Universidad de las Ciencias Informáticas
Facultad 5

Título: Biblioteca de Efectos de Post Procesamiento

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Yeisnier Domínguez Silva

Tutor(es): Frank Puig Placeres

Co-tutor: Jaime González Campistruz

La Habana, Junio 2009

DATOS DE CONTACTO

Nombre y Apellidos: Frank Puig Placeres

Edad: 26 años.

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 Universidad de las Ciencias Informáticas, con siete años de experiencia en el tema de Gráficos por Computadora. Coautor de los libros ShaderX5, Game Programming Gems 5, Game Programming Gems 6 y AI Wisdoms.

Nombre y Apellidos: Jaime González Campistruz.

Edad: 25 años.

Ciudadanía: cubano.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Informática.

Categoría Docente: Profesor Instructor.

E-mail: jgonzalezc@uci.cu

Graduado de la Universidad de las Ciencias Informáticas, con seis años de experiencia en el tema de Gráficos por Computadoras.

AGRADECIMIENTOS

A mi madre por su apoyo incondicional

A Ferna por haber cuidado como un hijo más

A Lisbey por ayudarme a conseguirlo y estar todo el tiempo a mi lado

A mis tutores, Frank y Jaime por supervisar cada uno de mis pasos

A Yosbel por demostrarme que es un verdadero hermano

A Brenda y Jandrich por su gran ayuda brindada

Y a aquellas personas que de una forma u otra hicieron posible este trabajo

A todos ustedes, Gracias.

DEDICATORIA

A mis abuelos y a mi madre…

RESUMEN

En los últimos años el empleo de efectos de post procesamiento ha hecho posible el realismo gráfico presentado por los Sistemas de Realidad Virtual (SRV). Actualmente existen cientos de estos efectos, lo que conlleva a que la mayoría de las aplicaciones que contengan efectos de post procesamiento sean de gran aceptación por parte de los usuarios debido al nivel de realismo que alcanzan las mismas. A causa de esto las aplicaciones de Realidad Virtual (RV) son empleadas en diferentes ramas de la sociedad como la Medicina, Educación, Entretenimiento y la Defensa.

El presente trabajo abarca el diseño de una biblioteca de efectos de post procesamiento para que las aplicaciones de Realidad Virtual, simuladores y Video Juegos creados en la UCI tengan una mayor aceptación en el mercado actual. Para alcanzar este objetivo se analizarán y describirán los diferentes efectos de post procesamiento más usados actualmente. Se diseñará y desarrollan algunos de estos efectos para incluir como ejemplos, pero a su vez se incluirá en la biblioteca la posibilidad de que se le adicionen nuevos efectos. De esta forma se obtendrá una biblioteca de efectos de post procesamiento de fácil acoplamiento en los diferentes proyectos de Realidad Virtual de la facultad 5 de la Universidad de la Ciencias Informáticas.

Palabras Claves: Efectos, Post Procesamiento, Realidad Virtual, Gráficos

Tabla de contenido

Introducción

Con el surgimiento y desarrollo de las técnicas de Realidad Virtual, los gráficos por computadoras han alcanzado un nivel tan avanzado que logran simular situaciones del mundo real, imprimiéndole a las mismas mayor realismo. Estas diferentes técnicas se pueden utilizar para elaborar prototipos de productos, en el campo de la medicina, en la visualización de datos (como la modelación de moléculas), en el entrenamiento de profesionales (simuladores de tiro y vuelo), en aplicaciones de entretenimiento (videojuegos), entre otras.

Es un hecho que la Realidad Virtual 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 estándares 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 manipulan las transformaciones en los vértices y se puede 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).

Los proyectos que se desarrollan en nuestra facultad necesitan que sus escenas tengan la suficiente calidad de visualización para presentar simulaciones competitivas en el mercado actual y de esta forma aumentar 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.

Debido a la situación antes expuesta se inicia la investigación descrita en este documento que plantea como **problema científico**: ¿Cómo implementar una biblioteca de post procesamiento que sea adaptable a los proyectos de la facultad?

En el presente trabajo se define como **idea a defender**, si se implementa correctamente la biblioteca de post procesamiento aumentará el realismo de las simulaciones en entornos de Realidad Virtual.

Para ello se propone la creación de un repositorio de efectos de post procesamiento que constituye el **objeto de estudio**. Definiendo como **campo de acción** el trabajo con las técnicas de efectos de post procesamiento. El **objetivo general** es diseñar una biblioteca capaz de aplicar los efectos de post procesamiento en los entornos virtuales.

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

- Analizar los lenguajes de programación, bibliotecas gráficas y técnicas relacionadas con la simulación de los efectos de post procesamiento.
- Analizar los efectos de post procesamiento más usados actualmente.
- Describir los efectos de post procesamiento más usados actualmente.
- Implementar una biblioteca que permita la aplicación de los efectos de post procesamiento en los entornos virtuales.
- Desarrollar un DEMO donde se muestre el uso de la biblioteca empleando los diferentes efectos de post procesamiento creados.

Capítulo 1: Fundamentación Teórica

La simulación y modelación de efectos visuales 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 una de las mayores preocupaciones y objetivos de los desarrolladores de simulaciones virtuales.

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 debido al gran volumen de cálculos que necesitaban.

La reciente generación del hardware gráfico nos ha transportado 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 autenticidad. Haciendo uso de la técnica per-pixel se puede aplicar una variedad de efectos basados en texturas, antes o después que esta haya sido aplicada. Entre este tipo de técnica se encuentra: 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. Además se abordarán los antecedentes del mismo así como su repercusión actual en la industria de la Realidad Virtual y la Simulación.

1.1. Efectos de Post Procesamiento.

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 “'detrás de' o 'después de'. Por ejemplo: *Posbélico, posponer, postónico.* A veces conserva la forma latina **post-.** *Postdorsal, postfijo*”.

A su vez, el diccionario define 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 el 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 vocablos y enmarcarlos en el campo de la Informática Gráfica se definen 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.*

Estos efectos son de gran utilidad para personalizar la apariencia de las simulaciones y permiten mejorar la calidad de las escenas. “En la biblioteca 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 usando un shader especializado para obtener una imagen diferente debido a una mejora o alteración”. (Microsoft Corporation, 2009)

Los antecedentes de la utilización de esta técnica 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 pueden 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 alcanzar mejor calidad 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 varios puntos. Los puntos principales que se usarán en este trabajo son dos. El primero trabaja sobre los vértices (geometría) y se le conoce como los Vertex Shader (Sombreado por vértices) programables. Siendo el segundo punto el que opera sobre los pixeles y se le conoce como Pixel Shader (Sombreado por pixel), permitiendo aplicar efectos personalizados a cada pixel.

A finales de 1999 y principios de los 2000 la compañía NVIDIA dominó el mercado de las tarjetas gráficas, en este período se caracterizó por 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 Tercera 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 Cuarta 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 10 series, esta última serie incluye el Directx 11.

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 mejor apariencia 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.

Ejemplos de estos efectos son el efecto Bloom que se realiza en aquellas situaciones que requieran 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. Este efecto se usó en el video juego Oblivion (2006) cuarta entrega de la saga “The Elder Scroll” como se muestra en la Figura 1.

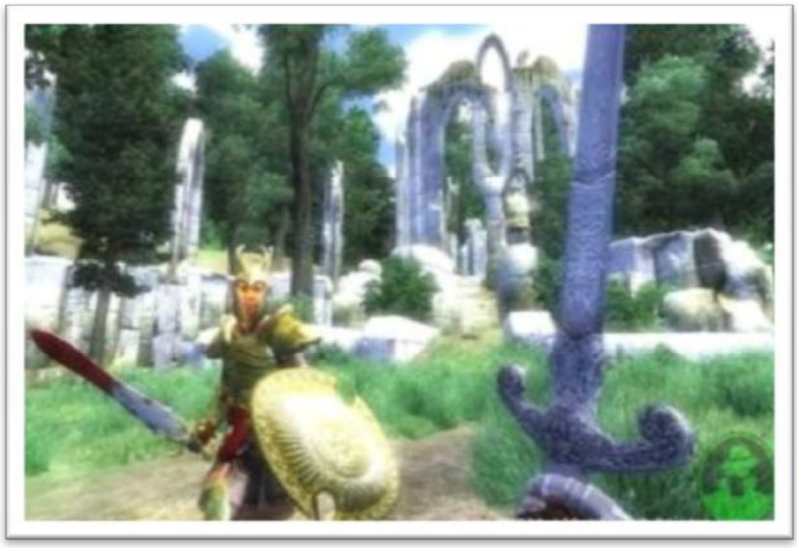

Figura 1: Efecto Bloom en el video-juego Oblivion: "The Elder Scroll".

Otro post-proceso ampliamente utilizado es el Depth of Field o profundidad de campo. 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 empleó por el video juego *Unreal Tournament 2007,* ver Figura 2.

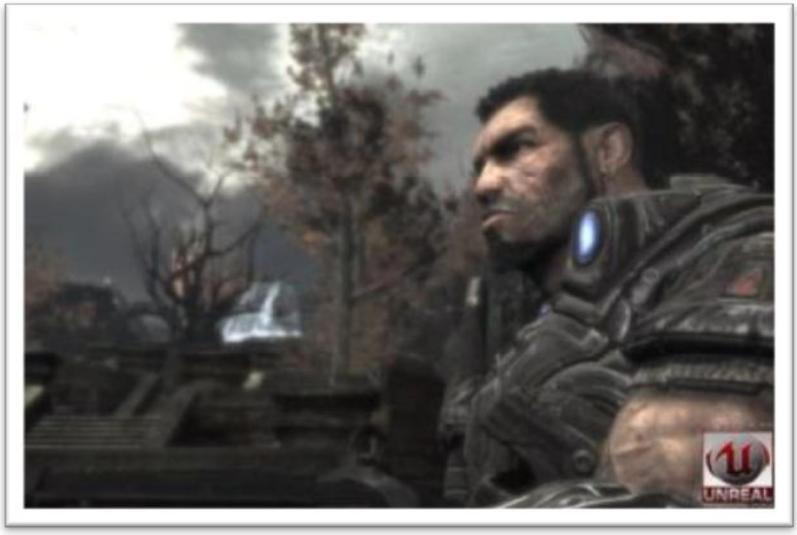

Figura 2: Efecto Depth of Field en el Unreal Tournament 2007.

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. Podemos apreciarlo en

el motor gráfico *Cry Engine* de los creadores del Crisis que fue lanzado en el año 2007, ver Figura 3.

Figura 3: Efecto Motion Blur en el video juego Crisis.

1.3. Shader

Gracias al avance en el hardware gráfico, se ha podido incrementar notablemente el rendimiento de las aplicaciones de visualización. Permitiendo representar escenas virtuales de mayor tamaño y complejidad. Sin embargo, aunque el aumento en el rendimiento ha posibilitado ejecutar las secuencias de visualización en un tiempo cada vez menor, se puede afirmar que al mismo tiempo estos sistemas no se han desarrollado en cuanto a 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 hardware 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).

En los últimos años se ha elaborado una alternativa al Fixed-Function para aumentar la flexibilidad gráfica del hardware de video. En varios momentos claves del pipeline gráfico se ha permitido 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 Shader** son los encargados de transformar todos los vértices de la escena. En ellos 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 todas

las operaciones a nivel de vértices como son los cálculos procedurales de coordenadas de texturas, iluminación per-vertex, entre otras.

