UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5
REALIDAD VIRTUAL

Módulo de carga dinámica para la herramienta SceneToolkit

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Alexey Broche Medina
Yordankis Alonso Cardoso

Tutor: Ing. Yanoski Rogelio Camacho Román
Co-Tutor: Ing. Katherine Gómez Cuello

Ciudad de la Habana

Mayo de 2009

DATOS DE CONTACTO

Nombre y apellidos: Yanoski Rogelio Camacho Román
Especialidad de graduación: Ing. en Informática
Categoría docente: Asistente
Categoría Científica: Ingeniero
Años de experiencia en el tema: 6 años
Años de graduado: 5
Correo electrónico: rcamacho@uci.cu

Nombre y apellidos: Katherine Gómez Cuello
Especialidad de graduación: Ing. en Ciencias Informáticas
Categoría docente: Adiestrado
Categoría Científica: Ingeniero
Años de experiencia en el tema: 3 años
Años de graduado: 2
Correo electrónico: kgomezc@uci.cu

Agradecimientos

A Yenifer, por su apoyo en todo momento.

A mi hermana, sencillamente mi ejemplo a seguir.

A mis padres, por todo el amor que me han brindado.

A mis amigos, por hacerme más fácil estos años de estudio.

A todos los que de una manera u otra han hecho posible este sueño.

Alexey

A Yeilín, por su apoyo incondicional.

A mis padres, por todo el amor brindado.

A mi tío José, por su confianza en mí y por todo el apoyo brindado.

A mis amigos, por hacerme más fácil estos años de estudio.

A todos los que de una manera u otra han ayudado en mi formación personal y profesional.

Yordankis

Dedicatoria

A Tata, César y Óscar.
A Eremia, mi gorda linda.
A Dionisio, el mejor de los padres.
Alexey

A mi tío José
A Yeilín, el amor de mi vida.
A mi hermano mayor, José Monnar.
A mis padres, los mejores del mundo.
A mis abuelos, aunque no estén presentes.
A toda mi familia.
Yordankis

Resumen

La Realidad Virtual (RV) se puede considerar como la percepción por parte del usuario de una realidad no existente, ilusoria, por medio de *hardware* y *software*, permitiéndole al mismo examinar un mundo o entorno tridimensional generado por computador.

El objetivo del presente trabajo de diploma es desarrollar un módulo para la carga y descarga de grandes entornos virtuales basado en la programación paralela (hilos), que permita mantener en memoria gran cantidad de datos.

En este trabajo de diploma se abordan los conceptos y las técnicas más empleadas en la manipulación de hilos y el trabajo con grandes entornos virtuales tridimensionales (3D) en tiempo real.

Como resultado de esta tesis de grado se obtuvo un módulo con una serie de funcionalidades que facilitan la construcción de Sistemas de Realidad Virtual (SRV) y la carga de grandes entornos virtuales.

Dicho módulo presenta las condiciones idóneas para acoplarse a la herramienta *SceneToolkit* (STK).

Palabras claves

Realidad Virtual, ambiente 3D, programación multihilo, hilos, carga dinámica, textura, motores gráficos.

Tabla de contenido

Introducción

Los Sistemas de Realidad Virtual (SRV) son representaciones no reales de situaciones que suceden a diario y que se apoyan en el uso de *hardware* especializado, dando la sensación de estar en una situación real en la que se puede interactuar con el ambiente circundante. Estos sistemas presentan una serie de características como la inmersión, la navegación y la manipulación que los hacen diferentes de otros sistemas informáticos.

En la actualidad la Realidad Virtual (RV) ha alcanzado un elevado nivel. Se pueden encontrar aplicaciones para la defensa, la medicina, la educación, la arquitectura, el entretenimiento y muchas más, que permiten interactuar con mundos tridimensionales de una manera más natural, realizar acciones dentro de un modelo virtual y de esta forma experimentar situaciones que se asemejan al mundo real.

El mundo de la simulación por computadora ha adquirido gran importancia a lo largo de los últimos años, pudiéndose encontrar generadores tanto de simuladores como de videojuegos. Estos generadores, llamados motores gráficos, incluyen funcionalidades básicas que hacen más viable y factible el trabajo a los programadores.

En una aplicación de RV, la capacidad de un visualizador no sólo viene determinada por el *software*, sino que también presenta un límite impuesto por el *hardware*. Este límite es fijado principalmente por dos factores: la memoria presente en la tarjeta gráfica y la memoria RAM del ordenador. (1)

Cuba está trabajando aceleradamente en el desarrollo de la RV por sus grandes aportes a la economía y a la defensa de la Isla. En la Universidad de las Ciencias Informáticas, específicamente en la Facultad 5, el grupo de trabajo Herramientas de Desarrollo de Sistemas de Realidad Virtual (HDSRV) está desarrollando la herramienta *SceneToolkit* (STK). La misma es capaz de manipular escenas tridimensionales en tiempo real, pero **necesita** optimizar sus funciones para una mejor utilización de los recursos de *hardware* (memoria RAM y de video) que permita mantener en memoria gran cantidad de datos y así poder mostrar grandes entornos virtuales.

El presente trabajo viene dado por el siguiente **problema científico**: ¿Cómo mostrar grandes entornos virtuales en las aplicaciones de realidad virtual de la Facultad 5?

Constituye **objeto de estudio** de este tema los motores gráficos para la realización de aplicaciones de realidad virtual, siendo el **campo de acción** la carga dinámica de entornos virtuales en los motores gráficos.

El **objetivo** de este trabajo es desarrollar un módulo que permita cargar y destruir dinámicamente entornos virtuales de grandes dimensiones.

Para lograr el total cumplimiento del objetivo se trazan las siguientes **tareas investigativas:**

- ✓ Propuesta de soluciones técnicas según estado del arte.
- ✓ Diseño e implementación de funcionalidades básicas para ejecución paralela (por hilos) en la *SceneToolkit*.
- ✓ Diseño e implementación del módulo de carga dinámica basado en la ejecución paralela.
- ✓ Acople de la solución a la *SceneToolkit*.

Con este módulo se pretende brindar funcionalidades a la STK para cargar y destruir grandes entornos virtuales, a través de la sectorización de los mismos, de manera que se mantengan en memoria solamente las áreas cercanas, y se carguen y destruyan otras áreas en dependencia del movimiento del objeto dinámico. Con esta solución se podrán mostrar grandes entornos virtuales, lo que repercutirá en un mayor realismo en las aplicaciones de realidad virtual de la Facultad 5.

Este trabajo de diploma consta de cuatro capítulos y está estructurado de la siguiente manera:

En un primer capítulo, **Fundamentación teórica**, se abordan todos los conceptos relacionados con el tema a tratar. Se hace un análisis bibliográfico donde se investigan las características de la carga dinámica y de los hilos, tendencias actuales, cómo se realiza la programación paralela, los tipos de carga dinámica existentes y las técnicas necesarias para llevarlas a escenarios 3D. En el segundo capítulo,

Soluciones técnicas, se exponen las características que presentará el sistema como solución al problema planteado. En el tercero, **Descripción de la solución propuesta**, se analiza con mayor profundidad el objeto de estudio, se crea el modelo del dominio, se hace la captura de requisitos, se definen los modelos de casos de uso del sistema y se hace una descripción textual de cada uno de ellos. En el cuarto capítulo, **Diseño e implementación del sistema**, se presentan los diagramas de clases de diseño así como una descripción de ellas, los diagramas de secuencia y el diagrama de componentes.

Capítulo 1 Fundamentación teórica

Introducción

En este capítulo se proporcionará una breve explicación sobre los motores gráficos ya que el módulo de carga dinámica se realizará con el objetivo de incluirlo en la herramienta *SceneToolkit*, que es un motor gráfico que se está desarrollando en la Universidad de las Ciencias Informáticas, y se necesitan conocer las peculiaridades de este tipo de herramienta a partir de los principales motores gráficos existentes en el mercado hoy en día, así como las características de la *SceneToolkit*.

Luego se expone qué es la carga dinámica, los tipos de carga dinámica que existen y lo referente a todo el trabajo con hilos. Finalmente se analizan las diferentes formas de representar y almacenar información de un entorno virtual.

1.1 Motores gráficos

El término “motor gráfico” (*graphic engine*) es ampliamente utilizado en todo el mundo principalmente por programadores de videojuegos y no es más que una parte de un proyecto de *software* que se encarga de propulsar ciertas funcionalidades en el *software*.

Actualmente en el mercado se pueden encontrar un gran número de aplicaciones que permiten visualizar mundos virtuales incluyendo lógicamente gráficos tridimensionales. Algunas de ellas se centran únicamente en el campo de la representación gráfica, sin permitir la introducción de lógica, mientras que otras permiten su introducción por medio de complejas interfaces, siendo el control sobre los elementos limitado. (1)

Básicamente un motor gráfico es un conjunto de clases o bibliotecas que proporcionan funciones de renderizado 2D, 3D, se suele apoyar en librerías/APIs gráficas como *OpenGL* o *DirectX* y definen una capa de abstracción entre el *hardware*, el sistema operativo y el videojuego.

El trabajo de un motor gráfico es el de realizar todas las tareas de bajo nivel tales como comunicarse con el adaptador gráfico, administrar la escena, transformar la geometría del mundo 3D y lidiar con diversos procesos matemáticos que afectan su comportamiento. Todo este trabajo es necesario, pero es algo con lo cual el programador inexperto no desea lidiar al desarrollar una aplicación 3D y es entonces cuando se recurre a un motor gráfico en el cual ya se encuentran implementadas las funcionalidades requeridas.

Los motores gráficos incluyen editores y más recientemente editores en tiempo real, estos nos permiten posicionar objetos, crear terrenos, vegetación, incluso sonido y luces y ver en tiempo real (tal como va sucediendo la acción) cómo se modifica el juego o aplicación.

Existen motores 3D libres para uso no comercial, motores gratis para uso comercial, motores 3D comerciales, que desafortunadamente son los mas rápidos para desarrollar.

1.1.1 Principales motores gráficos existentes en el mercado

A continuación se hará una revisión de los motores gráficos más empleados y que están presentes en el mercado, algunas de sus características y en qué basan su dibujado.

Ogre (Object-Oriented Graphics Rendering Engine)

Es un motor gráfico desarrollado en C++, multiplataforma, libre y diseñado para hacer más fácil e intuitiva la creación de juegos y aplicaciones de realidad virtual, haciendo uso de las capacidades 3D presentes dentro del *hardware*. Presenta una biblioteca de clases basada en *Direct3D* y *OpenGL* que contiene un conjunto de clases que permiten la generación de mundos virtuales. Permite que las escenas se dividan en "páginas" (pedazos de un mapa muy grande se procesan uno a la vez, permitiendo cargar escenas muy grandes). Cada página puede tener sus propios *HeightMaps*. Su renderizado está basado en Octree, BSP (*Binary Space Partition*), y portales. (2)

Crystal Space

Es una biblioteca de clases multiplataforma de desarrollo de software para gráficos 3D en tiempo real, con especial atención a los juegos. Escrita en C++ y basada en *OpenGL*. Su dibujado lo realiza mediante portales-sectores. (3)

OpenSceneGraph

Es una herramienta de alto rendimiento empleada para el desarrollo de aplicaciones encaminadas a la simulación gráfica, juegos, realidad virtual, modelado, y visualización científica. Está desarrollada completamente en el estándar C++ y *OpenGL* lo que permite su utilización en diferentes sistemas operativos. El proceso de carga y descarga de la escena se realiza mediante una base de datos que pagina las diferentes áreas de la escena y que permite decidir que parte o partes de la escena es necesario tener en cuenta en cada momento. (4)

Fly 3D

Está basado en renderizado por árboles BSP, PVS (*Potentially Visible Sets*) y portales. Entre sus características se destaca la escalabilidad proporcionada por su sistema de *plugins*. La plataforma que soporta es *Windows*. Licencia libre sin coste y acceso a todo el código fuente. (5)

Unreal

Motor gráfico del juego *Unreal* desarrollado por la empresa *Epic MegaGames* basado en una extensión del renderizado en portales conocido como DSG (*Dynamic Scene Graph Technology*), BSP y radiosidad. Su principal característica es la gran escalabilidad (muy modular). Se encuentra disponible para las plataformas *GNU/Linux*, *Windows*, *Macintosh*, *PlayStation 2* y *Xbox*. Para desarrollar con este motor se tiene que adquirir una licencia (USD 250 000 - USD 500 000) que da acceso a todo el código fuente, a las herramientas y juegos. (5)

Quake 2

Es el motor gráfico del videojuego *Quake 2*, desarrollado por John Carmak de *Id Software*. Se basa en el renderizado por árboles BSP y radiosidad. Soporta plataformas *Windows*, *GNU/Linux* y *Macintosh*. Su Licencia oscila entorno a los USD 250 000 que da acceso al código fuente. (5)

Genesis3D

Basado en renderizado por portales, BSP y radiosidad. La única plataforma que soporta es *Windows*. Bajo licencia libre y derecho a modificar el código fuente, a cambio de tener que mostrar el logotipo del motor en las aplicaciones que lo utilicen o bien bajo licencia comercial de costo USD 10 000 por título, para que no salga el logotipo de génesis 3D y código abierto. (5)

Torque (V12)

Motor gráfico utilizado en el juego *Tribes 2* de *Dinamix* basado en renderizado en portales. Desarrollado para las plataformas, tanto el cliente como el servidor, *Windows*, *Mac OS 9/X* y en preparación *GNU/Linux*. Es necesaria licencia (USD 100) que da acceso al código fuente. (5)

WorldToolKit (WTK)

Es un entorno de desarrollo multiplataforma avanzado, para aplicaciones gráficas 3D en tiempo real con un alto rendimiento, que posee una librería de más de 1000 funciones y herramientas programadas en C para usuarios finales, que les servirán para crear y manejar sus productos. WTK aprovecha al máximo las características disponibles de *OpenGL*. La escena en el WTK se estructura y maneja a través de una arquitectura jerárquica conocida como grafo de la escena (*Scene Graph Architecture*). De esta manera, se pueden ensamblar nodos en un árbol jerárquico donde cada uno representa una parte del simulador, indicando como la simulación será interpretada a la hora de dibujar el mundo virtual, lo que le da mucha eficiencia al simulador. (6)

3D-GameStudio

Es un *kit* de desarrollo para la realización de juegos de ordenador. Provee de un motor 3D, un motor 2D, un editor de niveles y modelos, compilador de *scripts* y librerías de modelos, texturas, etc. Manipula con igual rendimiento escenas de interior y de exterior. Visibilidad por árboles BSP, portales y PVS. El principal objetivo que esta aplicación persigue es que el creador del juego no necesite ser un programador experimentado. Ofrecen tres posibilidades para crear un juego: juegos diseñados a base únicamente de ratón, juegos o efectos diseñados con algo de programación utilizando *C-scripts* y juegos o efectos programados en C++ o *Delphi*. (6)

SimpEngine

Es un esqueleto o armazón orientado a objeto y basado en *OpenGL*, que se puede utilizar para construir juegos. Debido a su simplicidad, no significa que se pueda usar para desarrollar juegos grandes, pero sí para simples juegos, rápida y eficientemente. En un ciclo simple de juego, el subsistema responde a las entradas, ejecuta los cálculos físicos necesarios con los objetos del juego, manipula las colisiones, carga y destruye objetos, mueve la cámara por la escena, y ejecuta cualquier sonido que se necesite durante el juego. (6)

1.1.2 Motor gráfico *SceneToolkit* (STK)

Cuba ha tenido un crecimiento acelerado en el desarrollo de las técnicas de RV. Jugando un papel fundamental y formando parte de una de sus líneas de investigación, en la Universidad de las Ciencias Informáticas se desarrolla la herramienta básica para el desarrollo de SRV, *SceneToolkit* (STK), a cargo de un equipo de estudiantes y profesores de la Facultad 5.

