UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5

Visualización de multitudes en tiempo real utilizando impostores

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

Autor: Yandris Rodríguez Rodríguez

Tutor: Ing. Alexey Broche Medina.
Ing. Mariela Nogueira Collazo.

Ciudad de la Habana

Julio, 2010

DATOS DE CONTACTO

Nombre y apellidos: Alexey Broche Medina.
Especialidad de graduación: Ing. en Ciencias Informáticas
Categoría docente: Instructor Recién Graduado
Categoría Científica: Ingeniero
Años de experiencia en el tema: 3 años
Años de graduado: 1 año
Correo electrónico: abroche@uci.cu

Nombre y apellidos: Mariela Nogueira Collazo.
Especialidad de graduación: Ing. en Ciencias Informáticas
Categoría docente: Instructor
Categoría Científica: Ingeniero
Años de experiencia en el tema:
Años de graduado: 3 años
Correo electrónico: mnogueira@uci.cu

Dedicatoria

A Olga, la mejor madre del mundo.

A Bryan, Antonio, Isel y Beatriz.

A Hilario, mi padre y amigo.

A Edecia, mi abuela linda.

Agradecimientos

A mi madre por todo su amor, apoyo y comprensión.

A mi padre por sus consejos, amistad y cariño.

A mis hermanas y hermanos por creer en mí.

A mis tíos y tías por confiar siempre en mí.

A mi abuela por todo su amor y ayuda.

A mi novia por su amor y paciencia.

A mis amistades por su ayuda.

A todos aquellos que contribuyeron de una forma u otra a la realización de este sueño.

Resumen

La Realidad Virtual (RV) se puede definir como la recreación por medio de los recursos de *hardware* y *software* de un mundo artificial, en la que los usuarios tienen una amplia participación por medio de diferentes dispositivos de entrada y salida. En la actualidad el desarrollo de la RV ha dado paso a nuevos mundos investigativos en aras de alcanzar el fortalecimiento de estos sistemas.

En esta tesis se abordan importantes conceptos, técnicas y *software* empleados para la visualización de multitudes, con el propósito de arribar a la solución del presente trabajo.

Como resultado de esta tesis se obtuvo un módulo para visualizar multitudes en tiempo real, haciendo uso de la técnica *Geopostors*, que facilita el mejoramiento de la calidad de visualización de multitudes.

Palabras claves

Impostor, Realidad Virtual, Visualización de multitudes.

Tabla de contenido

Introducción

La RV es simulación por computadora, dinámica y tridimensional, con alto contenido gráfico, acústico y táctil, orientada a la visualización de situaciones complejas, en la cual el usuario ingresa a través de la utilización de sofisticados dispositivos de entrada, a mundos que aparentan ser reales, resultando inmerso en ambientes altamente participativos de origen artificial. (1)

En nuestros días la RV ha alcanzado un desarrollo vertiginoso, esto ha dado paso a la incorporación de novedosas técnicas que han potenciado la modelación 3D de alta calidad, la confección de complejos modelos matemáticos capaces de simular sucesos del mundo real, entre otros elementos que han propiciado el incremento de la interactividad y el realismo en los entornos virtuales. Al mismo tiempo estos elementos implican que a la hora de poner en marcha una aplicación de RV puedan aparecer dificultades que obstaculicen su procesamiento en tiempo real y su visualización fluida, algunas de ellas son: la insuficiente velocidad de cómputo y la poca disponibilidad de almacenamiento.

La mayor parte de los sistemas de RV que se utilizan en la actualidad emplean polígonos gráficos. Esto significa que todos y cada uno de los objetos del universo virtual están formados por polígonos o superficies lisas con múltiples caras. Por tanto cuanto mayor sea el número de polígonos que se utilice, más suave será su aspecto. No obstante, considerándose que debe calcularse la localización y la forma de cada uno de los polígonos, al aumentar la cantidad de polígonos aumenta también la cantidad de cálculos necesarios para trazar un fotograma (cada imagen estática que forma la secuencia animada). (2)

En la Universidad de las Ciencias Informáticas existen proyectos que desarrollan *software* de RV. Estos grupos de trabajo se dedican a la creación de productos de *software* como: Paseos Virtuales y Simuladores, dichos productos carecen de eficiencia en la visualización de multitudes.

Actualmente uno de los productos de *software* que se desarrolla es el Entrenador Aduanero, el cual será el caso de estudio para el presente trabajo. Para su desarrollo se utiliza el motor gráfico (*graphics engine*) “OGRE” *(Object-Oriented Graphics Rendering)*.

En la visualización de las escenas del Entrenador Aduanero cuando los espacios visualizados están muy poblados de modelos geométricos, la velocidad de visualización disminuye considerablemente, por ejemplo: al cargar en escena 50 modelos, la velocidad de la aplicación disminuye cuantiosamente. No

importa la distancia a la que se encuentren los modelos dentro del campo de visión de la cámara, siguen visualizándose con la misma cantidad de polígonos, por tanto la resolución no varía, tornándose lenta la visualización e incómoda la navegación por el entorno simulado, restándole de esta manera calidad y viveza a la escena simulada y a la aplicación como tal.

Ante estos inconvenientes surge la siguiente interrogante: **¿Cómo mejorar el rendimiento en la visualización de multitudes en tiempo real?**

Se definió como objeto de estudio del presente trabajo: **las técnicas para la visualización de multitudes en tiempo real,** siendo el campo de acción**: las técnicas para la visualización de multitudes en tiempo real utilizando impostores.**

El objetivo de este trabajo es: **Desarrollar un módulo que permita visualizar multitudes en tiempo real**, de manera que se logre una mejor visualización y realismo de las multitudes que se simulen.

Para dar cumplimiento al objetivo planteado se definieron las siguientes **tareas a cumplir**:

- ✓ Estudiar las técnicas y *software* existentes usados para la visualización de multitudes.
- ✓ Proponer las soluciones técnicas a partir del estudio realizado de las técnicas y *software* existentes usados para la visualización de multitudes.
- ✓ Diseñar e implementar un módulo para visualizar multitudes en tiempo real.
- ✓ Acoplar el módulo al juego Entrenadores Aduaneros.

El presente trabajo está conformado por cuatro capítulos y se ha estructurado de la siguiente manera:

En un primer capítulo, **Fundamentación Teórica**, se hace un análisis bibliográfico que muestra las características de las principales técnicas que se emplean para visualizar multitudes en tiempo real, los productos de *software* más utilizados para la simulación, una breve descripción del juego Entrenadores Aduaneros y algunos detalles del motor gráfico que usa el juego. En el segundo capítulo, **Soluciones Técnicas**, se muestran las características del sistema a desarrollar como solución al problema planteado. En el tercer capítulo, **Descripción de la Solución Propuesta**, se hace un análisis con mayor profundidad del objeto de estudio, se define el modelo del dominio, los requisitos del sistema, el modelo de casos de uso del sistema, y se hace una descripción textual de cada uno de ellos. En el cuarto y último capítulo, **Diseño e Implementación del Sistema**, se presentan los diagramas de clases de diseño y una descripción de los mismos, los diagramas de secuencia y el diagrama de componentes.

