último lo que hacemos es combinar la imagen original con la imagen resultante de los tres pasos anteriores.

Después de analizar el algoritmo general a seguir para crear un efecto Bloom en cualquier simulación, vamos a profundizar en los métodos más utilizados actualmente.

“Existen diferentes formas de designar las aéreas relucientes en una escena. Aparte de trabajar con el buffer de puntos flotantes e implementar un real HDR, también puede usarse un filtro Bright-pass para extraer las aéreas brillosas de la escena regular, o el método alfa channel usado para renderizar por separado la geometría del efecto”. (Kylmamaa, 2006)

La técnica filtro Bright-pass (Kylmamaa, 2006) (Microsoft Corporation, 2005)**.**

El filtro Bright-pass es una de las técnicas más utilizada en el reconocimiento de las zonas brillosas de una escena. Este filtro utiliza la técnica de Tone mapping para determinar las aéreas serán brillosas

en la escena final y excluye los demás datos. “El Tone mapping simula el automático control de la exposición de la luz que realiza el ojo humano o sea este método regula la cantidad de luz que entra a una escena” (Microsoft Corporation, 2005).

El filtro del Bloom generalmente consta de los siguientes pasos:

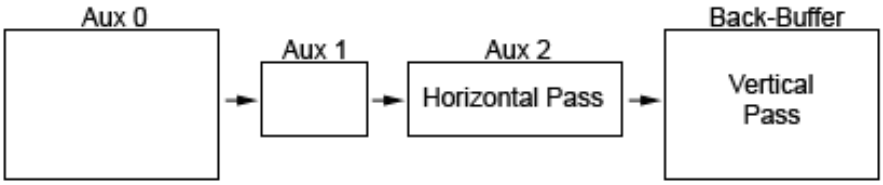

Figura 19: Etapas y pasos del filtro Bright-Pass.

El proceso de filtrado mostrado en la Figura 19 (Kylmamaa, 2006) se ejecuta después que la escena es renderizada hacia una textura de punto flotante que tiene la misma dimensión que el Back-Buffer. Por regla general el efecto Bloom es iniciado con la extracción de la información necesaria hacia interior de un reducido buffer de trabajo (Aux1), en este paso la imagen se vuelve borrosa y es escalada por debajo de 1/8x1/8 de la escala de la textura original de la escena. Después se le aplica a un buffer de gran tamaño (Aux2) el efecto de Gaussian blur horizontal y al buffer de tamaño normal (Back-buffer) se el aplica el efecto de Gaussian blur vertical.

Esta secuencia de pasos es ilustrada en la Figura 20 (Microsoft Corporation, 2005).

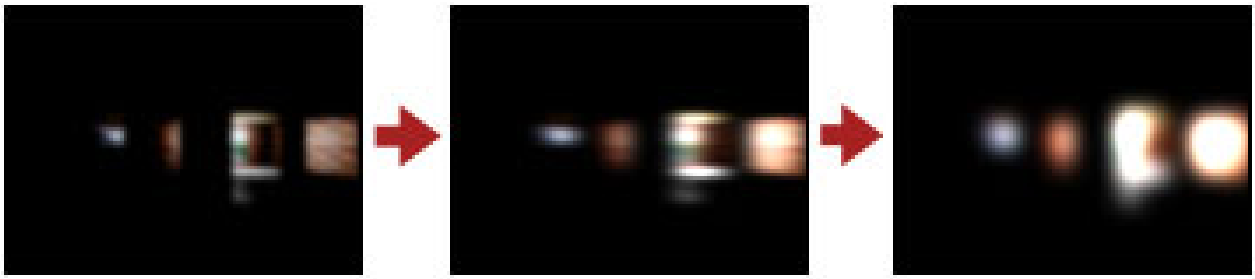

Figura 20: Secuencia de imágenes que ilustra que ilustra la técnica filtro Bright-Pass.

Después de aplicado el filtro Bloom, este efecto muestra un numero de texel en las proximidades del origen del texel que está en proceso y asigna los texels auxiliares decreciendo los pesos de mezcla(conocido como el peso Beta) a partir del centro; los pesos de mezcla son valores de puntos flotantes que están en el rango que va de 0.0 a 1.0 y están codificados en el formato del vértice donde el valor de 0.0 significa que el vértice no está mezclado con la matriz y 1.0 significa que el vértice es afectado por la matriz. Para lograr que la muestra de texel sea suficientemente larga es necesario

poner borrosa la textura a lo largo de la dirección horizontal, luego esta textura se hace borrosa a lo largo de la vertical para completar el proceso, como se muestra en la Figura 21 (Kylmamaa, 2006):

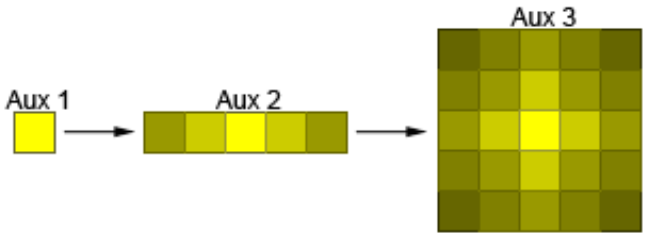

Figura 21: Proceso que permite poner borroso los texel de una textura a lo largo de la dirección horizontal (Aux2) y vertical (Aux3).

"Como parte final de este post-proceso la textura es escalada al tamaño del back buffer usando un filtro bilineal (modalidad del filtro linear) y adicionada en la salida de la escena" (Microsoft Corporation, 2005). Este método primero calcula la dirección de los texel, usualmente esta dirección no es un entera, luego busca los texel de dirección entera, la cual es enmarcada para calcular la dirección.

Una implementación inexperta podría mostrar cada pixel en el centro del texel con la apropiada colección de pesos para ese texel en cuestión. Sin embargo, es posible optimizar este proceso utilizando un filtro bilinear para el origen de la textura y el muestreo de los limites de los texel, este filtro permite que un gran número de muestras sean tomadas para mejorar el efecto o haciendo el mismo efecto con menos muestra y de este modo optimizando función.

Figura 22: Muestras de texel sacados por el filtro bilineal.

En el método de 3 muestras, ver Figura 22 (Kylmamaa, 2006), el punto de muestreo no está situado exactamente en las fronteras del texel. En este caso tomamos en consideración la cantidad de pesos de mezcla para cada texel y debido a que los texel están de cierta forma inquieta o en movimiento, el filtro bilineal muestra automáticamente el peso relativamente correcto de cada texel comparándolo con su vecino.

2.6 Efectos de perturbación de texturas.

La técnica de perturbación de la escena renderizada se puede utilizar en el desarrollo de múltiples efectos, tales como la simulación de aturdimiento, ver Figura 23 (Kylmamaa, 2006), el *efecto de radiación calorífica de la superficie (heat-shimmer)* (Oat, 2004), la simulación de las ondas expansivas de las explosiones (blast waves), *efecto de neblina (wispy clouds)* (Isidoro, 2002), la ondulación del agua (rippling water), entre otros.

Figura 23: Imagen tomada del juego Battlefield 2, las fuerzas especiales exhibieron un efecto de perturbación de imagen cuando el jugador se encontraba aturdido por un gas lacrimógeno.

"Estos efectos están generalmente basados en perturbar las coordenadas del origen de la textura usando las instrucciones de seno y coseno. En el caso de los pixel shader modelo 1.4, estas instrucciones no están disponibles y podrían simularse a través del muestreo de una textura de seno. La perturbación también podría moverse al nivel per-vertex si usa un blanco cuadrado altamente formado por mosaicos. En este caso, como mínimo un vertex shader modelo 2.0 sería requerido para usar las instrucciones de coseno y seno dentro del GPU. Alternativamente, los vértices serian alterados dentro del CPU para cada frame" (Kylmamaa, 2006).

2.6.1 Efecto de radiación calorífica.

Cualquier persona que ha estado al aire libre durante un día de verano intensamente soleado probablemente ha observado el calor reluciente que se desprende de una superficie caliente como cuando un sol enfurecido irradia sobre el asfalto de una carretera o autopista.

"Este efecto es una técnica conocida como "espejismo de calidad inferior" y es el resultado de rayos de luces mezclados como si cruzaran entre las capas de un denso aire de frio sobre la carretera y la expansión del aire caliente sobre la carretera. Porque el índice de refracción de los diferentes gases

exhibidos como su cambio de densidad, refractan diferentes luces a medida que pasa el aire a través de diferentes temperaturas. Una fuente de calor como una piscina de lava, con un enorme volumen, un movimiento rápido de la corrientes, exhibe un pronunciado y animado efecto de radiación calorífica". (Oat, 2004)

El algoritmo básico para la reproducción del fenómeno visual de la conversión del calor en gas es el siguiente (Oat, 2004):

1) Crear una textura RGBA renderizada que es del mismo tamaño como el back buffer.
2) Limpiar la textura RGBA a (0.0, 0.0, 0.0, 1.0)
3) Renderizar la escena completa hacia la textura renderizada, escribiendo el color a RGB y una escala de profundidad/distorsión para alfa.
4) Unir la textura renderizada a una de las unidades de la textura y renderizar una escena cuádruple alineada hacia el back buffer. Usar la normalización de las coordenadas del dispositivo (NDC) como las coordenadas de la textura dentro de la textura para dar la escala de la distorsión almacenada en el alfa channel o canal de alfa. Compensar las coordenadas NDC de acuerdo a la escala de distorsión y volver a probar la textura para obtener los valores distorsionados de RGB y la salida de estos del back buffer.

En el proceso de simulación del efecto de radiación calorífica es necesario determinar que cantidad de distorsión es aconsejable aplicar a cada pixel de la escena. Tres de los métodos utilizados para generar estos valores de distorsión son *Cálculo de los valores de profundidad per-pixel* (Oat, 2004), modelar una *Geometría de calor* (Oat, 2004) y modelar una *textura de calor* (Oat, 2004), los cales serán reseñados a continuación.

Cuando la luz hace contacto con las capas de la atmósfera lo hace a su vez con múltiples bolsas de aire de diferentes densidades, por lo cual mientras mayor sea el número de las bolsas de aire interceptadas también aumentan la probabilidad de que este fenómeno se realice. Por lo tanto en las simulaciones de este fenómeno óptico juega un papel fundamental la profundidad de la escena. La distancia desde el ojo puede ser calculada proyectando el vértice dentro del espacio del ojo y extrayendo la componente z de la posición del vector. El valor per-vértice de profundidad podría ser transmitido por los pixel shader, pero primero es importante limpiar el buffer de destino usando un valor alfa que representa el valor de distorsión máxima, con el objetivo de descartar cualquier pixel que no este escrito en el frame. Este proceso es solamente y estrictamente necesario si no estamos dibujando cada pixel en el frame buffer.

El método de Geometría de calor propone renderizar un valor de distorsión dentro del alfa channel de un buffer fuera de la pantalla. En una simulación la Geometría del calor es dibujada en forma de malla sobre la escena completa y su único propósito es identificar los valores de distorsión del calor para el alfa channel. Al observar el calor reluciente que se desprende de una superficie caliente, nos podemos dar cuenta de que no existe una línea distintiva que muestra donde empieza y termina este efecto, de este modo para mantener la integridad de la simulación de este efecto de radiación calorífica es necesario quitar los llamados bordes duros o picos.

Después de modelada la "Geometría de calor" o malla sobre la escena es importante evitar dibujar filos cortantes dentro del alfa channel del buffer fuera de la pantalla con el propósito de que la simulación sea lo mas real posible, con este objetivo el valor alfa generado por la malla de calor podría desteñirse ligeramente en los bordes del calor reluciente simulado.