La misma surge por la necesidad de concebir herramientas de trabajo propias, que permitan desarrollar productos con rapidez y calidad, a las que se les puedan incluir fácilmente nuevos módulos y funcionalidades.

Actualmente, la STK cuenta con una amplia documentación así como una serie de aplicaciones que demuestran las funcionalidades alcanzadas hasta el momento. Entre sus características principales se encuentran que es multiplataforma, brinda soporte de render para *DirectX* y *OpenGL*, optimización de la visualización, animaciones realistas y comportamientos físicos a través de controladores, efectos visuales, componentes visuales a través de imágenes, entre otras.

Esta herramienta basa su dibujado en una arquitectura jerárquica conocida como grafo de la escena (*Scene Graph Architecture*). Esta estructura permite el agrupamiento de objetos por tipos o por estado, permite la instanciación de geometrías y de sub-árboles enteros del diagrama de la escena, brindando un mejor uso de la memoria.

1.2 Carga dinámica

Hoy en día los escenarios de una aplicación de realidad virtual son generados por ordenador, típicamente a la frecuencia de 60Hz. La generación de estas imágenes, por ejemplo para un simulador de vuelo, en apenas 0,02 milisegundos, con una calidad visual realista presenta ciertos problemas que surgen por la limitación de los sistemas informáticos en cuanto a capacidad de representación y almacenamiento de la información (polígonos y texturas). (7)

Cuando la memoria ocupada por las texturas asociadas a un entorno es superior a la memoria de la tarjeta gráfica, se produce una caída de rendimiento de una gran magnitud. Lo mismo ocurre con la memoria ocupada por las geometrías. Cuando su tamaño supera la memoria RAM disponible, se produce una disminución del tiempo asociado al redibujado, debido a las continuas transferencias de memoria entre la memoria virtual del disco duro y la memoria principal del ordenador. (1)

Por lo tanto se debe recurrir al empleo de técnicas más sofisticadas como la carga dinámica. Gracias a la carga dinámica se pueden reducir el número de texturas y geometrías almacenadas en memoria a aquellas que sean necesarias en cada instante, teniendo en cuenta que la carga o descarga de geometrías y texturas es en función de la posición que ocupa el objeto dinámico. De esta manera se pueden cargar escenarios con mayor número de polígonos y texturas en tiempo real. (1)

A continuación se detallan las técnicas de carga dinámica estudiadas con sus ventajas e inconvenientes.

1.2.1 Técnicas de carga dinámica

Carga dinámica de texturas

Uno de los aspectos importantes a la hora de simular terrenos con un nivel aceptable de realismo es la aplicación de texturas, que en caso de pretender una buena calidad, tendrán un tamaño considerable.

Se mantiene en memoria solo aquellas texturas que sean necesarias. La carga dinámica de texturas se empleará en aquellos escenarios en los cuales la memoria ocupada por las geometrías no supere la memoria RAM disponible del ordenador. Se trata de una carga de gran rapidez. Pero en entornos de grandes dimensiones, las texturas utilizadas suelen superar la memoria de la tarjeta de video si no se hace un correcto dimensionado de las mismas. En este tipo de carga la presencia de tirones es menor pero no puede ser utilizada en entornos cuya memoria asociada a las geometrías supere la memoria RAM del ordenador. (1)

Carga dinámica de geometrías y texturas

La limitación de los polígonos es diferente al de las texturas, la cantidad de polígonos a procesar no está afectada por la capacidad de almacenamiento de la tarjeta gráfica. El problema de los polígonos es un problema de transferencia y procesamiento. Se mantiene en memoria solo las texturas y geometrías que sean necesarias para representar el fotograma. Se empleará en los escenarios donde la memoria ocupada por las geometrías sea muy elevada. Es una carga de mayor lentitud pero permite la representación de escenarios muchos más grandes. A cambio trae consigo la presencia de tirones y es ideal para simulaciones en ordenadores de pequeña capacidad, dentro de las cuales se pueda permitir bajadas esporádicas del número de fotogramas por segundo. (1)

El mayor problema de la carga dinámica surge en aquellos instantes en los cuales se produce la carga y descarga de la información. Por una parte se debe representar la imagen al menos a 30 fotogramas por segundo y por otra se debe guardar en memoria aquellas geometrías y texturas que pueden ser representadas en instantes sucesivos.

Uno de los puntos más importantes es la decisión de qué celdas son las que se quiere tener en memoria en cada momento. Esta decisión no tiene una solución única ni óptima y depende sobre todo del tipo de aplicación que se esté construyendo, de la velocidad del disco y del tamaño de la memoria RAM disponible. Siempre se busca minimizar el número de accesos a disco, además del tiempo de CPU que consumirá la elección de las celdas a cargar, porque como es sabido, el acceso continuo a disco puede suponer una pérdida de rendimiento considerable en la aplicación. (7)

Se debe crear un hilo dedicado a paginar el terreno y las texturas, ya que si se produjera dentro del hilo principal, la aplicación iría a saltos. La posible presencia de tirones es debida principalmente a que la carga de una geometría compleja puede tardar más de 1/33 milisegundos, tiempo máximo de transición entre dos fotogramas, por lo que se debe recurrir a la creación de un hilo secundario que se encargue exclusivamente de la carga de texturas y geometrías. El hilo principal, además de representar la escena, debe controlar el tiempo máximo por fotograma dedicado a la carga. (1)

1.3 Programación multihilo

Cuando se habla de sistemas operativos multitarea se dice que pueden ejecutar varias tareas simultáneamente de forma concurrente. Esto no es del todo cierto, tan solo los sistemas multiprocesador (con más de un procesador) pueden realizar multitarea real, en un sistema con un solo procesador se habla de sensación de multitarea.

En ambos casos interviene un mecanismo del sistema operativo denominado planificador. El planificador del sistema operativo es el que se encarga de administrar las distintas tareas que están en ejecución y decidir cuando ejecuta la tarea y (si es aplicable) en que procesador lo hace. Cuando una tarea está en ejecución tiene disponible la totalidad del procesador, no lo comparte con ninguna otra, lo mismo ocurre con el espacio de memoria, para la tarea que está ejecutando dispone de toda la memoria disponible del sistema. (8)

Para que haya realmente multitarea o por lo menos sensación de ella, las tareas deben compartir el procesador, es decir, eventualmente la tarea que está en ejecución en un determinado momento deberá retirarse y dejar paso a otra tarea que ocupará su lugar, esto se denomina cambio de contexto y, entre otras cosas implica una cierta sobrecarga de recursos (hay que guardar ciertos datos relativos al proceso que sale y cargar otros tantos relativos al proceso que entra). (8)

En la naturaleza se presentan innumerables ejemplos de procesos que se realizan de manera paralela, simultánea o concurrente. En el plano de los sistemas artificiales, la concurrencia es un denominador común en los juegos, en los sistemas sofisticados de los automóviles modernos, las actuales comunicaciones inalámbricas, etc. Por tales razones, la concurrencia es un fenómeno importante a tratar ya que permite el desarrollo de sistemas capaces de adelantar múltiples procesos simultáneamente. (8)

Se puede definir hilo (*thread*) como una secuencia única de control de flujo dentro de un programa. Entendiendo que puede haber más de una secuencia de control o hilos. En un entorno de multitarea basada en hilos, el hilo es la unidad de código más pequeña que se puede seleccionar para ejecución.

Cuando se habla de programación multihilo se está haciendo referencia a que, dentro de la propia aplicación haya diversas tareas ejecutando, es decir, el proceso reparte de alguna forma el trabajo entre distintas tareas hijas.

En este sentido hay que resaltar que para el sistema operativo sigue siendo un solo proceso (aunque este sabe que hay varias tareas hijas), de forma que el tiempo asignado por el sistema operativo no varía y por otro lado tan solo una de las tareas hijas del proceso se estará ejecutando en un momento dado. La diferencia radica en que, cuando un proceso deja de ejecutar y entra a ejecutar otro proceso se produce un cambio de contexto (se guarda el bloque de control de proceso (BCP) del proceso saliente, se carga el BCP del proceso entrante y el proceso entrante comienza a ejecutar) con el consiguiente desalojo de información del proceso, sin embargo cuando se produce el mismo caso entre tareas hijas dicho cambio de contexto no tiene lugar. Esto es debido a que dichas tareas comparten el mismo espacio de memoria, el del proceso padre. (8)

La programación multihilo entonces consiste en programas que contienen dos o más partes que se ejecutan de manera concurrente, aunque solo se tenga una Unidad Central de Proceso (CPU). La motivación inicial que dio lugar a este tipo de programación es la solución de problemas que conllevan la necesidad de estar ejecutando simultáneamente dos tareas al mismo tiempo por el mismo programa. La programación multihilo es una herramienta poderosa y peligrosa. En máquinas monoprocesador, mediante su uso se consigue un mayor rendimiento efectivo pero un menor rendimiento computacional. Esto básicamente quiere decir que el sistema está ocioso una menor parte del tiempo, pero el número de instrucciones útiles ejecutadas por unidad de tiempo desciende. Esto se debe a que al propio proceso de los hilos hay que sumarle el tiempo de conmutación entre ellos.

1.3.1 Características generales de la programación multihilo

Existen varias motivaciones para la programación concurrente: reducir el tiempo de ejecución, incrementar la tolerancia a fallas y explotar el paralelismo explícito e inherente de ciertas aplicaciones, entre otras.

La programación basada en multihilo permite acceder a los recursos de tiempo libre del CPU mientras se realizan otras tareas. Además permite escribir programas más eficientes, ya que optimizan los recursos del CPU, al reducir al mínimo los tiempos de inactividad. Este es un factor muy importante a tener en cuenta en el manejo de entornos interactivos, las animaciones y los juegos, por ejemplo, que requieren de una mayor optimización de los recursos de CPU.

Los puntos fuertes de la programación multihilo se pueden dividir en cuatro categorías principales:

- ✓ La capacidad de respuesta: el uso de múltiples hilos en una misma aplicación interactiva permite que un programa continúe con su ejecución incluso cuando una parte esté bloqueada o realizando una operación costosa, lo que incrementa la capacidad de respuesta ante el usuario.

- ✓ Compartición de los recursos: por omisión, los hilos comparten la memoria y los recursos del proceso al que pertenecen. La ventaja de compartir el código y los datos radica en que una aplicación tenga varios hilos de actividad distintos dentro del mismo espacio de direcciones.

- ✓ Economía: la asignación de recursos para la creación de los procesos es más bien costosa. Dado que los hilos comparten los recursos del proceso al que pertenecen, es más fácil crear y realizar cambios de contexto entre uso y otros hilos. Generalmente, se consume mucho más tiempo en la creación y gestión de los procesos que de los hilos.

- ✓ El uso en arquitecturas multiprocesador: las ventajas de usar el modelo multihilo pueden verse muy incrementadas en una arquitectura multiprocesador, donde los hilos pueden ser ejecutados de forma totalmente paralela en los diferentes procesadores. Un proceso con un único hilo solo puede ejecutarse en un procesador, independientemente de los que se tengan disponibles.

Decidir cuándo se debe orientar una aplicación hacia el multihilo no es una decisión sencilla. En muchos casos la aplicación es lo suficientemente simple como para que no sea necesario siquiera planteárselo, en otros casos en que la aplicación es lo suficientemente compleja quizá la sobrecarga o la complejidad introducida por el paso a multihilo anule las posibles ventajas que pudiera generar.

En general la decisión de usar varios hilos debe tomarse cuando existan tareas bien definidas con un coste computacional alto en las que buena parte de su ejecución no dependa del resultado del resto de las tareas (puesto que no tendría sentido programar tres tareas que van a estar continuamente esperándose las unas a las otras) y que además se prevea que puedan ser retenidas en ciertos momentos de su ejecución. (8)

Las razones para estos tres aspectos, radican en lo explicado anteriormente sobre el modelo de ejecución del sistema operativo. (8)

- ✓ Si el costo computacional de una tarea no es alto, introducir cambios de contexto hace que desaparezca cualquier mejora que se pueda conseguir gracias a las técnicas de programación con multihilo.

- ✓ Si una tarea depende de otra en gran medida (por ejemplo necesita el resultado que proporciona otra tarea para empezar) entonces no se gana nada ya que si la tarea B necesita que la tarea A termine para poder empezar la concurrencia no se producirá, acabará la tarea A y a continuación empezará la tarea B (que es lo mismo, de hecho es peor, que si la tarea A acabara la primera fase y ella misma comenzara la segunda).

1.3.2 Principales bibliotecas de clase para el trabajo con hilos

Muchas son las bibliotecas de clase que se emplean para poder efectuar la programación paralela. A continuación se describirán las más conocidas.

Simple DirectMedia Layer (SDL)

Provee una *Application Programming Interface* (API) de hilos que luce como una versión simplificada de *pthreads*. SDL provee toda la funcionalidad básica que un programador necesita en un paquete de hilos, mientras que esconde los detalles de bajo nivel que confunden a muchas personas y hacen que el desarrollo de código con hilos sea tan difícil. Además, la SDL proporciona funciones para manejar diferentes hilos de manera portable (a excepción de *Mac OS* anterior a *Mac OS X*). El aprovechamiento de estas funciones puede distar mucho del que se podría hacer en un sistema operativo concreto, pero aún así puede ser muy útil. Permite el uso de semáforos, *mutex*, condiciones y *timers*. (9)

GLib

Oculta las implementaciones específicas de cada plataforma y provee una API que puede ser accedida de la misma forma desde cualquier sistema operativo. No obstante, internamente se usarán las llamadas nativas disponibles en la plataforma donde se encuentre la aplicación. Es decir, *GLib* abstrae la implementación de los hilos. (10)

Boost

Proporciona un conjunto portátil de primitivas que les viene muy bien a los programadores de C++. Es multiplataforma. Permite ejecutar programas múltiples, asíncronos, independientes de hilos de ejecución. Cada hilo tiene su propia máquina de estado incluyendo el contador de instrucción del programa y los registros. (11)

Pthreads-w32

Normalmente se implementa como una biblioteca de enlace dinámico (DLL). Esto tiene algunas ventajas desde el punto de vista de *Win32*, pero también en los modelos existentes en las bibliotecas de hilos de *Unix*, que normalmente son objetos compartidos (por ejemplo, *libpthread.so*). La biblioteca está siendo utilizada en muchos proyectos, ya sea a partir de la migración de plataformas *Unix* o el desarrollo de aplicaciones multiplataforma. Estos incluyen comercial, de investigación y otras aplicaciones. (12)

OpenThreads

Esta biblioteca tiene como objetivo proporcionar una interfaz de hilos Orientado a Objetos (OO) para programadores de C++. La arquitectura de la biblioteca está concebida en torno a modelos de hilos "intercambiables" que se definen en tiempo de compilación en una biblioteca de objetos compartidos.