La figura 4 muestra como trabaja el pipeline gráfico donde la interface de DirectX u OpenGL pasa los vertex al driver, la GPU recibe estos datos y luego el vertex shader transforma los vértices recibidos y realiza otras operaciones a nivel de vértices, 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 pixels que conforman el triángulo.

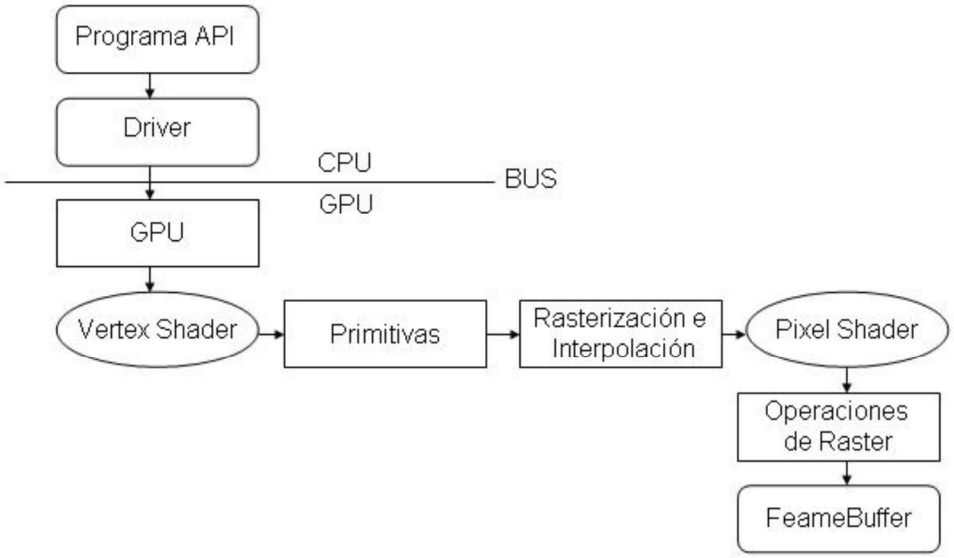

Figura 4: Pipeline Gráfico Conceptual usando Shaders.

Cada uno de estos píxeles se introduce en el **Píxel Shader** para que este 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, entre otras.

De esta forma, el uso de los shader permite introducir nuevos algoritmos en el hardware de video. 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 hardware.

1.4. Unidad de Procesamiento Gráfico.

GPU es un acrónimo utilizado para abreviar Graphics Processing Unit, que significa "Unidad de Procesado de Gráficos". Este se inventó como analogía a la sigla "CPU".

Básicamente, la GPU es una CPU dedicada exclusivamente al procesamiento de gráficos, para aligerar la carga de trabajo del procesador del ordenador en aplicaciones de Realidad Virtual. 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.

El hardware gráfico actual es muy complejo; normalmente se utiliza la GPU para ofrecer múltiples canales de ejecución, con algo de Memoria de Acceso Aleatorio (VRAM) para los buffer y el rendereado de texturas y alguna instrucción en específico.

A diferencia del ensamblador estándar de los CPU, la GPU usa para casi todo operaciones ternarias, esta optimizado para el trabajo matemático con vectores y su diseño está pensado para una alta capacidad de procesamiento en paralelo.

Todo dentro del CPU funciona en 10-24 pipelines que corren al mismo tiempo. No hace falta tener 24 GPUs paralelos, sino que uno sólo puede ejecutar 24 operaciones en paralelo (o más según la generación).

Si se usan varios GPU al mismo tiempo, las operaciones se multiplican.

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 donde en lugar de pintar la imagen 2D en la pantalla, se pinta sobre una textura.

Esta técnica es fundamental para la implementación de algoritmos que utilizan las potencialidades 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 una 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 costo 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, entre otras. 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 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:

- Crear una textura como el render target.
- Activar el render target.
- Renderizar la escena.

1.6. Metodologías de desarrollo.

Todo desarrollo de software es riesgoso y difícil de controlar, y si no se emplea una metodología que guíe este proceso, los resultados obtenidos son clientes insatisfechos y desarrolladores aún más descontentos. Actualmente a nivel mundial, en dependencia del tiempo de vida y la complejidad del proyecto que se vaya a desarrollar se proponen diferentes metodologías. Una metodología es el conjunto de técnicas y procedimientos que permiten conocer los elementos necesarios para definir un proyecto de software, es la base para la edificación de un producto de este tipo. Esencialmente, sirve para aumentar la "calidad" del software y controlar de manera transparente todo el proceso de desarrollo. Si se quiere que un proyecto sea escalable y flexible a los cambios es lógico pensar que para lograr ese propósito se necesite tomar en cuenta una de las muchas metodologías para el proceso de desarrollo de

software que existen. Las metodologías dadas sus características, se enmarcan en dos grandes grupos, los llamados "métodos pesados" y los "métodos ágiles". La diferencia más notable entre estos dos grupos es que mientras los métodos pesados intentan obtener los resultados apoyándose principalmente en la documentación ordenada, los métodos ágiles tienen como base de sus resultados la comunicación e interacción directa con todos los usuarios involucrados en el proceso.

La presente investigación se referirá a dos metodologías en específico. La primera de tipo pesada y la segunda se clasifica entre las de tipo ágil. La metodología pesada que se estudió fue Rational Uni ed Process (RUP) y la ágil fue eXtreme Programing (XP).

1.6.1. RUP

Es una de las metodologías más usadas y mayormente probadas a nivel internacional para el desarrollo de software.

Características de RUP

Dirigido por Casos de Uso: Tiene a los Casos de uso como el hilo conductor que orienta las actividades de desarrollo. Se centra en la funcionalidad que el sistema debe poseer para satisfacer las necesidades de un usuario (persona, sistema externo, dispositivo) que interactúa con él.

Centrado en la arquitectura: Propone arquitectura de forma similar a la de un edificio. Es necesario tener varios planos con diferentes aspectos, para tener una imagen completa del edificio antes que comience su construcción, aquí entra a jugar el término Arquitectura de Software, que abarca las diferentes vistas que se mencionaron anteriormente.

Iterativo e incremental: Propone la descomposición de proyectos grandes en proyectos más pequeños o subproyectos, cada uno de estos subproyectos es una iteración, y cada iteración debe estar controlada y tratar un determinado grupo de casos de uso.

RUP provee un acercamiento a disciplinas para asignar tareas y responsabilidades en un desarrollo organizado. Su objetivo es asegurar la producción de software de alta calidad que conozca la necesidad de sus clientes dentro de un predecible espacio de tiempo y presupuesto. La figura 5 ilustra la arquitectura de RUP, la cual tiene dos dimensiones:

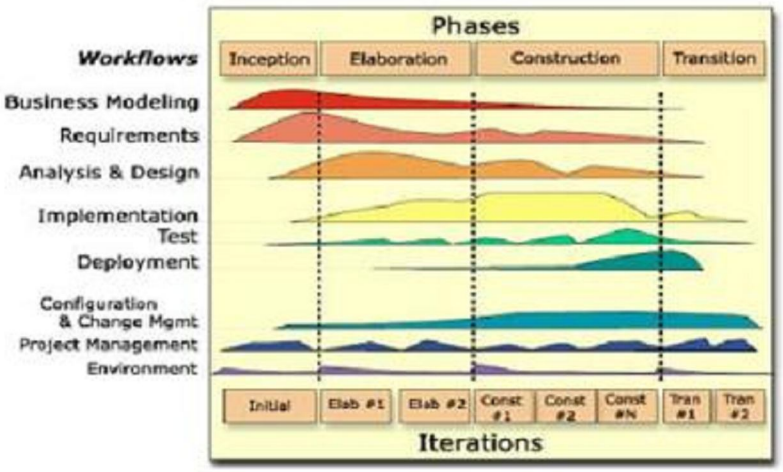

Figura 5: Arquitectura de RUP.

Fases de desarrollo

1. **Inicio**: Determinar la visión del proyecto.
2. **Elaboración**: Determinar la arquitectura óptima.
3. **Construcción**: Llegar a obtener la capacidad operacional inicial.
4. **Transición**: Llegar a obtener una primera versión del proyecto (release).

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.

Está metodología cuenta con varias disciplinas dentro de las que se encuentran:
Disciplina de Desarrollo

Ingeniería de Negocios: Entendiendo las necesidades del negocio.

Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.

Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.

Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.

Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo los solicitado está presente.

Disciplina de Admitidas

Configuración y administración del cambio: Guardando todas las versiones del proyecto.

Administrando el proyecto: Administrando horarios y recursos.

Ambiente: Administrando el ambiente de desarrollo.

Distribución: Hacer todo lo necesario para la salida del proyecto.

Dentro de esta metodología se identifican como elementos fundamentales:

Actividades: Procesos que se llegan a determinar en cada iteración.

Trabajadores: Las personas o gentes involucrados en cada proceso.

Artefactos: Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

1.6.2. eXtreme Programming (XP)

XP es una metodología ágil basada en cuatro principios: simplicidad, comunicación, retroalimentación y valor. Además está orientada por pruebas y refactorización, se diseñan e implementan las pruebas antes de programar la funcionalidad y el programador crea sus propios tests (pruebas) de unidad. (Programming)

Esta metodología se apoya en el trabajo orientado directamente al objetivo, basándose en las relaciones interpersonales, en la velocidad de reacción para la implementación y los cambios que puedan surgir durante el desarrollo del proceso. Por ello se requiere mantener dentro del equipo a un representante "competente" del cliente, quién responderá a todas las preguntas y dudas que surjan del equipo de desarrollo durante el proceso, de forma que no se retrase la toma de decisiones (DESAROLLO). XP se basa en Use Stories (historias de usuario), las cuales son escritas por el cliente o su representante dentro del equipo y describen los escenarios claves del funcionamiento del software. A partir de estas historias se generan los releases (entregas) entre el equipo y el cliente. Los releases son los que permiten definir las iteraciones necesarias para cumplir con los objetivos, de manera que cada resultado de la iteración sea un programa aprobado por el cliente de quien depende la definición de las siguientes

iteraciones. XP genera solo cuatro artefactos, de los cuales solo tres son documentados; el último de ellos, las tarjetas CRC, son pequeñas tarjetas de cartón que se mantienen en poder del equipo de desarrollo mientras dura el proceso de construcción. (Programming) Una característica importante de XP es la producción siempre en parejas del código, las cuales van cambiando constantemente para lograr así que todo el equipo sepa y pueda modificar el código generado según las necesidades. Esto logra que los integrantes del equipo de desarrollo aprendan entre sí y compartan dicho código totalmente. En la actualidad XP se proyecta a ser un modelo de desarrollo común, sencillo y adaptable a las características cambiantes y exigentes de empresas y clientes. (DESAROLLO)

Para la realización del presente trabajo se decidió utilizar RUP ya que es una de las metodologías más utilizadas y mayormente probadas a nivel internacional para el desarrollo del software. Para mantener un proceso homogéneo entre todos los grupos del proyecto y la experiencia del equipo de desarrollo.

