Universidad de las Ciencias Informáticas
Facultad 5

Título: “Módulo de Shader para el proyecto Laboratorios Virtuales”

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Juan Carlos Ayala Alonso

Tutor: MSc. Orlay García Ducongé

Co-tutor: Ing. Yasmany Alfonso Monteagudo

La Habana 2012
“Año 54 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____
Juan Carlos Ayala Alonso

_____ _____

Tutor: Orlay García Ducongé Co-tutor: Yasmany Alfonso Monteagudo

DATOS DE CONTACTO

AUTOR:

Juan Carlos Ayala Alonso.
Universidad de las Ciencias Informáticas,
La Habana, Cuba.
E-mail: jcayala@estudiantes.uci.cu

TUTOR:

MSc. Orlay García Ducongé.
Universidad de las Ciencias Informáticas,
La Habana, Cuba.
E-mail: oducunge@uci.cu

CO-TUTOR:

Ing. Yasmany Alfonso Monteagudo.
Universidad de las Ciencias Informáticas,
La Habana, Cuba.
E-mail: yamonteagudo@uci.cu

AGRADECIMIENTOS

A mi mamá Milvia, que es la persona más importante en mi vida.

A mi papá Juan Carlos.

A mis abuelos Lolita, Caridad y Melquiades.

A toda mi familia en general.

A mi novia Lianet.

A mis suegros Emma y Reynaldo.

A toda la familia de mi novia en general.

A mis tutores Orlay y El Doctor.

A todas mis amistades.

A todas las personas que de una forma u otra han contribuido con mi formación personal y profesional.

DEDICATORIA

A mi mamá, mi papá, mis abuelos y a mi novia.

RESUMEN

En la facultad 5 de la Universidad de las Ciencias Informáticas se han desarrollado varios productos relacionados con la Realidad Virtual. Los últimos productos realizados por esta facultad son tres laboratorios virtuales que se desarrollaron con el objetivo de apoyar el proceso de aprendizaje en la carrera de Informática. Uno de los principales problemas que se identificaron en estos productos fue la representación de los objetos tridimensionales, debido a que son representados con texturas planas y no muestran un aspecto similar al que poseen en la realidad. También se detectó una pobre iluminación de las escenas representadas en estos productos. El trabajo que se presenta tiene como objetivo elaborar un módulo para el proyecto Laboratorios Virtuales que permita resaltar los relieves de los objetos tridimensionales que se utilizan. Algunos de los principales conceptos estudiados para la realización del trabajo fueron los *shaders* y las técnicas de gráficos computacionales tridimensionales, para lograr resaltar el relieve en los objetos simulados. Además se investigó sobre el renderizado de alto rango dinámico y algunas de las técnicas que utiliza para el caso de la iluminación. Con el trabajo realizado se obtuvo un módulo que resalta los relieves de los objetos simulados en los laboratorios virtuales, además se mejoró la iluminación en las escenas.

PALABRAS CLAVE

Laboratorios virtuales, píxel, *shader*, unidad de procesamiento gráfico.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En el mundo existe un constante cambio producto al desarrollo tecnológico, es una realidad la necesidad de poseer nuevas y mejores tecnologías que ayuden al desarrollo de la humanidad. Las nuevas tecnologías han propiciado que se expanda el campo del conocimiento del hombre ampliando su capacidad para innovar y crear nuevas soluciones a los problemas que se enfrenta en la vida.

La creación de los ordenadores o computadoras fue un gran paso de avance para el desarrollo de la humanidad, ya que le permitió al hombre almacenar, procesar y transmitir información. El mejoramiento de estas computadoras ha permitido la aparición de las tarjetas gráficas, para aumentar el rendimiento de estos equipos en cuanto a la calidad visual de sus aplicaciones. Con este creciente desarrollo de los ordenadores han surgido nuevas tecnologías de visualización y modelación como la Realidad Virtual, que permite un mejor acercamiento a fenómenos o hechos reales mediante su simulación tridimensional e interacción con diferentes equipos especializados.

La Realidad Virtual es una tecnología con un futuro impresionante debido a las potencialidades que presenta, las cuales pueden ser puestas en práctica en diferentes campos de la sociedad. Del hombre depende su desarrollo para bienes comunes y para el crecimiento de las nuevas tecnologías que son de vital importancia para el desarrollo de la humanidad.

En la facultad 5 de la Universidad de las Ciencias Informáticas (UCI), se han desarrollado aplicaciones de Realidad Virtual con buenos resultados obtenidos. Como ejemplo de estos productos se tiene el Simulador Quirúrgico, el Simulador de Auto, los juegos Rápido y Curioso, Meteorix y Energía para Aprender. Los últimos productos realizados en la facultad 5, específicamente en el Centro de Informática Industrial (CEDIN), fueron tres laboratorios virtuales. Estos productos se desarrollaron con el objetivo de apoyar el proceso de aprendizaje en la carrera de Informática, ofreciéndoles a los estudiantes una herramienta que beneficia su educación, además de reducir el costo en cuanto a materiales de enseñanza ya que estos se muestran virtualmente.

Uno de estos productos fue el laboratorio virtual Ensamblaje de un Computador, el cual consiste en armar una computadora en piezas utilizando diferentes herramientas. Otro producto fue el laboratorio virtual Diseño e Instalación de una Red de Área Local (LAN), que permite a los usuarios confeccionar una red conectando los dispositivos necesarios así como entrenarse en la confección del cableado. El tercero es el

laboratorio virtual Administración y Configuración de una Red de Área Local (LAN), que permite el entrenamiento en los procesos de administración y configuración de una red de área local sobre los sistemas operativos Windows y Linux.

En estos productos los objetos simulados no cuentan con un buen nivel de realismo, debido a que son representados con texturas planas. Los diseñadores que crearon los objetos no pudieron realizar modelos complejos que tuvieran una gran cantidad de polígonos, producto al bajo nivel de cómputo que presentaban las computadoras donde serían puestos en práctica los tres laboratorios. En estos productos no se muestran los objetos tridimensionales con un nivel de relieve que simulen objetos de la vida real. En el Anexo 1 se representa un chasis simulado en el laboratorio virtual Ensamblaje de un Computador y en el Anexo 2 un chasis real [A1], [A2].

Existe gran diferencia entre el chasis real y el chasis simulado en el laboratorio virtual. El simulado no muestra varios relieves que posee el real, debido a que es representado con texturas planas. Los detalles en el chasis simulado son realizados en la geometría del modelo, incrementando la cantidad de polígonos en el objeto y atentando contra la capacidad de procesamiento de las máquinas, debido a que la representación de los objetos en tres dimensiones requiere de un alto rendimiento de procesamiento matemático. También se puede notar que existe una pobre iluminación en las escenas, por esta causa no se percibe la superficie metálica que posee el chasis. Además este chasis simulado carece de sombras al igual que el resto de los objetos que se representan en la escena.

Debido a la necesidad de dar solución a la problemática planteada, se identifica como ***problema científico:*** ¿cómo resaltar los relieves y detalles de los objetos tridimensionales en los laboratorios virtuales?

Este problema determinó como ***objeto de estudio:*** las características de los objetos tridimensionales en los laboratorios virtuales y se precisa como ***campo de acción:*** las características topológicas de los objetos tridimensionales en los laboratorios virtuales.

Para dar solución al problema planteado se define como ***objetivo general:*** elaborar un módulo para el proyecto Laboratorios Virtuales que permita resaltar los relieves de los objetos tridimensionales que se utilizan.

Para dar cumplimiento a los objetivos se proponen las siguientes ***tareas:***

- Elaboración del marco teórico de la investigación a partir del estado del arte actualmente existente sobre el tema.
- Identificación de las técnicas existentes para resaltar los relieves de los objetos tridimensionales.
- Definición de las funcionalidades que debe tener el módulo.
- Selección de herramientas y lenguajes para la implementación del sistema.
- Diseño del módulo.
- Implementación del módulo.
- Realización de pruebas, con el objetivo de validar la solución obtenida.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

Este capítulo agrupa una serie de conceptos e informaciones referentes a las técnicas, tecnologías y programas informáticos que se utilizarán. Además se darán a conocer sus características, aspectos fundamentales y potencialidades que presentan. Algunos de los temas que se abordarán serán los laboratorios virtuales, los *shaders*, los lenguajes de programación para *shaders* y el motor de render Ogre3D.

1.1 Laboratorios virtuales

Un laboratorio virtual es un sistema computacional que pretende aproximar el ambiente de un laboratorio tradicional donde ocurren sucesos reales. Los experimentos se realizan paso a paso, siguiendo un procedimiento similar al de un laboratorio real, se visualizan instrumentos y fenómenos mediante objetos dinámicos, imágenes o animaciones. Se obtienen resultados como datos numéricos y gráficos, tratándose estos matemáticamente para la obtención de los objetivos perseguidos en la planificación docente de las asignaturas (1).

También se define como una simulación en una computadora u ordenador personal de una amplia variedad de situaciones, que pueden ser desde prácticas manipulables hasta visitas guiadas. Se desarrolla en un ambiente interactivo y quienes aprenden lo pueden usar fuera del campus universitario, sin ayuda alguna del personal docente. El lugar del laboratorio virtual es la computadora personal, donde no se mantiene una comunicación directa con el personal docente, pues ha sido estructurado de forma tal que pueda manipularse desde cualquier computadora (2).

Actualmente existe una gran cantidad de laboratorios virtuales desarrollados en diferentes regiones del mundo y enfocados hacia diferentes esferas de la vida. Un ejemplo es el laboratorio virtual *Livewire* desarrollado por la compañía inglesa *New Wave Concepts*, el cual permite hacer simulaciones que demuestran los principios de funcionamiento de los circuitos electrónicos basados en leyes fundamentales como la ley de Ohm y las leyes de Kirchhoff. *Livewire* ofrece la ventaja de visualizar lo que ocurre con el funcionamiento del circuito cuando se realiza alguna transformación y así poder modificar y mejorar el diseño electrónico del mismo (3). Este laboratorio virtual realiza las simulaciones en dos dimensiones o

2D, mostrando los objetos con insuficiente realismo debido a que se representan de forma plana y sin profundidad.

La Universidad Nacional de Educación a Distancia (UNED), es una institución española donde se han desarrollado diferentes laboratorios virtuales. Uno de los ejemplos que se puede encontrar en esta escuela es la simulación de un sistema de control de cuatro tanques interconectados. Mediante este sistema se pueden modificar los niveles del líquido y la geometría de los tanques, así como el porcentaje de apertura de las válvulas, el diámetro de los desagües y los parámetros de los dos controladores que se utilizan. Otro ejemplo de laboratorio virtual desarrollado por la UNED es la simulación de un intercambiador de calor. Este laboratorio virtual consiste en simular la variación de temperatura que experimentan dos tuberías, una dentro de otra, por las que fluyen líquidos con diferentes temperaturas. Las longitudes de las tuberías, el grosor de la tubería interna y la temperatura son algunos de los parámetros modificables en este intercambiador de calor (4). Estos laboratorios desarrollados por la UNED no muestran los objetos como son en la realidad, debido a que se representan en 2D. De esta forma los usuarios que hacen uso de estos laboratorios tienen que abstraerse para asimilar la información que se les brinda.

En Cuba se cuenta con diferentes laboratorios virtuales como el de Química General, que surge como parte de un proyecto llamado Programas Informáticos para la enseñanza universitaria de la Química experimental (5). También en la Universidad Virtual del Ministerio de Ciencia, Tecnología y Medio Ambiente (CITMA), se han desarrollado varios laboratorios virtuales. Ejemplos de estos se tienen los laboratorios de modelación y experimentación numérica para Matemáticas y los laboratorios virtuales de Física (6). Las simulaciones generadas por estos productos carecen de realismo debido a que son en dos dimensiones. Los usuarios que interactúan con estos laboratorios perciben las escenas en un plano y con insuficiente calidad visual. Otro ejemplo en nuestro país son los tres desarrollados en la UCI, a continuación se explica en qué consiste cada uno de ellos.