El último método analizado es de Textura de calor, el cual es aconsejable utilizarlo para volúmenes de gases homogéneos y cerrados. Cuando en una simulación aumenta la distribución aleatoria de la densidad del volumen de gases, la geometría de calor modelada podría ser mejorada usando la modelación de una textura de calor. La textura de calor desplazada a través de la superficie de la geometría de calor es modulada per-pixel con los valores de distorsión aportados por la misma geometría de calor y profundidad.

2.6.2 Perturbación basada en fuego.

Un interesante efecto creado a través de la técnica de perturbación de imágenes es el *aparente movimiento tembloroso* (*flickering fire)* (Isidoro, 2002) que se observa en el fuego. Tradicionalmente este efecto fue logrado como una animación, un efecto de *retroalimentación de texturas* (Forsyth, 2004) (McCurskey, 2002) o un *sistema de partículas* (González Campistruz, y otros, 2007), sin embargo cada uno de estos métodos tiene sus limitaciones.

"Renderizar un fuego como una animación tiene la desventaja de parecer poco realista y necesita mucha memoria de textura para almacenar los fotogramas de la animación. En adición a esto, si existe muchos fotogramas en la escena y todos usan la misma animación, la imagen incluso podría parecer más artificial. Para simular la aleatoriedad del fuego, la carrera de las animaciones no es una buena elección". (Isidoro, 2002)

La textura de retroalimentación es otra de las técnicas utilizadas para la simulación en tiempo real del fuego. La idea que guía este método es generar por cada fotograma una fila de pixel aleatorios en el

fondo de la textura, en un segundo paso hacemos borrosa y desplazamos la imagen hacia arriba en la textura. Este método tiene a su favor la característica de ser completamente aleatorio, pero la técnica repetitiva de hacer borroso el proceso de la textura de retroalimentación no simula los bordes afilados que tiene el fuego real, por lo tanto este efecto en tiempo real es poco realista.

“Los sistemas de partículas también pueden ser usados para crear el efecto del fuego. Aunque esta técnica puede simular el fuego utilizando valores naturales aleatorios; requiere mucho tiempo del CPU para calcular los valores físicos del sistema de partículas y limita la cantidad de partículas necesarias para simular un fuego realista. En adición, las partículas de fuego no tienen la opacidad del fuego real”. (Isidoro, 2002)

El efecto flickering fire del fuego basado en la perturbación de textura supera a las técnicas analizadas anteriormente. Esta técnica sólo requiere dos texturas que contienen valores de color de la escala de grises: un mapa de perturbación y un mapa base; y a su vez requiere mucho menos memoria que la técnica del fuego animado. Debido a la forma en que usamos los mapas de perturbación, la animación en tiempo real del fuego basada en perturbación de textura elimina la forma repetitiva en que se realiza este efecto con la técnica de la textura de retroalimentación y a la vez conserva los bordes afilados presentes en el mapa base, esta característica es fundamental para lograr una eficiente y realista simulación del fuego. Con la programación de los shader este efecto podría usar los datos de los vértices para sembrar los offsets de los mapas de perturbación a fin de que los múltiples fotogramas en la misma escena puedan tener diferentes apariencias.

“El objetivo de este shader consiste en desplazar tres imágenes de texturas perturbadas por delante unas de otras en la dirección vertical y adicionar los resultados junto a los valores de perturbación obtenidos; utilizándose la misma textura para cada desplazamiento” (Isidoro, 2002). Con el uso de estos tres mapas aseguramos que la repetición del efecto ocurra después de un largo tiempo.

Los valores de perturbación son ligeramente atenuados utilizando los valores de coordenadas de la textura perturbada. La atenuación es usada para debilitar la perturbación cerca de la base del fuego con el objetivo de hacer el tope del fuego más parpadeante que la base. El resultado final de este shader es un arreglo de los valores de color de los pixels de la imagen perturbada; donde el valor 0 representa las zonas frías y el valor 255 representa las zonas de más calor de la imagen, ver Figura 24.

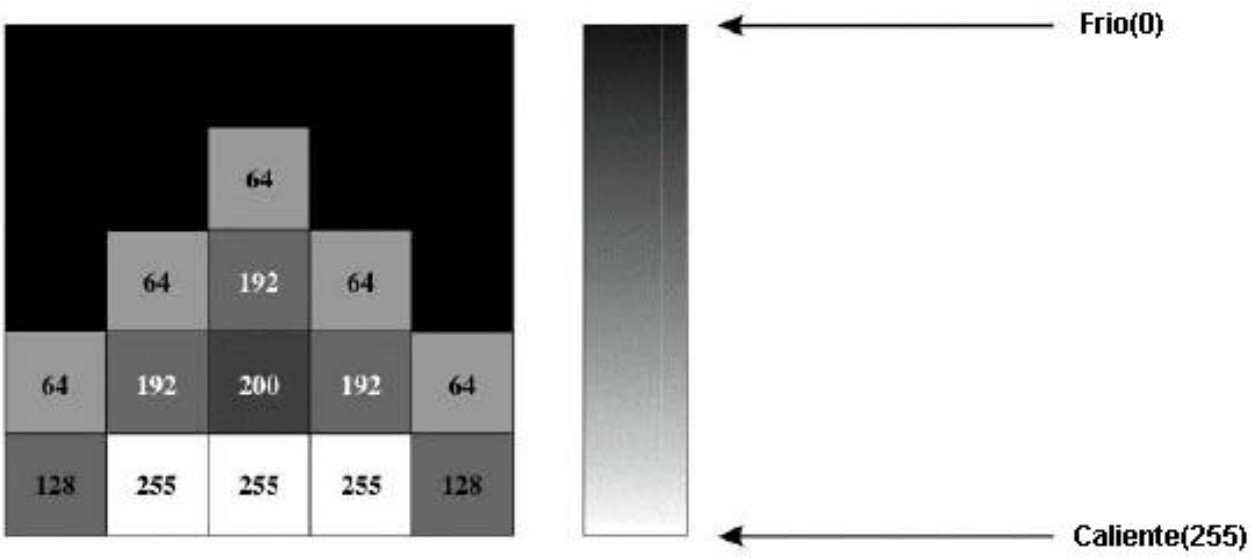

Figura 24: Arreglo de los pixel de una textura perturbada para la simulación del efecto fuego. Cada pixel determina la cantidad de calor de cada zona de la imagen.

2.6.3 Efecto neblina.

Uno de los efectos más fáciles que pueden ser realizados usando perturbación de texturas es el efecto de neblina (Wispy Clouds). “La idea es desplazar dos texturas las cuales se cruzan en direcciones opuestas. La primera textura es la textura de perturbación, la cual tiene almacenadas las perturbaciones u y v en los canales de la textura rojo y verde respectivamente” (Isidoro, 2002).

La segunda textura es la textura de nube, la cual tiene sus coordenadas de perturbación de textura para el mapa de perturbación. Para la simulación de este efecto se propone la utilización de una textura de nube en la escala de grises y luego se introduce dentro del canal alfa del mapa de perturbación. Debido a este proceso de empaquetamiento, solo una textura es necesaria para cada capa de nube. Entonces esta textura es tomada dos veces desde los shader, primero para obtener los valores de perturbación, luego para leer las texturas perturbadas.

Con el objetivo de hacer que las nubes parezcan más reales, es mejor usar múltiples capas de nubes. Las capas podrían ser combinadas en muchas formas diferentes. Una sencilla propuesta para combinar las capas es la interpolación linear entre ellas sin el uso de los pixel shader. Esto brinda la apariencia de una capa de nube en el tope de otra. Las capas de nubes son desplazadas por encima unas de otras a diferentes velocidades para dar un efecto de profundidad.

En orden de dar a las nubes un color gradiente desde una escala de grises del mapa de textura, usamos operaciones de multiplicación, entonces la textura de nube es escalada por 2 y sus valores de color son mezclados con los valores de la escala de grises representados en el mapa de textura, con el objetivo de variar el *nivel de pureza (saturación)* del pixel en proceso. El gradiente (transición suave y sin saltos de un color a otro) es el resultado de los rangos de modulación (operación de mezcla) entre los colores que componen la textura. Por ejemplo al mezclar un pixel naranja brillante RGB (255, 128,0) saturado al 100% con otro de color gris medio RGB (128, 128,128) de saturación nula se obtiene un pixel de color intermedio entre los dos colores anteriores: el naranja gris; el pixel resultante de la modulación anterior tiene el mismo matiz y luminosidad que el naranja brillante pero con un 50 % de saturación.

Debido a que la misma escala de grises de la textura es usada para los valores de intensidad y transparencia, las regiones negras de las nubes son completamente transparentes. Las dos capas de nubes son interpoladas linealmente con el fondo. El fondo de la textura es justamente un gradiente desde color morado al color azul.

2.6.4 Cristal de plasma.

En el campo de la simulación de fenómenos de la vida real, es posible implementar otros efectos utilizando la técnica de perturbación de textura, uno de estos efectos es la *deformación de plasma* (Isidoro, 2002). Esta técnica produce algo similar a las esferas de cristal de plasma que surgen de la corriente de plasma hacia la superficie.

El algoritmo de implementación de este post-proceso propone usar dos desplazamientos de texturas, usando cada una como un mapa de perturbación y un mapa base a la vez. “El mapa base es almacenado en las componentes RGB de la textura y el mapa de perturbación es almacenado en la componente alfa. Cada mapa de textura perturba el pixel subsiguiente buscado del otro mapa de textura” (Isidoro, 2002).

Matemáticamente este efecto es casi idéntico al método de dos capas de nubes utilizado en el efecto de neblina, pero la apariencia de estos post-procesos es radicalmente diferente. La mayor parte de estas diferencias podría ser atribuida al uso de diferentes texturas bases y de perturbación. (Isidoro, 2002)

El efecto de plasma podría combinarse como un bloque constructivo para crear más complejos efectos. Por ejemplo, el efecto del plasma podría ser combinado con un efecto de vidrio para crear una esfera de plasma de vidrio:

El primer paso es usar la técnica de mapeo de ambiente, usando la modalidad de *mapeo de ambiente esférico* (Microsoft Corporation, 2005). Esta técnica se puede observar como una textura especial que contiene una imagen de la escena alrededor de un objeto. Un mapa esférico es una representación 2D de una vista de 360 grados de la escena alrededor de un objeto, ver Figura 25 (Microsoft Corporation, 2005):

Figura 25: Mapa de ambiente esférico.

En un segundo paso le aplicamos a la textura una apariencia más parecida al cristal, para este proceso el mapa de ambiente esférico es combinado con el *término Fresnel per-pixel* (Brennan, 2002). Este término permite crear una apariencia semejante a un cristal lleno de baches o escarcha.

El término de Fresnel es calculado con un simple producto vectorial entre la normal per-pixel N y un vector E normalizado del ojo, ver Ecuación 7. El resultado del producto vectorial de N*E, entonces es usado como parámetro para una función F(x) con el objetivo de obtener una intensidad de reflexión.

$$F(x) = N * E$$

Ecuación 7: Intensidad de reflexión.

El tercer paso consiste en interpolar linealmente el entorno junto con el efecto de plasma para producir el resultado final. Para el valor final de los valores alfa, son adicionados el término Fresnel per-pixel y los valores de transparencia de plasma.

2.7 Efecto Cartoon.

El área de la graficación por computadora ha estado avanzando a pasos gigantescos desde su nacimiento, siempre manteniendo fija su meta de lograr generar imágenes que sean tan vívidas y parecidas a la realidad que puedan ser confundidas con una fotografía, a esta corriente o ideología se le llama *foto-realismo.* Pero con el avance actual se ha descubierto que la graficación por computadora además de producir imágenes foto-realista puede producir otro tipo de imágenes, como las que se encuentran en ilustraciones técnicas, dibujos animados o incluso las imitaciones de un estilo artístico.