Es la intención del grupo *OpenThreads* que las interfaces (archivos de cabecera) se utilicen para construir aplicaciones optimizadas. (13)

1.4 Representación espacial del terreno

1.4.1 Estructuras jerárquicas

Bounding Volume Hierarchy

La técnica de jerarquía de volumen frontera fue creada por Steven Rubin y Turner Whitted (1980). Se basa en encerrar los objetos de una escena en un volumen frontera (volumen de encierro que puede ser cuadrado, esférico, etc.) Y se organizan en una estructura jerárquica los múltiples volúmenes fronteras. Los rectángulos grises son los volúmenes de fronteras de los objetos y los otros rectángulos son los volúmenes fronteras que forman la jerarquía. (14)

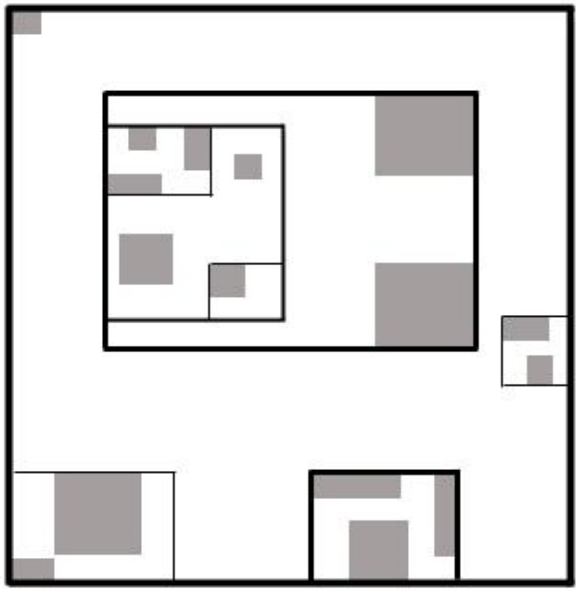

Figura 1 Ejemplo de la Jerarquía de volumen frontera.

Este tipo de estructura es apropiada para escenas con objetos dinámicos, pero la reubicación de la jerarquía de los objetos que se trasladan en la escena puede resultar complicada. Además, como muestra la figura no es nada fácil construir automáticamente una buena jerarquía, de hecho esto constituye un campo de investigación abierto.

Existen varios tipos de volúmenes frontera, que en dependencia de las características de los objetos de la escena son más o menos adecuados, uno de los más empleados es el *Bounding Spheres* o esferas fronteras. Sin embargo, cuando se tiene un objeto irregular, a menudo la esfera frontera ocupa un gran espacio en el que el objeto realmente no existe, lo que puede representar una deficiencia para algunos algoritmos. Para resolver este problema, se emplean entonces los *Bounding Boxes* o cajas fronteras, que se determinan buscando los mayores y menores vértices en relación con el centro del objeto, hallándose una caja que contiene al objeto. (6)

En dependencia de la aplicación que se esté haciendo, se pueden usar ambas técnicas como pruebas iniciales. Se puede usar una jerarquía de pequeñas esferas y cajas en el objeto. (6)

Otros volúmenes fronteras utilizados son: cápsulas, cilindros y elipsoides.

BSP- tree

El árbol de partición binaria del espacio *BSP-Tree* basa su estructura en un árbol binario en el cual cada nodo representa un plano. El subárbol izquierdo corresponde al medio espacio negativo y el derecho al positivo. La siguiente figura da muestra de ello. Las líneas denotan los planos y son representados por letras y las regiones por números. (14)

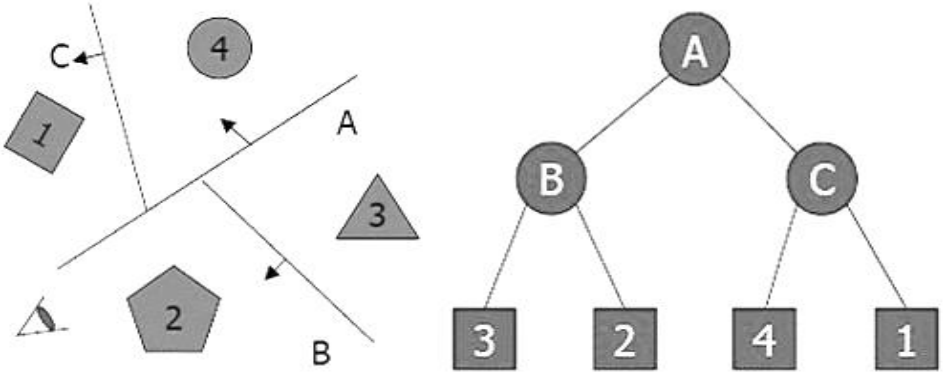

Figura 2 Ejemplo de BSP-Tree.

Esta técnica dada su estructura permite realizar grandes optimizaciones utilizando las potencialidades de la oclusión, de hecho fue esa una de las causas de su surgimiento.

La construcción de un modelo *BSP-Tree* a partir de otro modelo, resulta muy complicado y es esta precisamente una debilidad de esta estructura. La mayoría de los modelos están determinados en otros formatos, como la representación por volúmenes de frontera. En este caso por ejemplo, el proceso de conversión se hace difícil. Las técnicas conocidas para estas trasformaciones no son sensibles a la salida. Además, el *BSP-Tree* asume que los volúmenes de frontera son polígonos y que sus normales van dirigidas hacia fuera del objeto, pero como Sundarsky y Gotsman apuntan, actualmente sólo algunos modelos presentan estos requisitos. (14)

Octree

Esta estructura fue introducida por Andrew Glassner en 1984 para la subdivisión del espacio objeto. Para construir un *Octree* primeramente se inicializa un *voxel* que encierre todas las geometrías presentes en la escena, después se subdivide recursivamente cada *voxel* que encierre muchas geometrías en ocho *subvoxels* (octantes) de igual tamaño. La condición de parada de las subdivisiones puede estar dada por la cantidad de polígonos o la cantidad de objetos. (14)

En general, la subdivisión de los espacios mediante *Octree* se basa en colocar varios *voxels* pequeños en los sitios donde hay muchas geometrías y por el contrario, en los lugares que contengan pequeñas y pocas geometrías se utilizan pocos y grandes *voxels*. (14)

Esta técnica no es muy beneficiosa para tratar escenas dinámicas pues la actualización de la estructura de datos no es trivial como en otros casos, especialmente si es necesario reconstruir el *Octree* sobre la marcha, lo cual sucede muy a menudo y se hace muy engorroso.

Existen formas de disminuir la desventaja que trae consigo esta reconstrucción del árbol, como es el caso la técnica Mínimo Común Antecesor LCA (*Least Common Ancestro*) con la cual se obtienen resultados muy alentadores pero aún así en escenas dinámicas muy cargadas los beneficios son ínfimos.

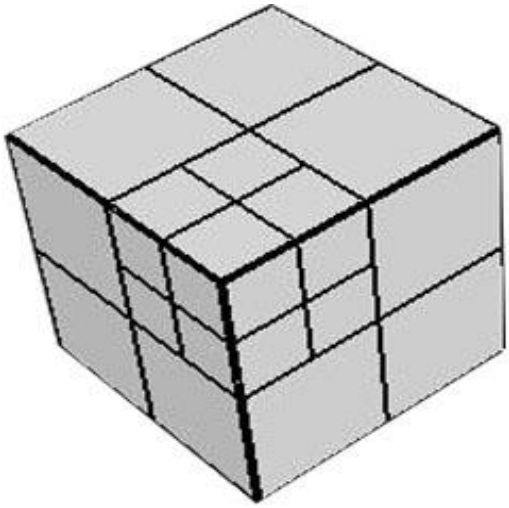

Figura 3 Ejemplo de Octree

1.4.2 Estructuras no jerárquicas

Rejilla regular

Las Rejillas Regulares son apropiadas para escenas dinámicas cuando se combinan con los volúmenes fronteras para los objetos, porque reubicar o relocalizar las geometrías resulta fácil: basta con sustraerla de la celda actual y colocarla en la nueva.

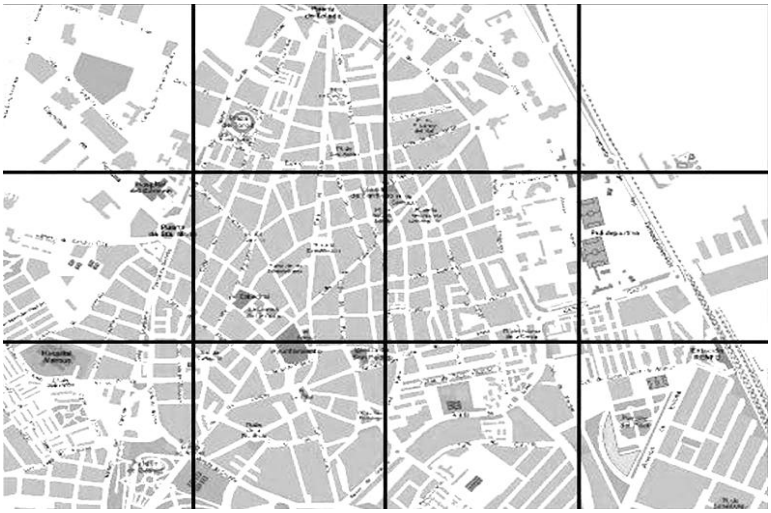

Figura 4 Ejemplo de Rejilla regular.

Una debilidad de esta estructura radica en que manejan áreas densas y escasas de objetos, con el mismo tamaño de subdivisión. Esto implica que no se puedan desechar extensas partes del modelo, cosa que con las estructuras jerárquicas si es posible; ejemplo: considere una escena de una cuidad, aquí puede que una celda encierre una tienda y por tanto contendrá muchas geometrías, mientras que otra celda encierra solo la porción de una calle, o sea, un pequeño número de geometrías. Se debe tratar de subdividir el espacio en 10 celdas que contengan la mayor cantidad posible de geometrías, porque esto ayuda a que la manipulación de las mismas sea menos costosa. (14)

Conclusiones del capítulo

En el capítulo anterior se hizo un estudio sobre el objeto de la investigación tratada, y quedaron enunciados importantes conceptos que están estrechamente relacionados con la carga dinámica de entornos virtuales.

Se expuso algunos de los tipos de carga dinámica utilizados para mostrar grandes entornos virtuales con altos niveles de realismo. Además, se obtuvo los elementos necesarios para dar respuesta al problema científico de este trabajo.

Capítulo 2 Soluciones técnicas

Introducción

En este capítulo se exponen las soluciones técnicas con el propósito de satisfacer el objetivo del trabajo. Para ello se ha realizado un amplio análisis de las técnicas y métodos vistos en el primer capítulo, seleccionándose los de mejores respuestas a las necesidades para resolver el problema.

2.1 Información que se maneja

La información que se maneja es toda la referente a los atributos que permitirán la carga y descarga de la información así como el trabajo con hilos.

2.2 Principales métodos para realizar la carga dinámica

El trabajo para la carga dinámica se realizará con la modelación e implementación de las siguientes funciones:

Configurar rejilla: Con esta funcionalidad el actor puede configurar su rejilla y el tamaño de las celdas previamente consultado con el diseñador del entorno.

Búsqueda de la celda: Esta función permite saber en que celda se encuentra inscrito el objeto dinámico. Además, debe permitir que se conozca si las celdas vecinas a ella están en rango o no, o sea, que pertenezcan a la Rejilla Regular.

Búsqueda de las celdas vecinas: Permite encontrar los vecinos de la celda donde se encuentra el objeto dinámico, adicionándolas a la lista de vecinos.

Carga dinámica: Cuando el objeto dinámico pasa a estar inscrito en una nueva celda esta función debe permitir la carga dinámica de los sectores vecinos, en todo momento se mantendrá en memoria los sectores vecinos cargados.

Destrucción dinámica: Posibilita eliminar dinámicamente sectores que dejan de formar parte de los vecinos de una celda determinada.

2.3 Manipulación de hilos

Todo el manejo de los hilos se realizará con la manipulación e implementación de las siguientes funciones:

Creación de hilos: Esta función es capaz de crear hilos de ejecución paralelos. Cuando se crea el hilo se adiciona automáticamente al grupo de hilos. El hilo maneja una función que se le pasa como parámetro.

Destrucción de hilos: Esta función destruye hilos de ejecución paralelos. Cuando se destruye el hilo se elimina automáticamente del grupo de hilos.

Bloqueo y desbloqueo de hilos: El trabajo con el bloqueo y el desbloqueo se debe hacer dentro de la función que se le pasa al hilo cuando se crea.

Dormir hilos: Esta función permite dormir un hilo por un tiempo determinado.

Despertar hilos: Esta función permite despertar un hilo cuando se desee.

2.4 Influencia del número de fotogramas por segundo

Un aspecto importante a tener en cuenta para la realización de este trabajo es la influencia del número de fotogramas por segundo. Uno de los problemas que presenta un bajo número de fotogramas por segundo es la sensación poco realista de la simulación ya que esta irá a saltos. No habrá continuidad visual en la representación de la imagen causando falta de realismo. El otro problema sería que de nada sirve mantener la simulación a 30 fotogramas por segundo si en algún instante se produce una disminución considerable de estos, presentando consigo tirones que dan una sensación poco realista.

Cabe destacar que si la frecuencia de dibujado es próxima a 30 fotogramas o superior, dota a la simulación de continuidad visual.

2.5 Estructura de datos utilizada para almacenar la información

Para el desarrollo de este proyecto se empleará la estructura de datos no jerárquica Rejilla Regular para guardar la información espacial de los sectores del entorno, y es la base para conocer el estado de visibilidad de cada sector de la escena.

Se escogió esta estructura de datos por su fácil implementación, por tener poco costo computacional, existir abundante bibliografía acerca de ella, por la experiencia acumulada en el trabajo con la misma y porque la carga dinámica de geometrías basadas solamente en las estructuras jerárquicas presenta un costo computacional muy grande, siendo muy crítico en entornos optimizados donde se instancian gran número de elementos.

División del terreno

Para la división del terreno se recurre a la sectorización de entornos, donde cada geometría se cargará inmediatamente que el objeto dinámico pasa a encontrarse en una nueva celda. Para optimizar los cálculos todos los sectores serán cuadrados y de iguales dimensiones. Además, con el fin de asegurar la correcta visualización del entorno en todo momento se mantendrán visible el sector en el cual se encuentra inscrito el objeto dinámico y los sectores presentes aledaños.

Se podrá configurar el tamaño del sector y la cantidad de filas y columnas de la Rejilla Regular.

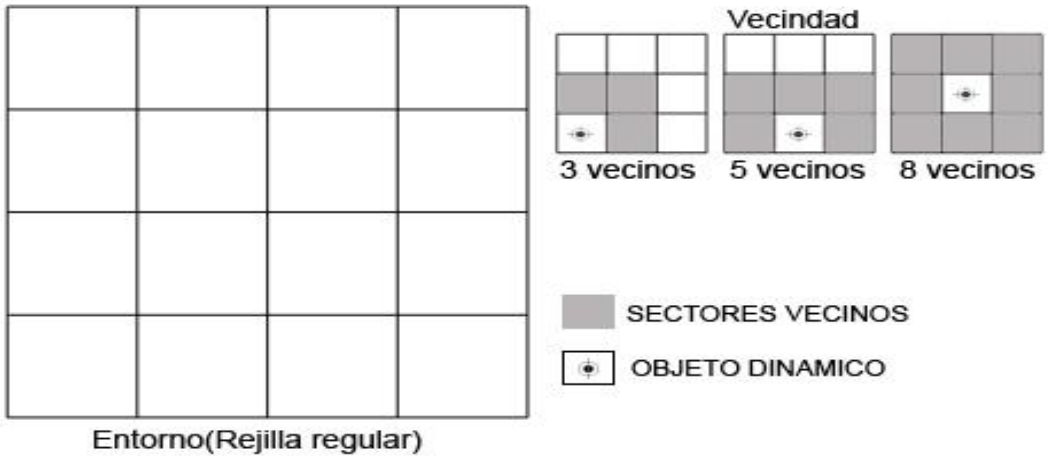

Figura 5 División del terreno.