El laboratorio virtual Ensamblaje de un Computador es un material de apoyo para la unidad curricular Arquitectura de un Computador. Este consiste en armar una computadora utilizando las diferentes piezas que se necesitan para hacerlo. Cuenta con dos fases que le permite al usuario desarrollar el proceso de ensamblaje de una computadora. La primera fase es piezas internas del computador, la cual tiene como principal objetivo lograr que el usuario desarrolle, a partir de los conocimientos adquiridos en el aula, el

ensamblaje interno de una computadora. En esta primera fase se manipulan piezas como el microprocesador, memoria RAM, ventilador y tornillos. La segunda fase es piezas externas del computador y su objetivo es lograr que el usuario desarrolle, partiendo de los conocimientos que ya ha adquirido, el ensamblaje externo de una computadora. En esta segunda fase se manipulan piezas como el lector de CD y DVD, monitor, *mouse* y discos duros (7).

El laboratorio virtual Diseño e Instalación de una Red de Área Local (LAN), como lo indica su nombre, consiste en permitir al usuario desarrollar el proceso de diseño e instalación de una red de área local. Este laboratorio virtual cuenta con tres fases. La primera fase es construcción de un cable de red y tiene como objetivo construir cables de red, mediante el uso de un entorno virtual, aplicando los conocimientos que el usuario posee. Algunos de los instrumentos que se utilizan en esta primera fase para la confección del cable son el alicate y la pinza. Al terminar la confección se realiza una prueba para verificar el estado del cable. La segunda fase es configuración de una red en 2D, su objetivo es construir una red de área local en dos dimensiones, mediante el uso de un entorno virtual, aplicando los conocimientos del usuario. En esta fase los usuarios seleccionan y ubican los dispositivos que van a utilizar y los conectan entre ellos. La tercera fase es configuración de una red en 3D, cuyo objetivo es diseñar e instalar una red de área local en tres dimensiones de un edificio, mediante el uso de un entorno virtual y aplicando los conocimientos que el usuario posee. Al igual que en la segunda fase los usuarios seleccionan y ubican los dispositivos que van a utilizar y los conectan entre ellos, pero se pueden desplazar por todo el edificio (8).

Por último, el laboratorio virtual Administración y Configuración de una Red de Área Local (LAN), cuenta con cinco actividades que tributan a la configuración de una red de área local. Este laboratorio virtual utiliza la red que fue diseñada previamente en el laboratorio virtual Diseño e Instalación de una Red de Área Local (LAN). El usuario puede administrar o configurar las distintas computadoras que aparecen en la red utilizando diferentes sistemas operativos. Las actividades que presenta este laboratorio virtual son: TCP/IP; Sistema de Nombre de Dominio o DNS, por sus siglas en inglés; *Proxy*; Protocolo de Transferencia de Archivos o FTP, por sus siglas en inglés y SAMBA. Con el uso de estas actividades es posible que ordenadores con GNU/Linux, Mac OS X o Unix en general se vean como servidores o actúen como clientes en redes de Windows. Con la excepción de SAMBA, que solo permite la configuración en el sistema operativo Linux (9).

1.2 Píxel

Un píxel es la menor unidad homogénea en color que forma parte de una imagen digital, ya sea una fotografía, un fotograma de video o un gráfico. Ampliando lo suficiente una imagen digital (*zoom*) en la pantalla de un ordenador, pueden observarse los píxeles de la imagen. Estas se forman como una sucesión de píxeles. La sucesión marca la coherencia de la información presentada, siendo su conjunto una matriz coherente de información para el uso digital. El área donde se proyectan estas matrices, suele ser rectangular. La representación homogénea del píxel en cuanto a la variación del color y densidad por pulgada, siendo esta variación nula y definiendo cada punto en función de la densidad, en lo referente al área. En las imágenes de mapa de bits o en los dispositivos gráficos cada píxel se codifica mediante un conjunto de bits de longitud determinada (profundidad de color). Un píxel puede codificarse con un byte (8 bits), de manera que cada píxel admite 256 variaciones, 2 elevado a la 8. También se pueden emplear tres bytes para definir un color; representándose un total de 2 elevado a la 24, colores, que suman 16777216 opciones de color (32 bits son los mismos colores que 24 bits, pero tiene 8 bits más para transparencia).

Para poder transformar la información numérica que almacena un píxel en un color, se debe conocer, además de la profundidad y brillo del color (el tamaño en bits del píxel), el modo de color que se está usando. El modo de color rojo, verde y azul o en inglés *red*, *green*, *blue* (RGB), permite crear un color compuesto por los tres colores primarios según el sistema de mezcla aditiva. De esta forma, en función de la cantidad de cada uno de ellos que se utilicen se apreciarán los resultados. Por ejemplo, el color violeta se obtiene mezclando el rojo y el azul. Las distintas tonalidades del violeta se obtienen variando la proporción en que intervienen ambos componentes. En el modo RGB es frecuente que se usen 8 bits para representar la proporción de cada uno de los tres componentes primarios. De esta forma, cuando uno de los componentes vale cero es porque no interviene en la mezcla y cuando vale 255 ((2 elevado a la 8) – 1) es porque interviene aportando el máximo de ese tono. La mayor parte de los dispositivos que se usan con un ordenador como el monitor y el escáner usan el modo RGB. Un píxel alcanza los 8 bits (2 elevado a la 8, colores), 24 bits (2 elevado a la 24, colores) o 48 bits (2 elevado a la 48, colores), este último valor de precisión solo se obtiene con escáneres o cámaras de gama alta (que utilicen formato raw o tiff) (10).

1.3 Unidad de procesamiento gráfico

Acrónimo del inglés *graphics processing unit* o unidad de procesamiento gráfico (GPU). Es un procesador dedicado exclusivamente al procesamiento de gráficos. Permite aligerar la carga de trabajo de la unidad de procesamiento central o en inglés *central processing unit* (CPU), en aplicaciones tridimensionales o 3D como los videojuegos. De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la GPU, la CPU puede dedicarse a otro tipo de cálculos (como la inteligencia artificial o los cálculos mecánicos en el caso de los videojuegos). Una GPU implementa varias operaciones gráficas llamadas primitivas optimizadas para el procesamiento gráfico. Una de las primitivas para el procesamiento gráfico en 3D es el *antialiasing*, que suaviza los bordes de las figuras para darles un aspecto más realista. Adicionalmente existen primitivas para dibujar rectángulos, triángulos, círculos y arcos. La GPU actualmente dispone de gran cantidad de primitivas, buscando mayor realismo en los efectos.

Una GPU está altamente segmentada, lo que indica que posee gran cantidad de unidades funcionales. Estas unidades funcionales se pueden dividir en dos fundamentales: aquellas que procesan vértices y aquellas que procesan fragmentos, que luego pasan a ser píxeles. Por tanto, se establecen el vértice y el fragmento como las principales unidades que maneja la GPU. Adicionalmente y no con menos importancia se encuentra la memoria. Esta se destaca por su rapidez a la hora de almacenar los resultados intermedios de las operaciones y las texturas que se utilicen. Inicialmente a la GPU le llega la información de la CPU en forma de vértices. Estos vértices reciben un tratamiento donde se les aplican algunas transformaciones. Posterior a esto, se define la parte de estos vértices que se va a observar (*clipping*) y los vértices se transforman en píxeles mediante el proceso de rasterización. Estas etapas no poseen una carga relevante para la GPU. Cuando se procesan los fragmentos, en el *chip* gráfico, se produce el principal cuello de botella. En este paso es donde se realizan las transformaciones referentes a los fragmentos. Cuando se ha realizado lo mencionado anteriormente y antes de almacenar los fragmentos en la caché, se aplican algunos efectos como el *antialiasing* y el efecto niebla (11).

1.4 *Shader*

Los *shaders* son pequeños programas que se ejecutan en la GPU. Con la aparición de estos pequeños programas se eliminó la barrera entre los programadores y la GPU, debido a que anteriormente no se podía acceder al *hardware* gráfico a través de *software*. Mediante los *shaders* es posible modificar la transformación de vértices, la iluminación, el mapeado de textura y otros aspectos que anteriormente eran controlados por una sección del *hardware* de video llamada *pipeline* de función fija o FFP, por sus siglas en inglés. Los *shaders* permiten a los programadores mayor control sobre sus aplicaciones, ya que pueden crear diferentes efectos que ayudan a la calidad visual en su trabajo. Algunos de los efectos que se pueden lograr con su uso son realmente impresionantes, como animaciones detalladas, efectos de partícula, fuegos realistas, niebla, humos, aguas, nubes, entre otros. Los efectos de iluminación o de la geometría de los objetos pueden ser mejorados considerablemente con la utilización de los *shaders*, debido a que los vértices y los fragmentos se tratan independientes. La GPU posee tres tipos de procesadores de *shaders*, que representan etapas por las que pasan los datos para al final ser mostrados. El *Vertex Shader*, el *Geometry Shader* y el *Pixel Shader* o *Fragment Shader* son los tres procesadores de *shaders* que presenta la GPU. A continuación se explicará en qué consiste cada uno de ellos.

El *Vertex Shader* es una función con la que se puede procesar los valores de un vértice en un plano de dos o tres dimensiones y recibe como parámetro un vértice. No opera sobre una primitiva, como un triángulo, solo trabaja con un vértice a la vez y no puede crearlo ni eliminarlo, solo transformarlo. Para ello, modifica propiedades del vértice mediante operaciones matemáticas para que repercutan en la geometría del objeto al que pertenece. Algunas de las propiedades que puede modificar son el color, las coordenadas de la textura, la posición y la orientación en el espacio. Con esto se puede lograr ciertos efectos específicos, como los que tienen que ver con la deformación en tiempo real de un elemento; por ejemplo, el movimiento de una ola. Donde toma gran importancia es en el tratamiento de las superficies curvas y su avance se ve reflejado en los videojuegos más avanzados de la actualidad. Particularmente, en el diseño de los personajes y sus expresiones corporales (12). Los programas pertenecientes a este procesador de *shader* se les conocen como programas de vértices.

El *Geometry Shader* se utiliza fundamentalmente para generar dinámicamente primitivas como puntos, líneas o triángulos, aunque también puede modificar primitivas existentes. Una de las principales aplicaciones del *Geometry Shader* se puede encontrar en la modificación automática de la complejidad de

una malla. Por ejemplo cuando una serie de líneas representa los puntos de control para una curva que es pasada al *Geometry Shader* y dependiendo de la complejidad requerida, automáticamente puede generar otras líneas, para de esta forma obtener una mejor aproximación de la curva. Otra aplicación se aprecia cuando se eliminan vértices de una malla poligonal según la posición del observador. Después de la ejecución del *Vertex Shader* se ejecuta el *Geometry Shader*, capturando las primitivas mediante los vértices y procesándolos para luego enviárselos al *Fragment Shader*.

El *Pixel Shader* o *Fragment Shader* no interviene en el proceso de la definición del "esqueleto" de la escena (*Wireframe*), sino que forma parte de la segunda etapa: la rasterización (*Rendering*). Allí es donde se aplican las texturas y se tratan los fragmentos que forman parte de ellas. Básicamente, un *Fragment Shader* especifica el color final de un píxel que se mostrará en la pantalla. Este tratamiento individual de los fragmentos permite que se realicen cálculos principalmente relacionados con la iluminación del elemento del cual forman parte en la escena y en tiempo real. Teniendo la posibilidad de iluminar cada fragmento por separado es como se logran crear buenos efectos en los videojuegos actuales. Una particularidad de los *Fragment Shader* es que, a diferencia de los *Vertex Shader*, requieren de un soporte de *hardware* compatible (12). Los programas que pertenecen a este procesador de *shader* se les conocen como programas de fragmentos.