El valor de una imagen generada por computadoras está dejando de ser evaluada sólo por su brillantez o la complejidad de los algoritmos utilizados para generarlos. Ahora se ha comenzado a tomar más en consideración el contenido de la imagen, la similitud que tenga con un estilo artístico y hasta el sentimiento que una imagen sea capaz de despertar sobre el espectador. A esta área de la informática gráfica que no intenta generar imágenes foto-realistas se le llama "*Representación No Foto-realista (NPR)*" (Card, y otros, 2002) (Fernando, y otros, 2003).

El efecto Cartoon o Cell Shading es una técnica informática de NPR basada en el trabajo de Walt Disney, Hanna-Barbera y otras compañías productoras de dibujos animados. Esta poderosa vía de rendering normalmente consiste en dotar a las imágenes de un aspecto de dibujos animados mediante el uso de bordes gruesos y colores vivos con o sin degradación. Esto le confiere un aspecto más simple a la imagen, como si se hubiese realizado con una técnica habitual de dibujo a mano.

En la mayoría de los mundos virtuales la iluminación es usada con el propósito de dar una apariencia realista a los escenarios, sin embargo el objetivo del efecto Cartoon consiste en reducir la variación del sombreado producido por el proceso de iluminación.

En el proceso de implementación del shader de este post-proceso existes tres pasos fundamentales (Fernando, y otros, 2003):

I. El sombreado difuso (Shading diffuse) de la escena necesita ser representado por dos valores: uno para regiones brillosas, y otro para las regiones oscuras.
II. La iluminación especular (Specular highlight) de la escena necesita ser identificada y representada como un color individual donde su intensidad es suficientemente alta.
III. Los bordes de la silueta de los objetos de la escena son perfilados para completar el look de Cartoon.

Modelo de iluminación difusa.

La *iluminación difusa* (Engel., 2002) simula la emisión de luz sobre un objeto desde una fuente de luz determinada usando el modelo de iluminación difusa. “Este modelo está basado en una ley física conocida como “La ley de Lambert”, donde se declara que la luz reflejada es determinada por el coseno entre el vector normal (N) y el vector de luz (L)” (Engel., 2002).

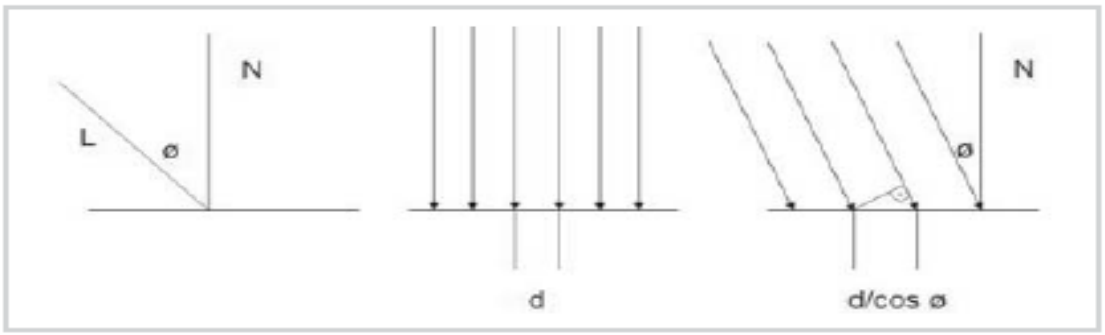

Figura 26: Iluminación difusa.

En la zona izquierda de la Figura 26 (Engel., 2002) se muestra la interpretación geométrica de la ley de Lambert. En la zona central se muestra los rayos de luz incidiendo perpendicularmente sobre la superficie en una distancia d de separación entre los rayos y la intensidad de los rayos es inversamente proporcional a esta distancia. En la zona de la derecha de la figura se muestra como los rayos de luz forman un ángulo ø con el vector normal del plano, ilustrando que la misma cantidad de luz que pasa por un costado del ángulo recto del triangulo es reflejada desde la región de la superficie correspondiente a la hipotenusa del triángulo. Debido a esta relación la longitud de la hipotenusa es calculada por d/cos ø.

Por definición, el producto vectorial para este modelo puede ser calculado mediante la Ecuación 8 (Engel., 2002):

$$N\ dot\ L = \| N \| \| L \| cos\emptyset$$

Ecuación 8: Iluminación difusa.

En el modelo usual de iluminación difusa usamos el valor resultante de la Ecuación 8 para escalar el color del vector de forma tal que este vector sea oscurecido basándose en la cantidad de luz que recibe. Este proceso podría verse como una ligera transición desde la luz proyectada sobre un objeto hasta las sombras proyectadas sobre la superficie. Pero realmente lo que el modelo Cartoon necesita es la realización de una abrupta transición entre unas pocas sombras de la escena.

Sombreado Difuso. (Fernando, y otros, 2003)

Un shader Cartoon necesita reducir los tonos de color de la iluminación difusa. Para este proceso de conversión de color utilizaremos la regla matemática conocida como función escalera (step function). A diferencia de las funciones tradicionales que tienen rango de valores continuos, la función escalera utiliza dos valores distintos. Esta función permite dividir un extenso rango de valores de color a dos rangos.

Basado en este principio, la zona difusa del shader calcula el producto vectorial entre las componentes N y L del modelo de iluminación difusa analizado. Este cálculo determina qué cantidad de luz recibe el vértice y luego clasifica este valor resultante como "Iluminación" o Oscuridad", ver Ecuación 9:

$$S = \hat{N} dot \hat{L}$$

Ecuación 9: Cantidad de luz que recibe un vértice.

Si S<0 el ángulo entre L y N es mayor de 90 grados, este implica que la superficie no recibe luz.

Para mapear estos valores desde un rango de valores continuos a una función escalera usamos una textura lookup. En este caso es suficiente con usar una textura 1D debido a que el producto vectorial de N y L es un valor escalar. Los valores de iluminación difusa son remplazados con los valores correspondientes de la función escalera y luego los codificamos en la textura lookup como la primera coordenada de textura.

Iluminación especular.

Comparado con el modelo de iluminación difusa, el modelo de (Engel., 2002) sólo depende de la posición del observador. Este modelo es conocido como el "modelo de iluminación de Blinn-Phong"; de acuerdo con el cual un tono de luz especular se observa cuando el observador está próximo a la dirección de reflexión especular. La intensidad de la luz disminuye bruscamente cuando el observador se aleja de la dirección del reflejo especular.

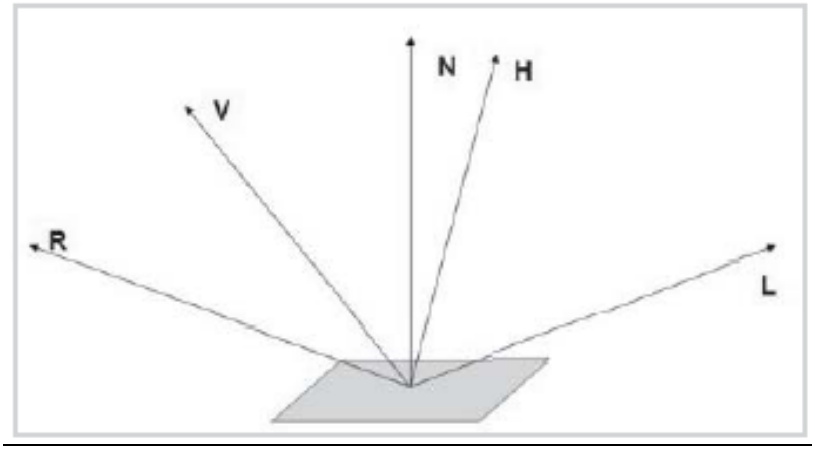

Figura 27: Vectores para iluminación difusa: vector de luz (L), posición del observador (V), orientación de la superficie (N), dirección del vector de reflexión (R), vector que divide el ángulo entre el vector L Y V (H).

Este modelo, ver Figura 27 (Engel., 2002), puede ser calculado a través del producto vectorial entre N y el vector imaginario H, este vector es definido como la mitad entre L y V.

$$(NdotH)^n = (Ndot((L+V)/2))^n$$

Ecuación 10: Iluminación especular**.**

Donde *n* representa el exponente que califica las propiedades brillosas de las superficies en un rango que va desde 1 hasta infinito.

El trabajo con la componente especular es muy similar al realizado con la componente difusa. La idea es aplicar la función escalera para calcular los valores de iluminación especular, de modo que en vez de tener una variación gradual en la intensidad, cada tono de luz está habilitado o no está habilitado en la escena. Estos valores son codificados en la textura lookup como la segunda coordenada de textura. (Fernando, y otros, 2003)

La iluminación difusa ayuda a visualizar la silueta de los objetos mientras que la iluminación especular brinda una nueva percepción sobre las propiedades de las superficies de los objetos. Si la idea del efecto Cartoon consiste en representar la estructura de los objetos, la implementación del shader podría omitir la porción de iluminación especular de la escena y remplazarla con algunas regiones difusas adicionales.

Perfilando las siluetas de los objetos.

Para completar este efecto, es necesario delinear los bordes de las siluetas de los objetos de la escena. Para este proceso buscamos los pixel que están ubicados sobre la silueta de los objetos y los colorea con el color del perfil (normalmente negro). (Fernando, y otros, 2003)

Una sencilla vía (Fernando, y otros, 2003) para buscar la silueta de los objetos es la utilización del producto vectorial N*V como una vara de medir. Al igual que el producto vectorial N*L mide la cantidad de luz que recibe una superficie, N*V mide que cantidad de superficie es visible desde el actual viewpoint. Cuando calculamos la luminosidad para un punto, este punto está iluminado cuando el valor de la ecuación N*L es positivo y esta borroso cuando el valor de la ecuación N*L es negativo. De igual forma un punto es visible cuando N*V es positivo y no esta visible cuando N*V es negativo. Los puntos en los cuales N*V esta limitado a cero representa una transición de visible a no visible.

Otra vía para identificar las siluetas de los objetos en una imagen es la utilización del post-proceso Detección de bordes (Edge Detection) analizado en epígrafes anteriores.

CAPÍTULO 3
DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

En el presente capítulo se realiza una modelación del entorno donde se usará la aplicación como respuesta al problema planteado. Se obtiene una visión del sistema del sistema propuesto desde el punto de vista del negocio y se dan los primeros pasos en su concepción práctica partiendo de las necesidades del producto.

3.1 Características de la biblioteca de efectos de Post-procesamiento.

La biblioteca que modela los efectos de post-procesamiento debe representar una colección de post procesos que faciliten a los programadores de sistemas de realidad virtual la inclusión de efectos de post-proceso con el objetivo de aumentar el realismo de las simulaciones en tiempo real. Estos sistemas deben realizar la modalidad de render a textura, debido a que la propuesta de la biblioteca está concebida para ser aplicada a imágenes.

Los efectos de post-proceso no dependen de la complejidad de la escena, debido a que pueden ser fácilmente incluidos después que el render de la escena es completado. El tiempo requerido para producirlos solamente depende de la resolución de la escena, sin tener en cuenta la complejidad de iluminación u otros detalles de la misma.

En la modelación y diseño la biblioteca se propone como estructura de datos básica al *Vector* de la *STL (Standard Template Library)* de C++, por las facilidades que brinda dicha estructura. El uso de esta lista de efectos permite una cómoda manipulación y acceso, aprovechando también la gestión dinámica de la memoria que hace, optimizando de esta forma el trabajo con biblioteca.

El lenguaje de programación que se empleará es el C++ debido a su alto nivel, el cual incorpora una variedad de características que facilitan una programación elegante y modular, además el código escrito en C++ estándar permite mantener su portabilidad hacia otras plataformas.