1.7. Herramientas CASE

El solo hecho de imaginar la realización de los diagramas de un ciclo de vida de RUP de forma manual demuestra la necesidad e importancia de estas herramientas. Se hizo un estudio sobre las principales herramientas CASE existentes y se seleccionó **Enterprise Architect** teniendo en cuenta sus características y las necesidades presentes. A continuación se exponen algunas de sus características:

- Soporta los 13 diagramas de UML 2.1.
- Soporte para perfil de estilo UML 2.0.
- Los diagramas de comportamiento incluyen: Casos de Uso, Actividades, Estado, Interacción, Secuencia y Comunicación.
- Los diagramas de Estructura Incluyen: Paquetes, Clases, Objetos, Composición, Componentes y Despliegue.
- Ingeniería de Código Directa e Inversa.
- Interfaz de usuario intuitiva.
- Soporte para Transformaciones MDA.
- Documentación flexible y comprensible.

- Plug-ins para vincular EA a Visual Studio.NET o Eclipse.
- Soporta Control de Versiones.
- Soporte de esquema XML.

Estas características son suficientes para resolver las necesidades de diseño del presente trabajo.

Capítulo 2: Técnicas Propuestas

En este capítulo se explicará el funcionamiento e implementación de algunos de los efectos de post procesamiento más usados. Además se introducirá el funcionamiento de OGRE que será usado como base para la creación de la biblioteca descrita en este documento.

2.1. Convolution 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 convolution espacial, el cual se fundamenta en el uso de máscaras de convolution; estos filtros son denominados filtros kernels o de convolution y son utilizados para modificar los valores de brillantez de la imagen original en función de un peso promedio creado mediante una combinación lineal.

Convolution es una técnica de post procesamiento de imágenes muy usada que cambia la intensidad del pixel en dependencia de la intensidad de sus pixeles vecinos. Un uso muy común de los Convolution son los Blur, Sharpen, Emboss y Edge Detection. Dichos efectos son muy usados en las populares aplicaciones de retoque de imagen como Photoshop, Photo Booth, iPhoto y Aperture.

En la figura 6 se muestra el antes y después de aplicado un convolution obteniendo el efecto Emboss. Para alcanzar este efecto se hace uso de una matriz matemática llamada Kernel.

Figura 6: Efecto Emboss empleado Convolution.

La figura 7 representa un Kernel de 3 x 3. Las dimensiones de dicho kernel no necesariamente deben ser iguales pero ambas si deben ser números impares. Los valores en la matriz del kernel indican cuánto va a influir dicho valor en el resultado a la hora de aplicar el convolution, el Kernel representado en la figura 7 muestra los valores del Kernel para un efecto Emboss.

El Kernel (específicamente, los valores de su matriz) son los que determinan como se transforma el pixel desde la imagen original en la imagen procesada. Puede parecer no intuitiva la forma en que un kernel de 9 números pueda producir un efecto Emboss como el ejemplo anterior.

-2	-2	0
-2	6	0
0	0	0

Figura 7: 3 x 3 kernel

Los Convolutions son operaciones por pixeles, la misma operación aritmética es repetida por cada pixel de la imagen. Por lo que imágenes grandes requieren mayor cantidad de operaciones que una imagen más pequeña. Un kernel es una matriz bidimensional de números que es sobrepuesta sobre cada pixel de una imagen secuencialmente, realizando las operaciones del convolution para obtener el resultado deseado.

Una imagen es representada por una matriz de números en la cual cada posición representa la intensidad de cada pixel (ver. Figura 8).

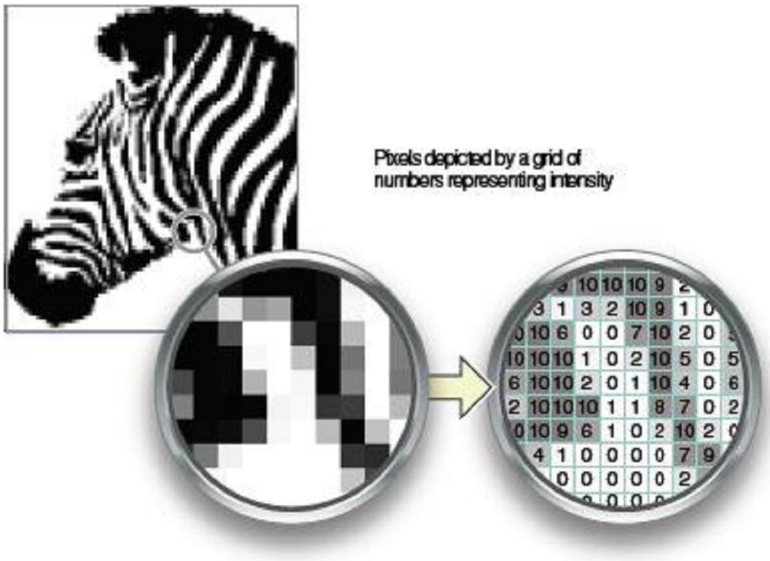

Figura 8: Una imagen como una matriz de números representando la intensidad.

Los números en el kernel representan la cantidad que se va multiplica por el pixel que quede por debajo de este. El número debajo representa la intensidad de los píxeles sobre los que el elemento se sitúa en el kernel. Durante el convolution, el centro del kernel pasa por cada píxel en la imagen. En el proceso se multiplica cada número en el kernel por el valor del píxel de intensidad directamente debajo de este.

En el paso final del proceso todas las cantidades de los productos juntos, los divide por la cantidad de números en el kernel, y este valor se convierte en el nuevo valor de intensidad del píxel que estaba directamente bajo el centro del kernel. La figura 9 muestra cómo funciona un kernel en un píxel.

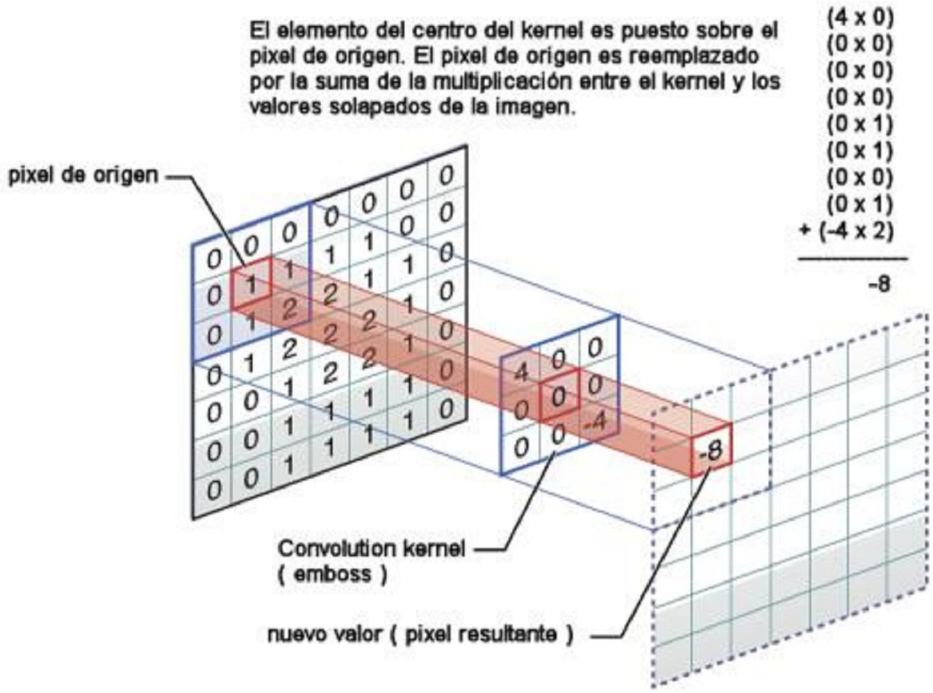

Figura 9: Kernel convolution

A pesar de que el kernel superpone varios píxeles (o en algunos casos, no a todos los píxeles), el único píxel que en última instancia recibe los cambios es el píxel de origen que está debajo del elemento central del kernel. La suma de todas las multiplicaciones entre el kernel y la imagen se llama la suma ponderada.

Cuando se hace la sustitución de un pixel con la suma ponderada de sus píxeles vecinos con frecuencia puede resultar una mayor intensidad en el píxel (y una imagen brillante en general), dividiendo la suma ponderada por un factor puede reducir la intensidad del efecto y así garantizar que el brillo inicial de la imagen se mantenga.

Este procedimiento se denomina normalización, en el cual se divide la suma ponderada entre el factor y este resultado se convierte en el nuevo valor del píxel. Este procedimiento se repite para cada píxel de la imagen original.

1	2	1
2	4	2
1	2	1

-1	-1	-1
0	0	0
1	1	1

-2	-2	0
-2	6	0
0	0	0

Figura 10: Kernel de efectos Glaussian Blur, Edge Detection y Emboss.

2.2. 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.

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 se analizará el efecto monocromático en su modalidad de blanco y negro.

Este efecto puede realizarse a través de un cálculo simple de luminosidad o brillo (**Luminance**) la cual es leída desde los pixeles la imagen original y les devuelve un nuevo color. Esta operación es realizada mediante un producto vectorial para calcular el valor de la luminosidad de la escena:

$$= 1 * \quad + 2 * \quad + 3 *$$

Ecuación 1: 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.

LuminanceConv = { 0.2125f, 0.7154f, 0.0721f };

Figura 11: Imagen de “Lenna” (a) imagen original (b) imagen monocromática.

2.3. Efecto Bloom.

El efecto *Bloom* 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 sobreiluminado 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.

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, se profundizará en los métodos más utilizados actualmente.

“Existen diferentes formas de designar las áreas 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 áreas brillosas de la escena regular, o el método alfa channel usado para renderizar por separado la geometría del efecto”. (Kylmamaa, 2006)

2.3.1. La técnica filtro Bright-pass

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 áreas que serán brillosas en la escena final y excluye los demás datos. “El Tone mapping simula el control automático 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, 2009). El filtro del Bloom generalmente consta de los siguientes pasos:

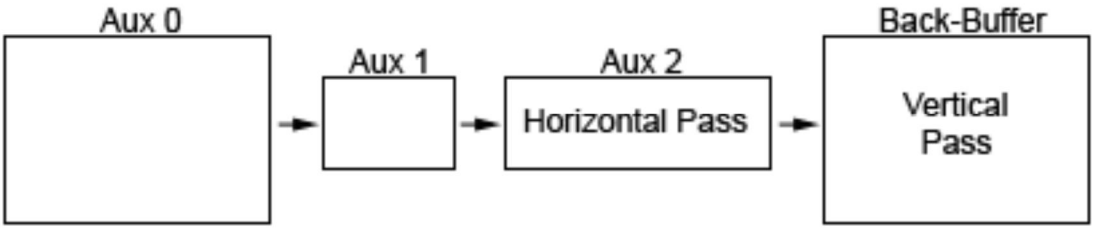

Figura 12: Etapas y pasos del filtro Bright-Pass.

El proceso de filtrado mostrado en la Figura 12 (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 el 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 le aplica el efecto de Gaussian blur vertical.

Esta secuencia de pasos es ilustrada en la Figura 13 (Microsoft Corporation, 2009).