1.5 Lenguajes de programación para *shaders*

Anteriormente los *shaders* se programaban con lenguajes de programación de bajo nivel como el lenguaje ensamblador, los programas escritos no eran legibles y resultaba difícil la reutilización del código. Con la creación de las interfaces de programación de aplicaciones (API) específicas para gráficos, se introdujo un nivel más entre el *hardware* y el *software*, estas API proporcionaron un lenguaje homogéneo para los modelos existentes en el mercado. La primera API usada ampliamente fue *Open Graphics Library* (*OpenGL*) y posteriormente la empresa *Microsoft* desarrolló *Direct3D*. Con el desarrollo de las API, se decidió crear un lenguaje más natural y cercano al programador: los lenguajes de programación de alto nivel para programar *shader*, los cuales han beneficiado considerablemente a los desarrolladores, facilitándoles el trabajo en gran medida. Los principales lenguajes de programación de alto nivel para programar *shader* que existen actualmente son Cg, HLSL y GLSL.

1.5.1 Lenguaje Cg

C para gráficos o Cg, por sus siglas en inglés, es un lenguaje de programación para *shader* que fue desarrollado por las empresas *NVidia* y *Microsoft*. Cg le permite a los desarrolladores la creación rápida de efectos especiales en tiempo real. Este lenguaje de programación se basa en el lenguaje C, que es un lenguaje poderoso y ha servido de base a otros lenguajes de programación. Cg es compatible con las API *OpenGL* y *Direct3D* y es un lenguaje multiplataforma. Generalmente este lenguaje trabaja con los *Vertex Program* o programas de vértices y con los *Fragment Program* o programas de fragmentos. Para lograr mejor portabilidad de los programas escritos en el lenguaje Cg se utilizan perfiles, donde se definen las características que deben tener las computadoras y sistemas donde serán ejecutados los programas. Además este lenguaje cuenta con una amplia biblioteca de funciones que facilita el trabajo de los desarrolladores. Entre las funciones que posee esta biblioteca se encuentran las funciones matemáticas, para facilitar los cálculos que se llevan a cabo, las funciones de geometría y las funciones para el trabajo con texturas que son de gran utilidad, debido a que reducen considerablemente el trabajo (13).

Algunas de las ventajas de utilizar Cg son:

- Lenguaje fácil de entender y de programar, con una sintaxis similar al lenguaje C.
- El código Cg es portable a varios sistemas operativos.
- El compilador Cg puede optimizar el código y realizar otras tareas de bajo nivel automáticamente.
- Es fácil para depurar errores.

1.5.2 Lenguaje HLSL

Lenguaje de sombreado de alto nivel o HLSL, por sus siglas en inglés, es uno de los lenguajes de programación para *shader*. Este lenguaje fue creado por la empresa *Microsoft* y como principal limitante tiene que solo se puede usar con la API *Direct3D*. Es uno de los lenguajes más utilizados en la industria de los videojuegos. Al igual que el lenguaje Cg está basado en el lenguaje de programación C y tiene gran similitud con el lenguaje Cg de forma general. La ventaja del HLSL es la de todo lenguaje de alto nivel: facilidad para llevar a cabo operaciones mediante pocas instrucciones de fácil aprendizaje. HLSL trabaja con los *Vertex Shader*, los *Fragment Shader* y los *Geometry Shader*, aunque con estos últimos en menor medida. Debido a que fue el último procesador de *shader* que se incorporó a la GPU, además de que se

necesitan mayores capacidades de *hardware* y *software* para soportarlos. En muchas situaciones los programas de vértices, geometría y fragmentos se asocian entre sí. Se podría tener un programa de vértices que puede ser utilizado por varios programas de fragmentos diferentes (14) (12) (15).

1.5.3 Lenguaje GLSL

Lenguaje de sombreado de *OpenGL* o GLSL, por sus siglas en inglés, es uno de los lenguajes de programación para *shader*. GLSL fue desarrollado por el subgrupo de Khronos que atiende *OpenGL* llamado *OpenGL ARB Working Group*. Es un lenguaje que de igual forma que Cg y HLSL está basado en el lenguaje de programación C. Posee funciones y operadores que presenta el lenguaje C, además de contar con funciones propias. El lenguaje cuenta con una variedad de instrucciones desde sus inicios, unificando el procesado de vértices y fragmentos (16) (17).

Algunos de los beneficios de la utilización de GLSL son los siguientes:

- Compatibilidad en múltiples sistemas operativos, incluyendo GNU/Linux, Mac OS X y Windows.
- La capacidad de escribir *shaders* que pueden ser utilizados en la tarjeta de cualquier proveedor de *hardware* de gráficos compatible con GLSL.
- Cada proveedor de *hardware* incluye el compilador GLSL, permitiendo así que cada proveedor pueda crear código optimizado para la arquitectura de su tarjeta gráfica en particular.

1.6 Técnicas de gráficos computacionales 3D

En la actualidad existe una variedad de técnicas de gráficos computacionales 3D que utilizando los *shaders* pueden lograr otras representaciones de gran calidad visual. El uso de estas técnicas permite que la superficie de los objetos que aparecen en las aplicaciones sea mostrada con un mayor nivel de detalle, acercándolos cada vez más a la realidad. Entre las técnicas existentes se encuentran *Bump Mapping*, *Normal Mapping*, *Parallax Mapping*, *Relief Mapping*, *Displacement Mapping*, *Interval Mapping* y *Steep Parallax Mapping*. A continuación se explicará en qué consisten algunas de estas técnicas.

1.6.1 *Bump Mapping*

Bump Mapping consiste en proveer un aspecto irregular a la superficie de los objetos, para lograr una mejor representación de estos. En esta técnica se usa un método matemático creado por James F. Blinn

en 1978, el cual simula protuberancias y orificios en los objetos sin modificar su topología. *Bump Mapping* cuenta con un mapa de normales, que es una textura donde se guarda información para modificar las normales de los fragmentos que se procesan. Las normales originales del modelo se encuentran perpendiculares a la superficie inicialmente. Posteriormente esta técnica modifica las normales perpendiculares en función del aspecto que se quiera lograr para la superficie del objeto (18). Con el uso de esta técnica el resultado que se obtiene es considerablemente bueno, permitiendo representar variedades de formas en los elementos simulados, logrando similitud con respecto a elementos de la vida real. A continuación se presenta un ejemplo sin el uso de la técnica y otro donde se utiliza, para ilustrar los resultados que se pueden obtener (19).

Figura 1: Textura sin *Bump Mapping*.

Figura 2: Textura con *Bump Mapping*.

1.6.2 *Normal Mapping*

Otra de las técnicas es *Normal Mapping*, la cual es una versión mejorada de la técnica *Bump Mapping*. La técnica se usa para lograr irregularidades en la superficie de los objetos sin añadir polígonos al modelo. Al igual que *Bump Mapping*, esta técnica utiliza un mapa de normales, pero este mapa guarda nuevas normales y no información para modificar las normales existentes. Por lo que en este caso se cambian las normales de los fragmentos que se procesan por las normales que se almacenan en el mapa de normales (18). Mediante esta técnica se logran mejores resultados que usando la técnica *Bump Mapping*, pudiéndose representar la superficie de los objetos con mayor detalle. Algunos de los detalles que se

pueden representar en la superficie de los objetos con esta técnica son realmente impresionantes, como poros o arrugas. En la siguiente figura se puede observar la técnica *Normal Mapping*.

Figura 3: Técnica *Normal Mapping*.

1.6.3 *Parallax Mapping*

Parallax Mapping es una mejora de las dos técnicas vistas anteriormente. Con el uso de esta técnica se logran mejores resultados en la representación de la superficie de los objetos. *Parallax Mapping* para calcular las nuevas normales utiliza un mapa de normales y un mapa de altura. Esta técnica se logra implementar desplazando las coordenadas de la textura a un punto en el modelo por una función desde el ángulo de vista en el espacio de la tangente, que es un espacio local a la superficie del modelo que se analiza, y el valor del mapa de alturas en tal punto. Desde un punto de vista inclinado, las coordenadas de la textura se desplazan más y así se logra la ilusión de profundidad debido a los efectos *Parallax* mientras la vista se mueve.

En otras palabras: cuando el *hardware* gráfico está rasterizando una textura sobre un triángulo, para cada píxel rasterizado se define ¿pintar el píxel correspondiente o el más cercano al observador que lo ocluye? También se analizan varios puntos vecinos que estén más cerca del observador y que tengan una altura

mayor que este, finalmente es seleccionado el punto más alto y este será el dibujado. Los vecinos a comprobar son aquellos que siguen la línea del observador al punto y la calidad de esta técnica depende del número de vecinos explorados. A mayor número de vecinos explorados mayor será la calidad final, pero mayor será el trabajo para el *hardware* gráfico. La figura que se muestra a continuación muestra un ejemplo de la técnica *Parallax Mapping* (20).

Figura 4: Técnica *Parallax Mapping*.

1.6.4 *Displacement Mapping*

Displacement Mapping a diferencia de las técnicas vistas anteriormente, modifica la malla original del modelo. La modificación en la geometría del objeto se puede percibir cuando se observan los bordes del objeto al que se le aplicó la técnica. *Displacement Mapping* utiliza una textura para representar el desplazamiento que deben hacer los vértices. Con el desplazamiento que realizan los vértices se logran las elevaciones o grietas en las superficies de los objetos. Una forma eficiente de emplear esta técnica es haciendo uso del *Geometry Shader* porque se modifica la malla del modelo. Con esta técnica se obtienen buenos resultados en los objetos, ya que les proporcionan un verdadero relieve en su superficie. La principal desventaja de *Displacement Mapping* es que tiene un alto costo computacional, porque al desplazar los vértices se generan nuevos polígonos. En la siguiente figura se muestra un ejemplo donde se puede observar la técnica de *Displacement Mapping* (15).

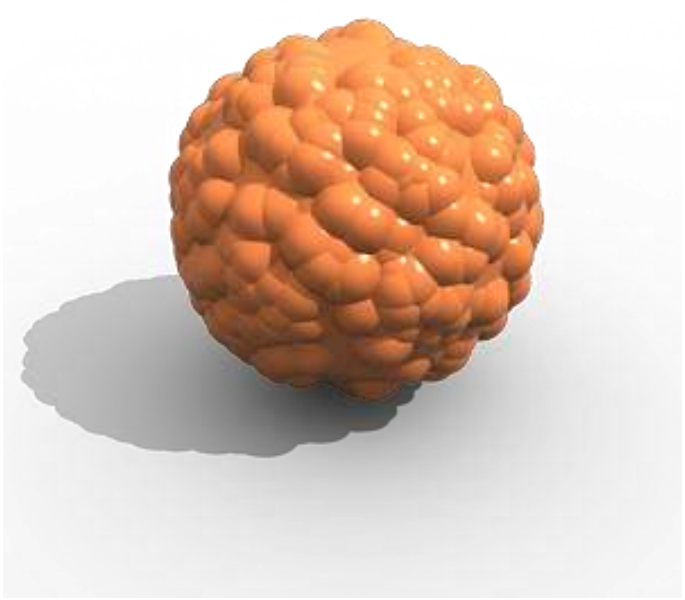


Figura 5: Técnica *Displacement Mapping*.

1.7 Renderizado de alto rango dinámico

Renderizado de alto rango dinámico o HDRR, por sus siglas en inglés, es un conjunto de técnicas o funciones que se utilizan dinámicamente para lograr mejores efectos de iluminación en escenas 3D. HDRR incrementa el rango de colores que se pueden representar, anteriormente representadas por 8 bits para cada componente de color, llegando a poder representar 32 bits por cada componente de color. Con este aumento del rango de colores se pudo cambiar el tipo de dato para representar los colores de entero a *float*. Esto permite que se mejoren considerablemente los detalles de contraste, logrando mejores resultados. El uso del HDRR permite que el ojo humano perciba un nivel real de contraste de luz y oscuridad, debido a que dispositivos como los monitores o televisores de alta definición no son capaces de hacerlo por sí solos. Existe una variedad de técnicas de las cuales el HDRR hace uso para lograr una buena cantidad de efectos. Algunas de estas técnicas son el *Tone Mapping*, *Blooming* o efecto *Bloom*, *SkyBox*, *Water Reflection* y el efecto *Refraction*. A continuación se explicará en qué consisten estas técnicas.