2.6 Procedimiento de carga dinámica

A partir de la posición espacial del objeto dinámico se puede conocer en que celda de la rejilla se encuentra el mismo y con esta información poder cargar dinámicamente las celdas “vecinas” a la celda donde se encuentra el objeto. Cada celda de la rejilla se corresponde con un sector del entorno.

Cuando el objeto dinámico pasa a estar contenido dentro de una nueva celda, el método de carga dinámica ejecuta el método de búsqueda de vecinos, este chequea que las celdas vecinas estén en rango y carga aquellas que no estén cargadas, las que ya dejen de serlo las destruye, con lo que solo quedan en memoria las texturas y geometrías necesarias en ese instante. Para poder realizar la carga dinámica se escoge la técnica de carga dinámica de texturas y geometrías por ser ideal para ordenadores de poca capacidad y por ser la técnica de carga que permite la representación de escenarios de grandes dimensiones.

Figura 6 Procedimiento de carga y descarga.

Se tendrá en cuenta para toda la manipulación de los hilos, la biblioteca de clases Boost, por tener basta experiencia acumulada ya que se han realizado trabajos que demuestran sus potencialidades, por ser multiplataforma, diseñado para ser altamente portable, por su fácil y sencilla comprensión, por ser diseñada para que el tratamiento de hilos sea tan seguro como sea posible y aislar el programador de construcciones peligrosas. También debe ser lo más eficiente posible, flexible a cambios y para tomar un dinámico e iterativo enfoque en su desarrollo. Además, se escoge esta biblioteca por su amplia bibliografía y documentación.

Conclusiones del capítulo

Como resultado de las soluciones técnicas, el módulo a desarrollar debe cumplir con el objetivo trazado en el comienzo de la investigación. Debe ser capaz de manipular adecuadamente hilos de ejecución así como realizar la carga y descarga de entornos virtuales sin que haya una bajada en el rendimiento de la aplicación.

Con el capítulo presentado quedan trazadas las guías que servirán para realizar las siguientes fases del trabajo. Además, se tienen las bases por las cuales se regirá el módulo encargado de la carga dinámica.

Capítulo 3 Descripción de la solución propuesta

Introducción

El propósito de este capítulo es brindar una descripción de la solución propuesta de manera conceptual. En él se muestran los conceptos necesarios para el desarrollo de la solución, luego se describe el producto a elaborar sobre la base de las dificultades y necesidades del cliente, así como las capacidades que tendrá el módulo de carga dinámica, descritas en forma de casos de uso.

3.1 Reglas del negocio

Con el propósito de definir las condiciones que deben satisfacerse para el correcto funcionamiento del módulo de carga dinámica se establecen como reglas del negocio las siguientes:

- ✓ El diseño del entorno debe comenzar en la coordenada (0, 0, 0).
- ✓ El diseño del entorno debe comenzar de abajo hacia arriba y de izquierda a derecha.
- ✓ El diseño del entorno debe ser cuadrado o rectangular.
- ✓ El diseño del entorno debe estar dividido por sectores cuadrados de iguales dimensiones.
- ✓ Los sectores deben ser salvados con el nombre “sector_” + # del sector, comenzando por el 000 (*Ejemplo: sector_000.3dx).*
- ✓ Los archivos a cargar deben ser sólo de extensión .*3dx,* debido a que este formato es el único que soporta la *SceneToolkit*.
- ✓ El objeto dinámico siempre debe ser cargado dentro del entorno.
- ✓ El usuario tendrá que configurar la rejilla antes de invocar al método de carga dinámica.

3.2 Modelo de dominio

A partir del entendimiento del problema se obtiene el siguiente modelo de dominio representado a continuación, que no es más que un acercamiento a la solución propuesta, donde se modelan los principales conceptos así como sus relaciones.

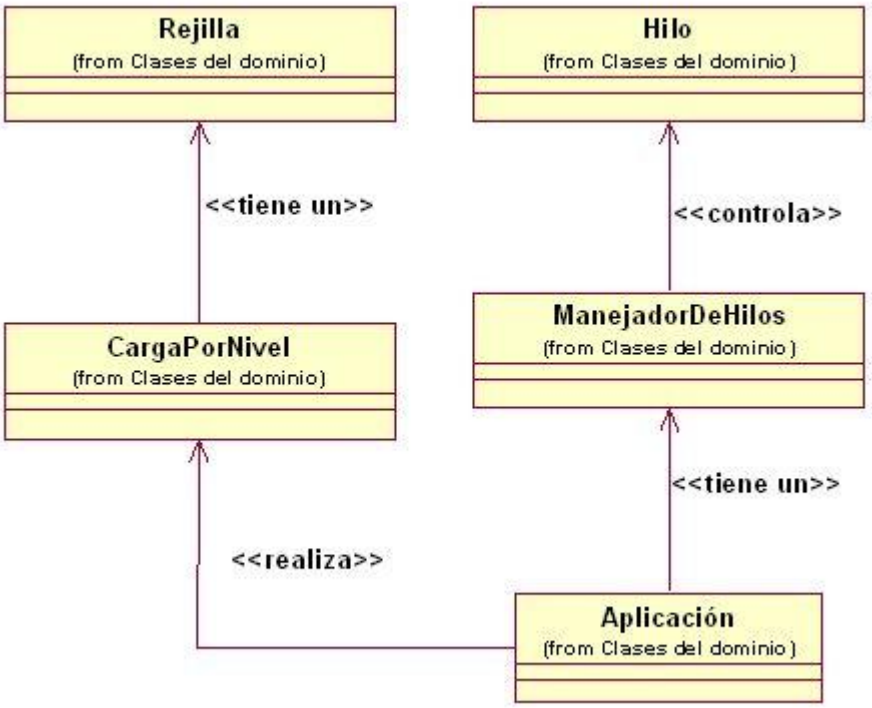

Figura 7 Modelo de dominio.

3.3 Especificación de los requisitos de software

3.3.1 Dependencias y relaciones con otro software

Este proyecto está concebido para que sea un módulo de la herramienta *SceneToolkit* que se desarrolla en el proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual, por lo tanto las clases que formarán parte del Módulo de carga dinámica para la herramienta *SceneToolkit* debe poseer una fuerte interrelación con la biblioteca de clase que conforma dicha herramienta ya que utilizará sus funcionalidades para crear sus propias responsabilidades.

Además, tendrá una relación de dependencia con la biblioteca de clases Boost.

3.3.2 Requisitos funcionales

Teniendo en cuenta las necesidades del cliente se pueden inferir los siguientes requerimientos funcionales:

1. Configurar la rejilla teniendo en cuenta la dimensión de las celdas y la cantidad de filas y columnas.
2. Conocer el sector donde se encuentra el objeto dinámico.
3. Conocer cuáles son los sectores aledaños al sector donde se encuentra el objeto dinámico.
4. Gestionar la carga de entornos.
   4.1. Cargar sectores aledaños a la posición actual del objeto dinámico.
   4.2. Destruir sectores que no son aledaños a la posición actual del objeto dinámico.
5. Gestionar la manipulación de hilos.
   5.1. Crear un hilo.
   5.2. Destruir un hilo.
6. Bloquear un hilo.
7. Desbloquear un hilo.
8. Dormir un hilo.
9. Despertar un hilo.

3.3.3 Requisitos no funcionales

Entre los requisitos no funcionales que debe tener el módulo a obtener se encuentran los siguientes:

Rendimiento: Como aplicación de tiempo real, debe tener alto grado de velocidad de procesamiento o cálculo, tiempo de respuesta y de recuperación, y disponibilidad.

Soporte: Debe ser compatible con las plataformas *Windows* y *GNU/Linux*. Además de proporcionar la biblioteca de clases Boost para el manejo de hilos; en esta primera versión solo para plataforma *Windows*.

Legales: Se utilizarán bibliotecas externas con licencias libres, nada propietario.

Software: Sistema operativo *Windows* y *GNU/Linux*, biblioteca de clases *Boost*, *Codeblock* versión 8.02, *Microsoft Visual Studio.Net 2003*.

Hardware: Compatibilidad con tarjetas gráficas de la familia *NVIDIA* (*Geforce* 3, *Geforce* 4, FX 5200).

Implementación: Debe ser implementado en el Leguaje C++ estándar. Se regirá por la filosofía de Programación Orientada a Objetos.

Usabilidad: La aplicación está concebida para ser reusable por aplicaciones finales.

Documentación: Se utilizará documentación de código con los estándares establecidos en el proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual para generación con *Doxygen*.

3.4 Definición de los casos de uso del sistema

A partir de las funcionalidades que deben ser implementadas en el módulo de carga, se reconoce el actor del sistema a desarrollar y se conciben los casos de uso del sistema. Además, se seleccionan los casos de uso correspondientes al primer ciclo de desarrollo para hacerles sus especificaciones textuales en formato expandido.

Los casos de usos definidos son divididos en dos vistas fundamentales. Una primera vista muestra los casos de uso encargados de la carga dinámica y la segunda vista donde aparecen los casos de uso encargados de la manipulación de hilos.

3.4.1 Diagrama de casos de uso del sistema

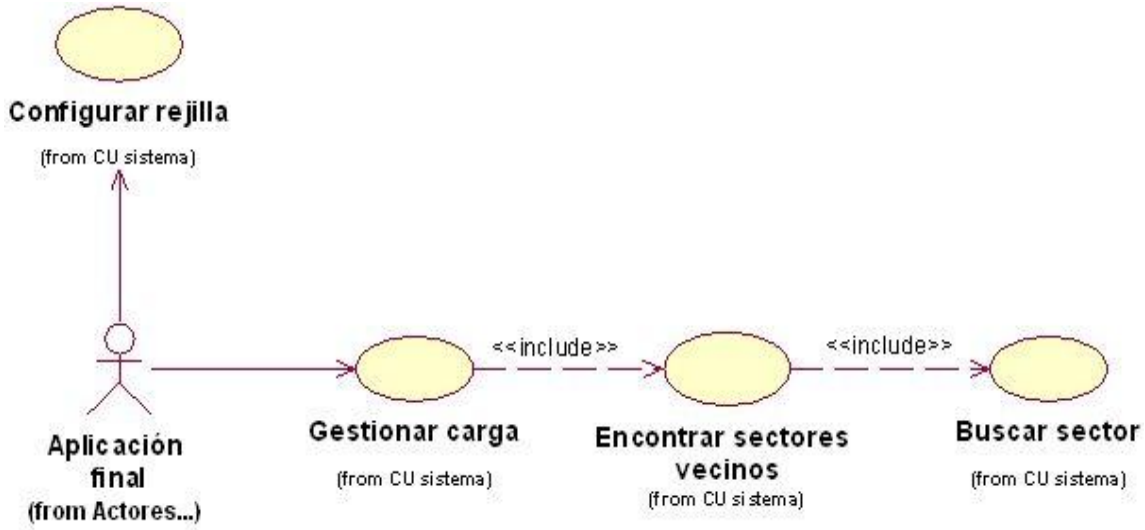

Figura 8 Diagrama de casos de uso del sistema (A).

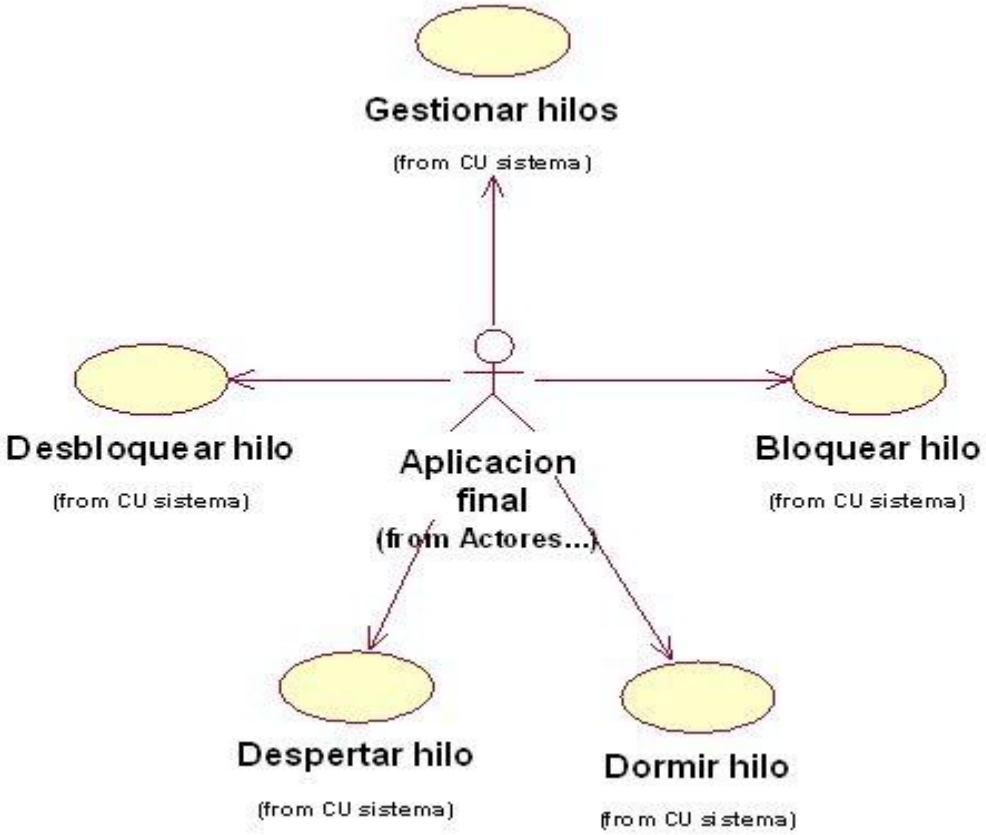

Figura 9 Diagrama de casos de uso del sistema (B).

3.4.2 Actores del sistema

Actores	**Justificación**
Aplicación final	La aplicación final es la que se beneficia ya que en ella van a haber casos de uso que invocan a los casos de uso del módulo de carga que es interno a la herramienta *SceneToolkit*.

Tabla 1 Justificación de actores.

3.4.3 Listado de casos de uso del sistema

CU_1	
Nombre del caso de uso	Configurar rejilla.
Actores	Aplicación final.
Propósito	Permitirle al actor poder configurar la rejilla con la que se va a trabajar.
Referencias	R1

Tabla 2 CU Configurar rejilla.

CU_2	
Nombre del caso de uso	Buscar sector.
Actores	Aplicación final.
Propósito	Permitirle al actor poder buscar el sector en el cual se encuentra inscrito el objeto dinámico.
Referencias	R2

Tabla 3 CU Buscar sector.