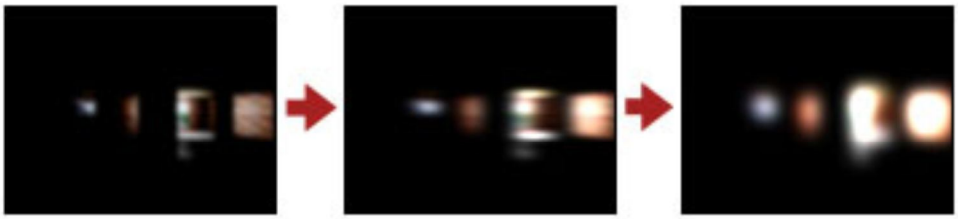

Figura 13: Secuencia de imágenes que ilustra que ilustra la técnica filtro Bright-Pass.

Después de aplicado el filtro Bloom, este efecto muestra un número 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 14 (Kylmamaa, 2006).

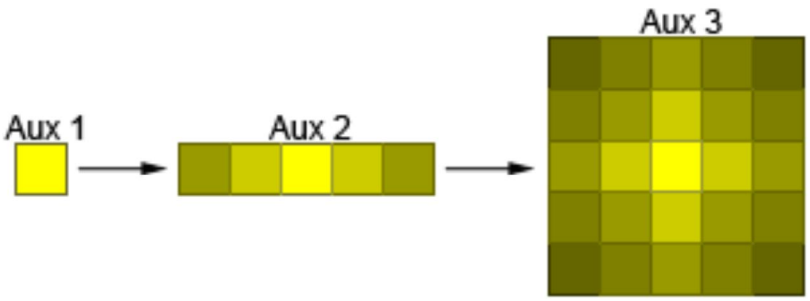

Figura 14: Proceso que permite poner borroso los texel de una textura a lo largo de la dirección horizontal (Aux2) y vertical (Aux3).

"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, 2009). 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 bilineal para el origen de la textura y el muestreo de los límites 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 15: Muestras de texel sacados por el filtro bilineal.

En el método de 3 Samples, ver Figura 15 (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.

2.4. Motor Gráfico

Para el desarrollo de las técnicas antes expuestas se hace uso del motor gráfico OGRE. Puesto que actualmente la facultad 5 de la Universidad de las Ciencias Informáticas está inmersa en extender el uso del mismo, para la creación de sus aplicaciones de Realidad Virtual.

2.5. Funcionamiento básico

OGRE (Object Oriented Graphics Engine) es un motor gráfico de fuente abierta orientado a objetos escrito en C++. Tiene licencia GNU Licencia Pública General Menor (LGPL) que permite su uso libre con algunas pequeñas restricciones. Fue diseñado con el objetivo de facilitar el trabajo de los desarrolladores que producen aplicaciones basadas en hardware de aceleración gráfica 3D. Utiliza librerías gráficas OpenGL y Direct3D de Microsoft, lo que permite la portabilidad del código a diferentes plataformas como Linux, Windows, Mac OS.

Con OGRE podemos hacer juegos o cualquier tipo de aplicación que requiera gráficos tridimensionales. También para hacer aplicaciones de simulación, corporativas o de investigación, sin tener que envidiar a los realizados por la mayoría de los motores del mercado.

Características generales

- Diseño orientado a objetos.
- Interfaz simple y fácil de usar, diseñada para que requiera poco esfuerzo el renderizado de escenas en tres dimensiones.
- Arquitectura basada en plugins muy flexible que permite extender las funcionalidades del motor.
- Sistema de Carga/Respaldo. Soporte de zip/pk3 para archivar.
- Diseño limpio y bien documentado de las clases del motor.
- Independiente de la API gráfica, se puede utilizar OpenGL o DirectX.

Por las características que presenta dicho motor gráfico y las ventajas que nos ofrece se hace necesario el uso del mismo, ya que nos centramos en la creación de los efectos sin concentrar esfuerzos en otras áreas como funcionalidad, rendimiento, apariencia entre otras sin tener que implementar a un nivel inferior las funcionalidades que el mismo nos brinda.

2.6. Post Procesamiento en OGRE

Para la inclusión de un efecto de post procesamiento haciendo uso del OGRE, es necesario el uso del compositor, ya que los mismos hacen la parte engorrosa del manejado del pintado a texturas haciendo el intercambio entre los diferentes búferes para el empleo de tales efectos.

Dejando la posibilidad al desarrollar de poder realizar la aplicación de los efectos sin el uso del compositor pero se requeriría un conocimiento más avanzado de Ogre.

La versión de OGRE usada para la creación de los efectos es la 1.6.0RC1 "Shoggoth" y en la misma se muestra el empleo de los compositores para aplicar efectos de post procesamiento, pero no existe una documentación de cómo hacer uso de los mismos así como integrarlos en sus aplicaciones y se muestran diferentes efectos de post procesamiento incluidos en un mismo ejemplo.

Además que para la creación de tales efectos se emplean diferentes materiales sin hacer un uso compartido de los mismos. Existiendo la misma situación en los shaders. Lo que demuestra la necesidad de crear un repositorio de efectos para que se pueda hacer un uso racional y compartido de los compositores, materiales y shaders. Lo que facilitaría el trabajo en colectivo ya que en la UCI se desarrolla en proyecto y se hace necesaria esta organización, tarea a la que se propone dar solución dicha investigación.

Capítulo 3: Solución Propuesta

La biblioteca representa una colección de efectos de post procesamiento que facilita a los desarrolladores la inclusión de nuevos efectos, con el objetivo de aumentar el realismo de sus simulaciones. Los efectos de post procesamiento no dependen de la complejidad de la escena, debido a que pueden ser fácilmente incluidos después de pintar la escena. El tiempo requerido para producirlos solamente depende de la resolución de la misma, sin tener en cuenta la complejidad de iluminación u otros detalles.

3.1. Modelo de Dominio

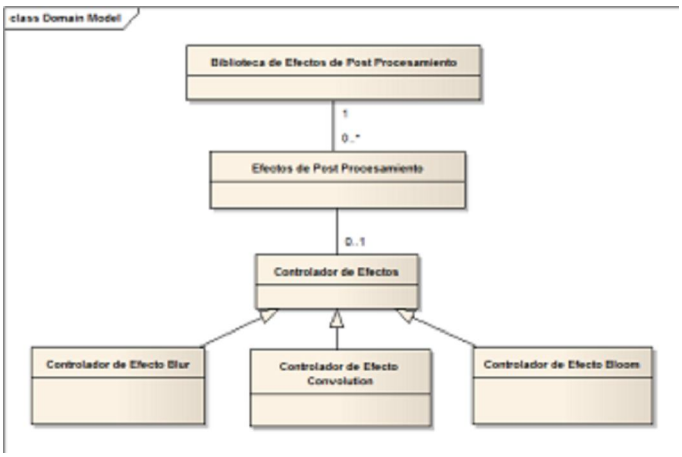

Figura 16: Diagrama del Modelo del Dominio.

3.2. Glosario de Términos del Dominio

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

Controlador de Efectos: Los efectos que necesitan cambiar los valores de sus variables en tiempo real necesitan un controlador.

Controlador de Efecto Blur: Encargado de actualizar las variables del efecto Blur. **Controlador de Efecto Convolution**: Controlador para controlar los parámetros del efecto Convolution.

Controlador de Efecto Bloom: Encargado de actualizar las variables del efecto Bloom.

3.3. Captura de requisitos

A continuación se expondrán los requisitos funcionales y no funcionales del sistema.

3.3.1. Requisitos funcionales

R1 - Crear efecto de Post Procesamiento.

R2 - Buscar recurso del efecto de Post Procesamiento.

R3 - Crear materiales asociados al efecto.

R4 - Cargar los shaders.

R5 - Crear las texturas a usar por el efecto.

R6- Buscar efecto previamente creado.

R7- Activar efecto ya creado.

R8- Desactivar efecto ya creado.

R9- Eliminar efecto de Post Procesamiento.

R10 - Asignar controlador.

3.3.2. Requisitos no funcionales

Usabilidad: Los futuros usuarios del sistema serán programadores con conocimientos básicos con respecto al tema de los efectos de post procesamiento.

Rendimiento: Debe ser una aplicación en tiempo real que tenga alta velocidad de procesamiento de los cálculos, tiempo de respuesta y recuperación del sistema.

Soporte: Debe ser compatible con la plataforma Windows, pero con pequeñas modificaciones puede migrar hacia Linux.

Hardware: Compatibilidad con tarjetas gráficas de la familia NVIDIA, y tarjeta de video de más de 128 megabyte de RAM y soporte gráfico para Modelo de Shader 2.0 o superior.

Diseño e implementación: Se regirá por la filosofía de Programación Orientada a Objetos, haciendo uso del motor gráfico OGRE.

Extensibilidad: El sistema es muy extensible dándoles la posibilidad a los programadores que hagan uso de la biblioteca de incluir nuevos efectos de post procesamiento.

3.4. Modelo de casos de uso.

En este epígrafe se darán a conocer los actores del sistema a desarrollar y los principales CU que se obtuvieron a partir de los requisitos funcionales mostrados en el epígrafe anterior.

3.4.1. Definición de los actores del sistema.

Tabla 1: Justificación de los actores.

Actores	Justificación
Aplicación	Es la que hará uso de la biblioteca, activando y desactivando los diferentes efectos de post procesamiento de la biblioteca propuesta.

3.4.2. Casos de usos del sistema

- Crear Post Procesamiento.
- Eliminar Post Procesamiento.
- Activar Post Procesamiento.
- Desactivar Post Procesamiento.
- Asignar controlador a Post Procesamiento.

3.4.3. Diagrama de casos de uso.

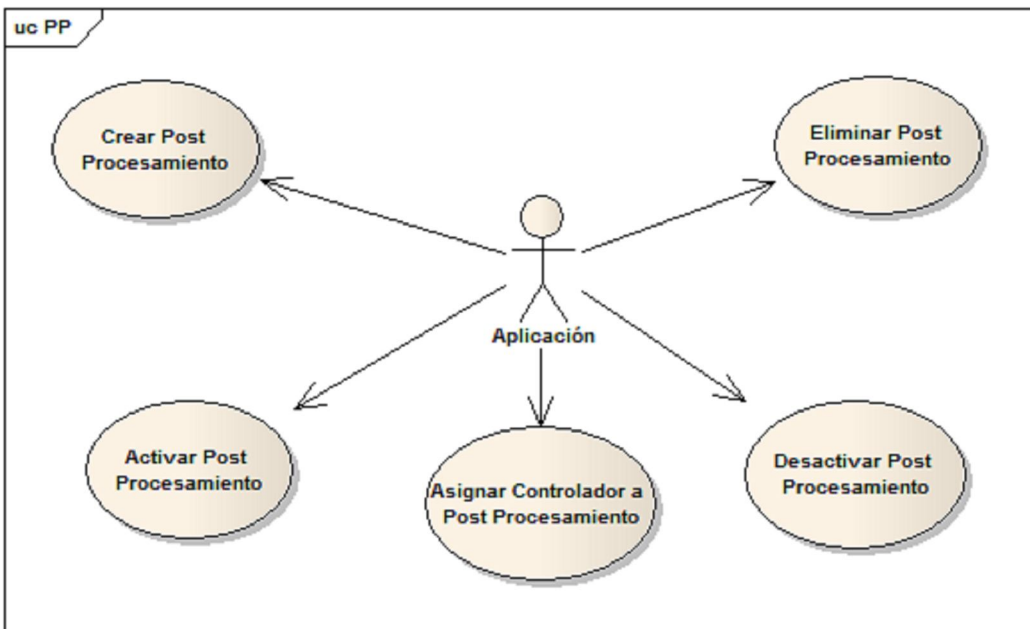

Figura 17: Diagrama de Casos de Uso.