La técnica *Tone Mapping* se encarga de adaptar las posibilidades del HDRR a las capacidades reales de los dispositivos de salida, logrando resultados que, si bien no son los ideales, se acercan bastante a lo que sucede realmente con el ojo humano. Esta técnica es una función primordial dentro del conjunto de efectos que sigue la norma del HDRR. *Tone Mapping* efectúa el mismo proceso que realiza el iris en las transiciones de la oscuridad a la luz y viceversa, en los escenarios tridimensionales, según los cambios de perspectiva adoptados. Cuando ocurre el cambio de la oscuridad a la luz, esta se nota más intensa de lo que es en realidad. La intensidad se observa debidamente cuando pasa un tiempo y el iris del ojo hace el proceso de adaptación. Al efectuarse el cambio de la luz a la oscuridad no se notan inicialmente los detalles en el entorno sombrío. Los detalles se perciben cuando el iris vuelve a acostumbrarse a este grado de iluminación. En pocas palabras, esta técnica ajusta la visibilidad de acuerdo a la exposición de la luz ambiente en un lugar dado.

Otra técnica es el *Blooming* aunque también se puede emular sin HDRR. Esta técnica crea un efecto que es el que habitualmente se observa cuando un sector está iluminado tan intensamente que desbordan sus límites geométricos naturales. Por ejemplo, si una luz muy potente proviene de la abertura de una puerta, puede verse claramente como el espectro de iluminación excede el perímetro de la misma. Para que sea posible, se coloca un valor de brillo que supera el radio de contraste que el dispositivo proyectante como el monitor puede brindar.

Además de estas dos técnicas que son las más importantes, se encuentran otras con las que se logran otros efectos impresionantes. El *SkyBox* es un ejemplo de las técnicas, esta realiza algo similar a lo que hace la técnica *Tone Mapping* pero tomando como referencia el cielo. Esto significa que el grado de iluminación ambiental varía según el sol, ya sea al descubierto con nubes livianas volando levemente o con terribles nubarrones que lo oculten, como sucede en la realidad. Otra de las técnicas es el efecto *Refraction*, la cual se apoya en el concepto que afirma que la luz que se refracta sobre un determinado tipo de material, adopta ciertas propiedades del mismo. Este es el caso de la luz que entra por la ventana, en la cual se nota como los rayos se observan del mismo color que el vidrio que la compone. Por último, aunque existen otras técnicas, está el denominado *Water Reflection*. Como su nombre en inglés lo indica, aplica el efecto que hace la técnica *Blooming* en las reflexiones de fuentes de luz sobre el agua, mejorando notablemente el nivel de saturación del reflejo. A continuación se presenta un ejemplo sin el uso de HDRR y otro ejemplo utilizando el HDRR (21).

Figura 6: Ejemplo sin y con el uso de HDRR.

1.8 Motor de render Ogre3D

Ogre3D, por sus siglas en inglés, significa motor de renderizado gráfico orientado a objetos y está hecho principalmente para realizar aplicaciones tridimensionales. Este motor gráfico está escrito en el lenguaje de programación C++ y es multiplataforma. Es un motor de *software* libre y está licenciado bajo la Licencia Pública General Reducida o GLPL, por sus siglas en inglés. Ogre3D brinda soporte para las API *OpenGL* y *Direct3D*, además les facilita el trabajo a los desarrolladores en comparación con dichas API. Cuenta con una documentación detallada y con una comunidad activa, además presenta una interfaz entendible y fácil de usar. Una de las principales características de Ogre3D es que presenta soporte para *shader*, aprovechando las posibilidades que brindan las tarjetas gráficas para lograr mejores efectos en las aplicaciones. Además Ogre3D soporta los tres tipos de procesadores de *shaders*, *Vertex Shader*, *Geometry Shader* y *Fragment Shader* (22).

Los *shaders* en Ogre3D pueden programarse a bajo nivel utilizando el lenguaje ensamblador, o de igual forma, se pueden programar utilizando lenguajes de alto nivel como Cg, GLSL o HLSL. Permite la

manipulación de parámetros vinculados a las aplicaciones tridimensionales, como matrices e información sobre el estado de las luces. Ogre3D cuenta además con una poderosa sintaxis para la declaración de sus materiales, logrando una independencia de los materiales con respecto al código. Los materiales emplean varias técnicas para lograr diferentes efectos como asignación de texturas y texturas animadas. Ogre3D es compatible con operaciones de funciones fijas como la mezcla *multitexture*, además de la modificación y generación de coordenadas de textura.

Compositor es un *framework* que posee Ogre3D, este permite definir con facilidad efectos después del procesamiento de la escena. Al igual que los materiales cuenta con unos archivos llamados *scripts* donde se pueden definir los efectos y pueden ser reutilizados y modificados fácilmente. Con el uso del *Compositor* se pueden lograr efectos de HDRR, el efecto *Bloom*, escenas en blanco y negro, visión nocturna, nitidez y desenfoque de movimiento en los bordes de los objetos, efectos de embaldosado, visión de calor entre otros. La realización de efectos posterior al procesamiento consiste en representar la escena en una textura, ya sea como complemento o en una ventana principal. Cuando la escena se encuentra en la textura, se puede llevar la imagen a un programa de fragmentos, donde se realizan diferentes operaciones. Este *framework* es similar a un método flexible que gestiona la generación de gráficos en uno o varios pasos. Para facilitar el trabajo, el *Compositor* presenta una serie de características. Algunas de estas características son: *compositor instance* (instancia del *compositor*), *compositor chain* (cadena de *compositors*), *target* (objetivo), *output target* (objetivo de salida), *target pass* (pasada objetivo) y pass (pasada).

Una instancia de un *compositor* se aplica a una vista única y puede crear efectos basados en las definiciones del *compositor*. Con la cadena de *compositors* es posible habilitar más de una instancia del *compositor* en un *viewport* al mismo tiempo, tomando los resultados del *compositor* anterior como entrada. Cada vista que tiene al menos un *compositor* que se la ha unido, posee una cadena de *compositor*. El objetivo es el lugar donde se envía el resultado de una serie de operaciones de renderizado. Un objetivo puede ser el resultado final o puede ser una textura de renderizado intermedia. Cuando el objetivo no es de salida tiene un tamaño definido y el formato de píxel que se puede controlar. El objetivo de salida es similar al objetivo, con la diferencia que es el resultado final y único de todas las operaciones, además el tamaño y formato de píxel de este objetivo no puede ser controlado por el *compositor*. Un objetivo puede ser renderizado muchas veces cuando se usa una circunvolución entre un par de texturas. En este caso

se tendrá más de una pasada objetivo por objetivo. En el objetivo de salida se tiene una sola pasada objetivo y es la pasada objetivo de salida final. Dentro de una pasada objetivo hay una o más pasadas individuales que realizan acciones específicas como el procesamiento de la escena original.

Conclusiones parciales

En este capítulo se mostraron los principales conceptos teóricos necesarios para comprender el trabajo que se presenta, dentro de los cuales se encuentran laboratorio virtual, píxel y GPU. Se analizó el significado y las potencialidades de los *shaders*. Se expusieron los principales lenguajes de programación para *shaders* que existen actualmente y sus características. Se presentaron algunas de las técnicas de gráficos computacionales 3D y se explicó para qué se utilizan. Además se investigó sobre el renderizado de alto rango dinámico y las técnicas que utiliza. Por último, se realizó un estudio de las principales cualidades del motor de render Ogre3D.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

Introducción

En este capítulo se presentan las diferentes tecnologías y herramientas que se utilizan en el desarrollo del módulo. Se muestra el modelo de domino para entender los principales conceptos con los que se trabajan. Se presentan los requisitos funcionales y no funcionales para conocer las condiciones o capacidades con las que debe cumplir el módulo de forma general.

Este trabajo se basa en que un laboratorio virtual es el reflejo de lo real llevado hasta una computadora. Se pueden realizar distintas operaciones que se hacen en la realidad y se obtiene el equivalente a los resultados esperados. La diferencia radica en que las operaciones se realizan en la computadora y los resultados que se obtienen son mostrados de diferentes formas.

2.1 Selección del ambiente de desarrollo

Teniendo en cuenta los problemas detectados en los laboratorios virtuales relacionados con la falta de realismo en los objetos tridimensionales, debido a que se representan con poco relieve y a la carencia de iluminación en las escenas. Se propone desarrollar un módulo que permita representar los relieves en la superficie de los objetos tridimensionales, así como dotar a las escenas de una mayor iluminación. A continuación se presenta el ambiente de desarrollo que se escogió para el módulo.

- Como técnica de gráficos computacionales 3D se escogió *Normal Mapping*, ya que con esta se pueden lograr buenos resultados en la representación de los relieves en la superficie de los objetos. Además es una técnica que no es costosa computacionalmente debido a que no modifica la geometría de los objetos.

- Como lenguaje de programación para *shader* en el caso de la técnica *Normal Mapping* se escogió el lenguaje Cg. Debido a que es un lenguaje multiplataforma y compatible con las API *OpenGL* y *Direct3D*.

- Se utilizará *Bloom* para lograr los efectos de iluminación en las escenas, y de esta forma representar una mejor iluminación sobre los objetos tridimensionales que se muestran. Este efecto se emulará sin HDRR debido a que no se necesitan grandes capacidades de *hardware*.

- Solo se utilizarán los *Vertex Shader* y los *Fragment Shader* ya que con ellos se pueden lograr los efectos de representación del relieve en la superficie de los objetos y de la iluminación de la escena, sin necesidad de utilizar los *Geometry Shader*, que requieren mayores capacidades de *hardware* y de *software*.

- Como herramienta CASE se escogió Visual Paradigm debido a que es una herramienta de *software* libre y multiplataforma. Es una herramienta que soporta el ciclo de vida completo del proceso de desarrollo de *software* a través de varios tipos de diagramas. Además es fácil de instalar y actualizar.

- Se utilizará C++ como lenguaje de programación porque es un lenguaje muy potente, orientado a objetos y es multiplataforma. También es uno de los lenguajes más utilizados en el mundo actualmente para la creación de aplicaciones relacionadas con la Realidad Virtual.

- Como metodología de desarrollo de *software* se utilizará Programación Extrema (XP). El equipo de desarrollo necesita una metodología ágil como esta, para poder implementar el módulo en poco tiempo y cumplir satisfactoriamente con el plazo de entrega del producto. Es preciso utilizar XP debido a que potencia las relaciones interpersonales, beneficiando al grupo de desarrolladores al mantener una buena comunicación entre ellos. Además promueve el trabajo en equipo, necesario para aumentar la productividad y economizar el tiempo de desarrollo del módulo. Otra de las razones por la cual se seleccionó esta metodología fue porque se preocupa por el aprendizaje de los desarrolladores y es importante asegurar la superación del equipo para proyectos futuros.

- Se utilizará como entorno de desarrollo integrado o IDE, por sus siglas en inglés, Microsoft Visual Studio 2008. Posee herramientas que ayudan en el completamiento de código de sus programas y cuenta con una interfaz amigable para el desarrollador.

- Como lenguaje de modelado se escogió el lenguaje unificado de modelado, UML por sus siglas en inglés. UML es uno de los lenguajes de modelado más conocido y utilizado en el mundo que se utiliza fundamentalmente para construir y documentar un *software*. Este lenguaje representa de forma gráfica un sistema y define las características de este antes de su construcción.