Capítulo 1 Fundamentación teórica

Introducción

En el presente capítulo se explica qué es la simulación de multitudes en tiempo real, el comportamiento de multitudes como parte del proceso de simulación, y dentro de la simulación, se centrará fundamentalmente en los conceptos referentes a la visualización. Se ofrecen ejemplos de las principales técnicas y *software* de desarrollo existentes para la visualización de multitudes en los sistemas de RV, así como su estudio y caracterización. Se brinda además una pequeña descripción del juego Entrenadores Aduaneros, con el cual el módulo va a interactuar, y por último una breve presentación de las características del motor gráfico Ogre.

1.1 Simulación de multitudes en tiempo real

La simulación en tiempo real y orientado a la RV se puede definir como la reproducción o recreación artificial de procesos que se dan en la realidad, o la técnica que imita el funcionamiento de un sistema del mundo real en el tiempo. (1)

Ahora, uniendo este concepto al término de multitudes se puede decir que la simulación de multitudes en tiempo real es la recreación artificial o simulación de una población o grupos de personas enmarcadas en un entorno virtual, reflejando su comportamiento y movimiento en el mismo.

Como parte también del concepto de simulación de multitudes se verá la visualización o representación de multitudes y su comportamiento como parte del proceso de simulación.

1.1.1 Visualización de multitudes

La visualización de multitudes orientada a la RV y enmarcada en términos humanos se puede definir como la representación de un grupo de modelos geométricos (modelos creados en programas de diseño 3D) o fotos, que representen figuras humanas (plano que tiene como textura la foto de un modelo humano) en un entorno virtual determinado, recreando así una población de individuos. La visualización utilizando modelos geométricos o imágenes, está dada por el objetivo que se persiga y la sensación de realismo que se le quiera dar a la escena. Por lo general los modelos geométricos se emplean cuando se va a representar una multitud pequeña, pues se requiere de muchos recursos, tanto de *software* como de *hardware*. Los modelos geométricos pueden combinarse con fotos que representen modelos humanos para usarse en medianas multitudes, así como en multitudes de gran volumen.

1.1.2 Comportamiento de multitudes

El comportamiento de multitudes se puede ver como el conjunto de acciones y reacciones de los individuos que representan una determinada población, así como las interacciones de dichas multitudes con el entorno virtual en que se encuentran.

El comportamiento de las multitudes depende del comportamiento de cada individuo con sus semejantes y del medio que los rodea. Para lograr implementar el comportamiento realista de una multitud se debe comenzar por la unidad mínima: el individuo. La capacidad de percibir, adaptarse, y reaccionar con su entorno; entre otras capacidades, como la de ser consciente de los cambios que se producen a su

alrededor y ser capaz de responder a ellos de manera autónoma, son el conjunto de capacidades que se le asignan a dicho individuo para exhibir gráficamente un comportamiento como el de un ser vivo. (3)

Son los algoritmos de planificación de caminos, detección de colisiones, entre otros, los que les dan a los individuos simulados la capacidad de desplazarse de un lugar a otro y evitar colisiones entre ellos y con los objetos del entorno. Todo esto va acompañado de la animación, técnica con la cual se crea la ilusión de movimiento, que puede ser generada mediante la forma tradicional, o sea, creada por un animador u obtenida a partir de la captura de movimiento. (3)

1.2 Principales Técnicas existentes para la visualización de multitudes

Hoy en día se emplean numerosas técnicas para la visualización de multitudes, a continuación se expondrán las principales y sus características.

1.2.1 Imágenes Basadas en *Render* (Impostores)

La idea básica de la técnica Imágenes Basadas en *Render* (por sus siglas en inglés “IBR”) se centra en poner la foto de un modelo en un plano rectangular, creándose así un impostor, de este modo se hace uso de un solo polígono por humano. Dichos impostores pueden ser estáticos o dinámicos.

Impostor estático

La construcción de un impostor estático consiste en tomarle fotos a un modelo desde diferentes puntos de vista (Fig. 1). A las fotos tomadas se les aplica un canal de transparencia y luego se almacenan en la memoria de textura (4), donde en tiempo de ejecución, dependiendo de la posición del impostor en relación con el espectador, la imagen más adecuada del impostor se selecciona y se muestra en el cuadrilátero que está orientado de forma dinámica hacia el espectador(*Billboard*) (5).

Impostor dinámico

La generación de un impostor dinámico parte del principio de la construcción de un impostor estático, excepto que las fotos tomadas se realizan desde diferentes puntos de vista para cada cuadro de la animación de un modelo. El almacenar fotografías de cada uno de los cuadros de la animación en la memoria de textura y seleccionar la imagen para cada cuadro en tiempo de ejecución, permitirá visualizar figuras animadas para diferentes puntos de vista. (4)

Unas de las cualidades que le brinda ventaja a la técnica impostores es el uso del *hardware* programable, que permite generar las texturas de los impostores de forma dinámica, brindándole más realismo a la escena.

Esta técnica aumenta considerablemente la velocidad de visualización y el resultado es lo suficientemente rápido como para generar grandes multitudes; (4) pero presenta los siguientes inconvenientes:

Consume gran cantidad de memoria en textura, pues para cada cuadro de la animación de un impostor se requiere una imagen.

La variedad de animaciones que pueden ser mostradas resulta muy restringida.