3.4.4. Descripción de los casos de uso en formato expandido

Tabla 2: CU “Crear Post Procesamiento”.

Nombre del Caso de Uso	Crear Post Procesamiento.
Actores	Aplicación
Propósito	Crear los diferentes efectos de post procesamiento que contiene la biblioteca.
Resumen: Se inicia cuando el actor solicita crear un efecto especificándole el nombre. En caso de no encontrarse el recurso asociado al efecto según el nombre no se crea el efecto y se le informa al actor. El CU finaliza cuando el efecto elegido, es creado con todas sus texturas y variables.	
Referencias	**R1, R2, R3, R4, R5**
Precondición	**-**
Postcondición	El efecto de post procesamiento ha sido creado.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1 - Solicita crear efecto pasando el nombre del efecto.	1.1- Buscar el recurso asociado al efecto según el nombre.
	1.2 - Cargar materiales y shaders del efecto.
	1.3 - Crear las texturas necesarias para aplicar el efecto.
	1.4 – Se crea el efecto de post procesamiento y se inserta en la lista de los efectos creados.
Curso Alterno:	
Línea 2	
	2.1 - Si algún material o shader no se encuentra, no se crea el efecto y se le informa al programador.
Prioridad: Crítico.	

Tabla 3: CU "Eliminar Post Procesamiento".

Nombre del Caso de Uso	Eliminar Post Procesamiento.
Actores	Aplicación
Propósito	Eliminar el efecto que se encuentra creado pasándole el nombre.
Resumen: Se inicia cuando el actor solicita eliminar un efecto especificándole el nombre. Se busca el efecto en la lista de post procesamiento. El CU finaliza cuando el nombre del efecto es encontrado y es eliminado o sino es encontrado el efecto.	
Referencias	**R6, R9**
Precondición	Efecto creado.
Postcondición	Efecto eliminado.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1 - Solicita eliminar efecto pasando el nombre del efecto.	1.1- Buscar el efecto según el nombre en la lista de efectos creados.
	1.2 – Se eliminado el efecto de post procesamiento.
Curso Alterno:	
Línea 2	
	2.1 - Si el efecto no se encuentra no ocurre nada.
Prioridad: Crítico.	

Tabla 4: CU "Activar Post Procesamiento".

Nombre del Caso de Uso	Activar Post Procesamiento.
Actores	Aplicación
Propósito	Activar el efecto que se encuentra creado pasándole el nombre.
Resumen: Se inicia cuando el actor solicita Activar un efecto especificándole el nombre. Se busca el efecto en la lista de post procesamiento. El CU finaliza cuando el nombre del efecto es encontrado y es activado.	
Referencias	**R6, R7**
Precondición	Efecto creado.

Postcondición	Efecto activado.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1 - Solicita activar efecto pasando el nombre del efecto.	1.1- Buscar el efecto según el nombre en la lista de efectos creados.
	1.2 – Se activa el efecto de post procesamiento.
Curso Alterno:	
Línea 2	
	2.1 - Si el efecto no se encuentra no ocurre nada.
Prioridad: Secundario.	

Tabla 5: CU "Desactivar Post Procesamiento".

Nombre del Caso de Uso	Desactivar Post Procesamiento.
Actores	Aplicación
Propósito	Desactivar el efecto que se encuentra creado pasándole el nombre.
Resumen: Se inicia cuando el actor solicita Desactivar un efecto especificándole el nombre. Se busca el efecto en la lista de post procesamiento. El CU finaliza cuando el nombre del efecto es encontrado y es desactivado.	
Referencias	**R6, R8**
Precondición	Efecto activado.
Postcondición	Efecto desactivado.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1 - Solicita activar efecto pasando el nombre del efecto.	1.1- Buscar el efecto según el nombre en la lista de efectos creados.
	1.2 – Se desactiva el efecto de post procesamiento.
Curso Alterno:	
Línea 2	

	2.1 - Si el efecto no se encuentra no ocurre nada.
Prioridad: Secundario.	

Tabla 6: CU "Asignar controlador a Post Procesamiento".

Nombre del Caso de Uso	Asignar controlador a Post Procesamiento.
Actores	Aplicación
Propósito	Asignar controlador a un efecto de post procesamiento.
Resumen: Se inicia cuando el actor solicita asignar un controlador a un efecto especificándole el nombre y el controlador. Se busca el efecto en la lista de post procesamiento. El CU finaliza cuando el nombre del efecto es encontrado y se le asigna el controlador.	
Referencias	**R6, R10**
Precondición	Efecto creado.
Postcondición	Controlador asignado al efecto especificado.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1 - Solicita asignar controlador a un efecto pasando el nombre del efecto y el controlador.	1.1- Buscar el efecto según el nombre en la lista de efectos previamente creados.
	1.2 – Se le asigna el controlador pasado en caso de encontrarse el efecto.
Curso Alterno:	
Línea 2	
	2.1 - Si el efecto no se encuentra no ocurre nada.
Prioridad: Secundario.	

Capítulo 4: Diseño del Sistema

A continuación se presentarán los diagramas de clases de diseño los cuales tendrán un paquete “Post Procesamiento” con las relaciones existente que pueden ser de agregación, generalización/especialización y composición. Además se presentan los diagramas de secuencia de los casos de usos que intervendrán en el primer ciclo de desarrollo de software.

4.1. Modelo de Diseño

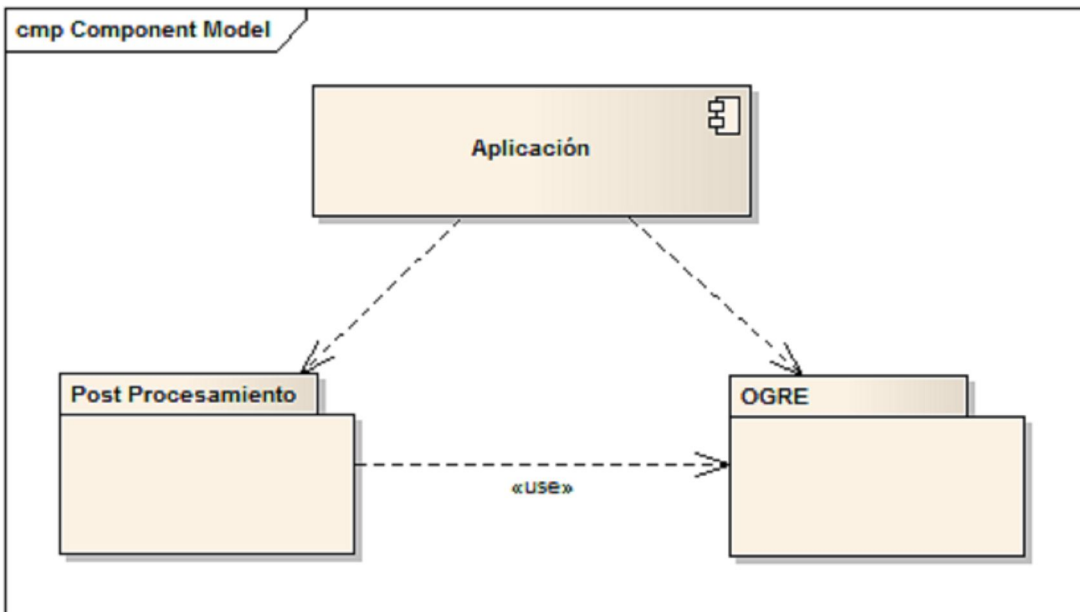

Figura 18: Diagrama de Modelo de Diseño del Sistema.

4.2. Paquete “Post Procesamiento”.

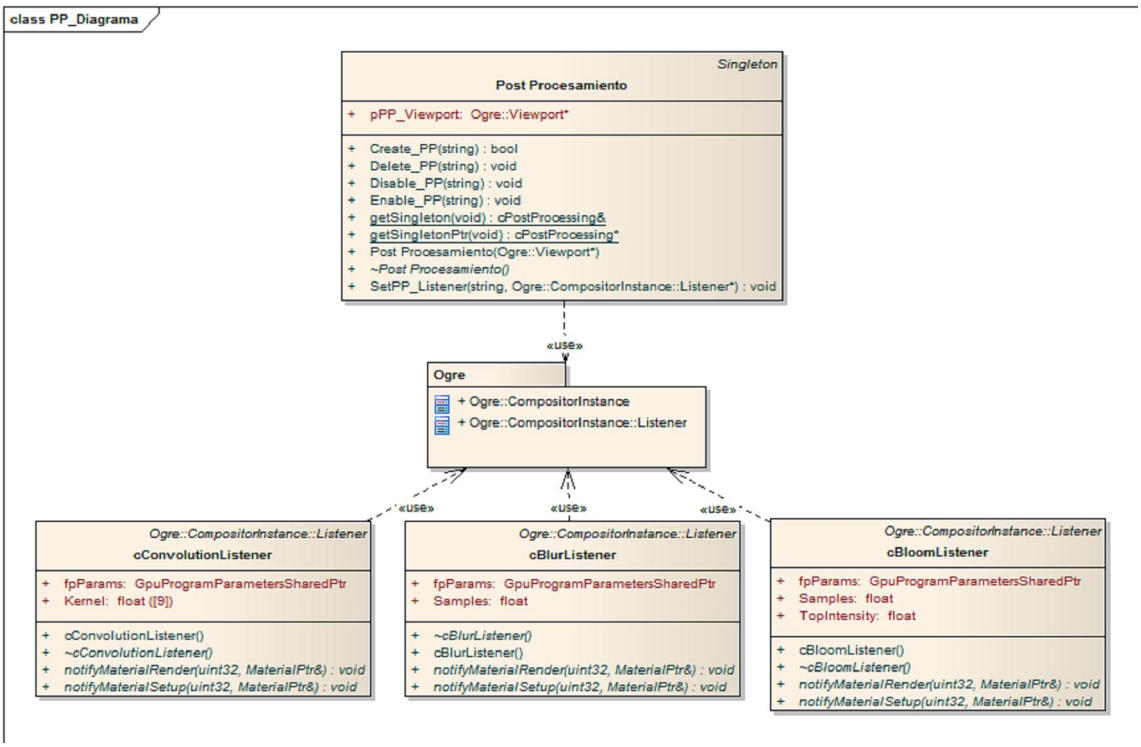

Figura 19: Diagrama de clases del paquete “Post Procesamiento”.