2.2 Modelo de dominio

El modelo de dominio es un diagrama que muestra los objetos relacionados con el proyecto y las relaciones que hay entre ellos. Es una representación visual estática del entorno real del proyecto que se centra en una parte del negocio, la relacionada con el ámbito del proyecto. El modelo de dominio tiene como objetivo ayudar a comprender los conceptos que utilizan los usuarios, con los que trabajan y con los que deberá trabajar la aplicación. A continuación se presenta el modelo de dominio.

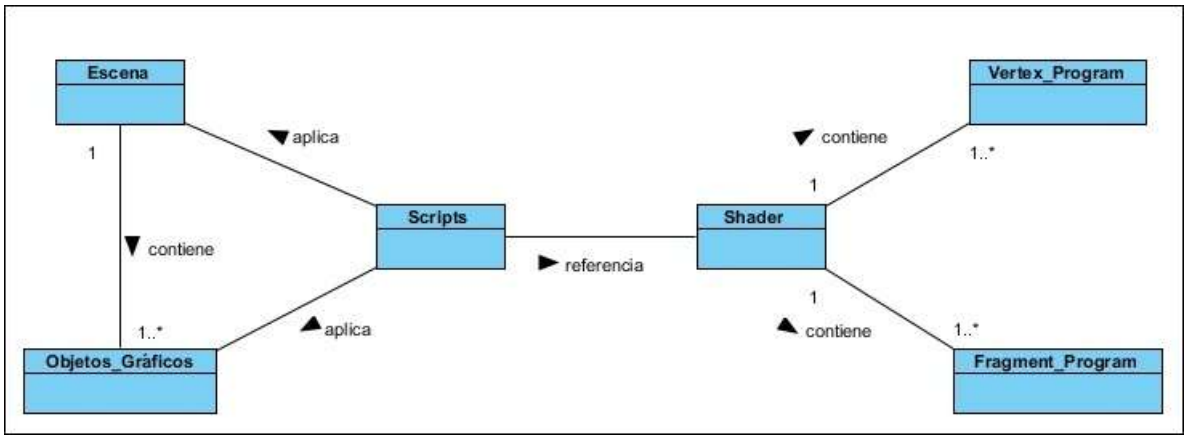


Figura 7: Modelo de dominio.

2.2.1 Descripción de las clases conceptuales

Escena: Es el lugar donde se encuentran ubicados los objetos gráficos y a la cual se le aplican varios *scripts*, para lograr una mejor iluminación de la misma.

Objetos Gráficos: Constituyen los objetos tridimensionales que se encuentran en la escena a los cuales se les aplican varios *scripts*, para lograr el relieve en sus superficies.

***Scripts*:** Constituyen los archivos que tienen referencia a los *shaders*, estos pueden ser archivos *.material* para los dos casos, la iluminación de la escena y la representación del relieve en la superficie de los objetos gráficos. Además pueden ser archivos *.compositor* para el caso de la iluminación de la escena.

***Shader*:** Constituyen los archivos donde se encuentran los *Vertex Program* y los *Fragment Program*, estos archivos pueden tener varios *Vertex Program* y *Fragment Program*.

***Vertex Program*:** Constituyen los programas de vértices que se utilizan para proporcionarle relieve a la superficie de los objetos gráficos.

***Fragment Program*:** Constituyen los programas de fragmentos que se utilizan para proporcionarle relieve a la superficie de los objetos gráficos, además se utilizan para lograr los efectos de iluminación en la escena.

2.3 Personal relacionado con el sistema

El personal relacionado con el sistema son aquellas personas que se encuentran relacionadas con los procesos que se efectúan en el sistema. A continuación se presenta el personal que se relacionará con el módulo.

Personal relacionado con el sistema	Justificación
Desarrollador	Persona encargada de interactuar con el sistema para ejecutar las funcionalidades del módulo.

Tabla 1: Personal relacionado con el sistema.

2.4 Requisitos de *software*

Los requisitos o requerimientos de *software*, como también se le conoce, se han convertido en un punto clave para el desarrollo de las aplicaciones informáticas. Un gran número de proyectos de *software* naufragan debido a una incorrecta definición, especificación o administración de requisitos. Factores tales como requisitos incompletos o mal manejo de los cambios de los requisitos llevan a proyectos completos al fracaso total.

El Glosario de Terminología Estándar de Ingeniería de *Software* (IEEE: *Standard Glossary of Software Engineering Terminology*) define al requisito como:

1. Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.

2. Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar u otro documento impuesto formalmente.

3. Una representación en forma de documento de una condición o capacidad como las expresadas en 1 o en 2.

La calidad con que se realice la captura de los requisitos tendrá influencia en todo el proceso de desarrollo del *software* repercutiendo en el resto de las fases de desarrollo. Una definición eficiente de los requisitos permite mostrar un nivel de disciplina en el proceso de desarrollo, facilitar un mejor soporte a la gestión de cambios y ganar una mayor eficiencia en las pruebas reduciendo el riesgo, mejorando la calidad y permitiendo la automatización. Además contribuye a tomar mejores decisiones de diseño y de arquitectura. También le permite al equipo de desarrollo reducir los problemas de mantenimiento (23). Los requisitos de *software* se clasifican en requisitos funcionales y requisitos no funcionales.

2.4.1 Requisitos funcionales

Los requisitos funcionales de un sistema describen lo que el sistema debe hacer. Estos requisitos dependen del tipo de *software* que se desarrolle, de los posibles usuarios del *software* y del enfoque general tomado por la organización al redactar requisitos. Además estos requisitos describen con detalle la función del sistema, sus excepciones, entradas y salidas (24). A continuación se presentan los requisitos funcionales del módulo.

RF1. Detectar capacidades del *hardware* gráfico.

RF2. Activar la técnica *Normal Mapping* a los objetos por entidad.

RF3. Activar la técnica *Normal Mapping* a los objetos por modelo.

RF4. Habilitar efecto *Bloom*.

RF5. Desactivar la técnica *Normal Mapping* a los objetos por entidad.

RF6. Desactivar la técnica *Normal Mapping* a los objetos por modelo.

RF7. Inhabilitar efecto *Bloom*.

2.4.2 Requisitos no funcionales

Los requisitos no funcionales, como su nombre sugiere, son aquellos requisitos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de este como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada, de salida y las representaciones de datos que se utilizan en las interfaces del sistema (24). A continuación se presentan los requisitos no funcionales del módulo.

- Requisitos de *Software*
  - Las computadoras deben tener instalado el sistema operativo Windows XP o cualquier sistema de Windows superior a este. También pueden tener instalado cualquier distribución de GNU/Linux.
  - Las computadoras deben tener instalados los *drivers* de video.
- Requisitos de *Hardware*
  - Las computadoras deben tener 256 MB de memoria RAM o una superior.
  - El *hardware* gráfico de las computadoras deben presentar soporte para *Vertex Shaders* y *Fragment Shaders*.
- Requisitos de restricciones en el diseño y la implementación
  - Se utilizará como lenguaje de programación C++.
  - Se empleará Ogre3D como motor gráfico.

2.5 Historias de usuario

Las historias de usuario (HU) son la técnica utilizada en XP para especificar los requisitos del *software*. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (25).

2.6 Fase de exploración

En la fase de exploración los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología (25). A continuación se presentan las historias de usuarios que se identificaron en esta fase.

Historia de Usuario	
Número: 1	**Usuario:** Desarrollador
Nombre: Detectar capacidades del *hardware* gráfico.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1	**Iteración asignada:** 1
Descripción: Detecta si el *hardware* gráfico presenta soporte para los programas de vértices y de fragmentos.	
Observaciones: Las funcionalidades del módulo se podrán ejecutar si el *hardware* gráfico presenta soporte para los programas de vértices y de fragmentos.	

Tabla 2: HU Detectar capacidades del *hardware* gráfico.

Historia de Usuario	
Número: 2	**Usuario:** Desarrollador
Nombre: Activar la técnica *Normal Mapping* a los objetos por entidad.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta

Puntos estimados: 2	**Iteración asignada:** 1
Descripción: Aplica la técnica *Normal Mapping* a todos los objetos de la escena que lo permitan. La técnica se aplicará a cada objeto para mostrar su aspecto individual.	
Observaciones: Los objetos de la escena que no se les pueda aplicar la técnica mantendrán su apariencia.	

Tabla 3: HU Activar la técnica *Normal Mapping* a los objetos por entidad.

Historia de Usuario	
Número: 3	**Usuario:** Desarrollador
Nombre: Activar la técnica *Normal Mapping* a los objetos por modelo.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1	**Iteración asignada:** 2
Descripción: Aplica la técnica *Normal Mapping* a todos los objetos de la escena que presenten un modelo que lo permita. Los objetos mostrarán el aspecto correspondiente a su modelo.	
Observaciones: Los objetos de la escena que no presenten un modelo al que se le pueda aplicar la técnica mantendrán su apariencia.	

Tabla 4: HU Activar la técnica *Normal Mapping* a los objetos por modelo.

Historia de Usuario	
Número: 4	**Usuario:** Desarrollador
Nombre: Habilitar efecto *Bloom*.	

Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 2	**Iteración asignada:** 2
Descripción: Habilita el efecto *Bloom* a la escena para lograr un mayor grado de iluminación.	
Observaciones: En caso de que se quiera observar la escena con mayor iluminación se debe habilitar el efecto *Bloom*.	

Tabla 5: HU Habilitar efecto *Bloom*.

Historia de Usuario	
Número: 5	**Usuario:** Desarrollador
Nombre: Desactivar la técnica *Normal Mapping* a los objetos por entidad.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1	**Iteración asignada:** 3
Descripción: Anula la técnica *Normal Mapping* a todos los objetos de la escena que la tengan aplicada. La técnica se anulará a cada objeto para mostrar su aspecto individual.	
Observaciones: Los objetos de la escena que no se les pueda anular la técnica mantendrán su apariencia.	

Tabla 6: HU Desactivar la técnica *Normal Mapping* a los objetos por entidad.

Historia de Usuario	
Número: 6	**Usuario:** Desarrollador
Nombre: Desactivar la técnica *Normal Mapping* a los objetos por modelo.	

Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1	**Iteración asignada:** 3
Descripción: Anula la técnica *Normal Mapping* a todos los objetos de la escena que presenten un modelo al que se le pueda anular la técnica. Los objetos mostrarán el aspecto correspondiente a su modelo.	
Observaciones: Los objetos de la escena que no presenten un modelo al que se le pueda anular la técnica mantendrán su apariencia.	

Tabla 7: HU Desactivar la técnica *Normal Mapping* a los objetos por modelo.

Historia de Usuario	
Número: 7	**Usuario:** Desarrollador
Nombre: Inhabilitar efecto *Bloom*.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 1	**Iteración asignada:** 3
Descripción: Inhabilita el efecto *Bloom* restándole iluminación a la escena.	
Observaciones: En caso de que se quiera observar la escena con menor iluminación se debe inhabilitar el efecto *Bloom*.	

Tabla 8: HU Inhabilitar efecto *Bloom*.

2.7 Fase de planificación de la entrega

En esta fase el cliente establece la prioridad de cada historia de usuario, correspondientemente los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días (25).

2.7.1 Estimación de esfuerzo por historias de usuarios

Las estimaciones de esfuerzo asociado a la implementación de las historias de usuarios la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias de usuario generalmente valen de uno a tres puntos (25). A continuación se muestra una tabla con la estimación de esfuerzo por historias de usuario.

Historias de usuarios	**Puntos de estimación**
Detectar capacidades del *hardware* gráfico.	1 semana.
Activar la técnica *Normal Mapping* a los objetos por entidad.	2 semanas.
Activar la técnica *Normal Mapping* a los objetos por modelo.	1 semana.
Habilitar efecto *Bloom*.	2 semanas.
Desactivar la técnica *Normal Mapping* a los objetos por entidad.	1 semana.
Desactivar la técnica *Normal Mapping* a los objetos por modelo.	1 semana.
Inhabilitar efecto *Bloom*.	1 semana.
Total	**9 semanas.**

Tabla 9: Estimación de esfuerzo por historias de usuario.