Se ha escogido el uso de los aceleradores gráficos (GPU) como la vía de implementación para la simulación de los efectos de Post-procesamiento debido a que contribuye enormemente al aumento de la velocidad de la simulación. Además se conoce que el GPU tiene más capacidad de procesamiento que el CPU, ya que solamente está destinada para el gráfico y puede procesar cientos de gigaflops para el cálculo de un simple punto flotante mientras que en la CPU solamente serían 12 gigaflops.

Para la implementación de los shader de los efectos de post-proceso se propone el lenguaje de alto nivel para la programación de la GPU Cg, debido a su compatibilidad con las bibliotecas gráficas OpenGL y Directx.

3.2 Aportes de la biblioteca de efectos de Post-procesamiento.

Partiendo de la situación problémica planteada en este trabajo, se puede afirmar que la elaboración de la biblioteca para la simulación de efectos de post-procesamiento constituye un gran aporte a los

entornos virtuales realizados por los proyectos productivos de la Facultad 5. Entre los beneficios que brinda la biblioteca está:

- Permite que se pueda integrar los efectos de post-procesos a cualquier entorno virtual independientemente de la complejidad de sus escenarios.
- Brinda la posibilidad de mejorar el realismo de las simulaciones.
- Ofrece al programador un trabajo más fácil y cómodo.

3.3 Modelo del dominio.

El modelo de dominio es una de las alternativas que brinda RUP para la identificación de requisitos y la comprensión del contexto cuando existe poca estructuración en los procesos de negocio, y con la que se le puede mostrar al usuario de manera visual los principales conceptos que se manejan en el dominio del sistema, sus partes y sus relaciones. Este modelo se realiza a través de un diagrama de clases de UML simplificado, en el cual se representan las clases conceptúales que pueden intervenir en el sistema y sus asociaciones preliminares, así como los objetos más importantes en el mismo.

El modelo que se muestra en la Figura 28 refleja la relación entre los conceptos involucrados en el desarrollo de la aplicación. Estos conceptos constituyen la base de la arquitectura del sistema, la cual se encuentra descrita en el siguiente capítulo.

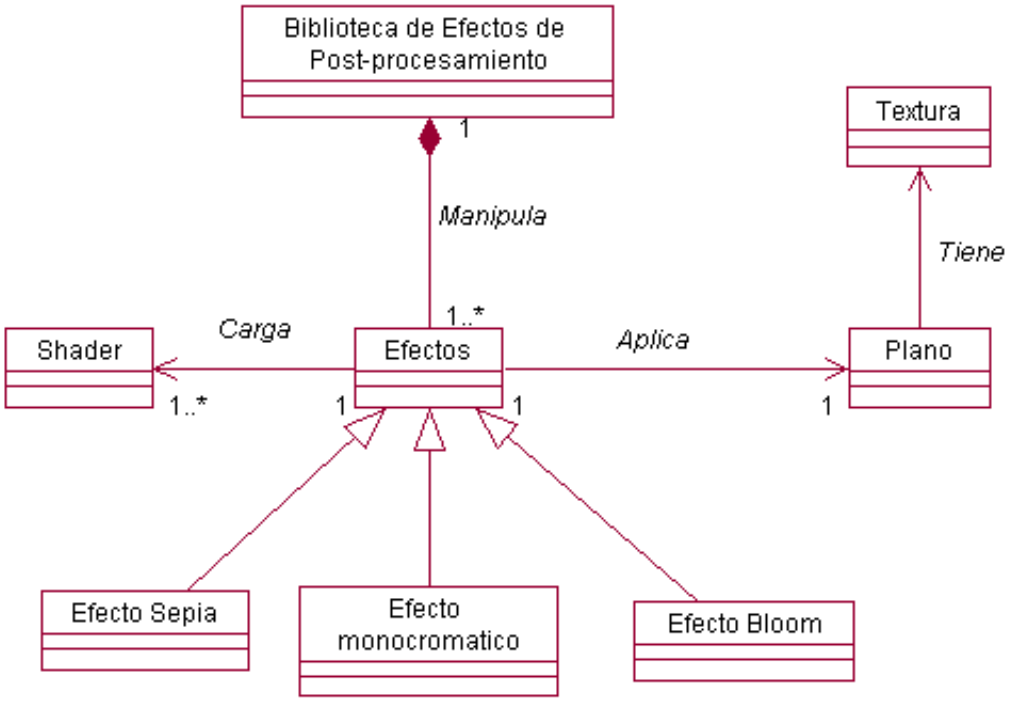

Figura 28: Diagrama del modelo del dominio.

3.3.1 Elementos del dominio.

Biblioteca de efectos de Post-procesamiento: Controlador o manipulador de los efectos de Post-procesamiento.

Plano: Superficie, normalmente rectangular, que delimita la parte de la escena donde se cargara una textura para aplicar los efectos de post-procesos.

Textura: Imagen de la superficie del plano.

Efecto Sepia: Post-proceso que convierte una textura determinada a un tono sepia.

Efecto Monocromatico: Post-proceso que convierte una textura determinada a un tono monocromático.

Efecto Bloom: Post-proceso que agrega resplandor a ciertas zonas brillosas de la textura.

Shader: Ficheros con el código a interpretar y almacenar para su posterior uso.

3.4 Captura de requisitos.

Un proyecto no puede ser exitoso sin una descripción detallada, correcta y exhaustiva de los requerimientos. Estos definen lo que debe hacer un sistema y la forma en que debe hacerlo. A continuación se presentan los requisitos funcionales y no funcionales con los que deberá cumplir el sistema.

3.4.1 Requisitos funcionales.

1. Crear efecto.
2. Verificar extensiones del fichero de efecto (shader).
3. Cargar shader.
4. Asignar textura.
5. Aplicar efecto a una textura.
6. Eliminar efecto insertado.

3.4.2 Requisitos no funcionales.

Usabilidad**:** Los futuros usuarios del sistema serán programadores con conocimientos básicos con respecto al tema de los efectos de post-procesamientos. El producto debe estar concebido para que el usuario piense en que desea hacer y no como hacerlo.

Soporte: En su versión inicial debe ser compatible con el sistema operativo Windows, pero debe estar preparado para que con rápidas modificaciones pueda migrar hacia Linux.

Hardware: Pentium 3, 128 Mb Ram, Acelerador Gráfico 128 Mb VRAM, soporte para shader Model 2.0.

Rendimiento: Diseñada para que 15 efectos de pantalla completa se ejecuten al menos en 30 FPS.

Diseño e implementación: Debe utilizar transparentemente la biblioteca gráfica OpenGL y DirectX. Se harán llamadas a dichas bibliotecas desde C++. Se regirá por la filosofía de Programación Orientada a Objetos.

3.5 Modelo de casos de uso.

En este epígrafe se darán a conocer los actores del sistema a desarrollar, y a partir de los requisitos funcionales del epígrafe anterior se obtendrán los principales casos de usos (CU) del sistema con sus

respectivas descripciones. En ellos se describe la secuencia determinada de eventos que realiza un actor en interacción con la aplicación.

3.5.1 Definición de los actores del sistema.

Tabla 1: Actor del sistema.

Actores	Justificación
Simulador	Es la aplicación que hará uso de la biblioteca, aplicando a sus texturas los efectos de post-procesamiento de la biblioteca propuesta.

3.5.2 Definición de los casos de uso del sistema.

Los casos de uso son la base del diseño del software, ya que a partir de ellos se derivan muchos de los posteriores procesos dentro del desarrollo del software.

1. Insertar Efecto
2. Aplicar Efecto.
3. Remover Efecto.

3.5.3 Diagrama de casos de uso.

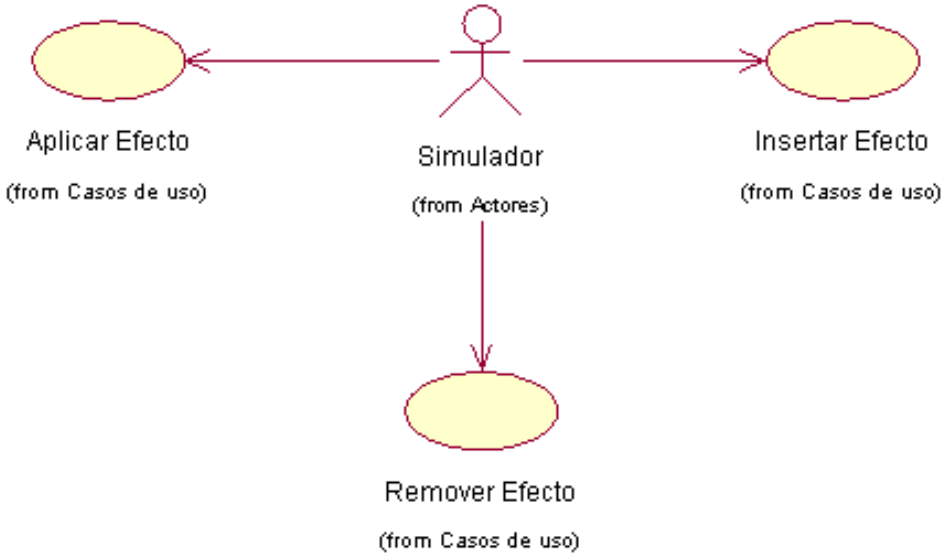

Figura 29: Diagrama de casos de uso.

Tabla 2 Listado de los casos de uso del sistema.

CU -1	**Insertar Efecto.**
Actor	Simulador.
Descripción	Permite adicionar un efecto a cualquier sistema de realidad virtual, se crea el objeto efecto, se cargan los shader.
Referencia	R1, R2, R3.
CU -1	**Aplicar Efecto.**
Descripción	Se asigna una textura a la cual se le aplican los efectos insertados en la lista de shader de la escena.
Referencia	R4, R5.
CU -1	**Remover Efecto.**
Actor	Simulador.
Descripción	Permite eliminar un efecto insertado en la lista de shader de la escena.
Referencia	R6.

3.5.4 Descripción de los casos de uso en formato expandido.

Tabla 3: CU "Insertar Efecto".

Nombre del caso de uso:	**Insertar Efecto**
Actores:	Simulador
Propósito:	Insertar los efectos de post-procesamiento, estructuras encargadas de almacenar e interpretar los shader.
Resumen: Se inicia cuando el simulador solicita la inserción de un efecto, se cargan los shader correspondientes. El caso de uso finaliza cuando el efecto es insertado en la lista de efectos de la escena.	
Referencias:	R1, R2, R3.
Curso normal de los eventos.	
Acción del actor	Respuesta del sistema
1- Solicita la inserción de un efecto pasándole el nombre del efecto.	
	2- Crea el objeto efecto.
	3- Verificar la existencia del archivo del shader.
	4- Verificar si la existencia es correcta.
	5- Carga el shader correspondiente al efecto seleccionado.
	6- Se inserta el efecto en la lista de shaders de la escena.
Curso Alterno	
Línea 2	
	2.1- Si el fichero no existe muestra un mensaje de error.
Línea 3	
	3.1- Si la extensión no es correcta muestra un mensaje de error.
Precondiciones:	-

Poscondiciones:	Efecto insertado
Prioridad:	Crítico

Tabla 4: CU "Aplicar efecto".

Nombre del caso de uso:	**Aplicar Efecto.**
Actores:	Simulador
Propósito:	Aplicar el efecto al entorno.
Resumen: Se inicia cuando el actor solicita ejecutar el efecto, se asignan las texturas. El CU finaliza cuando el efecto es visualizado.	
Referencias:	R4, R5.
Curso normal de los eventos.	
Acción del actor	Respuesta del sistema
1- Solicita aplicar el efecto pasándole como parámetro una textura.	
	2- Una vez de creado el efecto, se toma la lista de shader de la escena y se ejecutan los efectos insertados.
Precondiciones:	Efecto creado.
Poscondiciones:	Efecto aplicado.
Prioridad:	Crítico

Tabla 5: CU "Remover Efecto".