CU_3	
Nombre del caso de uso	Encontrar sectores vecinos.
Actores	Aplicación final.
Propósito	Encontrar los sectores aledaños al sector donde esta el objeto.
Referencias	R3, CU-Buscar sector

Tabla 4 CU Encontrar sectores vecinos.

CU_4	
Nombre del caso de uso	Gestionar carga.
Actores	Aplicación final.
Propósito	Gestionar la carga de forma tal que sea posible cargar sectores y destruirlos de manera dinámica.
Referencias	R4, CU-Encontrar sectores vecinos

Tabla 5 CU Gestionar carga.

CU_5	
Nombre del caso de uso	Gestionar hilos.
Actores	Aplicación final
Propósito	Gestionar hilos de forma tal que sea posible crearlos y destruirlos.
Referencias	R5

Tabla 6 CU Gestionar hilos.

CU_6	
Nombre del caso de uso	Bloquear hilo.
Actores	Aplicación final.
Propósito	Permitirle al actor poder bloquear un hilo previamente creado.
Referencias	R6

Tabla 7 CU Bloquear hilo.

CU_7	
Nombre del caso de uso	Desbloquear hilo.
Actores	Aplicación final.
Propósito	Permitirle al actor poder desbloquear un hilo previamente bloqueado.
Referencias	R7

Tabla 8 CU Desbloquear hilo.

CU_8	
Nombre del caso de uso	Dormir hilo.
Actores	Aplicación final.
Propósito	Permitirle al actor poder dormir un hilo previamente creado.
Referencias	R8

Tabla 9 CU Dormir hilo.

CU_9	
Nombre del caso de uso	Despertar hilo.
Actores	Aplicación final.
Propósito	Permitirle al actor poder despertar un hilo dormido.
Referencias	R9

Tabla 10 CU Despertar hilo.

3.5 Expansión de los casos de uso

3.5.1 CU Configurar rejilla

CU_1	
Nombre del caso de uso	Configurar rejilla.
Actores	Aplicación final.
Propósito	Permitirle al actor poder configurar la rejilla con la que se va a trabajar.
Resumen El caso de uso se inicia cuando el actor configura la rejilla, definiendo el tamaño del sector, las filas y las columnas y finaliza cuando ya está configurada.	
Referencias	R1
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1. El actor manda a configurar la rejilla.	
	1.1 El sistema toma los datos del tamaño del sector y la cantidad de filas y columnas y configura la rejilla.
	1.2 Se acopla la raíz de la rejilla al grafo de escena.
Precondiciones	-
Poscondiciones	El sistema configura la rejilla con el tamaño del sector y la cantidad de filas y columnas.

Tabla 11 Expansión del CU Configurar rejilla.

3.5.2 CU Buscar sector

CU_2	
Nombre del caso de uso	Buscar sector.
Actores	Aplicación final. CU-Encontrar sectores vecinos.
Propósito	Permitirle al actor poder buscar el sector en el cual se encuentra inscrito el objeto dinámico.
Resumen El caso de uso se inicia cuando el actor necesita conocer en todo momento el sector donde se encuentra inscrito el objeto dinámico para poder determinar los sectores vecinos aledaños y finaliza cundo se encuentra dicho sector.	
Referencias	R2
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1. El actor manda a buscar el sector donde se encuentra inscrito el objeto dinámico.	
	1.1 El sistema recorre la lista de sectores para determinar donde se encuentra el objeto dinámico.
	1.2 Si se encuentra dicho objeto se devuelve el número del sector donde está. Si no se encuentra se devuelve -1.
Precondiciones	-
Poscondiciones	El sistema devuelve la posición del sector donde se encuentra el objeto dinámico.

Tabla 12 Expansión del CU Buscar sector.

3.5.3 Encontrar sectores vecinos

CU_3	
Nombre del caso de uso	Encontrar sectores vecinos.
Actores	Aplicación final. CU-Gestionar carga.
Propósito	Permitirle al actor poder buscar el sector en el cual se encuentra inscrito el objeto dinámico.
Resumen El caso de uso se inicia cuando queremos encontrar los sectores vecinos al sector donde se encuentra el objeto dinámico y finaliza cuando se encuentran dichos sectores.	
Referencias	R3, CU-Buscar sector
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1. El actor quiere encontrar los sectores vecinos al sector donde se encuentra el objeto dinámico.	
	1.1 El sistema compara las listas de sectores, la actual y la pasada, si encuentra sectores nuevos los adiciona a la lista de sectores a cargar.
	1.2 Se invoca al CU-Buscar sector.
Precondiciones	-
Poscondiciones	El sistema devuelve la posición del sector donde se encuentra el objeto dinámico.

Tabla 13 Expansión del CU Encontrar sectores vecinos

3.5.4 CU Gestionar carga

CU_4	
Nombre del caso de uso	Gestionar carga.
Actores	Aplicación final.
Propósito	Gestionar la carga de forma tal que sea posible cargar y destruir entornos virtuales.
Resumen El caso de uso es iniciado por el actor, el cual quiere cargar o destruir por niveles entornos virtuales.	
Referencias	R4 CU-Encontrar sectores vecinos.
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
a) Si el actor desea cargar entornos por niveles, ir a la sección “Cargar por niveles”. b) Si el actor desea destruir entornos por niveles, ir a la sección “Destruir por niveles”.	
Sección: “Cargar por niveles”	
Acción del actor	**Respuesta del sistema**
1. El actor manda a cargar entornos por niveles.	
	1.1 El sistema carga los sectores vecinos al sector

	donde está el objeto dinámico que están adicionados en la lista de sectores a cargar.
	1.2 Se invoca al CU-Encontrar sectores vecinos.
Sección: "Destruir por niveles"	
Acción del actor	**Respuesta del sistema**
1. El actor manda a destruir entornos por niveles.	
	1.1 El sistema destruye los sectores que no son vecinos al sector donde está el objeto dinámico.
	1.2 Se invoca al CU-Encontrar sectores vecinos.

Tabla 14 Expansión del CU Gestionar carga.

3.5.5 CU Gestionar hilos

CU_5		
Nombre del caso de uso	Gestionar hilos.	
Actores	Aplicación final	
Propósito	Gestionar hilos de forma tal que sea posible crearlos y destruirlos.	
Resumen El caso de uso es iniciado por el actor, el cual puede invocar crear un hilo especificando que función será controlada por el mismo; o puede destruir un hilo previamente creado.		
Referencias	R5	
Curso normal de los eventos:		
Acción del actor		Respuesta del sistema
a) Si el actor desea crear un hilo, ir a la sección “Crear hilo”. b) Si el actor desea destruir un hilo, ir a la sección “Destruir hilo”.		
Sesión “Crear hilo”		
Acción del actor		**Respuesta del sistema**
1. El actor quiere crear un hilo.		
		1.1 Si no existe un hilo con el mismo nombre el sistema crea un hilo especificándole qué función será controlada por el mismo.
Sesión “Destruir hilo”		

Acción del actor	Respuesta del sistema
1. El actor desea destruir un hilo.	
	1.1 Si el hilo estaba previamente creado el sistema lo destruye.

Tabla 15 Expansión del CU Gestionar hilos.

3.5.6 CU Bloquear hilo

CU_6		
Nombre del caso de uso	Bloquear hilo.	
Actores	Aplicación final.	
Propósito	Permitirle al actor poder bloquear un hilo previamente creado.	
Resumen El caso de uso se inicia cuando el actor manda a bloquear un hilo previamente creado y finaliza cuando el hilo se bloquea.		
Referencias	R6	
Curso normal de los eventos:		
Acción del actor		**Respuesta del sistema**
1. El actor manda a bloquear un hilo.		
		1.1 Si el hilo estaba previamente creado el sistema bloquea el hilo.
Precondiciones		-
Poscondiciones		El sistema bloquea el hilo.

Tabla 16 Expansión del CU Bloquear hilo.

3.5.7 CU Desbloquear hilo

CU_7		
Nombre del caso de uso	Desbloquear hilo.	
Actores	Aplicación final.	
Propósito	Permitirle al actor poder desbloquear un hilo previamente bloqueado.	
Resumen El caso de uso se inicia cuando el actor manda a desbloquear un hilo previamente bloqueado y finaliza cuando el hilo se desbloquea.		
Referencias	R7	
Curso normal de los eventos:		
Acción del actor		**Respuesta del sistema**
1. El actor manda a desbloquear un hilo.		
		1.1 Si el hilo estaba previamente bloqueado el sistema desbloquea el hilo.
Precondiciones		-
Poscondiciones		El sistema desbloquea el hilo.

Tabla 17 Expansión del CU Desbloquear hilo.

3.5.8 CU Dormir hilo

CU_8	
Nombre del caso de uso	Dormir hilo.
Actores	Aplicación final.
Propósito	Permitirle al actor poder dormir un hilo previamente creado.
Resumen El caso de uso se inicia cuando el actor manda a dormir un hilo previamente creado y finaliza cuando el hilo se duerme.	
Referencias	R8
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1. El actor manda a dormir un hilo.	
	1.1 Si el hilo estaba previamente creado el sistema duerme el hilo.
Precondiciones	-
Poscondiciones	El sistema duerme el hilo.

Tabla 18 Expansión del CU Dormir hilo.

3.5.9 CU Despertar hilo

CU_9		
Nombre del caso de uso	Despertar hilo.	
Actores	Aplicación final.	
Propósito	Permitirle al actor poder despertar un hilo dormido.	
Resumen El caso de uso se inicia cuando el actor manda a despertar un hilo dormido y finaliza cuando el hilo se despierta.		
Referencias	R9	
Curso normal de los eventos:		
Acción del actor		**Respuesta del sistema**
1. El actor manda a despertar un hilo.		
		1.1 Si el hilo estaba dormido el sistema despierta el hilo.
Precondiciones		-
Poscondiciones		El sistema despierta el hilo.

Tabla 19 Expansión del CU Despertar hilo.

Conclusiones del capítulo

En este capítulo se recopilan los requisitos funcionales y no funcionales y se definen y describen los casos de uso de acuerdo con lo que espera el cliente que realice el módulo a desarrollar en el primer ciclo de desarrollo. Ya se puede proceder con el diseño del mismo.

Capítulo 4 Diseño e implementación del sistema

Introducción

En este capítulo se describen todos los aspectos relacionados con el módulo a construir mediante el diagrama de clases y los diagramas de secuencia de la realización de los casos de uso que intervendrán en el primer ciclo de desarrollo del proyecto. Este paso constituye el refinamiento de las etapas anteriores.

Además, se hace una descripción de cada una de las clases del módulo.

4.1 Estándares de codificación

Nombre de los ficheros:

Se nombrarán los ficheros .h y .cpp de la siguiente manera:

STKNameOfUnits.cpp

Se usará STK para identificar el nombre de la herramienta.

Constantes:

Las constantes se nombrarán con mayúsculas, utilizándose el "_" para separar las palabras:

MY_CONST_ZERO = 0;

Tipos de datos:

Los tipos se nombrarán siguiendo el siguiente patrón:

Enumerados:

enum EMyEnum {ME_VALUE, ME_OTHER_VALUE};

Indicando con "E" que es de tipo enumerado. Nótese que las primeras letras de las constantes de enumerados son las iniciales del nombre del enumerado.

Estructuras:

struct SMyStruct {…};

Indicando con "S" que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.

Clases:

class CClassName;

Indicando con "C" que es una clase

Interfaces:

IMyInterface

Indicando con “I” que es una interfaz.

Listas e iteradores STD:

vector<> TNameList;

TNameList::iterator TNameListIter;

map<> TNameMap;

TNameMap::iterator TNameMapIter;

multimap<> TNameMultiMap;

TNameMultiMap::iterator TNameMultiMapIter;

Declaración de variables:

Los nombres de las variables comenzarán con un identificado del tipo de dato al que correspondan, como se muestra a continuación. En el caso de que sean variables miembros de una clase, se le antepondrá el identificador “” (en minúscula), si son globales se les antepondrá la letra “g”, y en caso de ser argumentos de algún método, se les antepondrá el prefijo “arg_”.

Instancias de tipos creados:

EMyEnumerated eName;

SMyStructure kName;

CClassName kObjectName;

CClassName pkName*; //puntero a objeto

CClassName akName*; //arreglo de objetos

CClassName akName*; // variable miembro de clase

IMyInterface pIName*; //puntero interfaces

Tipos simples:

bool bVarName;

int iName;

unsigned int uiName;

float fName;

char cName;

char acName*; // arreglo de caracteres

char pcName*; // puntero a un char

*char** aacName*; // bidimensional

*char** apcName*; // arreglo de punteros

bool m_bMemberVarName; //variable miembro

char gcGlobalVarName; //variable global, no se le antepone “”

short sName;

Métodos**:**

En el caso de los métodos, se les antepondrá el identificador del tipo de dato de devolución, y en caso de no tenerlo (*void*), no se les antepondrá nada. Solamente los constructores y destructores comenzarán con “”.En el caso de los argumentos se les antepone el prefijo “arg_”

Constructor y destructor:

CClassName (bool arg_bVarName, float& arg_fVarName);

~CClassName ();

Funciones:

bool bFunction1 (…);

int piFunction2 (…);*

CClassName pkFunction3 (…);*

Procedimientos:

void Procedure4 (…);

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases no se nombrarán "*Gets*" y "*Sets*", sino como los demás métodos, pero con el nombre de la variable a la que se accede y sin "m_":

```
int iMyVar;                    //variable
```

Obtención del valor:

```
int iMyVar ();
{
return iMyVar;
}
```

Establecimiento del valor:

```
void MyVar (char* arg_iMyVar)
{
iMyVar = arg_iMyVar;
}
```

Obtención y establecimiento del valor:

```
int& iMyVar ();
{
return iMyVar;
}
```

4.2 Diagramas de clases de diseño

4.2.1 Diagrama de clase de diseño por paquetes

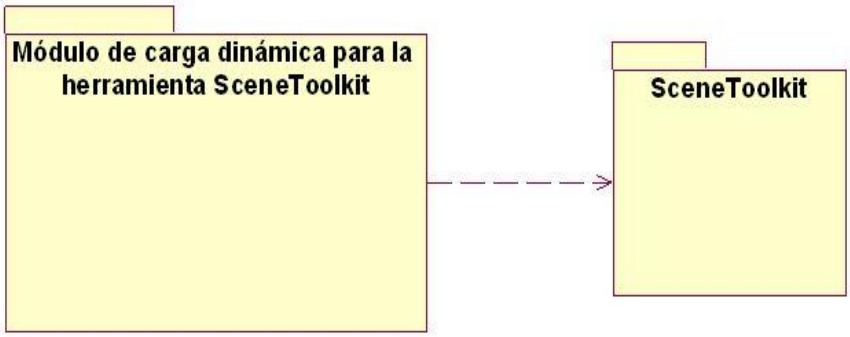

Figura 10 Diagrama de clases de diseño por paquetes.