2.8 Fase de iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El plan de entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible debido a que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción (25).

2.8.1 Plan de iteraciones

Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración

anterior y tareas no terminadas en la iteración anterior (25). Es importante aclarar que para realizar la planificación de la etapa de implementación del sistema es necesario que los clientes hayan descrito las historias de usuario. Además los desarrolladores deben estimar el esfuerzo para dichas historias. El plan de iteraciones se encarga de organizar las historias de usuario por iteraciones y especifica las que se desarrollarán en cada iteración del proceso de implementación. Luego de que fueron descritas las historias de usuario y se estimó el esfuerzo para cada una de ellas, se decide implementar el módulo en tres iteraciones. A continuación se describe cada una de ellas.

- **Iteración 1**

Esta iteración tiene como objetivo darle cumplimiento a las historias de usuario 1 y 2, consideradas las de mayor importancia para el módulo. En la iteración se comprueba si el *hardware* gráfico soporta los programas de vértices y de fragmentos, para poder ejecutar las demás funcionalidades del módulo. Se nota un mejoramiento en el aspecto de los objetos simulados, debido a que se representa el relieve de cada objeto en su superficie. Culminada esta iteración se efectúa la primera entrega del módulo, para de esta forma mostrarle al cliente lo realizado y recibir sus valoraciones con relación a esta entrega.

- **Iteración 2**

Esta iteración tiene como objetivo darle cumplimiento a las historias de usuario 3 y 4 que realizan dos funcionalidades importantes para el módulo. En esta iteración se mejora considerablemente la iluminación de la escena. Se muestra el relieve en la superficie de los objetos, en dependencia del modelo que representen. Se rectifican los errores de la iteración anterior. Con la culminación de esta iteración se tiene el módulo con las principales funcionalidades listas y se hace la segunda entrega a los clientes.

- **Iteración 3**

Esta iteración tiene como objetivo darle cumplimiento a las historias de usuario 5, 6 y 7 que son importantes para el módulo aunque en menor grado con relación a las anteriores. En esta iteración se nota la diferencia que existe en la escena sin el mejoramiento de la iluminación y en los objetos cuando no se resalta el relieve en sus superficies. Se rectifican los errores de la iteración anterior. Culminada esta iteración se tiene el módulo listo para entregarle la primera versión completa al cliente.

2.8.2 Plan de duración de las iteraciones

El plan de duración de las iteraciones se realiza con el objetivo de expresar el tiempo que tomará cada iteración para su cumplimiento. Mediante este también se muestra el orden en que serán implementadas las historias de usuario en cada iteración. A continuación se muestra el plan de duración de las iteraciones para el módulo.

Iteración	Historias de usuarios	Duración total de las iteraciones
Iteración 1	Detectar capacidades del *hardware* gráfico.	3 semanas.
	Activar la técnica *Normal Mapping* a los objetos por entidad.	
Iteración 2	Activar la técnica *Normal Mapping* a los objetos por modelo.	3 semanas.
	Habilitar efecto *Bloom*.	
Iteración 3	Desactivar la técnica *Normal Mapping* a los objetos por entidad.	3 semanas.
	Desactivar la técnica *Normal Mapping* a los objetos por modelo.	
	Inhabilitar efecto *Bloom*.	

Tabla 10: Plan de duración de las iteraciones.

Conclusiones parciales

Durante este capítulo se seleccionó el ambiente de desarrollo para el módulo. Se escogió el *Vertex Shader* y el *Fragment Shader* como los procesadores de *shader* que se utilizarán en la implementación del módulo. Se decidió utilizar *Normal Mapping* como técnica de gráficos computacionales 3D. Para la iluminación se usará el efecto *Bloom* emulado sin HDRR. Se definió XP como metodología de desarrollo de *software*, se presentó el lenguaje de modelado y los diferentes lenguajes de programación que se eligieron para desarrollar el módulo. Se mostraron las diferentes herramientas que se emplearán durante el desarrollo del trabajo. Además se presentó el modelo de dominio y sus clases conceptuales, así como el personal relacionado con el sistema. Se expuso el concepto de requisitos de *software*, requisitos

funcionales, requisitos no funcionales e historias de usuario. Se analizó la fase de exploración, la fase de planificación de la entrega y la fase de iteraciones, pertenecientes a la metodología XP.

CAPÍTULO 3: CONSTRUCCIÓN DE LA SOLUCIÓN PROPUESTA

Introducción

Este capítulo mostrará las clases que posee el módulo mediante las tarjetas de clase, responsabilidad y colaboración. Además se presentará el estándar de codificación utilizado en la implementación de estas clases. Para lograr una mayor comprensión de las iteraciones vistas en el capítulo anterior se llevará a cabo el desarrollo de las iteraciones. Por último se expondrán las pruebas de aceptación para verificar que el módulo cumple con los requisitos del cliente.

3.1 Diseño de la solución propuesta

En la metodología XP el diseño se realiza durante todo el tiempo de vida del proyecto, siendo revisado constantemente y es muy probable que se modifique producto a cambios presentados durante el desarrollo. Las historias de usuario que se diseñan solo son las que el cliente ha seleccionado para la iteración que se esté desarrollando. Esto sucede porque se considera que no es posible tener un diseño completo del sistema y sin errores desde el principio. Además dada la naturaleza cambiante del proyecto, hacer un diseño muy extenso en las fases iniciales del proyecto para luego modificarlo, se considera una pérdida de tiempo.

3.1.1 Diagrama de clases

A continuación se muestra el diagrama de clases perteneciente al módulo, para mostrar las clases que serán utilizadas, además de sus atributos y operaciones. Con este diagrama de clases se brinda la información que manejará el módulo.

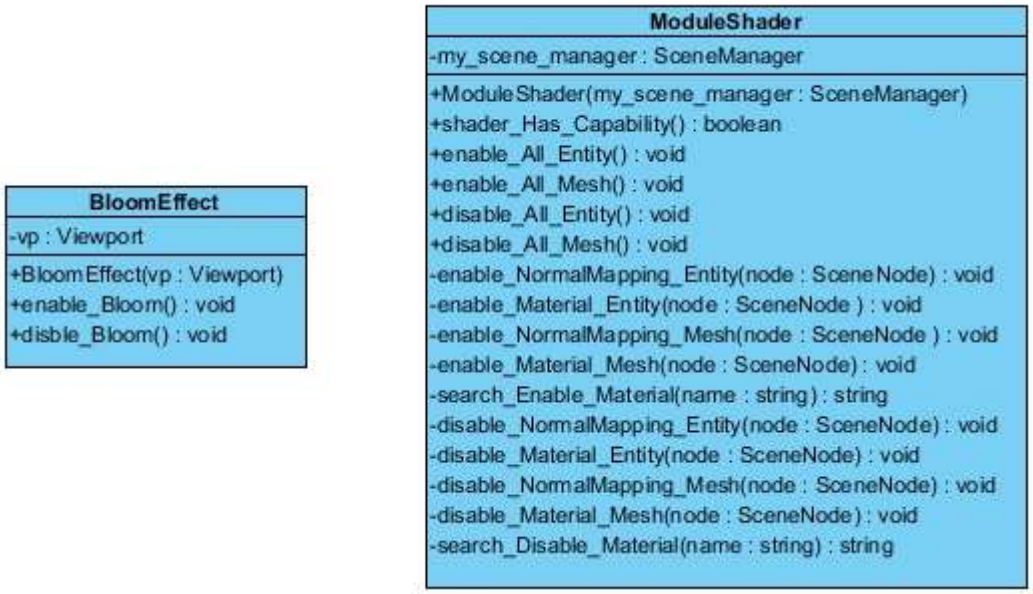


Figura 8: Diagrama de clases.

3.1.2 Tarjetas CRC

Las tarjetas CRC o tarjetas de clase, responsabilidad y colaboración son una propuesta de la metodología XP para dejar el pensamiento procedimental e incorporarse al enfoque orientado a objetos. Cada tarjeta representa una clase, y cuenta con tres secciones fundamentales. En la parte superior de la tarjeta se encuentra el nombre de la clase que representa, en la parte inferior izquierda se encuentran las responsabilidades o funcionalidades y en la parte inferior derecha los colaboradores o clases que le sirven de soporte. A continuación se presentan las tarjetas CRC que se identificaron en el módulo.

Tarjeta CRC	
Clase: ModuleShader	
Responsabilidades:	**Colaboradores:**
Detectar si el *hardware* gráfico soporta programas de vértices y de fragmentos.	

Recorrer todos los nodos de una escena.	
Recorrer todas las entidades de un nodo en específico.	
Buscar el material correspondiente a un modelo o entidad en específico para activar la técnica *Normal Mapping*.	
Buscar el material correspondiente a un modelo o entidad en específico para desactivar la técnica *Normal Mapping*.	
Conocer si la computadora posee tarjeta gráfica NVidia para aplicar la técnica *Normal Mapping* correctamente.	
Activar la técnica *Normal Mapping* a cada entidad de acuerdo a su material específico.	
Activar la técnica *Normal Mapping* a cada entidad de acuerdo al material específico de su modelo.	
Desactivar la técnica *Normal Mapping* a cada entidad de acuerdo a su material específico.	
Desactivar la técnica *Normal Mapping* a cada entidad de acuerdo al material específico de su modelo.	

Tabla 11: Tarjeta CRC ModuleShader.

Tarjeta CRC	
Clase: BloomEffect	
Responsabilidades:	**Colaboradores:**
Añade el efecto *Bloom* a un *viewport* en específico.	

Habilita el efecto *Bloom* sobre el *viewport*.	
Inhabilita el efecto *Bloom* sobre el *viewport*.	

Tabla 12: Tarjeta CRC BloomEffect.

3.2 Estándar de codificación

Los estándares de codificación constituyen pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física. Estos estándares crean un buen hábito en el equipo de desarrollo al utilizar un estilo de programación común para todos. También permiten que todos los involucrados en el proyecto entiendan el código fácilmente y en menor tiempo. A continuación se presenta el estándar de codificación utilizado en el módulo.

- **Indentación**

- La indentación será de cuatro espacios.
- No se usará la tabulación.

- **Líneas y espacios en blanco**

- Se insertará una línea en blanco antes y después de una declaración de datos que aparezca entre instrucciones ejecutables.

```
String material_name = "WO_NM/";

String resource;

material_name += name;
```

Figura 9: Líneas y espacios en blanco.

- **Declaración de variables**

- Se declarará cada variable separada por una línea.
- Se evitará nombrar a las variables con abreviatura siempre que sea posible.

- Se escribirán con todas las letras en minúsculas.
- En caso que sean varias palabras se separarán por el guión bajo.

```
Entity *entity_aux = NULL;

String name_entity_aux;
String name_mesh_aux;
String name_material_aux;
```

Figura 10: Declaración de variables.

❖ **Declaración de clases**

- Comenzarán con letra mayúscula.
- En caso que sean varias palabras se escribirán juntas.
- La primera letra de cada palabra estará en mayúscula.

```
class BloomEffect
{
public:
```

Figura 11: Declaración de clases.

❖ **Declaración de funciones**

- Comenzarán con letra minúscula.
- En caso que sean varias palabras se separarán por el guión bajo, excepto un nombre propio compuesto, en este caso los nombres se escriben juntos y con las letras iniciales en mayúscula.
- De la segunda palabra en adelante se escribirán con la letra inicial en mayúscula.

```
void enable_NormalMapping_Entity(SceneNode *node);
void enable_Material_Entity(SceneNode *node);
void enable_NormalMapping_Mesh(SceneNode *node);
void enable_Material_Mesh(SceneNode *node);
```

Figura 12: Declaración de funciones.