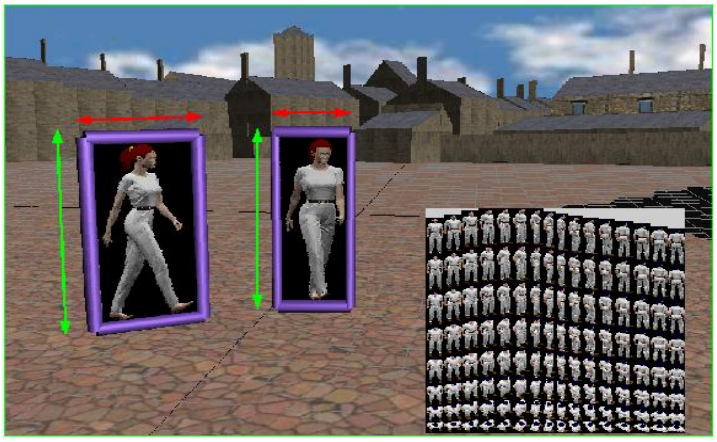

Figura 1: Imágenes Basadas en *Render* (IBR). (4)

1.2.2 Polypostors

El principio de esta técnica está basado en el uso de modelos poligonales 2D animados de 90 triángulos cada uno (Fig. 2); pero a diferencia de los impostores que almacenan una imagen por cada cuadro de una animación, los *Polypostors* están animados por el desplazamiento de los vértices del polígono 2D, usando solo una textura para toda la animación. El desplazamiento de los vértices se almacena de forma muy compacta, obteniéndose como resultado un uso más eficiente de la memoria. (6)

La obtención de un *Polypostor* está dada por un algoritmo que recibe como entrada un modelo 3D animado y como salida proporciona la conversión del mismo a un modelo 2D. La idea básica del algoritmo se centra en cortar el modelo en varias partes (Fig. 3), donde para el primer cuadro de la animación cada parte del cuerpo es renderizada y convertida en un polígono 2D texturizado. Para los fotogramas restantes un algoritmo basado en la programación dinámica cambia los vértices de los polígonos 2D, de modo que

se aproximen a la imagen actual renderizada en la mayor medida posible. En tiempo de ejecución los polígonos deformados están compuestos en orden de profundidad, creando la ilusión de una animación de personajes 3D. (6)

La semejanza que mantiene esta técnica de visualización con respecto a la técnica de impostores viene dada en que por cada punto de vista (ángulo) en el que se encuentre el modelo respecto a la cámara se generará una animación, además de usar el *hardware* programable para obtener una variación de textura de forma dinámica.

La técnica *Polypostors* logra la optimización de los recursos de la estación de trabajo, aunque computacionalmente es muy costosa. A continuación en la Fig. 4 se muestra una gráfica que ofrece una breve comparación entre impostores, modelos geométricos y *Polypostors*. (6)

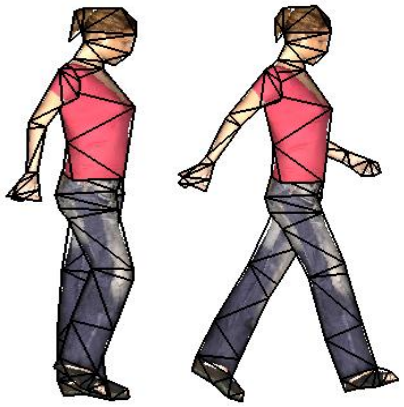

Figura 2: Animación con polígonos 2D. (6)

Figura 3: Fragmentación de un modelo. (6)

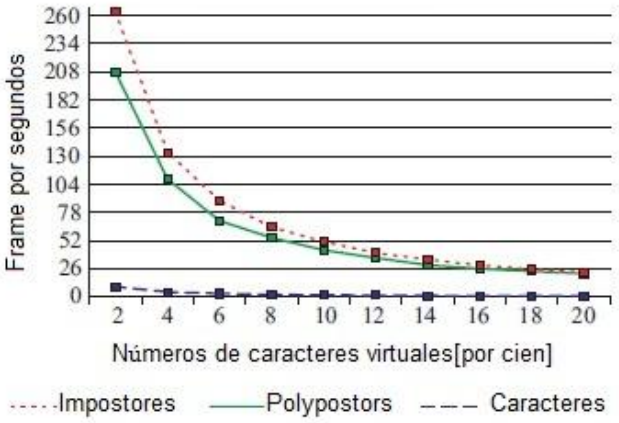

Figura 4: Modelos virtuales contra Fps. (6)

1.2.3 Modelos geométricos usando niveles de detalles (LOD)

Unos de los requisitos en los sistemas interactivos para la trama en tiempo real es el número limitado de polígonos que puede ser mostrado por un motor gráfico en cada fotograma de una simulación. Por lo tanto, las mallas con un alto número de polígonos tienen que simplificarse a fin de lograr tasas de pantalla aceptables.

Una de las técnicas para la visualización de multitudes es la creación y utilización de modelos geométricos usando niveles de detalles, la misma se basa en los niveles de detalles de la malla del modelo de acuerdo a la distancia de visualización que se encuentre de la cámara, es decir, se utilizaría el detalle alto (modelos de malla compleja o gran cantidad de polígonos) para modelos que se encuentren cerca del ángulo de visión de la cámara, nivel de detalle medio(modelos de malla de complejidad media) para modelos que se encuentren a una distancia media del ángulo de visión de la cámara y un bajo nivel de detalle para modelos que se encuentren lejos del ángulo de visión de la cámara. En síntesis, el nivel de detalle se centra en la importancia visual que tenga el modelo en la escena (Fig.5). (7)

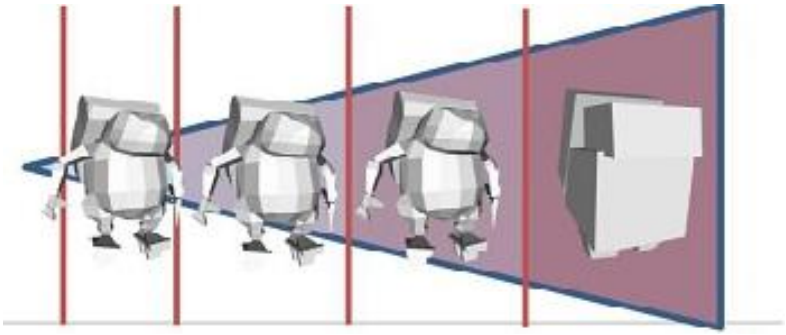

Figura 5: LOD. (7)

Existen actualmente tres técnicas para el manejo de los niveles de detalles, las cuales son:

LOD discreto

El LOD discreto, llamado también LOD basado en rangos es el más conocido. Consistente en crear múltiples versiones para cada objeto (modelo geométrico) durante un proceso *offline* o de pre-procesamiento, cada una de ellas correspondiente a cada nivel de detalle, y en tiempo de ejecución se selecciona la versión adecuada. Este método tiene como ventaja que el proceso de simplificación puede demorar lo que sea necesario para generar los LOD, además, el *hardware* gráfico moderno permite la creación de versiones de LOD, los cuales pueden ser compilados en pre procesamiento, adaptando las

estructuras de los modelos a determinadas ventajas del *hardware* como las tirillas de triángulos (*triangle strips*), listas de visualización (*display lists*) o arreglos de vértices (*vertex arrays*). Como esta es una operación pre-calculada, no se puede predecir desde qué dirección se estará viendo el objeto, por lo que este tipo de LOD se conoce también como isotrópico o independiente de la vista. (8)

LOD continuo o progresivo

Este método, es un tipo de LOD que parte de la idea del LOD discreto, con la diferencia de que el nivel de detalle de cada objeto es calculado en tiempo real cuando es necesario, en lugar de ser pre-procesado. El funcionamiento real consiste en pre-procesar el objeto para crear una estructura asociada al mismo, que contiene el espectro de detalles, la cual será utilizada en tiempo real para extraer los parámetros necesarios para el nivel deseado y modificar el objeto.