Nombre del caso de uso:	**Remover Efecto**
Actores:	Simulador
Propósito:	Eliminar el efecto que se encuentra en proceso.
Resumen: Se inicia cuando el simulador solicita eliminar un efecto aplicado, se destruye el objeto efecto creado.	
Referencias:	R8

Curso normal de los eventos.	
Acción del actor	Respuesta del sistema
1- Solicita eliminar el efecto insertado.	
	2- Se toma la lista de shader de la escena y se destruye la estructura del efecto seleccionado.
Precondiciones:	Efecto insertado.
Poscondiciones:	Efecto eliminado.
Prioridad:	Crítico

CAPÍTULO 4
CONSTRUCCIÓN DEL SISTEMA PROPUESTO

En la primera parte de este capítulo se presentan los diagramas de clases de análisis del sistema propuesto. Su objetivo es brindar una primera visión de las posibles clases del diseño a un alto nivel, pero donde se dejan ver sus responsabilidades y relaciones, aún sin el mayor rigor entre ellas.

A continuación se presentan los diagramas de clases como resultado del refinamiento de las etapas anteriores. Se presentan además los diagramas de secuencia de la realización de los CU. Además se muestran los diagramas de componentes y el modelo de despliegue pertenecientes al flujo de trabajo de implementación, los cuales guiarán una mejor implementación del producto.

4.1 Modelo de análisis.

El análisis consiste en obtener una visión del sistema que se preocupa de ver qué hace, de modo que sólo se interesa por los requisitos funcionales.

En la construcción del modelo de análisis se identifican las clases que describen la realización de los casos de uso, los atributos y las relaciones entre ellas. Con esta información se construye el diagrama de clases del análisis. Las clases que se identifican están asociadas con el contexto del dominio del problema por lo que representan conceptos y relaciones.

En la Figura 30 se muestran las clases principales que conforman el sistema, empleando para ello un modelo de análisis.

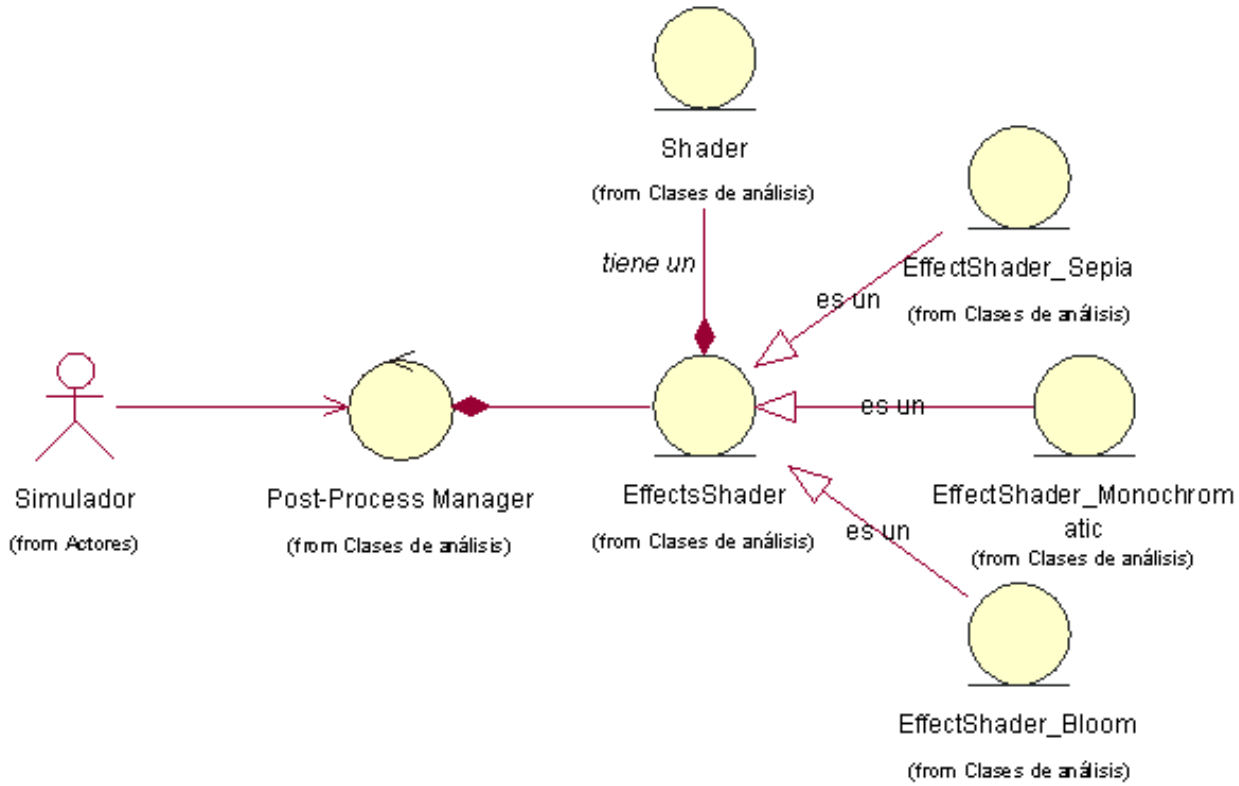

Figura 30: Modelo de análisis.

4.2 Modelo del diseño.

El diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, este debe ser suficiente para que el sistema pueda ser implementado sin ambigüedades.

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. En este modelo, los casos de uso son realizados por las clases del diseño y sus objetos.

4.2.1 Patrones de diseño.

"En la terminología de objetos, el patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas. Muchos patrones ofrecen orientación sobre cómo asignar las responsabilidades a los objetos ante determinada categoría de problemas. (…) Los patrones no se proponen descubrir ni expresar nuevos principios de la ingeniería de software. Todo lo contrario: intentan codificar el conocimiento, las expresiones y los principios ya existentes: cuanto más trillados y generalizados, tanto mejor." (Larman, 2004)

En el diseño de la aplicación se tuvieron en cuenta principalmente los patrones Experto, y Creador. El primero plantea que siempre se debe asignar una responsabilidad al experto en información, o sea, la clase con toda la información necesaria para llevarla a cabo. El segundo expresa que la responsabilidad de crear una instancia de una determinada clase debe asignarse a otra clase, siempre que esta agregue, contenga, registre o utilice específicamente los objetos de aquella.

4.2.2 Definición de los paquetes del diseño.

Los paquetes de diseño, ver Figura 31, constituyen una herramienta organizacional del modelo de diseño con el objetivo de agrupar elementos relacionados.

El paquete "Sistemas de realidad virtual" encapsula la clase que permite realizar el proceso de render, es necesario que se realice en su modalidad de *render a textura,* debido que los post-procesos son diseñados para ser aplicados texturas. Este paquete representa al actor del sistema analizado en el capitulo anterior, por lo cual no es necesario analizar su funcionamiento.

El paquete “Post-proceso Library” encapsula las clases necesarias para el funcionamiento de la propuesta de la Biblioteca de efectos de post-procesamiento, estas clases serán analizadas en el próximo epígrafe.

Figura 31: Diagrama de paquetes del diseño.

4.2.3 Diagramas de clases del diseño.

En este epígrafe se verá todo lo referente al diagrama de clases del paquete “Post-Process Library”, ver Figura 32. El diagrama de clases es uno de los elementos más importantes dentro de un proyecto de software, ya que brinda una visión bastante completa de todo el sistema, mostrando todas las clases con sus métodos y atributos, así como las relaciones entre estas.

En el diagrama de clases del diseño se modela clase de análisis “Shader” como un método de virtual de la clase de diseño “CEffects”.

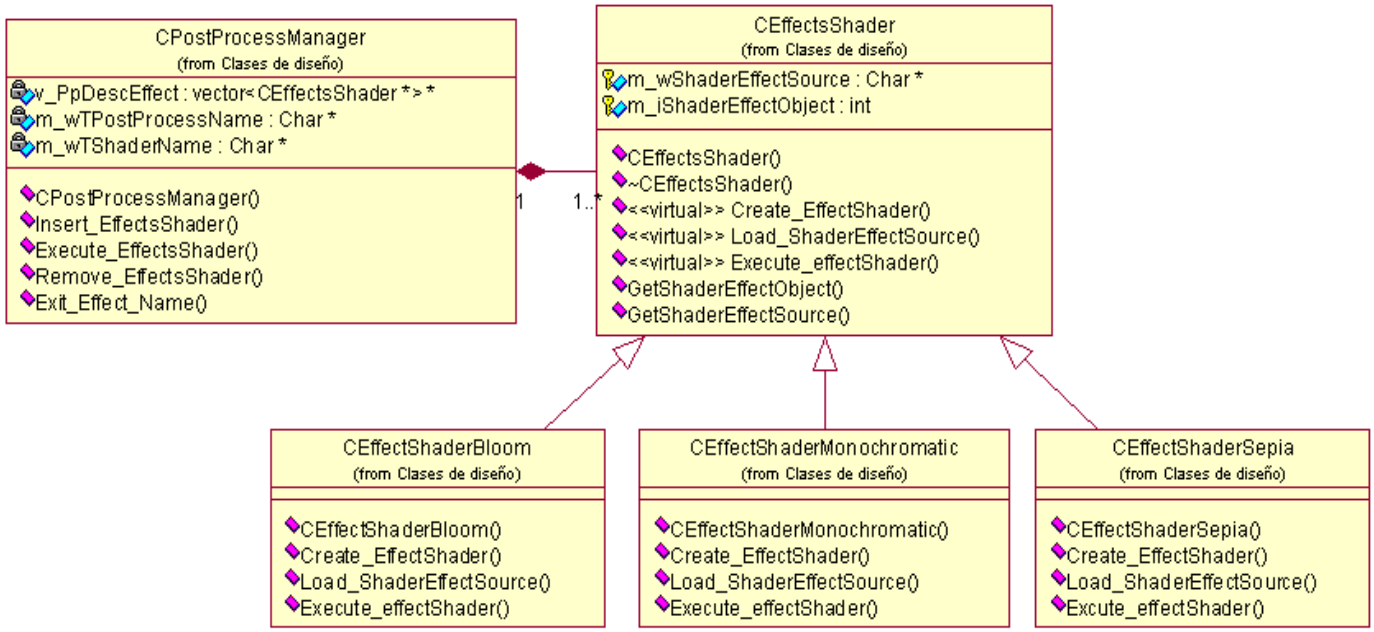

Figura 32: Diagrama de clases del paquete *“Post-Process Library”*.

La clase principal de la biblioteca efectos de post-procesamiento es la clase "**CPostProcessManager**", la cual consta de un arreglo que lista el nombre de los Post-Procesos disponibles, un segundo arreglo con los nombres de los ficheros de las técnicas (Shader) correspondientes a los efectos disponibles y un tercer arreglo polimórfico que contiene los efectos de Post-Procesos activados por el usuario para ser aplicado a una textura, este último atributo está representado mediante la relación de composición con la clase abstracta "**CEffectsShader**", de la cual heredan los futuros Post-Procesos implementados.

En este trabajo se propone para una primera versión de la biblioteca la inclusión de los efectos Bloom, Monocromático y Sepia representado en el diagrama de clases por las clases de diseño "**CEffectShaderBloom**", "**CEffectShaderMonochromatic**" y "**CEffectShaderSepia**" respectivamente. Estas clases conforman la jerarquía de clases dada en su raíz por la clase "**CEffectsShader**"; esta estructura jerárquica permite la inclusión de nuevos post-procesos sin afectar el diseño actual, logrando que la herramienta sea enriquecida con la posterior inclusión de nuevos efectos.