4.2.2 Paquete *SceneToolkit*

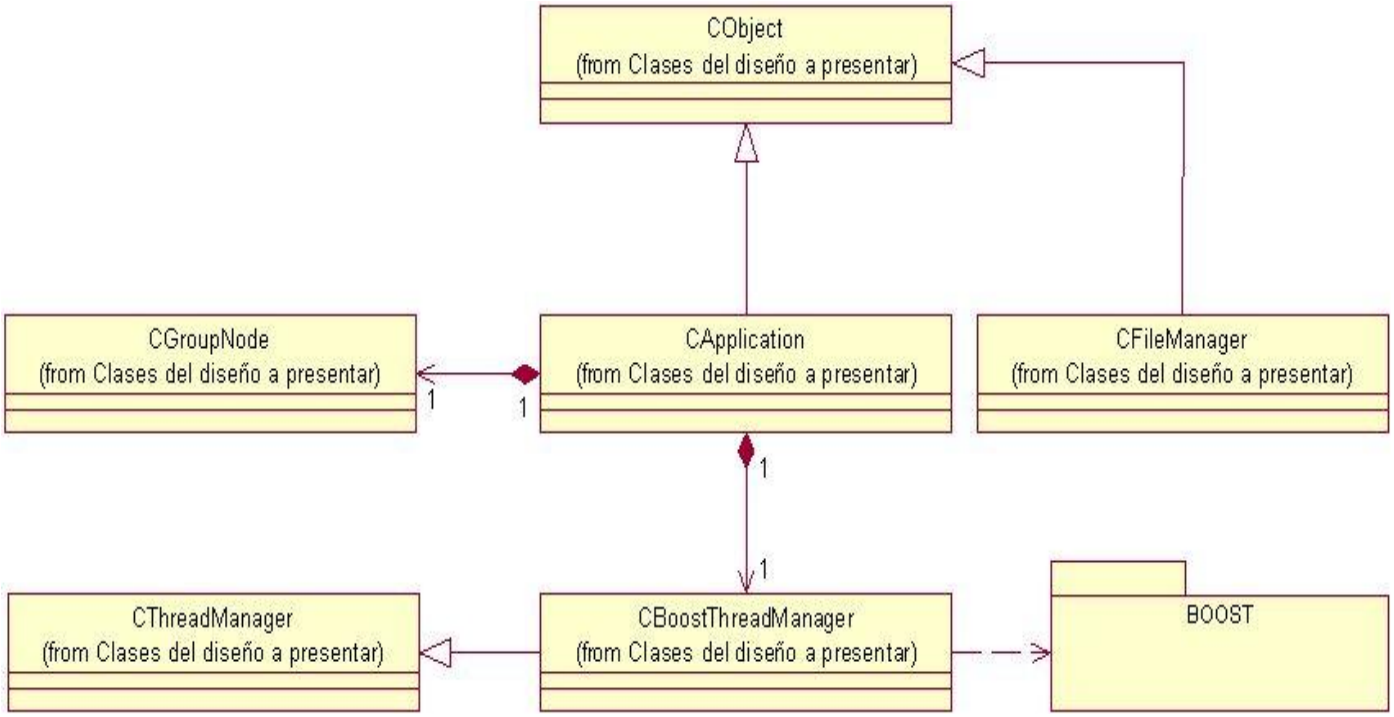

Figura 11 Diagrama del paquete *SceneToolkit*.

4.2.3 Paquete Módulo de carga dinámica para la herramienta *SceneToolkit*

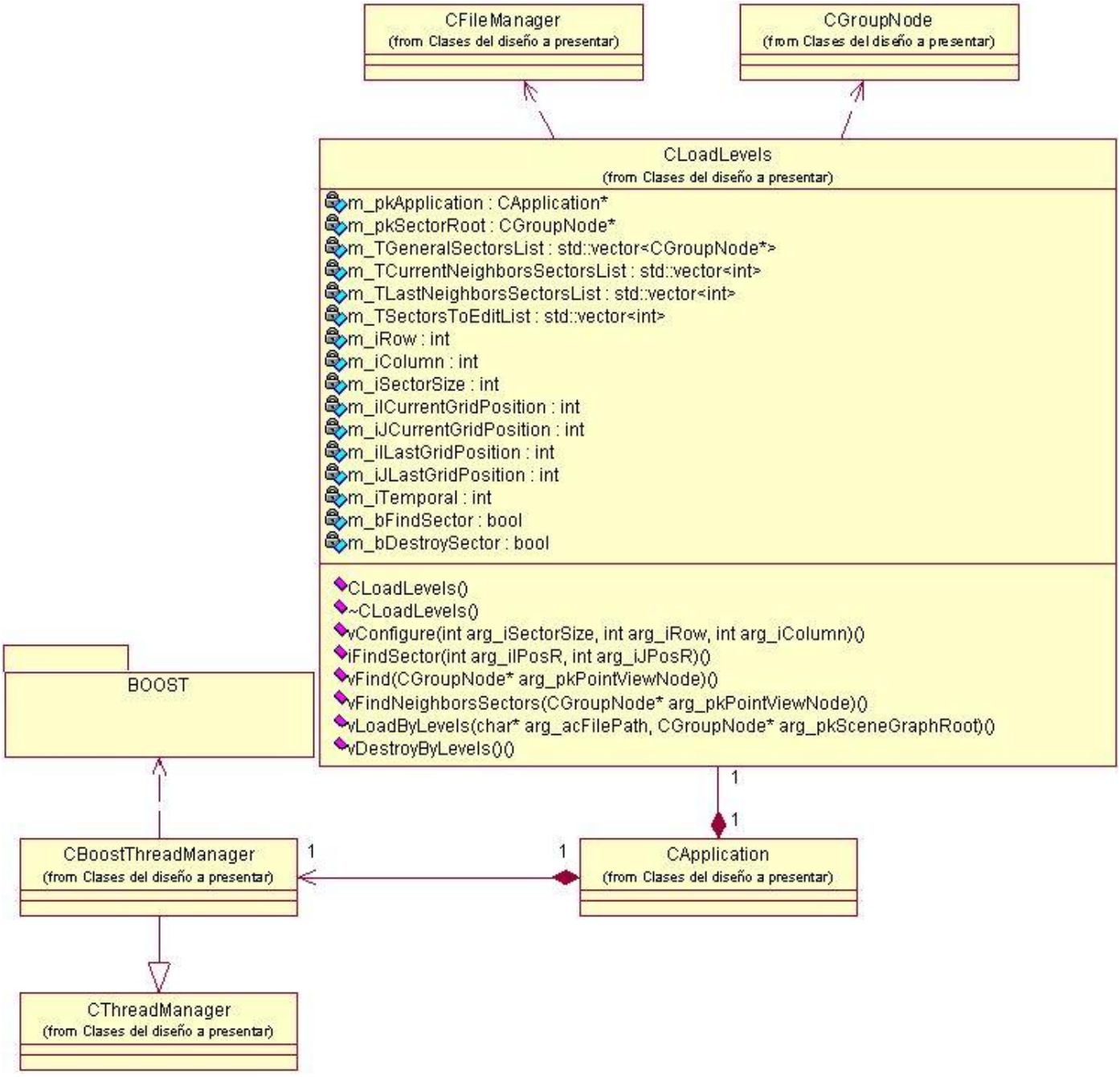

Figura 12 Diagrama del paquete Módulo de carga dinámica para la herramienta *SceneToolkit*.

4.3 Descripción de las clases de diseño

Después de haber visto los diagramas de clases es necesario especificar las características de ellas. En las tablas que aparecen a continuación se describen los atributos y métodos de las clases incluidas en el módulo, estas son: *CThreadManager*, *CBoostThreadManager*, *CLoadLevels*.

Nombre: *CThreadManager*	
Tipo de clase: Controladora	
Atributo	**Tipo**
Para cada responsabilidad	
Nombre:	*vCreateThread(void arg_vFunction()) = 0*
Descripción:	Es un método virtual puro para que se redefina en las clases hijas.
Nombre:	*iThreadQuantity() = 0*
Descripción:	Es un método virtual puro para que se redefina en las clases hijas.
Nombre:	*vRemoveThread(unsigned int arg_uiPos) = 0*
Descripción:	Es un método virtual puro para que se redefina en las clases hijas.
Nombre:	*vSleep(unsigned int arg_uiTime) = 0*
Descripción:	Es un método virtual puro para que se redefina en las clases hijas.
Nombre:	*vJoinThread(unsigned int arg_uiPos) = 0*
Descripción:	Es un método virtual puro para que se redefina en las clases hijas.
Nombre:	*vYieldThread(unsigned int arg_uiPos) = 0*
Descripción:	Es un método virtual puro para que se redefina en las clases hijas.
Nombre:	*vLockThread() = 0*
Descripción:	Es un método virtual puro para que se redefina en las clases hijas.
Nombre:	*vUnLockThread() = 0*
Descripción:	Es un método virtual puro para que se redefina en las clases hijas.
Nombre:	*vJoinAllThread() = 0*

Descripción:	Es un método virtual puro para que se redefina en las clases hijas.

Tabla 20 Descripción de la clase *CThreadManager*.

Nombre: *CBoostThreadManager*	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_pkThread	*boost::thread**
m_pkThreadGroup	*boost::thread_group**
m_pRecursiveMutex	*boost::recursive_mutex*
m_pkLock	*boost::detail::thread::scoped_lock<boost::recursive_mutex>**
m_pAuxThreadList	*std::vector <boost::thread*>*
Para cada responsabilidad	
Nombre:	*vCreateThread(void arg_vFunction())*
Descripción:	Crea un hilo y lo adiciona a la lista de hilos.
Nombre:	*iThreadQuantity() = 0*
Descripción:	Devuelve la cantidad de hilos.
Nombre:	*vRemoveThread(unsigned int arg_uiPos) = 0*
Descripción:	Destruye un hilo dada una posición.
Nombre:	*vSleep(unsigned int arg_uiTime) = 0*
Descripción:	Duerme un hilo dada una posición.
Nombre:	*vJoinThread(unsigned int arg_uiPos) = 0*
Descripción:	Libera un hilo dada una posición.
Nombre:	*vYieldThread(unsigned int arg_uiPos) = 0*
Descripción:	Despierta un hilo dada una posición.
Nombre:	*vLockThread() = 0*
Descripción:	Bloquea un hilo.
Nombre:	*vUnLockThread() = 0*
Descripción:	Desbloquea un hilo.

Nombre:	*vJoinAllThread() = 0*
Descripción:	Libera todos los hilos.

Tabla 21 Descripción de la clase *CBoostThreadManager*.

Nombre: *CLoadLevels*	
Tipo de clase: Controladora	
Atributo	Tipo
m_pkApplication	*CApplication**
m_pkSectorRoot	*CGroupNode**
m_TGeneralSectorsList	*std::vector<CGroupNode*>*
m_TCurrentNeighborsSectorsList	*std::vector<int>*
m_TLastNeighborsSectorsList	*std::vector<int>*
m_TSectorsToEditList	*std::vector<int>*
m_iRow	*int*
m_iColumn	*int*
m_iSectorSize	*int*
m_iICurrentGridPosition	*int*
m_iJCurrentGridPosition	*int*
m_iTemporal	*int*
m_iILastGridPosition	*int*
m_iJLastGridPosition	*int*
m_bFindSector	*bool*
m_bDestroySector	*bool*
Para cada responsabilidad	
Nombre:	*vConfigure(int arg_iSectorSize, int arg_iRow, int arg_iColumn)*
Descripción:	Configura la rejilla con la cual se va a trabajar.
Nombre:	*iFindSector(int arg_iIPosR, int arg_iJPosR)*

Descripción:	Devuelve la posición donde se encuentra inscrito el objeto dinámico.
Nombre:	*vFind(CGroupNode* arg_pkPointViewNode)*
Descripción:	Método auxiliar para poder hallar los sectores vecinos.
Nombre:	*vFindNeighborsSectors(CGroupNode* arg_pkPointViewNode)*
Descripción:	Método que halla los sectores vecinos.
Nombre:	*vLoadByLevels(char* arg_acPathFile, CGroupNode* arg_pkSceneGraphRoot);*
Descripción:	Método que se encarga de la carga dinámica de entornos virtuales.
Nombre:	*vDestroyByLevels()*
Descripción:	Método que se encarga de la destrucción dinámica de entornos virtuales.

Tabla 22 Descripción de la clase *CLoadLevels*.

4.4 Diagramas de secuencia

Los diagramas siguientes corresponden a las relaciones que se establecen de forma secuencial entre los objetos de las principales funcionalidades descritas en los escenarios de los casos de uso del sistema.

Los diagramas de secuencia son divididos en dos vistas fundamentales. Una primera vista muestra los diagramas encargados de la carga dinámica y la segunda vista donde aparecen los diagramas encargados del manejo de hilos.

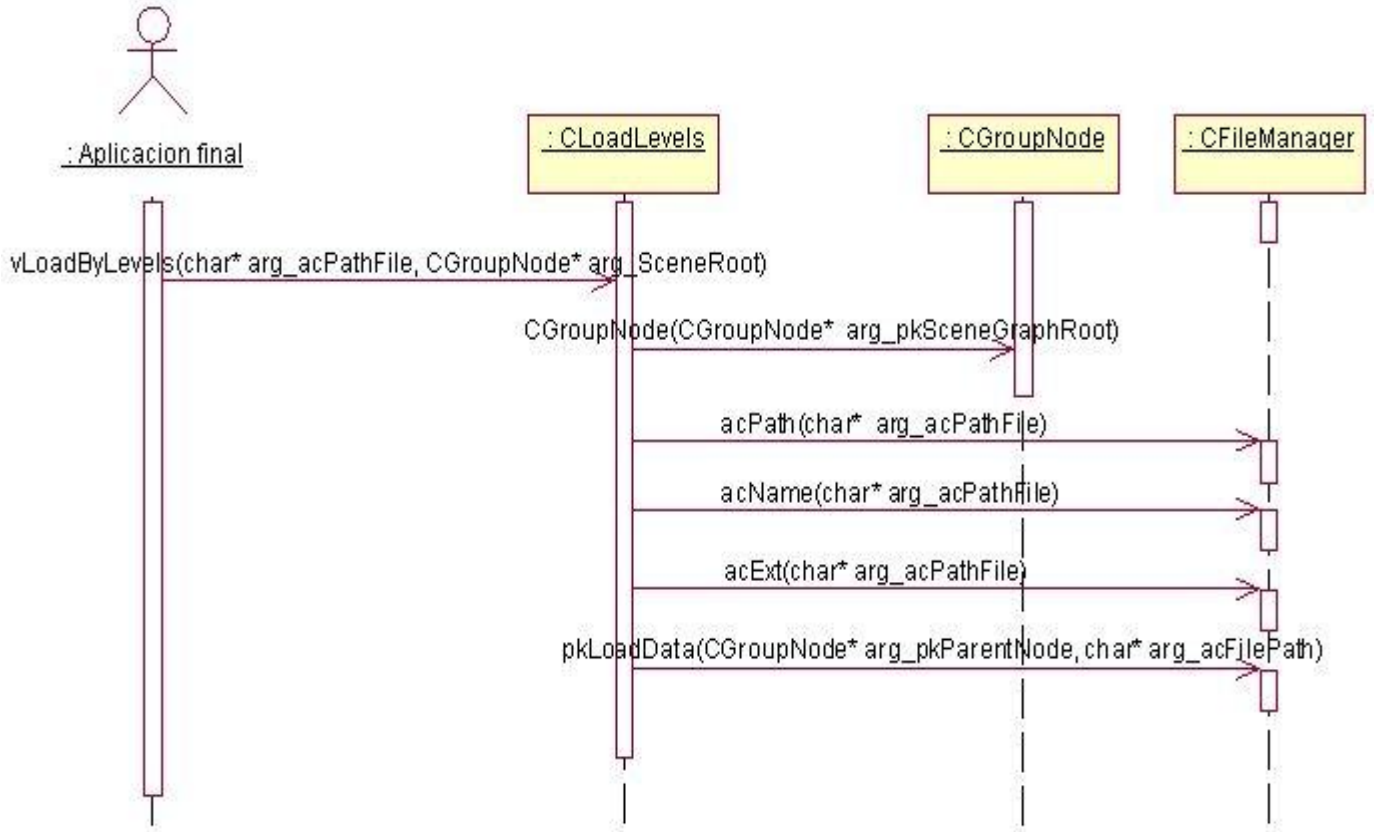

Figura 13 Diagrama de secuencia "Cargar por niveles".