3.3 Desarrollo de las iteraciones

Durante la fase de iteraciones se seleccionaron y detallaron las iteraciones que tendría el módulo. Cada iteración está compuesta por diferentes historias de usuario que fueron descritas anteriormente para el cliente. El desarrollo de las iteraciones consiste en descomponer estas historias de usuario en tareas de programación o ingeniería. Las tareas de ingeniería no tienen que usar la terminología del cliente. Estas tareas describen las historias de usuario con mayor profundidad y en un lenguaje entendible para los programadores. A continuación se describen las tareas de ingeniería pertenecientes al módulo.

3.3.1 Iteración 1

En esta iteración se implementaron las historias de usuario más importantes del módulo. A continuación se presentan las tareas de ingeniería correspondientes a esta iteración.

Historias de Usuario:	**Tareas de Ingeniería:**
Detectar capacidades del *hardware* gráfico.	Detectar si el *hardware* gráfico soporta programas de vértices y de fragmentos.
Activar la técnica *Normal Mapping* a los objetos por entidad.	Recorrer los nodos de una escena.
	Recorrer las entidades que presenta un nodo.
	Buscar el material correspondiente a una entidad.

Tabla 13: Tareas de ingeniería de la iteración 1.

Tarea de Ingeniería	
Número de la tarea: 1	**Número de la HU:** 1
Nombre: Detectar si el *hardware* gráfico soporta programas de vértices y de fragmentos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 28/2/2012	**Fecha de fin:** 5/3/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Detecta si el *hardware* gráfico soporta programas de vértices y de fragmentos para ejecutar las demás funcionalidades del módulo.	

Tabla 14: Tarea de ingeniería 1.1.

Tarea de Ingeniería	
Número de la tarea: 1	**Número de la HU:** 2
Nombre: Recorrer los nodos de una escena.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.6
Fecha de inicio: 6/3/2012	**Fecha de fin:** 9/3/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Recorre todos los nodos de una escena para acceder a las entidades.	

Tabla 15: Tarea de ingeniería 2.1.

Tarea de Ingeniería	
Número de la tarea: 2	**Número de la HU:** 2
Nombre: Recorrer las entidades que presenta un nodo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha de inicio: 10/3/2012	**Fecha de fin:** 14/3/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Recorre todas las entidades adjuntas a un nodo determinado para acceder a cada una de ellas.	

Tabla 16: Tarea de ingeniería 2.2.

Tarea de Ingeniería	
Número de la tarea: 3	**Número de la HU:** 2
Nombre: Buscar el material correspondiente a una entidad.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha de inicio: 15/3/2012	**Fecha de fin:** 19/3/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Busca el material que le corresponde a una entidad determinada, el material contiene la textura y el mapa de normales propios de la entidad, además hace referencia al programa de vértices y de fragmentos.	

Tabla 17: Tarea de ingeniería 2.3.

3.3.2 Iteración 2

En esta iteración se implementaron las historias de usuario 3 y 4 del módulo. A continuación se presentan las tareas de ingeniería correspondientes a esta iteración.

Historias de Usuario:	**Tareas de Ingeniería:**
Activar la técnica *Normal Mapping* a los objetos por modelo.	Recorrer los nodos de una escena.
	Recorrer las entidades que presenta un nodo.
	Buscar el material correspondiente a un modelo.
Habilitar efecto *Bloom*.	Habilitar efecto *Bloom*.

Tabla 18: Tareas de ingeniería de la iteración 2.

Tarea de Ingeniería	
Número de la tarea: 1	**Número de la HU:** 3
Nombre: Recorrer los nodos de una escena.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.3
Fecha de inicio: 20/3/2012	**Fecha de fin:** 21/3/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Recorre todos los nodos de una escena para acceder a las entidades.	

Tabla 19: Tarea de ingeniería 3.1.

Tarea de Ingeniería	
Número de la tarea: 2	**Número de la HU:** 3
Nombre: Recorrer las entidades que presenta un nodo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.3
Fecha de inicio: 22/3/2012	**Fecha de fin:** 23/3/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Recorre todas las entidades adjuntas a un nodo determinado para acceder a cada una de ellas.	

Tabla 20: Tarea de ingeniería 3.2.

Tarea de Ingeniería	
Número de la tarea: 3	**Número de la HU:** 3
Nombre: Buscar el material correspondiente a un modelo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.4
Fecha de inicio: 24/3/2012	**Fecha de fin:** 26/3/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Busca el material que le corresponde al modelo que usa la entidad determinada, el material contiene la textura y el mapa de normales propios del modelo, además hace referencia al programa de vértices y de fragmentos.	

Tabla 21: Tarea de ingeniería 3.3.

Tarea de Ingeniería	
Número de la tarea: 1	**Número de la HU:** 4
Nombre: Habilitar efecto *Bloom*.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 27/3/2012	**Fecha de fin:** 9/4/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Habilita el efecto *Bloom* a un *viewport* determinado.	

Tabla 22: Tarea de ingeniería 4.1.

3.3.3 Iteración 3

En esta iteración se implementaron las historias de usuario 5, 6 y 7 del módulo. A continuación se presentan las tareas de ingeniería correspondientes a esta iteración.

Historias de Usuario:	**Tareas de Ingeniería:**
Desactivar la técnica *Normal Mapping* a los objetos por entidad.	Recorrer los nodos de una escena.
	Recorrer las entidades que presenta un nodo.
	Buscar el material correspondiente a una entidad.
Desactivar la técnica *Normal Mapping* a los objetos por modelo.	Recorrer los nodos de una escena.
	Recorrer las entidades que presenta un nodo.
	Buscar el material correspondiente a un modelo.
Inhabilitar efecto *Bloom*.	Inhabilitar efecto *Bloom*.

Tabla 23: Tareas de ingeniería de la iteración 3.

Tarea de Ingeniería	
Número de la tarea: 1	**Número de la HU:** 5
Nombre: Recorrer los nodos de una escena.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.3
Fecha de inicio: 10/4/2012	**Fecha de fin:** 11/4/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Recorre todos los nodos de una escena para acceder a las entidades.	

Tabla 24: Tarea de ingeniería 5.1.

Tarea de Ingeniería	
Número de la tarea: 2	**Número de la HU:** 5
Nombre: Recorrer las entidades que presenta un nodo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.3
Fecha de inicio: 12/4/2012	**Fecha de fin:** 13/4/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Recorre todas las entidades adjuntas a un nodo determinado para acceder a cada una de ellas.	

Tabla 25: Tarea de ingeniería 5.2.

Tarea de Ingeniería	
Número de la tarea: 3	**Número de la HU:** 5
Nombre: Buscar el material correspondiente a una entidad.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.4
Fecha de inicio: 14/4/2012	**Fecha de fin:** 16/4/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Busca el material que le corresponde a una entidad determinada, el material contiene la textura propia de la entidad.	

Tabla 26: Tarea de ingeniería 5.3.

Tarea de Ingeniería	
Número de la tarea: 1	**Número de la HU:** 6
Nombre: Recorrer los nodos de una escena.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.3
Fecha de inicio: 17/4/2012	**Fecha de fin:** 18/4/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Recorre todos los nodos de una escena para acceder a las entidades.	

Tabla 27: Tarea de ingeniería 6.1.

Tarea de Ingeniería	
Número de la tarea: 2	**Número de la HU:** 6
Nombre: Recorrer las entidades que presenta un nodo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.3
Fecha de inicio: 19/4/2012	**Fecha de fin:** 20/4/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Recorre todas las entidades adjuntas a un nodo determinado para acceder a cada una de ellas.	

Tabla 28: Tarea de ingeniería 6.2.

Tarea de Ingeniería	
Número de la tarea: 3	**Número de la HU:** 6
Nombre: Buscar el material correspondiente a un modelo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.4
Fecha de inicio: 21/4/2012	**Fecha de fin:** 23/4/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Busca el material que le corresponde al modelo que usa la entidad determinada, el material contiene la textura propia del modelo.	

Tabla 29: Tarea de ingeniería 6.3.

Tarea de Ingeniería	
Número de la tarea: 1	**Número de la HU:** 7
Nombre: Inhabilitar efecto *Bloom*.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha de inicio: 24/4/2012	**Fecha de fin:** 30/4/2012
Programador responsable: Juan Carlos Ayala Alonso.	
Descripción: Inhabilita el efecto *Bloom* a un *viewport* determinado.	

Tabla 30: Tarea de ingeniería 7.1.

3.4 Pruebas

La metodología XP destaca los aspectos relacionados con las pruebas, clasificándolas en diferentes tipos y funcionalidades específicas, indicando quién, cuándo y cómo deben ser implementadas y ejecutadas. En la medida que se realicen correctamente las pruebas tendrán buenos resultados otras prácticas de XP como la propiedad colectiva del código y la refactorización. Según la metodología XP se debe ser muy estricto con las pruebas. El correcto funcionamiento del producto y el nivel de aceptación de los clientes se reflejan en las pruebas que se realicen. Para liberar una nueva versión del producto es necesario que pase satisfactoriamente las pruebas existentes. En el caso contrario se utilizará el resultado de las pruebas para identificar el error y solucionarlo con mecanismos ya definidos. La metodología XP divide las pruebas en pruebas unitarias y pruebas de aceptación.

3.4.1 Pruebas unitarias

La producción del código está dirigida por las pruebas unitarias. Estas pruebas son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Uno de los elementos más importantes de estas pruebas es que deben construirse antes que los métodos, para que el programador tenga bien definido lo que va a implementar. También es necesario que el desarrollador

conozca todos los casos de prueba que deberá pasar, para poder optimizar y mejorar la calidad del trabajo. Estas pruebas deben elaborarlas los programadores utilizando un mecanismo que permita su automatización. De esta forma se logrará la implementación y ejecución de las pruebas en un menor tiempo. Con el uso de las pruebas unitarias se facilita la liberación continua de versiones ya que las pruebas son ejecutadas de forma automática y se obtiene el resultado correspondiente a la nueva versión que se está liberando.

3.4.2 Pruebas de aceptación

Las pruebas de aceptación son conocidas también como pruebas funcionales y se realizan sobre los requisitos que fueron reflejados en las historias de usuario. El cliente es el encargado de controlar estas pruebas para verificar el correcto funcionamiento del sistema. Cada historia de usuario debe pasar por una o más pruebas de aceptación en cada una de las iteraciones. Estas pruebas deben identificar los errores que posteriormente serán erradicados y establecer los casos de prueba para cada iteración. Es preciso que una historia de usuario realice satisfactoriamente todas las pruebas de aceptación destinadas a ellas para que sea aceptada.

En el desarrollo del módulo solamente se realizaron las pruebas de aceptación. Se le realizó una prueba a cada historia de usuario para comprobar su correcto funcionamiento. A continuación se presentan las pruebas de aceptación correspondientes a todas las historias de usuario.

Caso de prueba de aceptación	
Código: HU1_P1	**Historia de usuario:** 1
Nombre: Comprobar las capacidades del *hardware* gráfico.	
Descripción: Verificar si el *hardware* gráfico soporta programas de vértices y de fragmentos.	
Condiciones de ejecución: El cliente debe comprobar si el *hardware* gráfico soporta los programas de vértices y de fragmentos, para ejecutar las demás funcionalidades del módulo.	
Entrada / Pasos de ejecución: Confirmar si el *hardware* gráfico soporta los programas de vértices y de fragmentos.	

Resultado esperado: Se captura exitosamente si el *hardware* gráfico soporta o no los programas de vértices y de fragmentos.
Evaluación de la prueba: Satisfactoria.

Tabla 31: Prueba de aceptación para la HU1.