La clase base "**CEffectsShader**" es diseñada como una clase abstracta con el objetivo de garantizar que las clases hijas mantengan cierta interfaz común, de esta forma esta clase sirve de patrón o modelo para la creación de nuevas clases que aporten la funcionalidad concreta que quedó pendiente en sus métodos virtuales. Esta clase representa cada entrada del arreglo que lista los Post-Procesos activados en la clase controladora, consta de dos atributos protegidos uno de ellos es un arreglo para almacenar el nombre del fichero que contiene el nombre de la técnica correspondiente a un determinado efecto y el otro atributo es utilizado para almacenar la posición en que se encuentra la técnica en el arreglo de técnicas de efectos disponibles en la biblioteca.

4.2.4 Descripción de las clases del diseño.

Tabla 6: Clase del diseño "CPostProcessManager".

Nombre: CPostProcessManager	
Tipo de clase: Controladora	
Atributo	**Tipo**
v_PpDescEffect	vector<CEffectsShader*>
m_wTPostProcessName	Char *
M_wTShaderName	Char*
Para cada responsabilidad	
Nombre:	**CPostProcessManager()**

Descripción:	Constructor de la clase.
Nombre:	**Insert_EffectsShader(Char * m_wNameShader)**
Descripción:	Función encargada de insertar un efecto pasándole como parámetro el nombre del post-proceso, el cual se busca en el arreglo de efectos disponibles de la biblioteca y se busca la técnica correspondiente a este efecto, se inicializa el objeto efecto y se inserta en la lista de Post-Procesos de la escena.
Nombre:	**Execute_EffectsShader(CTexture)**
Descripción:	Función encargada de aplicar a una textura los efectos guardados en la lista de post-procesos de la escena.
Nombre:	**Remove_EffectsShader(CEffectsShader * v_PpEffect)**
Descripción:	Función encargada de eliminar un efecto de la lista de Post-Procesos de la escena.
Nombre:	**Exit_Effect_Name(Char *)**
Descripción:	Función que se encarga de buscar en el arreglo de efectos disponibles de la biblioteca el nombre de un post-proceso dado y devolver la posición en que se encuentra.

Tabla 7: Clase de diseño "CEffectsShader"

Nombre: CEffectsShader	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_wShaderEffectSource	Char*
m_iShaderEffectObject	int
Para cada responsabilidad	
Nombre:	**CEffectsShader()**
Descripción:	Construye un objeto de tipo CEffectsShader.
Nombre:	**~CEffectsShader()**
Descripción:	Destruye y libera la memoria de un objeto de tipo CEffectsShader.
Nombre:	**Create_EffectShader()**
Descripción:	Función virtual heredada que permite crear la técnica

	correspondiente a un determinado tipo de efecto, se carga el fichero del shader correspondiente al efecto.
Nombre:	**Load_ShaderEffectSource()**
Descripción:	Función virtual heredada encargada de cargar el fichero del shader correspondiente guardado en el atributo m_wShaderEffectSource.
Nombre:	**Execute_effectShader(CTexture)**
Descripción:	Función virtual heredada encargada de ejecutar la técnica correspondiente a un efecto y aplicarlo a una textura.
Nombre:	**Get de cada uno de los atributos**
Descripción:	Manejador de datos que permite la obtención de los atributos m_wShaderEffectSource y m_iShaderEffectObject.

Tabla 8: Clase de diseño "CEffectsShaderBloom".

Nombre: CEffectShaderBloom	
Tipo de clase: Entidad	
Para cada responsabilidad	
Nombre:	**CEffectShaderBloom()**
Descripción:	Construye un objeto de tipo CEffectShaderBloom.
Nombre:	**Create_EffectShader()**
Descripción:	Método heredado y redefinido que se encarga de crear la técnica correspondiente al efecto Bloom, carga el shader correspondiente a este tipo de efecto.
Nombre:	**Load_ShaderEffectSource()**
Descripción:	Método heredado y redefinido que se encargada de cargar el fichero del shader correspondiente al efecto Bloom que se encuentra guardado en el atributo m_wShaderEffectSource heredado de la clase base.
Nombre:	**Execute_effectShader(CTexture)**
Descripción:	Método heredado y redefinido que se encarga de ejecutar la técnica correspondiente al efecto Bloom y lo aplica a una textura.

Tabla 9: Clase de diseño “CEffectShaderMonochromatic”.

Nombre: CEffectShaderMonochromatic	
Tipo de clase: Entidad	
Para cada responsabilidad	
Nombre:	**CEffectsShaderMonochromatic()**
Descripción:	Construye un objeto de tipo CEffectShaderMonochromatic.
Nombre:	**Create_EffectShader()**
Descripción:	Método heredado y redefinido que se encarga de crear la técnica correspondiente al efecto Monocromatico, carga el shader correspondiente a este tipo de efecto.
Nombre:	**Load_ShaderEffectSource()**
Descripción:	Método heredado y redefinido que se encargada de cargar el fichero del shader correspondiente al efecto Monocromático que se encuentra guardado en el atributo m_wShaderEffectSource heredado de la clase base.
Nombre:	**Execute_effectShader()**
Descripción:	Método heredado y redefinido que se encarga de ejecutar la técnica correspondiente al efecto Monocromático y lo aplica a una textura.

Tabla 10: Clase de diseño “CEffectShaderSepia”.

Nombre: CEffectShaderSepia	
Tipo de clase: Entidad	
Para cada responsabilidad	
Nombre:	**CEffectsShaderSepia()**
Descripción:	Construye un objeto de tipo CEffectShaderSepia.
Nombre:	**Create_EffectShader()**
Descripción:	Método heredado y redefinido que se encarga de crear la técnica correspondiente al efecto Sepia, carga el shader correspondiente a

	este tipo de efecto.
Nombre:	**Load_ShaderEffectSource()**
Descripción:	Método heredado y redefinido que se encargada de cargar el fichero del shader correspondiente al efecto Sepia que se encuentra guardado en el atributo m_wShaderEffectSource heredado de la clase base.
Nombre:	**Execute_effectShader()**
Descripción:	Método heredado y redefinido que se encarga de ejecutar la técnica correspondiente al efecto Sepia y lo aplica a una textura.

4.3 Diagramas de interacción del sistema.

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema. La mayoría de las veces, esto implica modelar instancias concretas o prototípicas de clases, interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un comportamiento. Los diagramas de interacción pueden utilizarse para visualizar, especificar, construir y documentar la dinámica de una sociedad particular de objetos, o se pueden utilizar para modelar un flujo de control particular de un caso de uso.

4.3.1 Realización del CU “Insertar Efecto”.

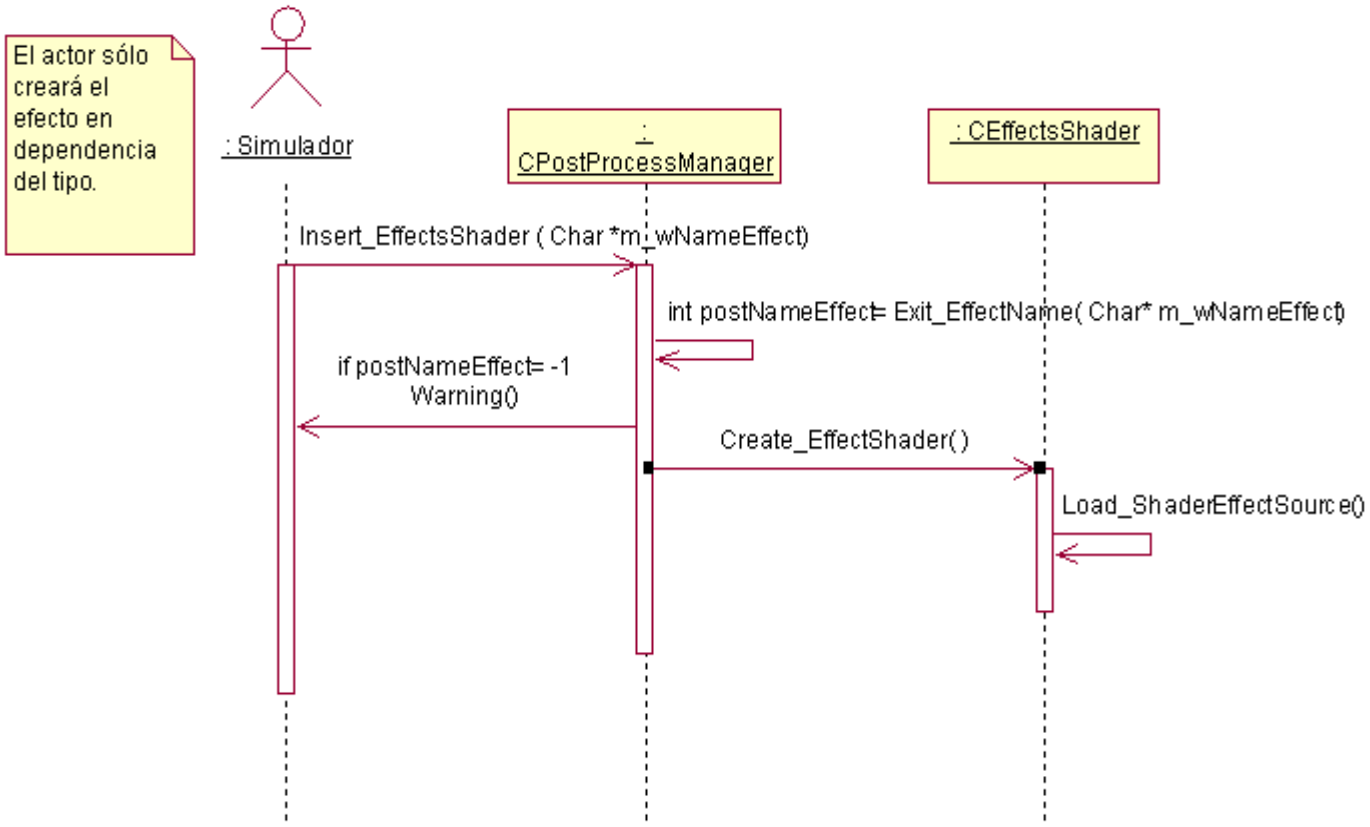

Figura 33: Diagrama de secuencia “Insertar Efecto”.

4.3.2 Realización del CU “Aplicar Efecto”.

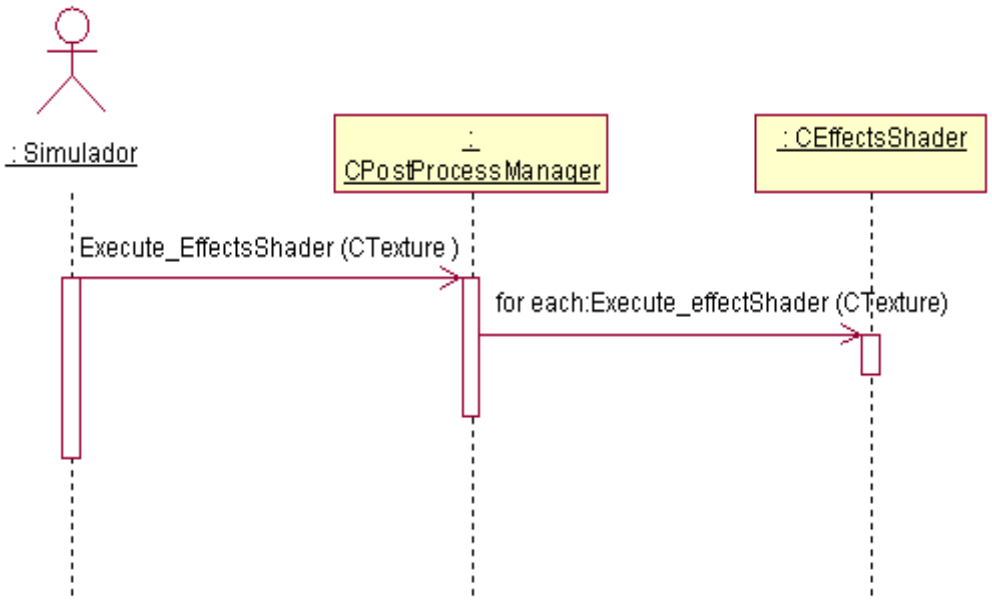

Figura 34: Diagrama de secuencia “Aplicar Efecto”.