De esta manera, nunca se tienen más polígonos de los necesarios pues se obtienen mejores aproximaciones a la cantidad de polígonos necesaria para cada nivel de detalle. (8)

LOD dependiente de la vista

El LOD dependiente de la vista es una extensión del LOD continuo o progresivo usándose un criterio de simplificación dependiente del punto de vista del observador, donde se selecciona dinámicamente el LOD más apropiado para la vista actual. Como es un modelo aniso trópico, un simple objeto puede abarcar múltiples niveles de simplificación; por ejemplo, las partes más cercanas del objeto se pueden representar con mejor detalle que las más lejanas (óptimo para objetos grandes y complejos como terrenos) o la silueta se puede representar con mejor resolución que las regiones interiores. (8)

1.2.4 Geopostors

La técnica *Geopostors* es una combinación de las técnicas de imágenes basadas en *render* y la geometría para los seres humanos virtuales. La misma se centra en los principios de las técnicas antes mencionadas, donde el cambio de un impostor por un modelo geométrico y viceversa se hace mediante una conversión de *pixel* a *texel* para evitar un salto brusco frente a la cámara a la hora de efectuarse el cambio. Esta técnica es usada para generar grandes multitudes generalmente y usa las posibilidades que brinda el *hardware* programable para aplicarles la textura a los impostores generados. Así mejora el realismo y la variedad de los impostores de forma dinámica, pues el almacenamiento de textura para cientos de impostores es algo costoso. (5)

Para cada cuadro de animación se generan fotos de 17 puntos de vista por 8 elevaciones, en un intervalo de 0-180 grados en el caso de los laterales y de 0-90 en el caso de las elevaciones (Fig. 6), para una animación de caminado cíclica simétrica, con el objetivo de ahorrar la cantidad de textura en memoria; de lo contrario se deberían tomar fotos en los 360 grados para los laterales del modelo y en los 180 para las elevaciones. Las fotos tomadas al modelo son almacenadas en una sola imagen de 1024 x 1024 (Imagen de Mapa Normal), equivalentes a un cuadro de animación (Fig. 7), donde según la posición del impostor respecto a la cámara se selecciona la imagen correspondiente y se pega en un *Billboard.* (5)

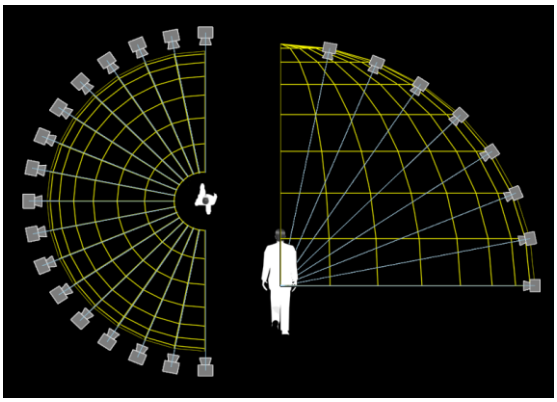

Figura 6: Generación de un impostor de 17x8 puntos de vista (5).

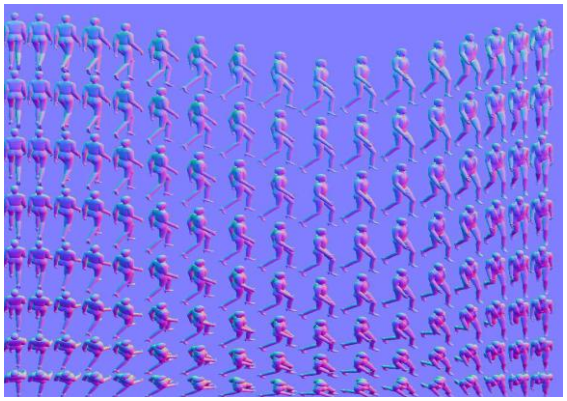

Figura 7: Imagen de Mapa Normal para un cuadro de animación. (5)

1.3 Productos de *software* existentes para la simulación de multitudes

En la actualidad existen diversos productos de *software* para simular y visualizar multitudes; ya sea en tiempo real para simular diferentes procesos de la vida diaria y el comportamiento o reacción de determinada cantidad de personas en un entorno dado, o los productos de *software* desarrollados para la industria del cine, que no simulan ni visualizan multitudes en tiempo real; pero en esencia se basan en los mismos principios que las aplicaciones que lo hacen en tiempo real. De acuerdo a la investigación realizada los productos de *software* más utilizados para realizar este tipo de trabajo de visualización son los que se mencionan a continuación. Dichos productos, a pesar de presentar el inconveniente de ser *software* propietario, fueron estudiados debido a que podían servir de apoyo a la investigación.

1.3.1 Software Legion

Legion es una de las herramientas líder en el sector de la simulación peatonal(Fig. 8), simula en detalle actividades como: llegadas de trenes, autobuses, colas, compra de tickets, utilización de cajeros automáticos, pasos por controles de seguridad, uso de los locales comerciales, esperas, utilización de escaleras, ascensores, pasillos o cruces señalizados, etc. Para esto usa la simulación microscópica (son las interacciones entre los individuos las que determinan el comportamiento de una masa) cuyo algoritmo está basado en las mediciones tomadas del mundo real, prediciendo el modo en el que miles de peatones se desplazarán en la realidad. (9)

En cuanto a la micro-navegación de *Legion* se tiene que las entidades inteligentes poseen características humanas como: las preferencias personales, limitaciones físicas, objetivos, conciencia de las circunstancias, memoria, negociación con otros, sincronización, conciencia del tiempo y percepción del comportamiento de otros, características que influyen en la manera en que tomamos decisiones acerca de los movimientos, y es el conjunto de estas decisiones lo que determina el comportamiento de las masas. Otras de las características de *Legion* es que sus entidades se desplazan por un espacio continuo, libre de limitaciones artificiales y el movimiento es vectorial, su modelo matemático trabaja con agentes individuales en un espacio continuo, maneja planos de múltiples niveles, a la hora de tomar decisiones los agentes no están sujetos a leyes físicas particulares, rejillas, reglas o cualquier otra asunción artificial, centrando sus decisiones en la comodidad, el esfuerzo y el tiempo empleado para llegar a su objetivo. (9)

Los usuarios de *Legion*, en lugar de tener que programar el comportamiento de los peatones al caminar, sólo tienen que seleccionar un perfil de peatones específico de una lista.