Caso de prueba de aceptación	
Código: HU2_P2	**Historia de usuario:** 2
Nombre: Comprobar la activación de la técnica *Normal Mapping* por entidad.	
Descripción: Verificar que se aplicó la técnica *Normal Mapping* a los objetos por entidad.	
Condiciones de ejecución: El cliente debe comprobar que se aplicó la técnica *Normal Mapping* correctamente a los objetos por entidad.	
Entrada / Pasos de ejecución: Confirmar visualmente que se aplicó la técnica *Normal Mapping* a los objetos por entidad.	
Resultado esperado: Se muestran los objetos con la técnica *Normal Mapping* aplicada para cada entidad.	
Evaluación de la prueba: Satisfactoria.	

Tabla 32: Prueba de aceptación para la HU2.

Caso de prueba de aceptación	
Código: HU3_P3	**Historia de usuario:** 3
Nombre: Comprobar la activación de la técnica *Normal Mapping* por modelo.	
Descripción: Verificar que se aplicó la técnica *Normal Mapping* a los objetos por modelo.	
Condiciones de ejecución: El cliente debe comprobar que se aplicó la técnica *Normal Mapping* correctamente a los objetos por modelo.	

Entrada / Pasos de ejecución: Confirmar visualmente que se aplicó la técnica *Normal Mapping* a los objetos por modelo.
Resultado esperado: Se muestran los objetos con la técnica *Normal Mapping* aplicada para cada modelo.
Evaluación de la prueba: Satisfactoria.

Tabla 33: Prueba de aceptación para la HU3.

Caso de prueba de aceptación	
Código: HU4_P4	**Historia de usuario:** 4
Nombre: Comprobar la activación del efecto *Bloom*.	
Descripción: Verificar que se aplicó el efecto *Bloom* a la escena.	
Condiciones de ejecución: El cliente debe comprobar que se aplicó el efecto *Bloom* a la escena correctamente.	
Entrada / Pasos de ejecución: Confirmar visualmente que se aplicó el efecto *Bloom* a la escena.	
Resultado esperado: Se muestra la escena con el efecto *Bloom* aplicado.	
Evaluación de la prueba: Satisfactoria.	

Tabla 34: Prueba de aceptación para la HU4.

Caso de prueba de aceptación	
Código: HU5_P5	**Historia de usuario:** 5
Nombre: Comprobar la desactivación de la técnica *Normal Mapping* por entidad.	
Descripción: Verificar que se anuló la técnica *Normal Mapping* a los objetos por entidad.	

Condiciones de ejecución: El cliente debe comprobar que se anuló la técnica *Normal Mapping* correctamente a los objetos por entidad.
Entrada / Pasos de ejecución: Confirmar visualmente que se anuló la técnica *Normal Mapping* a los objetos por entidad.
Resultado esperado: Se muestran los objetos sin la técnica *Normal Mapping* aplicada para cada entidad.
Evaluación de la prueba: Satisfactoria.

Tabla 35: Prueba de aceptación para la HU5.

Caso de prueba de aceptación	
Código: HU6_P6	**Historia de usuario:** 6
Nombre: Comprobar la desactivación de la técnica *Normal Mapping* por modelo.	
Descripción: Verificar que se anuló la técnica *Normal Mapping* a los objetos por modelo.	
Condiciones de ejecución: El cliente debe comprobar que se anuló la técnica *Normal Mapping* correctamente a los objetos por modelo.	
Entrada / Pasos de ejecución: Confirmar visualmente que se anuló la técnica *Normal Mapping* a los objetos por modelo.	
Resultado esperado: Se muestran los objetos sin la técnica *Normal Mapping* aplicada para cada modelo.	
Evaluación de la prueba: Satisfactoria.	

Tabla 36: Prueba de aceptación para la HU6.

Caso de prueba de aceptación	
Código: HU7_P7	**Historia de usuario:** 7

Nombre: Comprobar la desactivación del efecto *Bloom*.
Descripción: Verificar que se desactivó el efecto *Bloom* a la escena.
Condiciones de ejecución: El cliente debe comprobar que se desactivó el efecto *Bloom* a la escena correctamente.
Entrada / Pasos de ejecución: Confirmar visualmente que se desactivó el efecto *Bloom* a la escena.
Resultado esperado: Se muestra la escena sin el efecto *Bloom* aplicado.
Evaluación de la prueba: Satisfactoria.

Tabla 37: Prueba de aceptación para la HU7.

Conclusiones parciales

En este capítulo se muestra el diagrama de clases perteneciente al módulo. Se expone el concepto de las tarjetas CRC y se representan las clases del módulo en dichas tarjetas. Se investigó sobre el estándar de codificación, dándose su definición y se evidenció su utilización en este trabajo. Se realizó el desarrollo de las iteraciones para describir las historias de usuario con mayor detalle. Por último se analizó la importancia de las pruebas en la metodología XP. Para validar el correcto funcionamiento del módulo se aplicaron las pruebas de aceptación pertenecientes a XP.

CONCLUSIONES

Con el desarrollo de este trabajo se obtuvo un módulo para el proyecto Laboratorios Virtuales que permite resaltar los relieves de los objetos tridimensionales. Se seleccionó *Normal Mapping* como técnica de gráficos computacionales 3D a utilizar. Se logró la representación de escenas iluminadas a través del efecto *Bloom* emulado sin HDRR.

RECOMENDACIONES

Implementar la técnica *Parallax Mapping* con el objetivo de lograr una buena calidad sin tener en cuenta el rendimiento como limitante, para que pueda ser aplicada en las máquinas con alto procesamiento gráfico.

BIBLIOGRAFÍA

1. Rosado , L y Herreros, J R. *Nuevas aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la Física.* 2009.

2. Monge Nájera, Julián y Méndez Estrada, Victor Hugo. *Ventajas y desventajas de usar Laboratorios Virtuales en educación a distancia: la opinión del estudiantado en un proyecto de seis años de duración.* 2007.

3. ecuRed. [En línea] http://www.ecured.cu/index.php/Livewire.

4. UNED. [En línea] http://lab.dia.uned.es/rlab/contenido/labvirtual.html?page=3.

5. Medina, Hilda González. *EXPERIENCIAS DEL USO DE LAS TIC EN LA EDUCACIÓN QUÍMICA.*

6. Nodarse, Francisco A. Fernández. *Laboratorios virtuales en la Universidad virtual del CITMA.*

7. *Manual de Usuario: Práctica de Laboratorio Virtual "Ensamblaje de un computador".*

8. *Manual de Usuario Laboratorio Virtual: "Diseño e Instalación de una red de área local (LAN)".*

9. *Manual de Usuario Laboratorio Virtual: "Configuración y administración de una red de área local (LAN)".*

10. Píxel. [En línea] http://www.ecured.cu/index.php/Pixel.

11. GPU. [En línea] http://www.ecured.cu/index.php/GPU.

12. Shaders. [En línea] http://www.neoteo.com/pixel-shaders-y-vertex-shaders.

13. Nvidia. [En línea] [Citado el: 2 de Abril de 2012.] http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter01.html.

14. WindowsClient.net. [En línea] [Citado el: 10 de Abril de 2012.] http://windowsclient.net/wpf/wpf35/wpf-35sp1-hlsl-primer.aspx.

15. st-Laurent, Sebastien. *Shaders for game programmers and artists.*

16. OpenGL. [En línea] [Citado el: 10 de Mayo de 2012.] http://www.opengl.org/documentation/glsl/.

17. John Kessenich, Dave Baldwin, Randi Rost. *The OpenGL Shading Language.* 2011.

18. Mikkelsen, Morten. *Simulation of Wrinkled Surfaces Revisited.* 2008.

19. Nvidia. [En línea] [Citado el: 5 de Mayo de 2012.] http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter08.html.

20. ecuRed Parallax Mapping. [En línea] http://www.ecured.cu/index.php/Parallax_Mapping .

21. HDRR. [En línea] http://www.neoteo.com/high-dynamic-range-hdr.

22. Ogre. [En línea] http://www.ogre3d.org/docs/manual/manual_29.html.

23. Requisito de Software. [En línea] http://www.ecured.cu/index.php/Requisitos_de_Software.

24. Sommerville, Ian. *Ingeniería de Software 7ma edición.*

25. Penadés, Patricio Letelier y Mª Carmen. *Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).*

26. Dempski Kelly, Viale Emmanuel. *Advanced lighting and materials with shaders.*

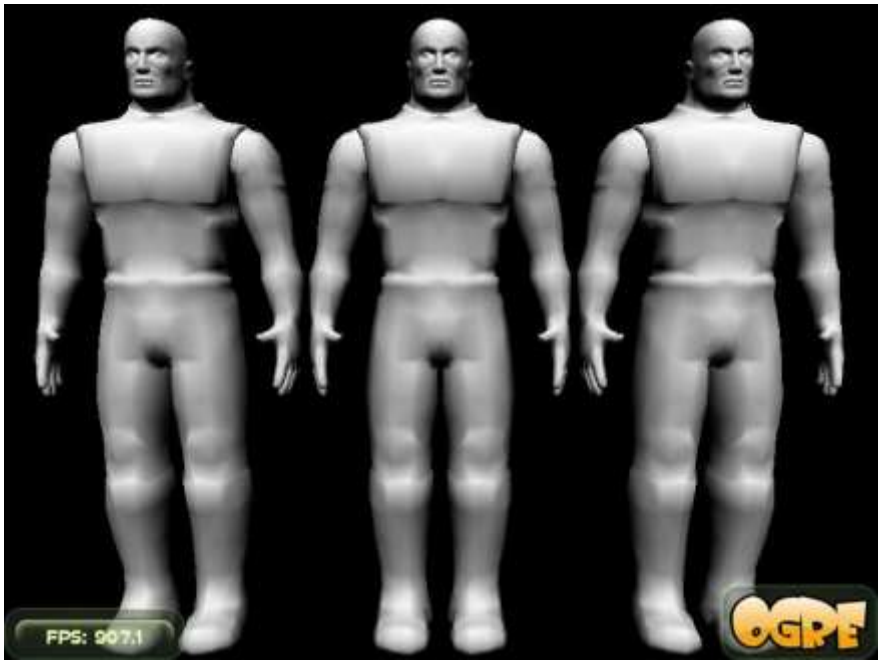
27. Calver Dean, de Boer Willem, Haines Erick. *ShaderX^3 Advanced Rendering with DirectX and OpenGL.*

ANEXOS

Anexo 1: Chasis simulado en el laboratorio virtual Ensamblaje de un Computador.

Anexo 2: Chasis real.

Anexo 3: Modelos sin material.



Anexo 4: Desactivación de la técnica *Normal Mapping* por modelo.

Anexo 5: Activación de la técnica *Normal Mapping* por modelo.

Anexo 6: Desactivación de la técnica *Normal Mapping* por entidad.

Anexo 7: Activación de la técnica *Normal Mapping* por entidad.

Anexo 8: Efecto *Bloom* habilitado a la escena.

GLOSARIO

API: Interfaces de Programación de Aplicaciones
CASE: Ingeniería de *Software* Asistida por Computación
CITMA: Ministerio de Ciencia, Tecnología y Medio Ambiente
Cg: C para Gráficos
Framework: Marco de Trabajo
GLSL: Lenguaje de Sombreado de *OpenGL*
GPU: Unidad de Procesamiento Gráfico
HDRR: Renderizado de Alto Rango Dinámico
HLSL: Lenguaje de Sombreado de Alto Nivel
HU: Historias de Usuario
IDE: Entorno de Desarrollo Integrado
IEEE: Instituto de Ingenieros Eléctricos y Electrónicos
LAN: Red de Área Local
Ogre3D: Motor de Renderizado Gráfico Orientado a Objetos
***OpenGL*:** Biblioteca Gráfica de Código Abierto
RGB: Modo de color Rojo – Verde – Azul
UML: Lenguaje Unificado de Modelado
UNED: Universidad Nacional de Educación a Distancia
XP: Programación Extrema