4.3. Diagramas de Interacción de Diseño

4.3.1. Realización del caso de uso “Crear Post Procesamiento”

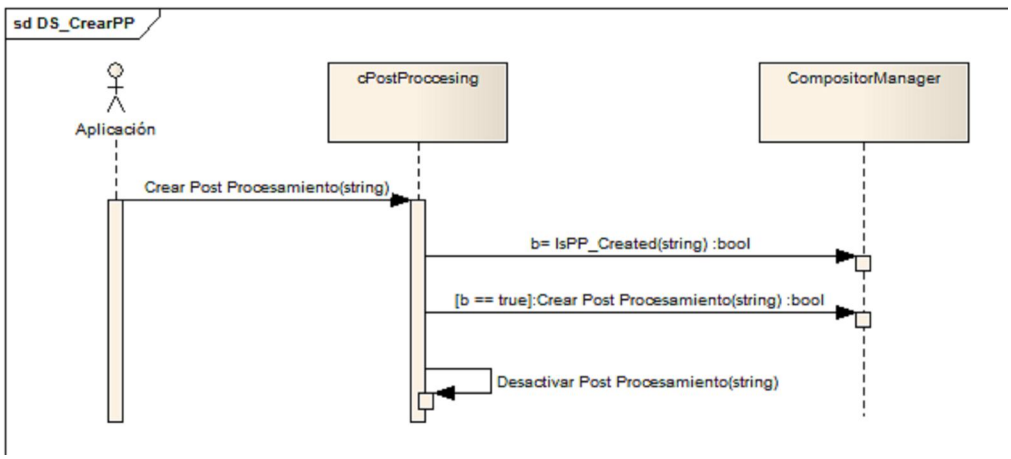

Figura 20: Diagrama de Secuencia "Crear Post Procesamiento".

4.3.2. Realización del caso de uso “Eliminar Post Procesamiento”

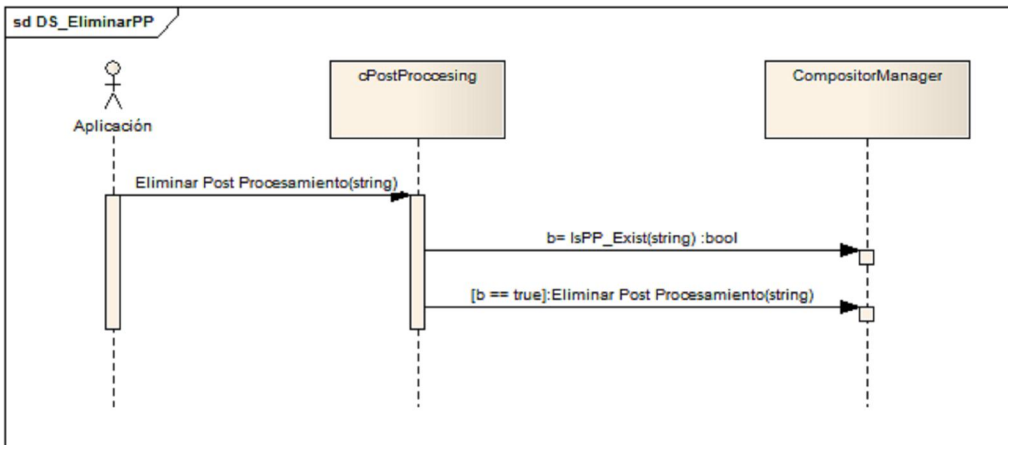

Figura 21: Diagrama de Secuencia "Eliminar Post Procesamiento".

4.3.3. Realización del caso de uso “Activar Post Procesamiento”

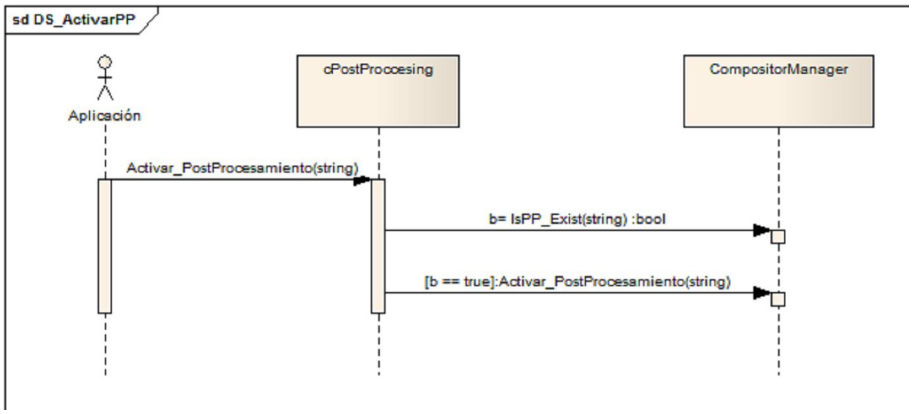

Figura 22: Diagrama de Secuencia "Activar Post Procesamiento".

4.3.4. Realización del caso de uso “Desactivar Post Procesamiento”.

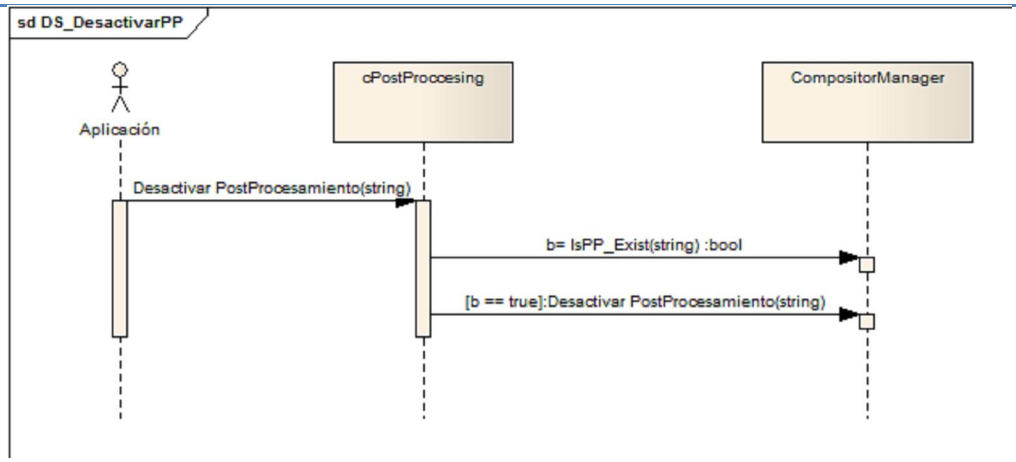

Figura 23: Diagrama de Secuencia "Desactivar Post Procesamiento".

4.3.5. Realización del caso de uso “Asignar Controlador a Post Procesamiento”.

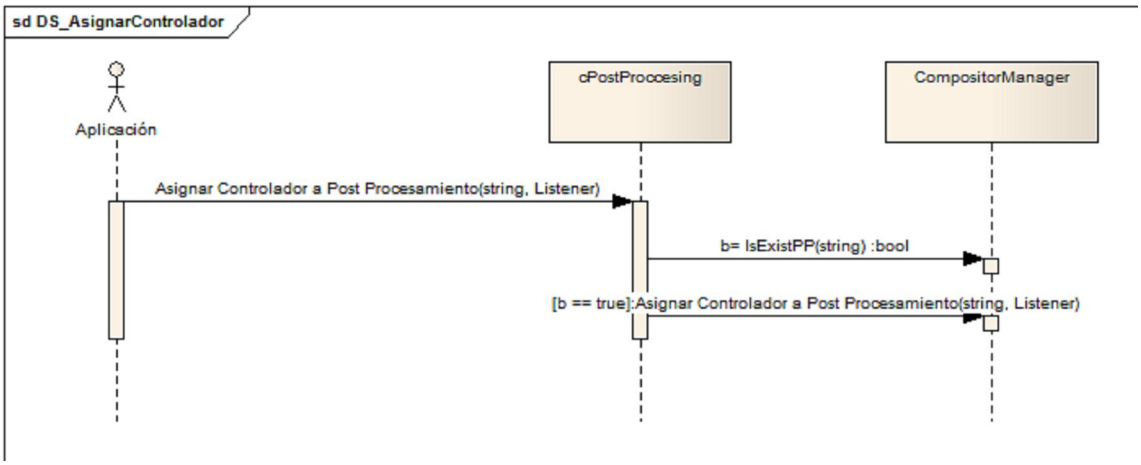

Figura 24: Diagrama de Secuencia "Asignar Controlador a Post Procesamiento".

4.4. Descripción de las clases

4.4.1. Paquete “Post Procesamiento”.

Tabla 7: Clase “cPostProccesing”.

Nombre: cPostProccesing
Tipo de clase: Interfaz
Descripción: Clase controladora contiene todo los efectos de Post Procesamientos creados, es la encargada de activarlos y desactivarlos.

Atributo	**Tipo**
`pPP_Viewport`	`Viewport`
Para cada responsabilidad	
Nombre:	`cPostProccesing()`
Descripción:	Constructor de la clase.
Nombre:	`Create_PP( string )`
Descripción:	Crea un efecto de Post Procesamiento con el nombre especificado.
Nombre:	`Delete_PP( string )`
Descripción:	Eliminar un efecto previamente creado según el nombre pasado por parámetro.
Nombre:	`Enable_PP( string )`
Descripción:	Activar un efecto de Post Procesamiento.
Nombre:	`Disable_PP( string )`
Descripción:	Desactivar un efecto de Post Procesamiento.
Nombre:	`SetPP_Listener( string, Listener )`
Descripción:	Asigna un controlador a un efecto de Post Procesamiento para cambiar las variables de las que hace uso el efecto.

Tabla 8: Clase “cConvolutionListener”.

Nombre: cConvolutionListener	
Descripción: Se encargará de actualizar los valores de los parámetros en la creación y en cada frame de los efectos de post procesamiento que usan Convolution Kernels.	
Tipo de clase (interfaz)	
Atributo	**Tipo**

`Kernel[9]`	`float`
Para cada responsabilidad	
Nombre:	`cConvolutionListener()`
Descripción:	Constructor de la clase.
Nombre:	`notifyMaterialSetup(uint32, MaterialPtr)`
Descripción:	Esta función es llamada cuando se crea el material.
Nombre:	`notifyMaterialRender(uint32, MaterialPtr)`
Descripción:	Es llamada cada vez que se va a pintar la escena, es decir en cada frame.

Tabla 9: Clase “cBlurListener”.

Nombre: cBlurListener	
Descripción: Estas clase actualiza los valores de los parámetros en la creación y en cada frame del efecto de post procesamiento Blur.	
Tipo de clase (interfaz)	
Atributo	**Tipo**
`Samples`	`float`
Para cada responsabilidad	
Nombre:	`cBlurListener ()`
Descripción:	Constructor de la clase.
Nombre:	`notifyMaterialSetup(uint32, MaterialPtr)`
Descripción:	Esta función es llamada cuando se crea el material.
Nombre:	`notifyMaterialRender(uint32, MaterialPtr)`
Descripción:	Es llamada cada vez que se va a pintar la escena, es decir en cada frame.

Tabla 10: Clase "cBloomListener".

Nombre: cBloomListener	
Descripción: Se actualizarán los valores de los parámetros en la creación y en cada frame del efecto de post procesamiento Bloom.	
Tipo de clase (interfaz)	
Atributo	**Tipo**
`Samples`	`float`
Para cada responsabilidad	
Nombre:	`cBloomListener()`
Descripción:	Constructor de la clase.
Nombre:	`notifyMaterialSetup(uint32, MaterialPtr)`
Descripción:	Esta función es llamada cuando se crea el material.
Nombre:	`notifyMaterialRender(uint32, MaterialPtr)`
Descripción:	Es llamada cada vez que se va a pintar la escena, es decir en cada frame.

En este capítulo se confeccionaron los diagramas de clases de cada uno de los paquetes, también los diagramas de secuencias por cada CU donde se tiene una secuencia más exacta de las funcionalidades que realizan. Se realizó una descripción de las clases donde se encuentran los atributos y las operaciones más importantes.