Figura 8: *Software Legion*. (9)

1.3.2 Software Massive

Massive es un programa que sirve para animar/simular multitudes en escenas en las que intervienen decenas, cientos o miles de personajes, de manera que sus reacciones parezcan naturales y aleatorias.

Para la animación tiene en cuenta lo que observa cada personaje, sus intenciones, sus rasgos básicos de personalidad y la situación geográfica para evaluar una serie de reacciones en forma de movimientos, para que la interacción entre estos personajes irreales produzca escenas visualmente lógicas y reacciones aparentemente realistas. Este programa tiene la capacidad de cargar cualquier modelo hecho en cualquier software 3D y crear una red de reacciones, objetivos y personalidades para dotar de reacciones aleatorias a los distintos personajes (Fig. 9). Si estos personajes son humanos, se tendrá que contar con una buena cantidad de capturas de animación real, adaptarla a los personajes y configurar que movimiento se corresponde a cada grupo de reacciones para que el programa los vaya mezclando entre sí. De esta manera se consigue una animación continua y fluida. (10)

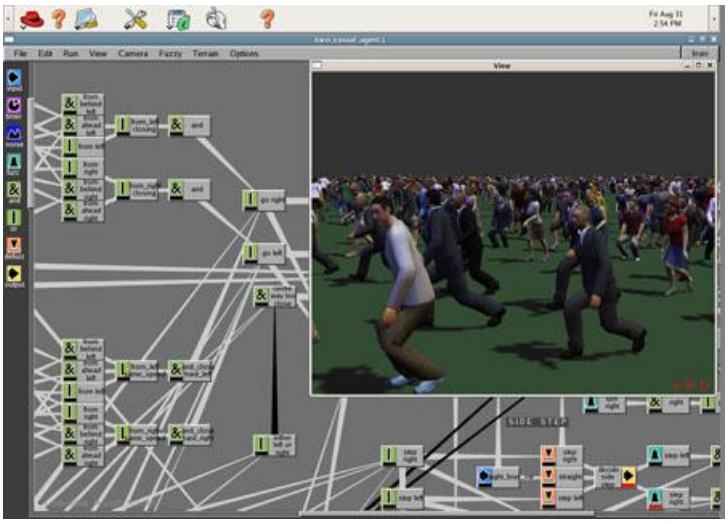

Figura 9: *Software Massive*. (10)

1.4 *Software* con los que se relacionará el módulo a implementar

El módulo a desarrollar será probado en el juego Entrenadores Aduaneros y utilizará las funcionalidades del motor gráfico Ogre para visualizar, de ahí que mantendrá una estrecha relación con dichos productos de *software*. A continuación se ofrece una breve descripción de estos.

1.4.1 Motor Gráfico Ogre

Para dar una pequeña descripción de Ogre, se comenzará con definir qué es un motor gráfico. El término "motor gráfico" (*graphic engine*) es ampliamente utilizado en todo el mundo, principalmente por programadores de videojuegos.

Es un conjunto de clases o bibliotecas que proporcionan funciones de redibujado 2D y 3D, se suele apoyar en bibliotecas/APIs gráficas como *OpenGL* o *DirectX* y define una capa de abstracción entre el *hardware*, el sistema operativo y el videojuego.

Quedado enunciado ya el concepto de motor gráfico, pasamos a dar una breve descripción de las características de Ogre.

Ogre es un motor gráfico desarrollado en C++, multiplataforma, libre y diseñado para hacer más fácil e intuitiva la creación de juegos y aplicaciones de RV, 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. (11)

1.4.2 Entrenadores Aduaneros

El juego Entrenadores Aduaneros es un juego desarrollado en la Facultad 5 en el Centro de Informática Industrial (CEDIN). En dicho juego se simula el comportamiento de una multitud de personas durante su estancia en un aeropuerto.

El objetivo del mismo es entrenar a los agentes de la aduana cubana para que sean capaces de detectar cualquier anomalía en el comportamiento de los pasajeros que entran y salen del aeropuerto, y evitar con ello la entrada de materiales peligrosos, el tráfico de drogas o cualquier otra actividad que ponga en peligro la integridad del país.

Este se divide en 4 niveles, en los que se refleja la llegada de los pasajeros, la recogida de equipajes, el chequeo en las puertas y la salida de los mismos.

Conclusiones del capítulo

De acuerdo al problema científico, a la situación problemática planteada anteriormente y el estudio realizado de las principales técnicas y *software* utilizados para la visualización de multitudes en tiempo real, se vuelve imperante la necesidad de desarrollar una técnica para visualizar multitudes en tiempo real a partir de las antes vistas.

De las técnicas tratadas, la más relevante es la de *Geopostors*, por sus ventajas en cuanto al realismo y la calidad de visualización que brinda por encima de las demás técnicas.

Capítulo 2 Soluciones técnicas

Introducción

A partir del estudio y el análisis realizado de las técnicas vistas en el capítulo anterior, en el presente capítulo se exponen las soluciones técnicas definidas para satisfacer el objetivo del trabajo.

Se describen los pasos de la técnica *Geopostors* para lograr la visualización de multitudes en tiempo real, así como los principales métodos para realizar la visualización y una breve explicación de cómo acoplar nuestra solución técnica al juego Entrenadores Aduaneros.

2.1 Información que se maneja

La información que se maneja es toda la referente a los atributos que permitirán la visualización de multitudes en tiempo real utilizando impostores.

2.2 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 (12).

El criterio antes expuesto se tiene en consideración a la hora de realizar el intercambio entre el modelo geométrico y el impostor.

2.3 Técnica para visualizar multitudes

Teniendo en cuenta las características y ventajas de las técnicas estudiadas y analizadas en el capítulo anterior, se decidió implementar una técnica híbrida, esta se basa en la combinación o solapamiento de impostores y modelos geométricos, conocida actualmente como *Geopostors*. Esta técnica no es muy costosa computacionalmente, es fácil de implementar, brinda una mejor calidad y realismo a las escenas de grandes multitudes por encima de otras técnicas existentes y mejora considerablemente la eficiencia de la visualización de multitudes. El módulo a desarrollar tendrá las siguientes características:

- Generación del modelo geométrico
- Generación del impostor.
- Cambio entre el humano virtual y el impostor.

2.3.1 Generación del Modelo Geométrico

La fase de construcción de un modelo geométrico parte del diseño a lápiz. Una vez obtenido un esbozo del modelo que se quiere construir, se pasa a hacer uso de la herramienta de modelado 3D Studio Max.

A partir de una primitiva (Fig.10), mediante las acciones de escalar, mover, rotar y extrudir, se van manipulando los vértices, aristas, caras y polígonos hasta llegar a la figura deseada. Ya construida la malla se pasa a darle vida a nuestro personaje, ajustándole un esqueleto previamente animado y asignándole un material.