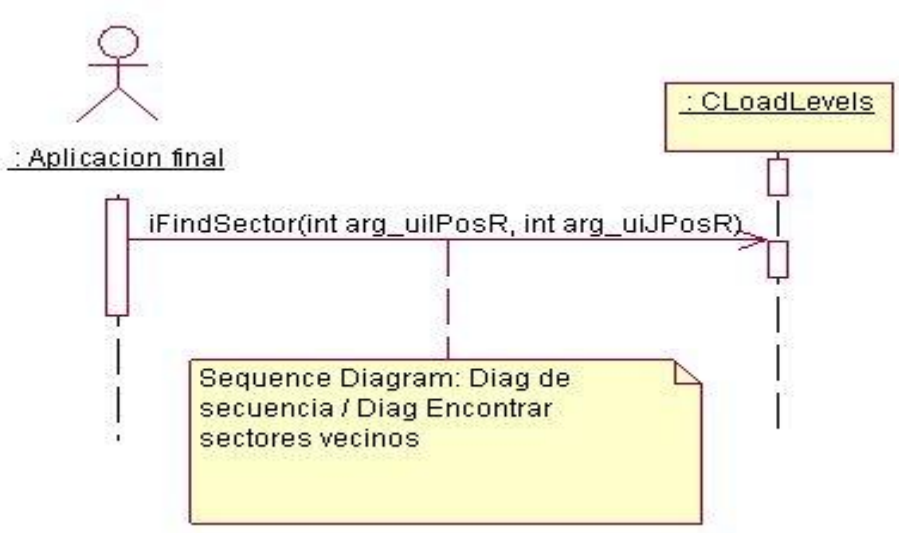

Figura 14 Diagrama de secuencia "Buscar sector".

Figura 15 Diagrama de secuencia "Encontrar sectores vecinos".

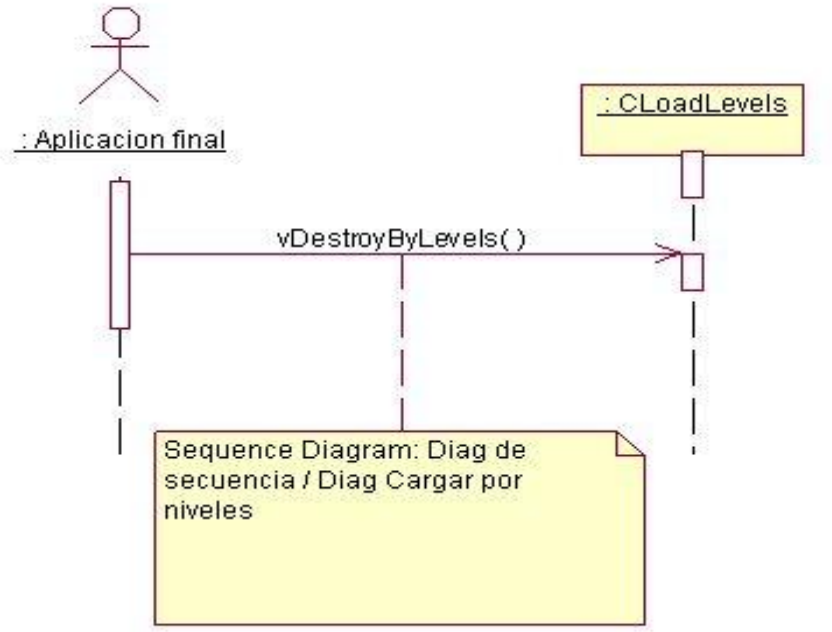

Figura 16 Diagrama de secuencia "Destruir sectores por niveles".

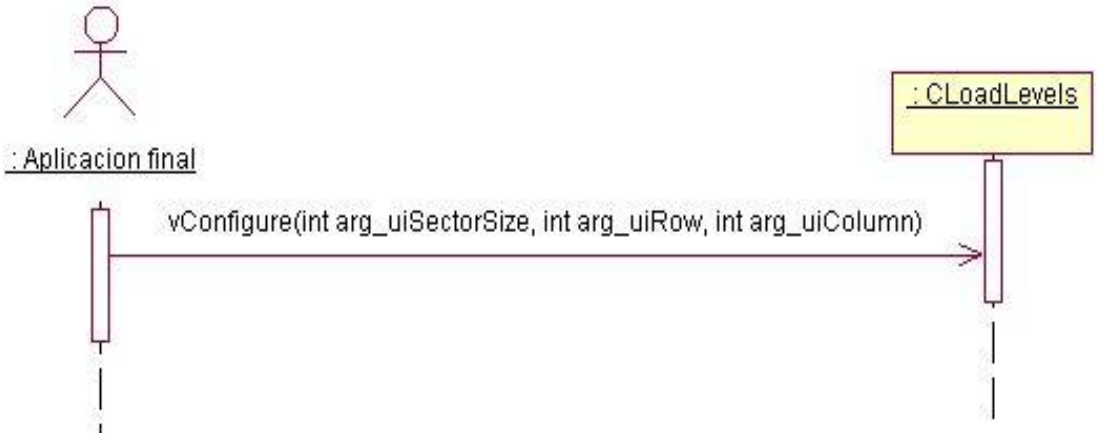

Figura 17 Diagrama de secuencia "Configurar rejilla".

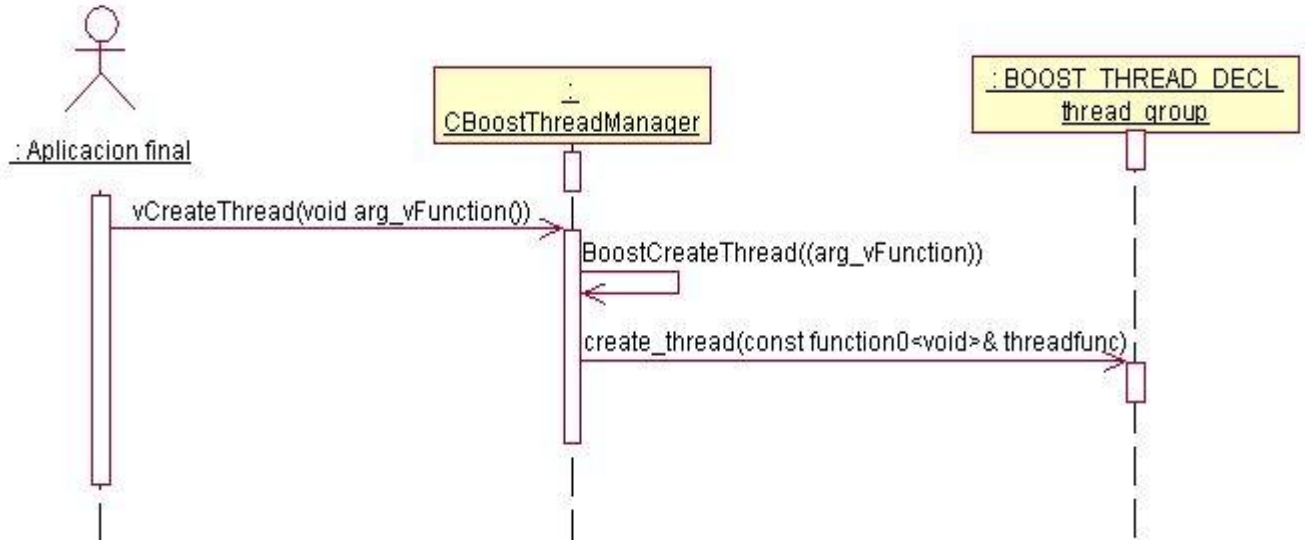

Figura 18 Diagrama de secuencia "Crear hilo".

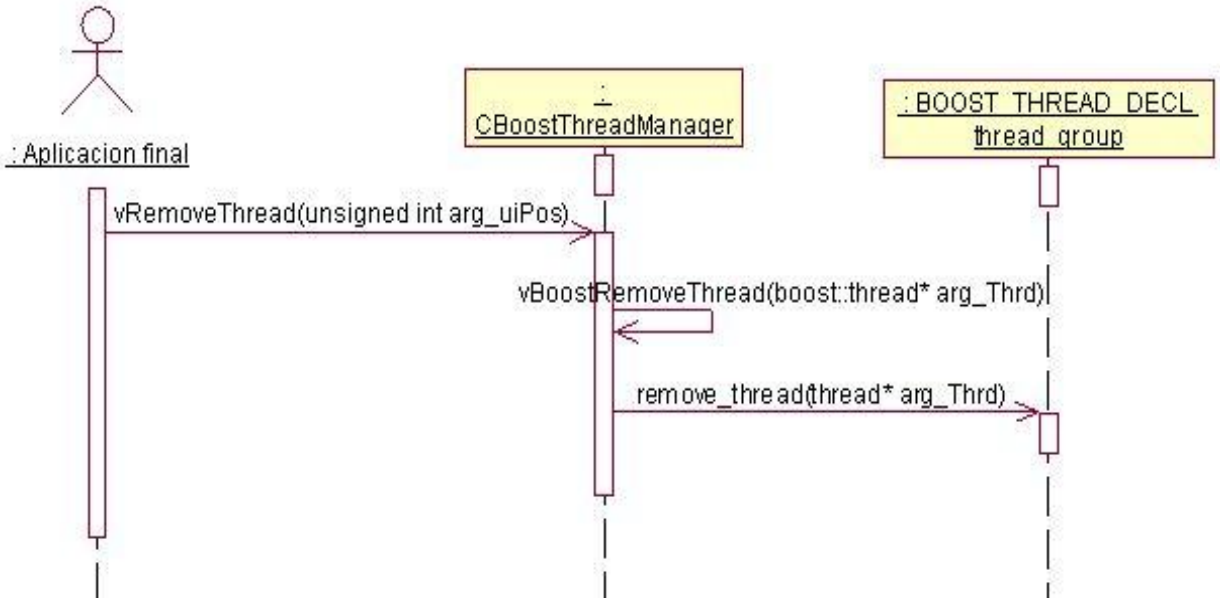

Figura 19 Diagrama de secuencia "Destruir hilo".

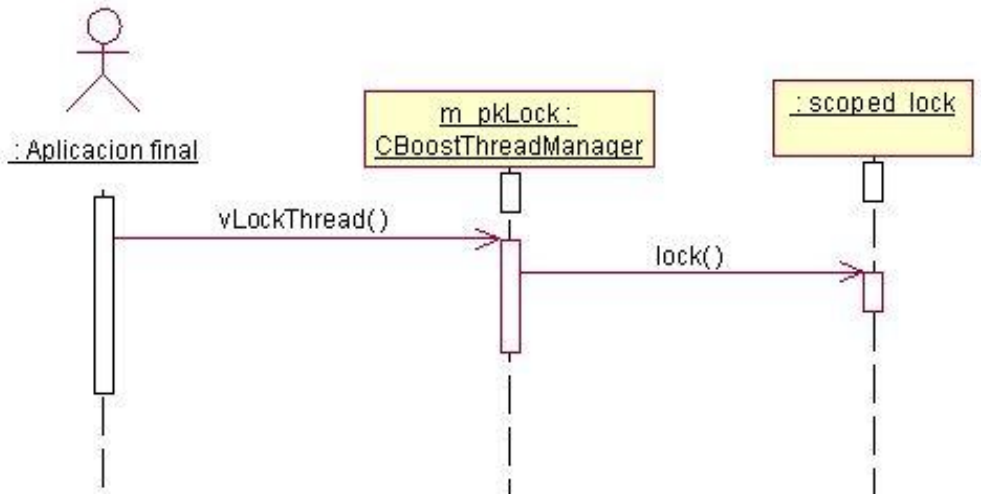

Figura 20 Diagrama de secuencia "Bloquear hilo".

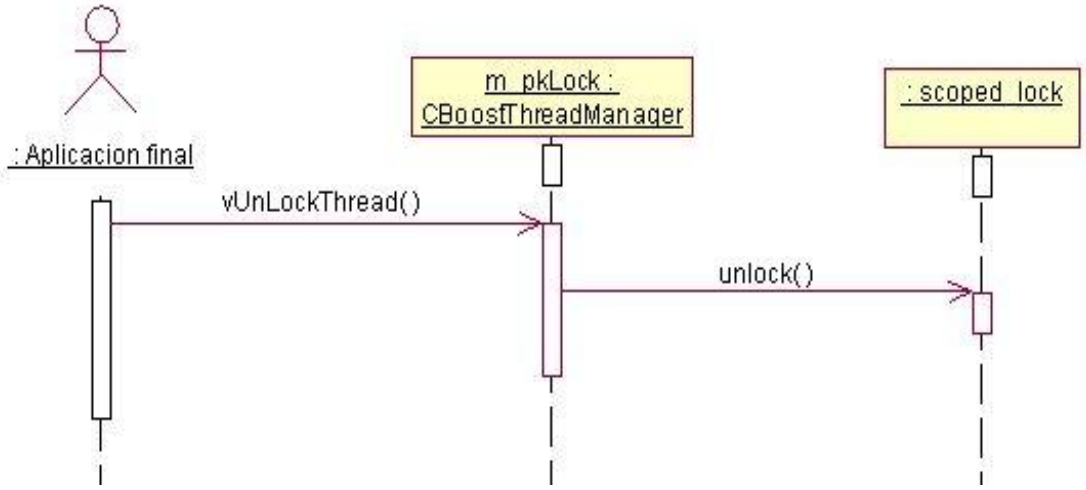

Figura 21 Diagrama de secuencia "Desbloquear hilo".