Capítulo 5: Implementación

Este capítulo describe la creación de los efectos de post procesamiento haciendo uso de los compositores en OGRE y de la integración en una aplicación básica para la inclusión de los efectos de la biblioteca.

Para la creación de los efectos de post procesamiento se hizo uso de los compositores de OGRE. Un compositor de OGRE, por decirlo de una manera sencilla es una forma de aplicar un efecto al resultado obtenido por el render de OGRE. Por ejemplo hacer que todos los colores de la pantalla se vayan fundiendo en un color, o se quiere aplicar un filtro de blur a la escena. De la forma tradicional, esto implica hacer primero un render de la escena completa a una textura y aplicar un algoritmo a esa textura, siendo este último el resultado que finalmente se tiene que mostrar en pantalla. Pues bien, los compositores nos permiten simplificar todos estos pasos. También permiten cosas como encadenar compositores, por ejemplo aplicando un efecto al render original, y sobre este aplicar otro efecto, y así sucesivamente.

Para utilizar un compositor necesitamos:

- Especificación del compositor, que puede ser por código o mediante un script (compositor).
- 1 o más materiales (lo mismo, pueden estar definidos en scripts (material)
- 1 o más shaders.

Para su mejor entendimiento se le muestra a continuación la especificación del compositor Blur creado para la biblioteca.

```
compositor PP/Blur
{
    technique
    {
        Texture        rt0        target_width_scaled        0.25
target_height_scaled 0.25 PF_R8G8B8
        texture        rt1        target_width_scaled        0.25
target_height_scaled 0.25 PF_R8G8B8

        target rt0
        {
            input previous
        }

        target rt1
        {
            input none
            pass render_quad
            {
                material BlurV
                input 0 rt0
            }
```

```
            }

            target_output
            {
                input none
                pass render_quad
                {
                    material BlurH
                    input 0 rt1
                }
            }
        }
    }
```

Un compositor es muy parecido en estructura a un material normal de Ogre, ya que tiene igualmente técnicas y pasadas (passes) y también pueden ser creados mediante código.

Una técnica es una especificación del compositor para un caso concreto, por ejemplo, el compositor puede variar dependiendo de las características de la tarjeta gráfica o de las opciones del programa, y puede que algunas técnicas sirvan para una tarjeta y no para otras.

Una pasada es un paso individual del render que se manda hacer a la tarjeta (análogo a los passes de los materiales). Hay varios tipos de passes, pero nos vamos a centrar en el render_quad que permite aplicar un efecto (material) sobre toda la escena (en realidad es sobre todo el RenderTarget, que puede ser una escena, una textura).

Imaginemos el caso en que queremos hacer un blur a toda la escena. Para hacer este blur lo que tenemos que hacer es para cada píxel de la escena mirar el color de unos cuantos píxeles de alrededor suyo y promediarlos. Este es un escenario perfecto para utilizar un compositor ya que se aplica una función sobre cada píxel obtenido en la escena. Para poder hacer este efecto necesitamos:

- La escena original, renderizada en una textura
- Un material que haga el blur teniendo como entrada la escena anterior y lo saque a otra textura, que es la que finalmente se pintará.

Hay dos pasos claramente definidos, primero la obtención de la imagen de la escena (lo que normalmente se denomina Render To Texture) y la aplicación del efecto para obtener el resultado final.

Cada uno de estos pasos corresponde a un target en el script del compositor. El primero tendrá como entrada la imagen original renderizada (o el compositor anterior en la cadena) y como salida una textura temporal que sólo sirve para comunicar los dos targets (`rt0`). El segundo target no tiene una entrada inicial, pero se le asigna a la

primera textura del material la salida del anterior target. Tal como indica el nombre de este target (`target_output`) el resultado es la salida del compositor.

En el caso del blur, necesitaremos un script que utilicé un fragment shader para hacer un promedio de píxeles sobre cada píxel de la escena. El script de material quedaría algo así:

```
fragment_program BlurV_ps20_hlsl hlsl
{
      source BlurV_ps20.hlsl
      target ps_2_0
      entry_point main
}

material Blur
{
      technique
      {
            pass
            {
                  cull_hardware none
                  cull_software none
                  depth_check off

                  fragment_program_ref BlurV_ps20_hlsl
                  {
                  }
                  texture_unit
                  {
                        tex_coord_set 0
                        tex_address_mode clamp
                        filtering trilinear
                  }
            }
      }
}
```

Para emplear los efectos de Post Procesamiento creado en la biblioteca se hace necesaria la creación de una instancia de la clase controladora `cPostProccesing`, la cual quedaría de esta forma:

```
cPostProcessing* pPostProcessing;

pPostProcessing = new cPostProcessing( currentViewport );
```

Después de creada la instancia de la biblioteca podemos referirnos a la misma haciendo uso directo de dicha instancia o usando el Singleton creado por la misma. En el siguiente código de muestra el empleo de las diferentes formas de usar la biblioteca.

```
pPostProcessing->Create_PP( PP_Name );

cPostProcessing::getSingleton().Create_PP( PP_Name );
```

Ya creada la instancia de la biblioteca podemos ir agregando los efectos de post procesamiento que esta posee, habiendo copiado anteriormente la carpeta *materials* que posee la misma para su proyecto y agregarla al archivo *resource.cfg* de su aplicación para que cuando desee crear un efecto dicha biblioteca encuentre la definición de los mismos. Con el siguiente fragmento de código podemos crear los efectos que contiene la biblioteca una vez realizado los pasos mencionados anteriormente.

```
pPostProcessing->Create_PP( “Bloom” );
```

De esta forma ya el efecto “Bloom” está cargado y listo para ser activado en la escena para su visualización y lograr el efecto creado por el mismo, lo cual se realiza de la siguiente manera:

```
pPostProcessing->Enable_PP( “Bloom” );
```

Y para desactivarlo en caso de no desear aplicarlo más, se usa el método `Disable_PP`, un ejemplo de cómo hacer uso de esta función:

```
pPostProcessing->Disable_PP( “Bloom” );
```

Después de haber creado un efecto de Post Procesamiento y haber hecho uso del mismo, es decir haberlo aplicado en la escena y disfrutado el efecto alcanzado por el mismo y ya no se desea usarlo más, podemos eliminarlo de los efectos creados en la aplicación para liberar recursos. Con la siguiente instrucción eliminamos un efecto:

```
pPostProcessing->Delete_PP( “Bloom” );
```

Una vez creado un efecto es posible que el Post Procesamiento que utilicemos no necesiten que se puedan cambiar sus valores en tiempo de ejecución (ejemplo siempre se hace el blur del mismo tamaño). En ese caso no haría falta asignarle un controlador en caso contrario es necesario crear un controlador, que se encargará de actualizar los valores de los parámetros en la creación y en cada frame. Esta es su estructura en el caso de los Convolution:

```
cConvolutionListener* pConvolution = new cConvolutionListener();
pPostProcessing->SetPP_Listener(“Convolution”, pConvolution);
```

4.5. Diagrama de Componentes

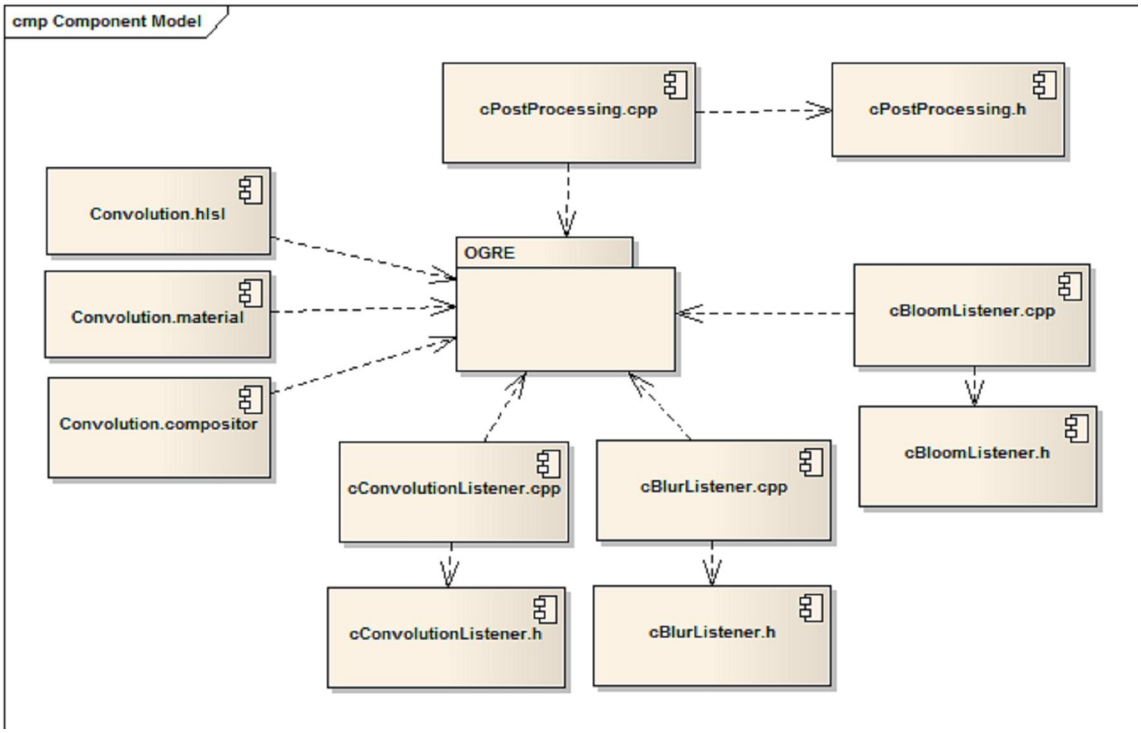

Figura 25: Diagrama de Componentes.

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 brindan las 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 propone la utilización de algunos efectos de post procesamiento 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 obtuvo un diagrama de clases lo suficientemente extensible que contribuye a la inclusión de otros post procesamiento 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. Se espera 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

Después de haber obtenido los resultados propuestos para esta investigación se recomienda:

- Que el presente trabajo sirva como punto de partida y base de estudio para futuras tesis o investigaciones relacionadas con las técnicas de generación de efectos de post procesamiento.
- Investigar con más profundidad las potencialidades que brindan las nuevas tecnologías del hardware gráfico para la inclusión de nuevos efectos para el uso compartido de los proyectos.
- Que se haga uso de la colección de efectos de post procesamiento creados por la biblioteca para aumentar la calidad de los Entornos Virtuales.

Referencias Bibliográficas

Card, Mitchell y Drew, Jason L. 2002. *Non-Photorealistic Rendering with Pixel and Vertex Shaders.* s.l. : Wordware Publishing, 2002., 2002.

DESAROLLO, PROCESOS DE. Metodologías RUP y XP. [En línea] [Citado el: Martes 17 de Marzo de 2009.] http://jackopc.blogspot.com/.

Engel, Wolfgang F. 2002. *Direct3d ShaderX: Vertex and Pixel Shader Tips and Tricks.* s.l. : Wordware Publishing, Inc, 2002. ISBN 1-55622-041-3.

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

Guinot, Jérôme. 2005. OZone3D.net. [En línea] 8 de Diciembre de 2005. http://www.OZone3D.net.