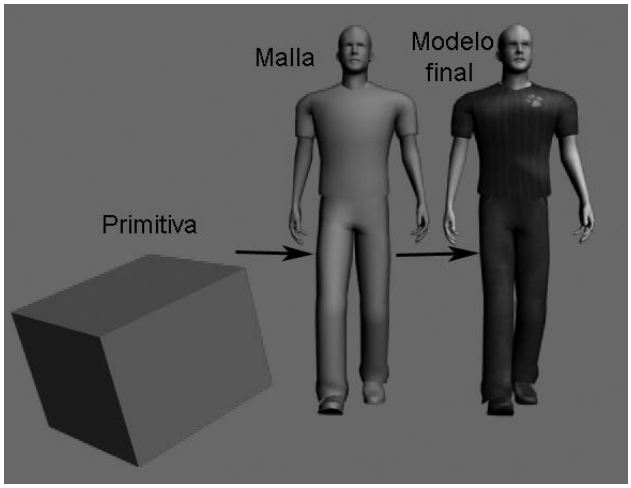

Figura 10: Generación del modelo geométrico a partir de una primitiva.

2.3.2 Generación del Impostor

A partir de un modelo geométrico previamente diseñado en un programa de diseño 3D, en este caso 3D Studio Max, se obtienen varias fotos del modelo desde diferentes puntos de vista, para este caso en particular, se saca para un modelo 3D 16 imágenes de 54x126, en un intervalo de 0 a 180 grados con 12 grados de diferencia entre cada ángulo. (Fig.11)

Esto se realiza para cada cuadro de una animación de caminado cíclico de 10 cuadros de animación, con una cámara a una distancia media del modelo.

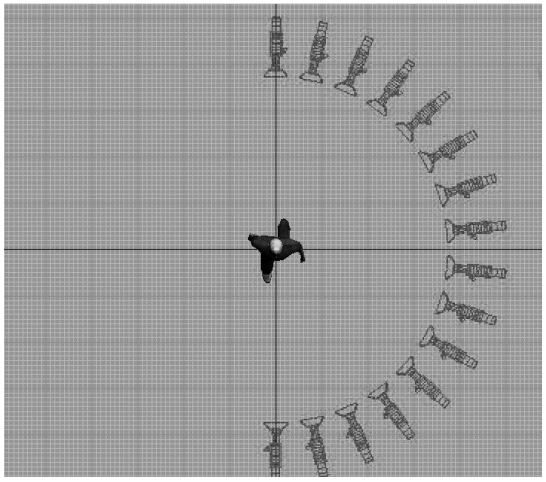

Figura 11: Generación del impostor desde 16 puntos de vista.

Para evitar tomar fotos en un rango de 0 a 360 grados y minimizar el uso de la memoria en cuanto al almacenamiento de textura, las posiciones y el caminado del modelo se hacen de forma simétrica. Estas fotos son almacenadas en una imagen de mapa normal (IMN) de 216x504 (Fig.12). Esta IMN se salva con la extensión .png. Se elige este tipo de extensión por la calidad que presenta además de tener un canal alfa o canal de transparencia, aunque pueden usarse otros tipos de extensiones siempre y cuando se le agregue un canal de transparencia. La imagen representativa del modelo geométrico es puesta en un *Billboard* en correlación a la posición del modelo respecto a la cámara, para así obtener al impostor.

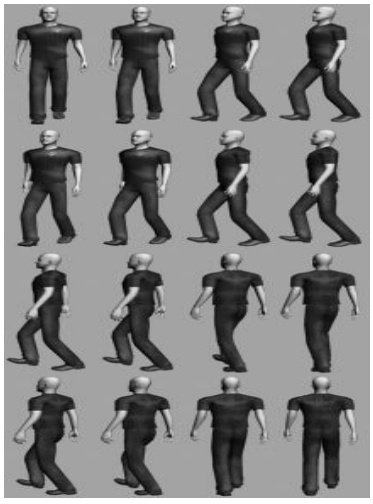

Figura 12: Imagen para un cuadro de animación.

2.3.3 Cambio entre el humano virtual y el impostor

Para lograr un cambio sin que exista una transición brusca y ponga en peligro la calidad visual y el realismo de la escena que se quiere obtener, es necesario que los bordes de los impostores tengan la mayor aproximación posible a la frontera de la imagen y el *Billboard* coincida con el ancho y el alto del modelo geométrico (Fig.13). El intercambio entre modelo e impostor se realiza a una distancia media de acuerdo a la importancia visual del modelo.

Para que las animaciones del impostor y del modelo coincidan y se evite un salto en la animación a la hora del cambio, se hacen coincidir los números de cuadros tomados como fotos al modelo original, con los cuadros de la animación almacenada por el modelo en el fichero cuando es exportado.

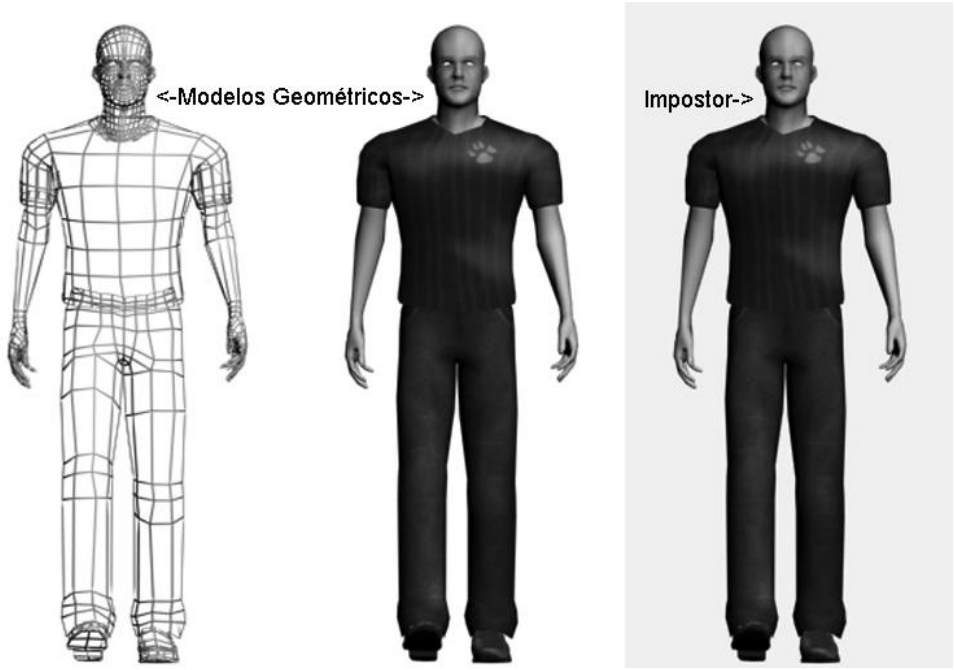

Figura 13: Tamaño del impostor y el modelo.