4.3.3 Realización del CU “Eliminar Efecto”.

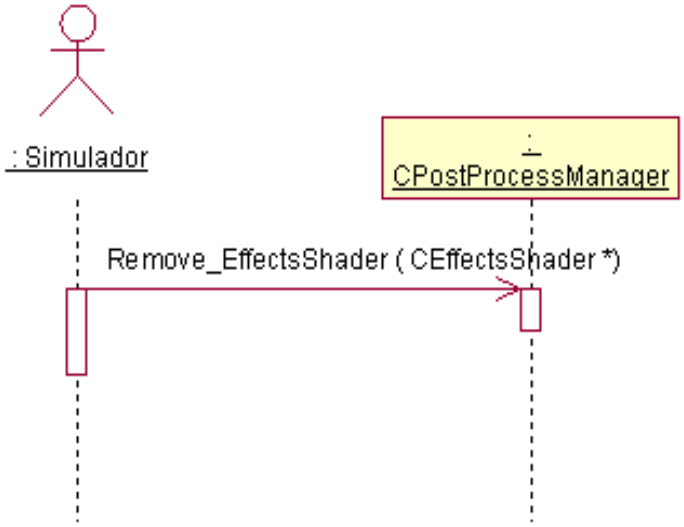

Figura 35: Diagrama de secuencia “Eliminar Efecto”.

4.4 Modelo de implementación.

En el modelo de implementación se describe cómo los elementos del modelo de diseño que se deben implementar en términos de componentes y cómo se organizan estos de acuerdo con los mecanismos de estructuración disponibles en el entorno de implementación y en el lenguaje de programación

utilizado; así como la dependencia entre los componentes. Esto se hace a través del diagrama de paquetes y del diagrama de componentes.

4.4.1 Diagrama de despliegue.

El diagrama de despliegue analizado en este documento es muy simple, sólo incluye la representación de una PC, por dicha razón quedó decidido que no es necesario representar dicho diagrama.

4.4.2 Subsistema de implementación.

Partiendo del concepto que enuncia al “Subsistema de Implementación” como un artefacto análogo a un “Paquete de Diseño”, se definieron dos subsistemas de implementación que corresponden con los paquetes de diseño analizados en el flujo de trabajo de diseño.

El subsistema “RV System” es análogo al paquete de diseño “Sistemas de realidad virtual”, este subsistema encapsula los componentes que permiten realizar el proceso de render, es importante que el sistema de realidad virtual que interactué con la biblioteca implemente la modalidad de *render a textura.* Debido a que este subsistema representa los componentes de implementación de un sistema externo, se decidió que no es necesario detallar sus componentes en un diagrama.

Figura 36: Subsistemas de implementación.

4.4.3 Diagrama de componentes.

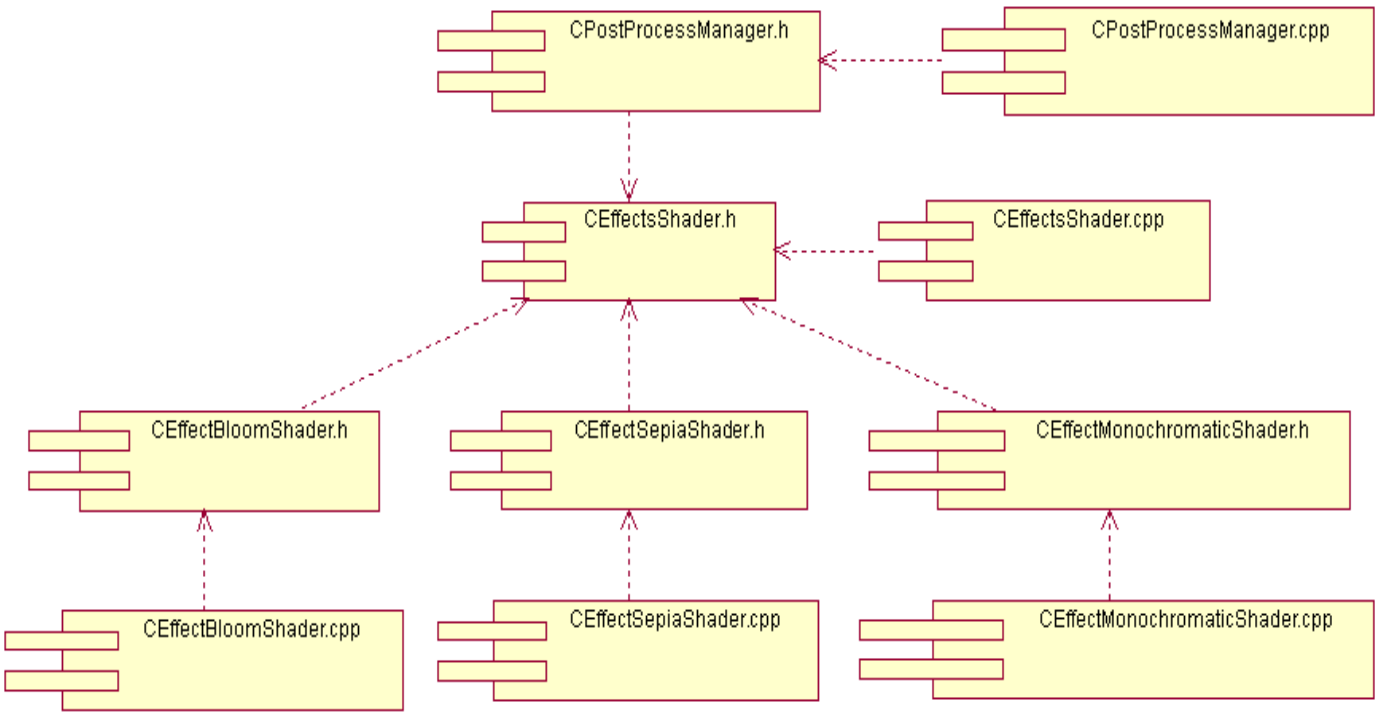

Figura 37: Diagrama de componentes.

CONCLUSIONES

Para el cumplimiento de los objetivos propuestos, primeramente se requirió realizar un estudio de las principales técnicas, tecnologías y tendencias actuales en relación con el tema de la visualización de efectos de post-procesamiento en tiempo real. En el estudio se analizó las potencialidades que brindas la nuevas tecnologías del hardware gráfico para la obtención de métodos más eficientes que contribuyan a mejorar la sensación de realismo y la calidad visual de los entornos virtuales. A partir de esta investigación se recopilaron las principales técnicas para la simulación de los efectos de post-procesamiento, se reseñan sus características y se hace un balance entre los algoritmos de cada efecto con el propósito de proponer los métodos más óptimos.

Posteriormente se elaboró una propuesta de diseño que guie una futura implementación de la biblioteca de efectos de post-procesamiento para entornos virtuales, para esta primera versión se propone la utilización de tres post-procesos con el objetivo de obtener la visualización de un modelo simple. Se hizo la captura de requisitos funcionales y no funcionales y la agrupación de los casos de uso del sistema. Se generaron algunos artefactos del flujo de trabajo análisis, diseño e implementación; destacándose la obtención de un diagrama de clases lo suficiente extensible que contribuye a la inclusión de otros post-procesos sin afectar la estructura de la biblioteca.

Este trabajo ha demostrado la importancia de utilizar las técnicas de efectos de post-procesamiento como una vía para mejorar el nivel de realismo e inmersión en la creación y visualización de los entornos virtuales.

Sin embargo, aún existen muy pocas investigaciones y trabajos en este campo. Esperamos que el continuo perfeccionamiento del hardware gráfico junto a las potencialidades que brinda la inclusión de estos efectos a los sistemas de realidad virtual, ofrezca un punto de partida para futuros estudios.

RECOMENDACIONES

Finalizado el trabajo de diploma donde se analizaron algunas técnicas para la simulación de efectos de post-procesamiento de imágenes empleadas para darle mayor sensación de realismo a los entornos virtuales. Se recomienda que:

Este trabajo sirva como punto de partida y base de estudio para futuras tesis o investigaciones relacionadas con las técnicas de generación de efectos, específicamente en el análisis de los métodos para la simulación de efectos de post-procesamiento de imágenes.

Investigar con más profundidad las potencialidades que brindan las nuevas tecnologías del HW gráfico para la simulación de efectos especiales en los EV.

Se genere un hábito de uso de shader para la creación de aplicaciones 3D en los grupos de proyecto del perfil “Realidad Virtual”.

Se haga uso del catálogo de efectos de post-procesamiento para apoyar la sensación de realismo de los EV, así como se implemente la biblioteca propuesta en los proyectos de la F5.

REFERENCIA BIBLIOGRÁFICA

Brennan, Chris. 2002. Per-Pixel Fresnel Terms. [aut. libro] Wolfgang F. Engel. *Direct3d ShaderX : vertex and pixel shader tips and tricks.* USA.: Wordware Publishing, 2002.

Card, Drew and Mitchell, Jason L. 2002. Non-Photorealistic Rendering with Pixel and Vertex Shaders. [aut. libro] Wolfgang F. Engel. *Direct3d ShaderX : vertex and pixel shader tips and tricks.* USA.: Wordware Publishing, 2002.

Chover, Miguel. 2004. Informática Gráfica. [Consultado en: diciembre/2007]. [En línea] 2004. http://www3.uji.es/~jromero/grafica.

Corral Martín, José María y García Sánchez, Álvaro F. Tema 7 Colores y Sombras. *Informática Gráfica.* [Consultado en: mayo/2008]. [En línea] http://gsii.usal.es/~corchado/igrafica/descargas/temas/Tema07.ppt.

Curso 2004/05. Tema 5 Animación por ordenador. [Consultado en: junio/2008]. [En línea] http://dis.um.es/~jfernand/0405/tsm/tema5.pdf.

Engel., Wolfgang F. 2002. Programming Vertex Shaders. *Direct3d ShaderX : vertex and pixel shader tips and tricks.* USA. : Wordware Publishing, 2002.

Fernando, Randima and Kelgard, Mark J. 2003. Nonphotorealistic Rendering. *The Cg Tutorial.* USA. : Addison Wesley, 2003.

Forsyth, Tom. 2004. Post-Process Fun with Effects Buffers. [aut. libro] Wolfgang F. Engel. *ShaderX2 : shader programming tips and tricks with DirectX 9.* USA. : Wordware Publishing,, 2004.

García Guzmán, Mario Ezequiel. 2004. Uso de las tecnologías del hardware gráfico en el apoyo al realismo en entornos virtuales arquitectónicos. Universidad de Colima. [Consultado en: diciembre/2007]. [En línea] septiembre /2004. http://digeset.ucol.mx/tesis_posgrado/Pdf/MARIO_EZEQUIEL_GUZMAN_GARCIA.pdf.

González Campistruz, Jaime y Octavio Herrera P, Lannie. 2007. *Efectos Especiales para Entornos de Realidad Virtual.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2007.