Kylmamaa, Marko. 2006. Creating a Post-processing Framework: Tips & Tricks. [En línea] 2006. http://gamasutra.com/features/20061003/kylmamaa_01.shtml..

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

Microsoft Corporation. 2009. *The DirectX Software Development Kit.* March de 2009. PostProcess Sample.

Ogayar Anguita, Carlos Javier. 2006. *Optimización de algoritmos geométricos básicos mediante el uso de recubrimientos simplistas.* [En línea] 2006. http://hera.ugr.es/tesisugr/16135143.pdf.

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

Programming, eXtreme. A propósito de programación extrema XP. [En línea] http://www.monografias.com/trabajos51/programacion-extrema/programacion-extrema.shtml.

Tirado Granados, Gonzalo. *Introducción a OGRE 3D.*

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

Glosario de Abreviaturas

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

CPU: Unidad Central de Procesamiento.

CU: Caso de Uso.

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

HW: Hardware.

SRV: Sistemas de Realidad Virtual.

UCI: Universidad de la Ciencias Informáticas.

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:

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.

L:

Luminancia: (Ver el término 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.

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.

Singleton:

T:

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

Textura: Colección o arreglo n-dimensional de texel.

V:

Vector: cantidad que expresa magnitud y dirección.

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

Índice de Figuras y Tablas

Anexos

Anexo No. 1: Efecto Bloom.

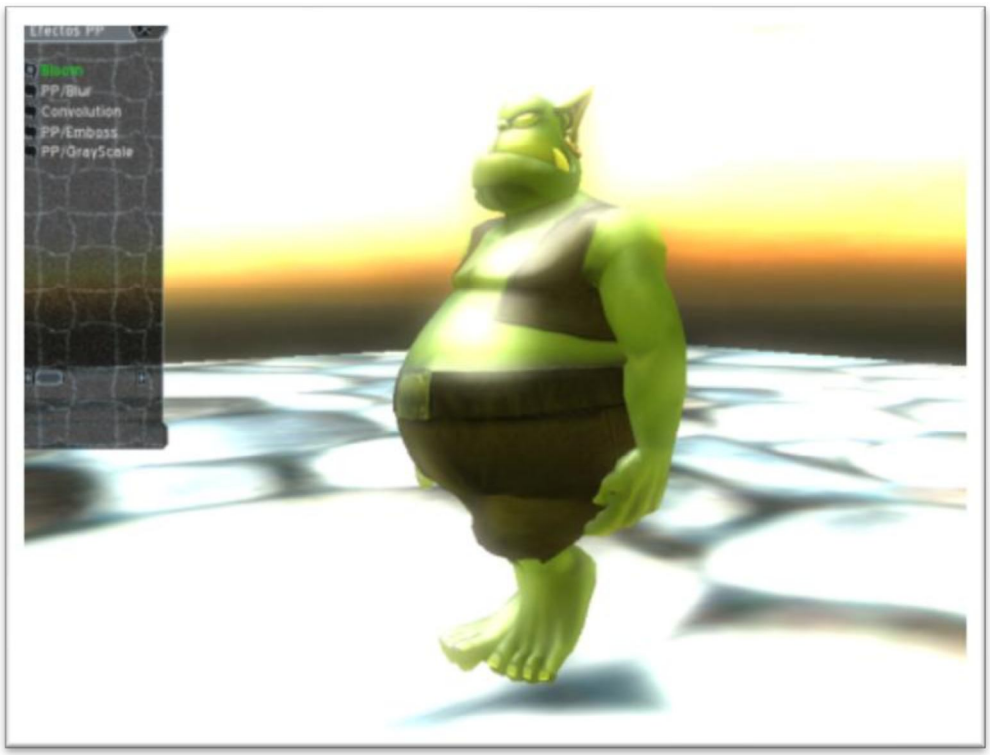

Figura 26: Efecto "Bloom".

Anexo No. 2: Efecto Blur.

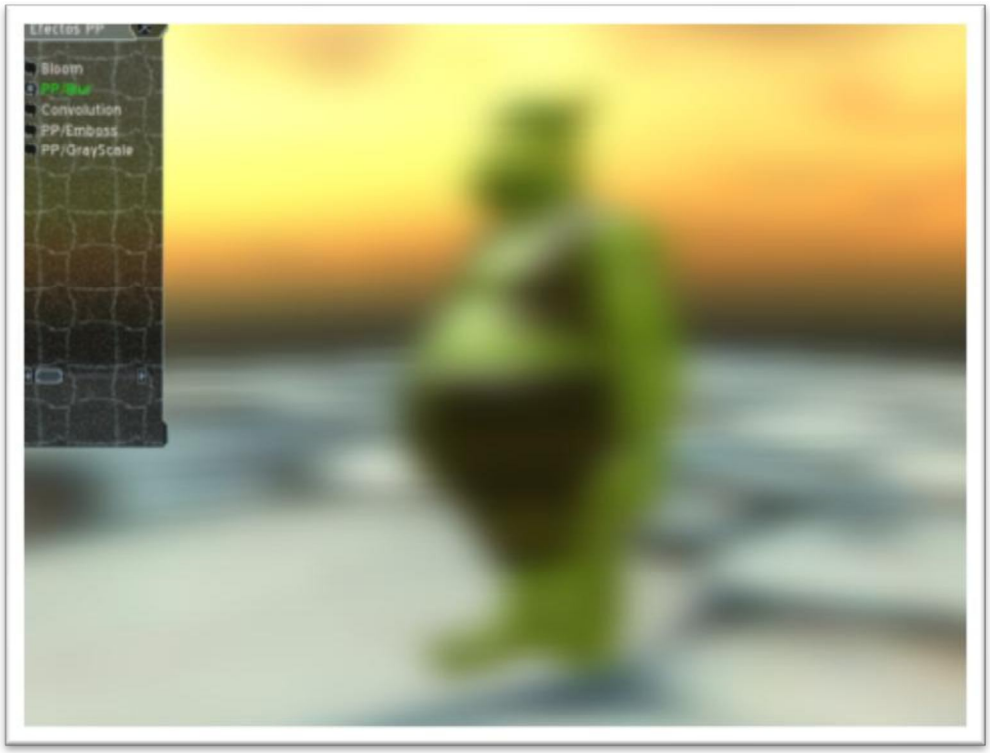

Figura 27: Efecto "Blur".

Anexo No. 3: Efecto Edge Detection.

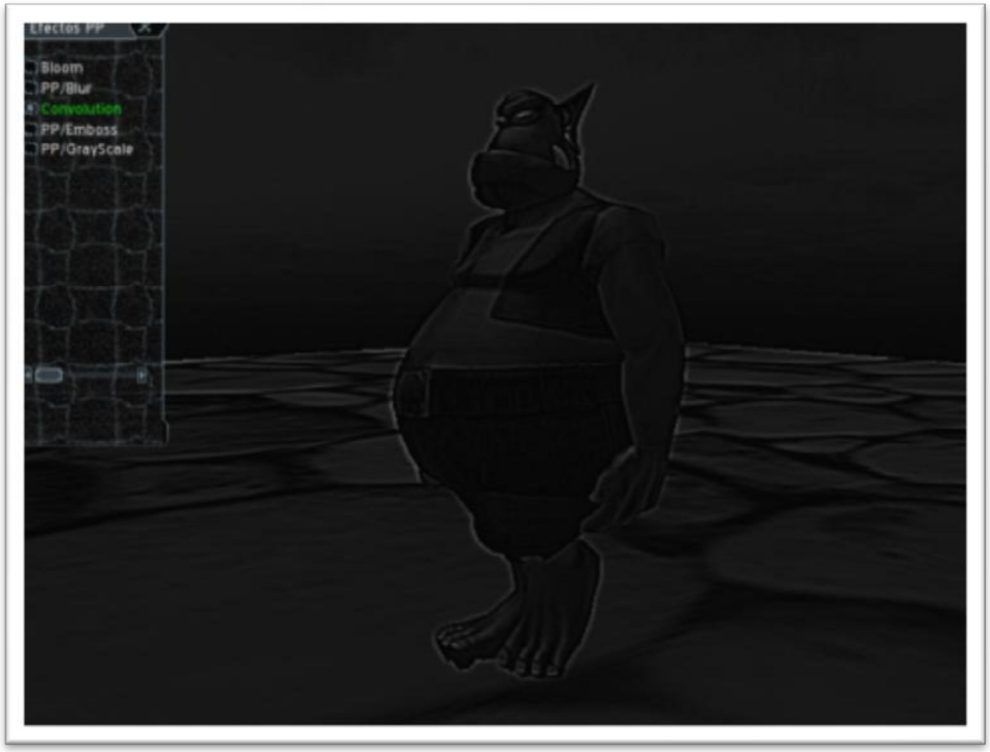

Figura 28: Efecto "Edge Detection".

Anexo No. 4: Efecto Emboss.

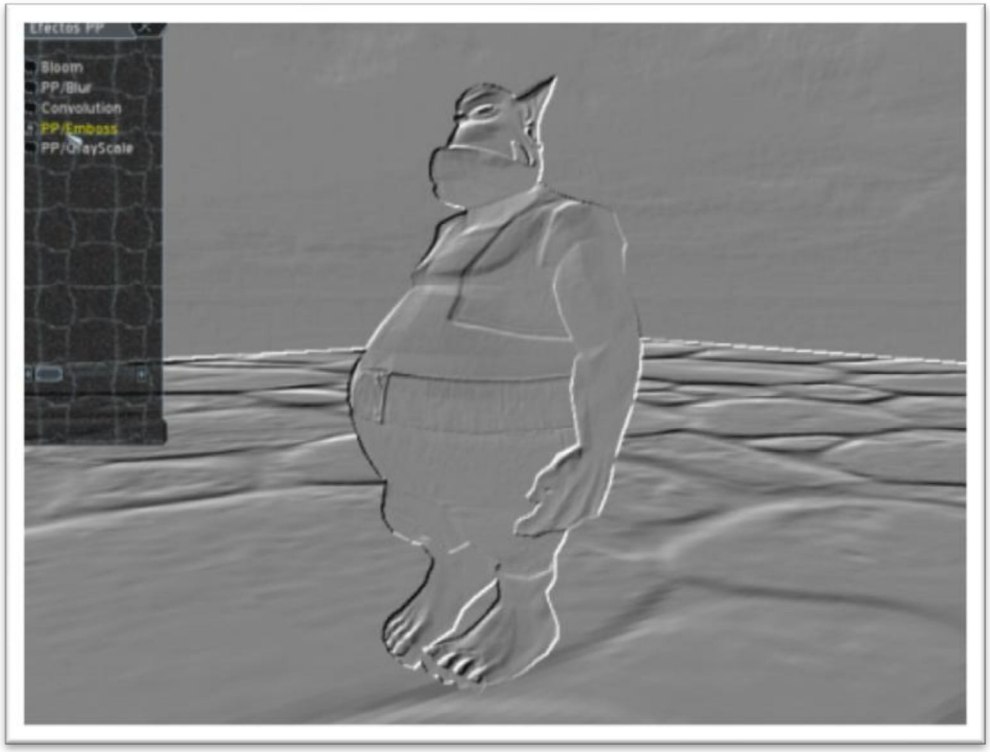

Figura 29: Efecto "Emboss".