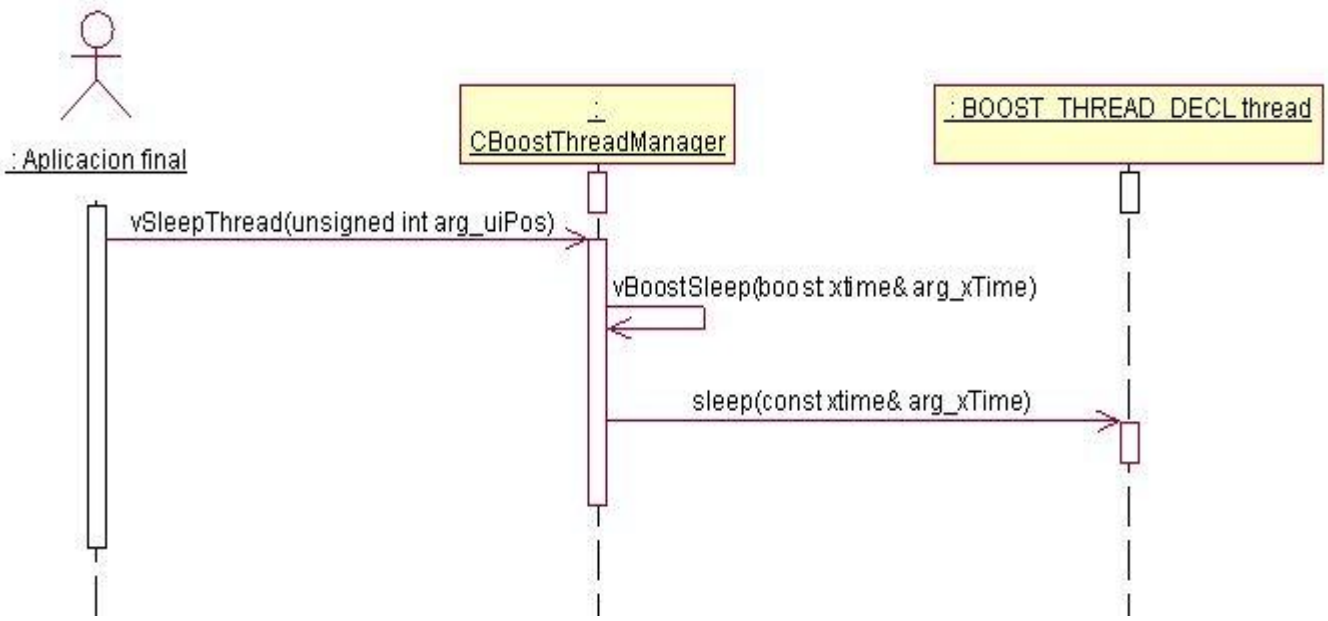

Figura 22 Diagrama de secuencia "Dormir hilo".

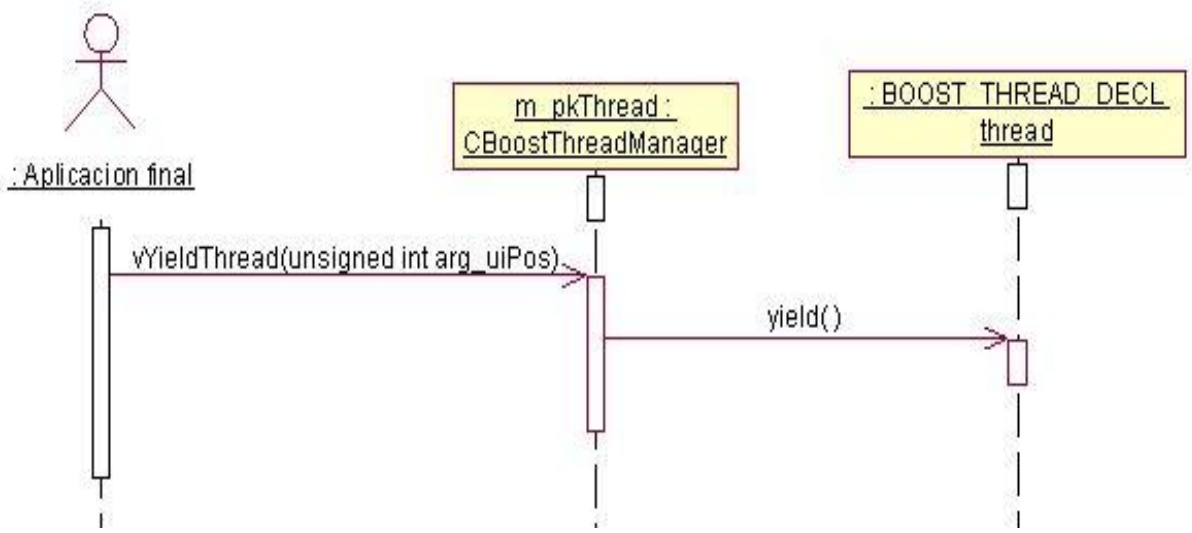

Figura 23 Diagrama de secuencia "Despertar hilo".

4.5 Diagrama de despliegue

Este diagrama consta de una sola computadora personal, por lo tanto se considera que no es necesario reflejarlo en este proyecto.

4.6 Diagrama de componentes

Este diagrama refleja la creación de componentes físicos, que se traducen en ficheros .h y ficheros .cpp correspondiente a la implementación en C++.

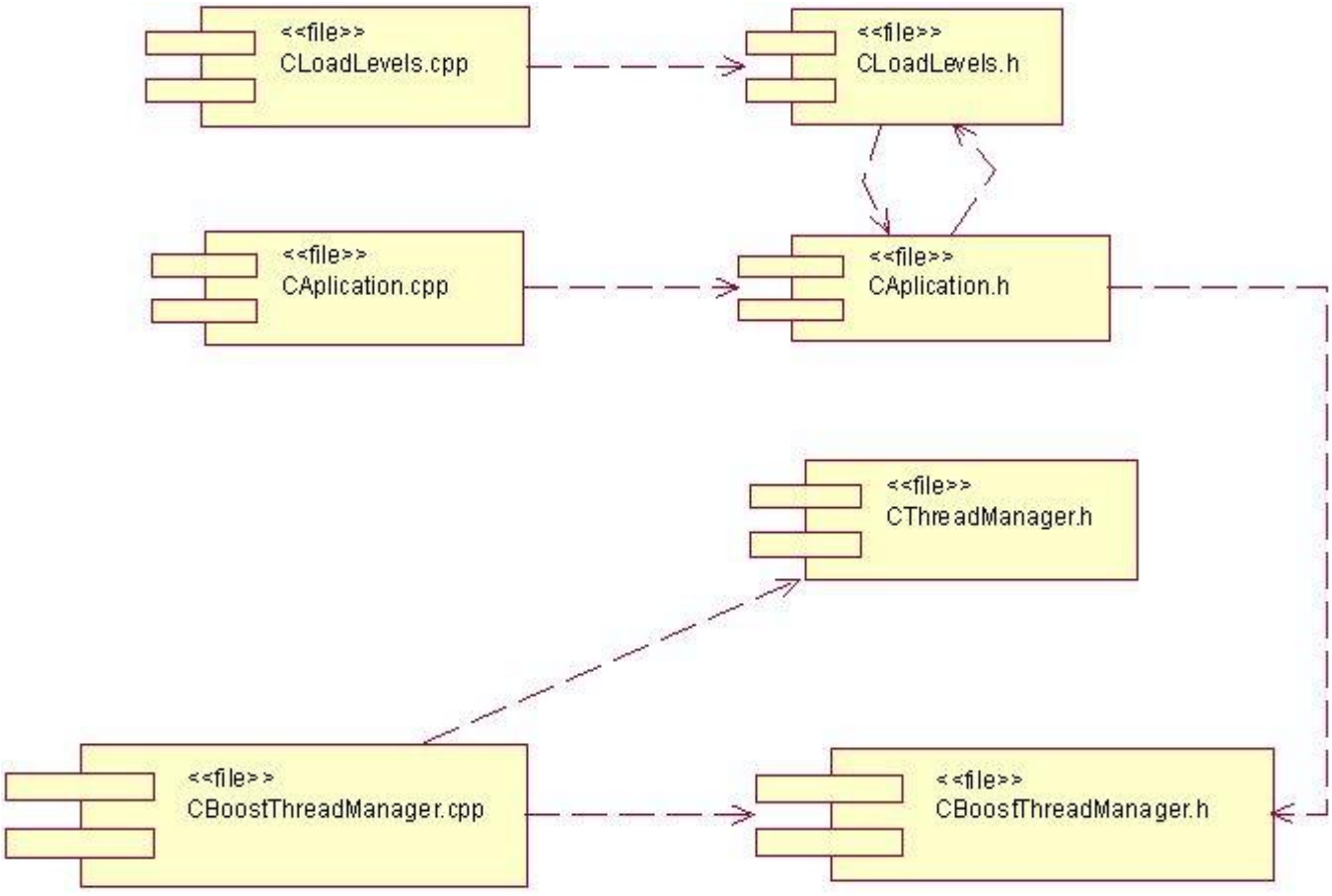

Figura 24 Diagrama de componentes.

Conclusiones del capítulo

A lo largo del este capítulo se muestran los artefactos necesarios para el desarrollo del módulo de carga dinámica. Se define una arquitectura compatible con la *SceneToolkit* y la mensajería entre las clases en forma secuencial.

Después se pasa a detallar cómo se harán físicas las clases de diseño, consolidando el proceso de desarrollo para pasar a la programación de los casos de uso correspondientes al primer ciclo de desarrollo.

Al concluir este capítulo se tiene concebido detalladamente el diseño completo del módulo.

Conclusiones generales

Dando cumplimiento al objetivo de este proyecto, y a los requisitos del cliente, se obtuvo el módulo de carga dinámica para la *SceneToolkit,* el cual brinda las funcionalidades básicas de carga y destrucción dinámica de entornos virtuales. Con esto se optimiza el proceso de visualización de entornos debido a que solo se mantiene en memoria aquellas texturas y geometrías que sean necesarias en cada instante, teniendo en cuenta que la carga o descarga de geometrías y texturas es en función de la posición que ocupa el objeto dinámico.

Otro resultado que se obtuvo con la realización de este trabajo fue darle soporte de trabajo multihilo a la herramienta *SceneToolkit,* ofreciendo un mayor rendimiento a las aplicaciones de realidad virtual.

Se realizó el acople del módulo de carga dinámica a la herramienta *SceneToolkit*. Por tanto, se puede asegurar que se obtuvo un producto totalmente funcional.

Recomendaciones

Se recomiendan los siguientes aspectos:

- ✓ Aplicar efectos visuales a la carga dinámica para mejorar el realismo en la aplicación.
- ✓ Automatizar la configuración de la rejilla regular en dependencia de las prestaciones del ordenador.
- ✓ Aplicar técnicas de optimización como impostores, para cambiar geometrías más complejas por otras más sencillas.

Bibliografía

1. **Polack, Trent.** *Focus on 3D terrain programing.* Ohio : Premier Press, 2003.

2. **Santamaría Peña, Jacinto, y otros.** *Motor gráfico interactivo para la visualización en tiempo real de gran volumen de información vectorial y texturizada.* La Rioja : s.n., 2005.

3. **J. Vénere, Marcelo, y otros.** *Editor de escenarios para aplicaciones de Realidad Virtual.* Buenos Aires : s.n., 2001.

4. **Aguilera García, A, Feito Higueruela, F. R. y Torres Cantero, J. C.** *Visualización de terrenos mediante niveles de detalles.* Jaén : s.n.

5. **Morer Camo, Paz, Naya Villaverde, Miguel Angel y Monzón Gómez, Luis.** *Creación de entornos 3D para un simulador de conducción de automóviles.* A Coruña : s.n., 2001-2003.

Referencias bibliográficas

1. **Maroto Ibánez, Joaquín.** *Metodología para la generación de entornos Virtuales distribuidos y su aplicación a simuladores de conducción.* Madrid : s.n., 2005.

2. http://www.ogre3d.org. *http://www.ogre3d.org.* [En línea] [Citado el: 20 de 11 de 2008.] http://www.ogre3d.org/about.

3. http://www.crystalspace3d.org. *http://www.crystalspace3d.org.* [En línea] [Citado el: 20 de 11 de 2008.] http://www.crystalspace3d.org/main/Main_Page.

4. http://www.openscenegraph.org. *http://www.openscenegraph.org.* [En línea] [Citado el: 20 de 11 de 2008.] http://www.openscenegraph.org/projects/osg.

5. **Catá, Jordi.** *Comparativa de motores gráficos para videojuegos.* 2002.

6. **Camacho Román, Yanoski Rogelio y Jiménez López, Fernando.** *Biblioteca gráfica para Sistemas de Realidad Virtual.* Ciudad de la Habana : s.n., 2004.

7. **Chehayeb, Nassouh, y otros.** *Representación realista de terrenos para simuladores de tiempo real.* Santander : s.n., 2002.

8. http://www.stackframe.net. *http://www.stackframe.net.* [En línea] [Citado el: 28 de 11 de 2008.] http://www.stackframe.net/es/content/11-2008/introduccion-la-programacion-multihilo.

9. **González, Bruno y Orduña, Pablo.** *Curso de introducción a videojuegos multiplataforma con SDL.* Deusto : s.n., 2004.

10. http://www.es.gnome.org. *http://www.es.gnome.org.* [En línea] 28 de 11 de 2008. http://www.es.gnome.org/Documentacion/Desarrollo/Glib/GlibAvanzado.

11. http://www.boost.org. *http://www.boost.org.* [En línea] [Citado el: 25 de 11 de 2008.] http://www.boost.org/doc/libs/1_34_1/doc/html/thread.html.

12. http://sourceware.org. *http://sourceware.org.* [En línea] [Citado el: 28 de 11 de 2008.] http://sourceware.org/pthreads-win32.

13. http://openthreads.sourceforge.net. *http://openthreads.sourceforge.net.* [En línea] [Citado el: 28 de 11 de 2008.] http://openthreads.sourceforge.net/index.html.

14. **Nogueira Collazo, Mariela y Correa Madrigal, Omar.** *Algoritmos para la manipulación eficiente de objetos dinámicos en escenas virtuales urbanas.* Ciudad de la Habana : s.n., 2007.

Glosario de abreviaturas

2D: Dos dimensiones o bidimensional.

3D: Tres dimensiones o tridimensional.

API: Interfaz de programación de aplicaciones. Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

BCP: Bloque de control de proceso

C++: Lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos (POO). C++ está considerado por muchos como uno de los lenguajes más potentes, debido a que permite trabajar tanto a alto como a bajo nivel.

CPU: Unidad central de procesos.

POO: Programación orientada a objetos.

PVS: Conjunto potencialmente visible.

RAM: Memoria de acceso aleatorio.

RUP: Proceso unificado de software.

RV: Realidad virtual.

SRV: Sistema de realidad virtual.

STK: *SceneToolkit*, herramienta para el manejo y representación de escenas tridimensionales.

Glosario de términos

Glosario de términos del dominio

Rejilla: Estructura de datos no jerárquica que se encarga de representar el entorno.

Hilo: Secuencia única de control de flujo de un programa.

Manejador de hilos: Secuencia de control de hilos.

Carga por nivel: Es aquella carga de entornos que se realiza de manera dinámica.

Aplicación: Vincula el manejador de hilos y la carga por nivel.

Glosario de términos generales

Carga dinámica: Carga de los entornos virtuales de manera continua y sectorizada.

Carga por niveles: Igual a la carga dinámica.

CPU: Componente en una computadora digital que interpreta las instrucciones y procesa los datos contenidos en los programas de la computadora.

Destrucción por niveles: Destrucción de los entornos virtuales de manera continua y sectorizada.

Fotogramas: Cada imagen estática que forma la secuencia animada.

Módulo: Componente autocontrolado de un sistema que posee una interfaz bien definida hacia otros componentes.

Objeto dinámico: Elemento del entorno virtual que tiene movilidad.

Polígono: Figura geométrica limitada por segmentos consecutivos no alineados, llamados lados.

RAM: Memoria de acceso aleatorio.

Realidad virtual: Término futurista el cuál pretende describir la interacción de los seres humanos en mundos virtuales o simulados.

Sectores vecinos: Aquellos sectores aledaños al sector donde se encuentra el objeto dinámico.

Sistema de Realidad Virtual: Sistema informático interactivo que ofrece una percepción sensorial al usuario de un mundo tridimensional sintético que suplanta al real.

Textura*:* Imagen que sirve de “piel” a los modelos en un mundo virtual**.**

Índice de figuras y tablas

Índice de figuras

Índice de tablas