Isidoro, John, Riguer, Guennadi and Brennan, Chris. 2002. Texture Perturbation Effects. [aut. libro.] Wolfgang F Engel. *Direct3d ShaderX : vertex and pixel shader tips and tricks.* USA. : Wordware Publishing, Inc., 2002.

Kylmamaa, Marko. 2006. Creating a Post-processing Framework: Tips & Tricks. [Consultado en: enero/2008]. [Online] 2006. http://gamasutra.com/features/20061003/kylmamaa_01.shtml.

Larman, Craig. 2004. UML y Patrones. *Introducción al análisis y diseño orientado a objetos.* La Habana : Editorial Félix Varela, 2004.

Marwan, Y. Ansari y ATI Research. 2004. Fast Sepia Tone Conversion. [aut. libro] Andrew Kirmse. *Game Programming Gems 4.* USA : Charles River Media, 2004.

McCurskey, Mason. 2002. Chapter 13 Image feedback. *Special Effects Game Programming with Directx.* USA. : Premier Press, 2002.

—. 2002. Chapter 8 Basic texturing. *Special Effects Game Programming with Directx.* USA. : Premier Press, 2002.

Microsoft Corporation. 2005. *Microsoft DirectX 9.0 SDK Update (April 2005) Documentation.* 2005.

—. 2005. Linear Texture Filtering. *Microsoft DirectX 9.0 SDK Update (April 2005) Documentation.* 2005.

Mitchell, Jason L. 2002. Image Processing with 1.4 Pixel Shaders in Direct3D. [aut. libro] Wolfgang F. Engel. *Direct3d ShaderX : vertex and pixel shader tips and tricks.* USA. : Wordware Publishing, 2002.

Mitchell, Jason L., Ansari, Marwan Y. and Hart, Evant. 2004. Advanced Image Processing with DirectX 9 Pixel Shaders. [aut. libro] Wolfgang F. Engel. *ShaderX2 : shader programming tips and tricks with DirectX 9.* USA. : Wordware Publishing, 2004.

Oat, Chris, Tatarchuk, Natalya and ATI Research, Inc. 2004. Heat and Haze Post-Processing Effects. [aut. libro] Andrew Kirmse. *Game Programming Gems 4.* sUSA : Charles River Media, 2004.

Ogayar Anguita, Carlos Javier. 2006. Optimización de algoritmos geométricos básicos mediante el uso de recubrimientos simplistas. Universidad de Granada. [Consultado en: enero/2008]. [En línea] mayo/2006. http://hera.ugr.es/tesisugr/16135143.pdf.

Rosado, Gilberto. 2007. Chapter 27. Motion Blur as a Post-Processing Effect. [aut. libro] Hubert Nguyen. *GPU Gems 3.* USA : Addison Wesley Professional, 2007.

Santos., J., S. 2006. *Introducción a Shaders.*[Consultado en: diciembre/2007]. [En línea] 2006. http://adva.com.ar/foro/viewtopic.php?t=2099.

Sanz Montemayor, Antonio, Cabido Valladolid, Raúl and Sánchez Calle, Ángel. 2004. Desarrollo de Aplicaciones 3D con Integración de Código Cg. Universidad Rey Juan Carlos. [Consultado en: enero/2008]. [En línea] abril/2004. http://www.escet.urjc.es/~asanz/documentos/TR_aplic3DCg_color.pdf.

Turkowski, Ken. 2007. Chapter 40. Incremental Computation of the Gaussian. [aut. libro] Hubert Nguyen. *GPU Gems 3.* USA : Addison Wesley Professional, 2007.

Walsh, Peter. 2003. Advanced Direct3D. *Advanced 3D Game Programming Using DirectX 9.0.* USA : Wordware Publishing, 2003.

Wikipedia. 2008. Black-and-white. [Consultado en: marzo/2008]. [En línea] marzo 2008. http://en.wikipedia.org/wiki/Black-and-white.

—. 2008. Monochrome. [Consultado en: marzo/2008]. [En línea] marzo 2008. http://en.wikipedia.org/wiki/Monochrome.

—. 2008. Motion Blur. [Consultado en: enero/2008] [En línea] enero 2008. http://es.wikipedia.org/wiki/Motion_Blur.

—. 2007. Shader. [Consultado en: diciembre/2007]. [En línea] diciembre de 2007. http://en.wikipedia.org/wiki/Shader.

Wolfgang, F. Engel. 2002. *Direct3D ShaderX: Vertex and Pixel Shader Tips and Tricks.* USA : Wordware Publishing, 2002.

BIBLIOGRAFÍA CONSULTADA

Nociones de fotogrametría digital. [Consultado en: junio/2008]. [En línea] http://webdelprofesor.ula.ve/ingenieria/iluis/publicaciones/Fotogrametr%EDa/FOTOGRAMETRIA%20DIGITALparte1.pdf.

Capítulo 2 Fundamento del color. *Biblioteca Virtual Miguel de Cervantes.* [Consultado en: junio/2008]. [En línea] http://descargas.cervantesvirtual.com/servlet/SirveObras/57915842105571617400080/008591_2.pdf.

LUNA, Frank D. 2003. *Introduction to 3D Game Programming with DirectX.* USA : WordWare Publishing, 2003.

NAVARRO, ALFONSO MIÑARRO. Luz y Color. [Consultado en: junio/2008]. [En línea] http://dis.um.es/grupos/sig/08BI/Color.pdf.

Porta, Paulo. 2005. Técnicas de filtrado. [Consultado en: junio/2008]. [En línea] septiembre/2005. http://www.quesabesde.com/camdig/articulos.asp?articulo=137.

Wikipedia. 2008. Canny edge detector. [Consultado en: marzo/2008]. [En línea] marzo/2008. http://en.wikipedia.org/wiki/Canny_edge_detector.

—. 2008. Edge detection. [En línea] mayo de 2008. http://en.wikipedia.org/wiki/Edge_detection.

GLOSARIO DE ABREVIATURAS

API: *Application Programming Interface*, (Interfaz de Programación de Aplicaciones).

CAD: *Computer Aided Design,* (Diseño Asistido por Computadoras): herramientas gráficas que permiten diseñar prototipos y evaluarlos antes de construirlos.

Cg: C for Graphic.

CPU: Unidad Central de Procesamiento.

CU: Caso de Uso.

GLSL: Open Level Shading Language.

GPU: *Graphic Processor Unit*, (Unidad de Procesamiento Gráfico).

HDR: *High Dynamic Range*, (Alto Rango Dinámico)

HLSL: High Level Shading Language.

HW: Hardware.

NPR: *Non-Photorealistic Rendering,* (Representación No Foto-realista).

OpenGL: *Open Graphics Library* (Biblioteca de gráficos abierta).

RGB: *Red Green Blue*. Los colores RGB consisten en tres canales, que representan los niveles de rojo, verde y azul, respectivamente, conocidos como colores aditivos primarios.

RGBA: *Red Green Blue Alpha*. El modelo de color RGBA consiste en los tradicionales canales rojo, verde, azul y la componente alfa, este último canal representa la transparencia.

RUP: *Rational Unified Process,* (Proceso Unificado de Desarrollo).

RV: Realidad Virtual.

SRV: Sistemas de Realidad Virtual.

UCI: Universidad de la Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado.

YIQ: *Luminance in-phase quadrature,* (Cuadratura sincronizada de luminancia)

VRAM: Video RAM.

1D: Primera Dimensión.

2D: Segunda Dimensión.

3D: Tercera Dimensión.

GLOSARIO DE TÉRMINOS

A:

Aliasing: las líneas, especialmente las que están casi horizontales o verticales, aparecen dentadas o irregulares debido a su representación por *pixels*. Este escalonamiento es llamado *aliasing*.

Antialiasing: técnicas que se utilizan para reducir el efecto de *aliasing*.

*****

C:

Coordenadas de texturas: Son los valores que definen cual punto de la textura se relaciona con el vértice del modelo tridimensional. Estos valores son usados para rasterizar la textura en el polígono y definir la relación que existe entre cada triángulo y el espacio correspondiente en la textura.

*****

F:

Fotograma: (ver *frame*).

Frame: cada uno de las imágenes que componen una animación.

Frame buffer: Memoria de pantalla.

*****

I:

Interpolación: algoritmo matemático que a partir de varios puntos en el espacio, describe una función que contiene a los puntos intermedios.

*****

H:

HDR (High Dynamic Range - Alto Rango Dinámico): Técnica que permite simular la luz que va más allá del 100% del brillo máximo de nuestros monitores. El resultado es mucho más real que el efecto Bloom, pues HDR va más allá, imitando el funcionamiento del iris en el ojo humano.

*****

L:

Luminancia:(Ver Luminosidad).

Luminosidad: es una variable que indica el grado de claridad u oscuridad que posee un color. Dentro del círculo cromático, el amarillo es el color de mayor luminancia y el violeta el de menor. Independientemente de los valores propios de los colores, éstos se pueden alterar mediante la adición de blanco que lleva el color a claves o valores de luminancia más altos, o de negro que los disminuye.

*****

M:

Mapa de Profundidad: Textura Procedural de Pre-cálculo en la que cada texel contiene información sobre la distancia a la que se encuentra el modelo de alta resolución en comparación con el de baja resolución en el polígono asignado a ese espacio de la textura. Son usadas para definir distorsiones en la superficie que dan sensación de grietas o elevaciones.

Mapa de Normales: Textura Procedural de Pre-cálculo en la que cada texel contiene información sobre la normal de la superficie polígono asignado a ese espacio de la textura.

Mapa de texturas: correspondencia entre vértices de una textura y los vértices del modelo.

*****

P:

Pipeline: Canal de proceso y comunicación en paralelo.

Píxel: abreviatura de “picture element”. Es la menor unidad de información de una imagen digital.

*****

R:

Rasterizar: convertir un modelo o imagen vectorial en una foto computarizada.

Realismo: se refiere a la calidad o detalles de los objetos en un entorno virtual, comparándolo con los objetos de la vida real y como estos interactúan con el usuario***.*** El realismo se logra empleando modelos complejos (muchos polígonos), texturas de alta definición y efectos visuales de alto impacto como el relieve en los objetos, sombras realistas, entre otros.

Rendering: crear en forma automática una imagen de acuerdo al modelo tridimensional que existe en el ordenador.

Renderizar: (ver *rendering*).

Render target: El buffer en memoria para el cual un procesador gráfico escribe los pixels. Un render target podría ser una superficie en memoria, parecida al back buffer, o el render target podría ser una textura.

Ruido: Perturbación al azar de una superficie basada en la interacción de colores o materiales.

*****

S:

Saturación: atributo visual que permite estimar la proporción de color cromático puro contenido en la sensación visual. Una saturación nula corresponde a una ausencia de color, a un color acromático. La escala de grises (blanco y negro incluido) posee una saturación nula.

Shader: Fragmento de código escrito en un lenguaje gráfico específico que se ejecuta en una plataforma programable a nivel de hardware, para procesar tanto píxeles (píxel shader) como vértices (vertex shader) y sustituyen etapas del pipeline gráfico.

*****

T:

Texel (Texture Element): Unidad mínima de una textura aplicada a una superficie.

Textura: Colección o arreglo n-dimensional de texel.

Texturas Procedurales de Pre-cálculo: Texturas creadas mediante algoritmos matemáticos que almacenan en los componentes RGBA valores pre-calculados de los modelos 3D. Ejemplo de estos valores son la iluminación, las normales, la radiosidad, entre otros.

*****

V:

Vector: cantidad que expresa magnitud y dirección.

Vector normal: vector cuyos puntos están en dirección perpendicular a una superficie.

Viewport**:** Porción de la ventana que establece una correspondencia entre las coordenadas espaciales transformadas sobre el plano de proyección y las coordenadas en pantalla.