2.4 Principales métodos para realizar la visualización de multitudes

El trabajo para la visualización de multitudes se realizará con la modelación e implementación de las siguientes funciones:

Cargar imágenes: Esta función permite cargar la imagen para cada cuadro de la animación del impostor.

Inicializar secuencia animada: Esta función permite inicializar una secuencia de imágenes para cada animación del modelo.

Inicializar impostor: Esta función permite inicializar un impostor con todos sus atributos.

Inicializar modelo geométrico: Esta función permite inicializar un modelo geométrico con todos sus atributos.

Actualizar secuencia animada: Esta función actualiza la secuencia animada para cada animación dependiendo del punto de vista.

Actualizar impostor: Actualiza constantemente cuadro a cuadro qué elemento se está pintando, el material y la traslación.

Actualizar modelo geométrico: Actualiza constantemente si el modelo está visible, así cómo las animaciones, la posición, la traslación y la rotación de dicho modelo.

Mostrar impostor: Esta funcionalidad nos permite visualizar el impostor según la importancia visual definida.

Mostrar modelo geométrico: Esta funcionalidad nos permite visualizar el modelo según la importancia visual definida.

Finalizar secuencia animada: Esta función permite terminar la secuencia de imágenes del impostor liberando la memoria ocupada.

Finalizar impostor: Esta función permite terminar la visualización del impostor, liberando la memoria ocupada.

Finalizar modelo geométrico: Esta función permite terminar la visualización del impostor, liberando la memoria ocupada.

2.5 Acople del módulo de visualización de multitudes al producto Entrenadores Aduaneros

Para realizar el acople del módulo de visualización de multitudes al producto Entrenadores Aduaneros se insertan las clases dentro del producto, luego se vinculan las clases del módulo con la clase del producto, con la cuál interactuarán, agregándole así las funcionalidades de dicho módulo al juego. Para lograr tal vinculación, la clase del producto Entrenadores Aduaneros que construye el modelo geométrico, crea un impostor por cada modelo geométrico y asigna los valores a sus atributos. Luego se actualiza constantemente el modelo geométrico o el impostor para visualizar uno u otro en dependencia de la importancia visual de la cámara y el ángulo de visión para mostrar la imagen del impostor correspondiente en cada momento.

Conclusiones del capítulo

En el actual capítulo se seleccionó para formar parte de la solución del presente trabajo la técnica *Geopostors*, por las ventajas que esta presenta de ser fácil de implementar, no es costosa computacionalmente y mejora la velocidad de visualización para escenas con un gran número de modelos geométricos.

Se abordaron las características que presentará la solución técnica, describiendo detalladamente cada uno de los pasos para su desarrollo. Con esto se logrará visualizar adecuadamente multitudes en tiempo real sin comprometer la calidad visual de la aplicación y el realismo en las escenas.

Capítulo 3 Descripción de la solución propuesta

Introducción

El objetivo del presente capítulo es conceptualizar desde la Ingeniería de *Software* la solución propuesta en el apartado anterior. Se definen las funcionalidades que presentará el sistema, descritas en forma de casos de uso. Además se describe el producto a elaborar, teniendo en cuenta las necesidades del cliente.

3.1 Restricciones del sistema

En el actual epígrafe se exponen las premisas que imponen las condiciones que deben satisfacerse para incorporar el módulo de visualización de multitudes.

- ✓ Los modelos geométricos deben ser exportados con la extensión *.mesh.*
- ✓ La cantidad de imágenes confeccionadas para la animación debe ser igual a la cantidad de cuadros de animación del modelo.
- ✓ La animación de caminado del modelo debe ser simétrica.
- ✓ Las imágenes deben ser guardadas con un canal de transparencia o canal alfa.
- ✓ El nombre de las imágenes confeccionadas para la animación debe ser igual al nombre del modelo exportado junto a su extensión, acompañado del identificador (ID) del cuadro de animación al que pertenece la imagen, seguido de la extensión de la imagen.Ejemplo:(nombre modelo. Extensión modelo +ID. Extensión imagen).

3.2 Modelo de dominio

A partir de la interpretación dada al problema, se obtiene el modelo de dominio que a continuación se muestra, siendo este, una aproximación a la solución propuesta, donde se modelan los conceptos fundamentales y sus relaciones, mostrándose así, como un *Geopostor* está formado por un plano y un modelo geométrico, y al mismo tiempo la relación que posee con el motor gráfico Ogre para visualizarlos. Se tiene un modelo de dominio debido a que no están bien definidas las fronteras del negocio y no están definidos los trabajadores ni artefactos del negocio.

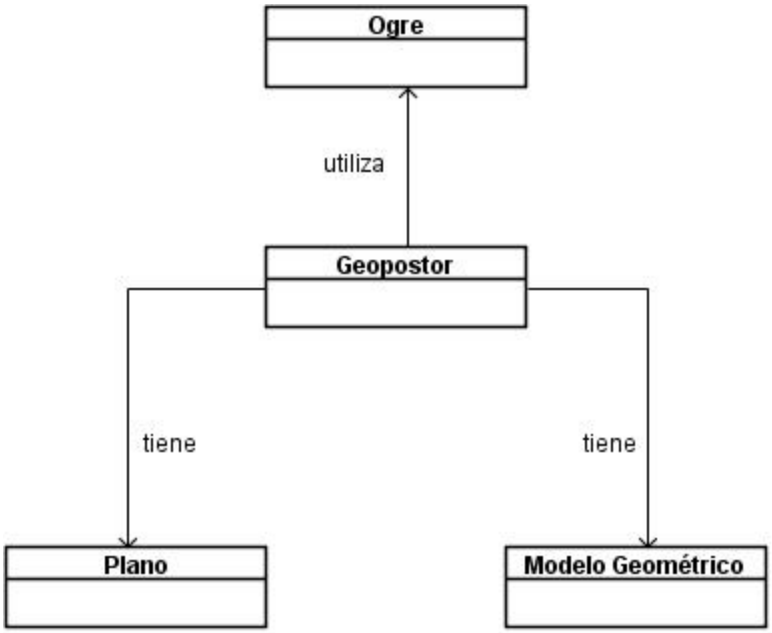

Figura 14: Modelo de Dominio.

3.3 Especificación de los requisitos de *software*

3.3.1 Dependencias y relaciones con otros *software*

Este trabajo ha sido creado para que sea un módulo que va a ser usado por todas las aplicaciones de RV que utilicen el motor gráfico Ogre, debido a que las clases que formarán parte del módulo de visualización de multitudes poseen una fuerte relación con dicho motor gráfico, ya que utilizará sus potencialidades para visualizar y realizar sus principales funciones.

También tendrá una estrecha relación con las clases que componen el juego Entrenadores Aduaneros debido a que el módulo se usará en este juego para generar las multitudes, que unido a la inteligencia artificial, le brindará el realismo necesario a las escenas simuladas.

3.3.2 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. Dichas capacidades o condiciones tienen que ser alcanzadas o poseídas por un sistema o componentes de un sistema para satisfacer un contrato, estándar u otro documento impuesto formalmente.

Teniendo en cuenta lo antes planteado podemos definir de acuerdo a las necesidades del cliente los siguientes requisitos funcionales:

RF1. Inicializar impostor por cada modelo geométrico.
RF2. Inicializar modelo geométrico.
RF3. Inicializar secuencia de imágenes para cada animación del modelo geométrico.
RF4. Mostrar y ocultar impostor.
RF5. Mostrar y ocultar modelo geométrico.
RF6. Actualizar impostor.
RF7. Actualizar modelo geométrico.
RF8. Actualizar la secuencia de imágenes para cada animación, dependiendo del punto de vista.
RF9. Finalizar visualización del impostor.
RF10. Finalizar visualización del modelo geométrico.
RF11. Finalizar la secuencia de imágenes para cada animación.

3.3.3 Requisitos no funcionales

Los requisitos no funcionales se pueden definir como propiedades o cualidades que el producto debe tener o como las características que hacen al producto atractivo, usable, ágil y confiable.

De acuerdo a lo antes dicho se definieron como requisitos no funcionales para el módulo los siguientes:

Usabilidad:
El módulo está creado para ser usado por aplicaciones que utilicen Ogre.

Diseño e implementación:
Se usará el lenguaje de programación C++ y se regirá por la filosofía de la Programación Orientada a Objetos (POO).

Soporte:
El módulo debe ser compatible con la plataforma de Windows.

Rendimiento:
El módulo debe brindar un mejor rendimiento en la visualización de multitudes.

Software:
Sistema operativo Windows, Microsoft Visual Studio.Net 2008, Ogre 1.6.

3.4 Definición de los casos de uso del sistema

En este epígrafe se reconoce el autor del sistema a desarrollar y se definen los casos de uso del sistema mediante la agrupación de los requisitos funcionales anteriormente presentados.

Además se muestra el diagrama de casos de uso del sistema y se les hace las especificaciones textuales, en formato expandido a cada uno de ellos.

3.4.1 Actores del sistema

Actores	**Justificación**
Aplicación	La aplicación es la que interactúa con el sistema debido a que es la aplicación la que invoca a los casos de uso del módulo de visualización de multitudes.

Tabla 1: Justificación de actores.

3.4.2 Diagrama de casos de uso del sistema

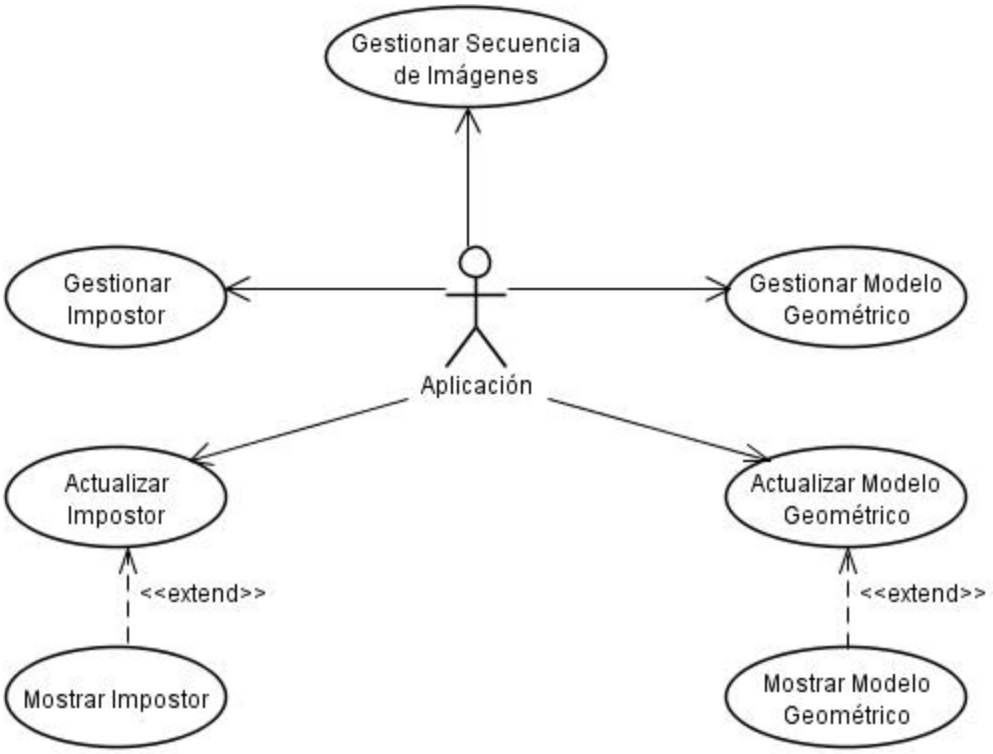

Figura 15: Diagrama de casos de uso del sistema.

3.4.3 Listado de casos de uso del sistema

Nombre del caso de uso	Gestionar Impostor.
Actores	Aplicación.
Propósito	Permitirle al actor poder inicializar y finalizar el Impostor.
Referencias	RF1, RF9

Tabla 2: CU Gestionar Impostor.

Nombre del caso de uso	Actualizar Impostor.
Actores	Aplicación.
Propósito	Permitirle al actor poder actualizar el estado del Impostor.
Referencias	RF6

Tabla 3: CU Actualizar Impostor.

Nombre del caso de uso	Mostrar Impostor.
Actores	Aplicación.
Propósito	Permitirle al actor poder mostrar u ocultar el Impostor.
Referencias	RF4

Tabla 4: CU Mostrar Impostor.

Nombre del caso de uso	Gestionar Modelo Geométrico.
Actores	Aplicación.
Propósito	Permitirle al actor poder inicializar y finalizar el Modelo Geométrico.
Referencias	RF2, RF10

Tabla 5: CU Gestionar Modelo Geométrico.

Nombre del caso de uso	Actualizar Modelo Geométrico.
Actores	Aplicación.
Propósito	Permitirle al actor poder actualizar el estado del Modelo Geométrico.

Referencias	RF7

Tabla 6: CU Actualizar Modelo Geométrico.

Nombre del caso de uso	Mostrar Modelo Geométrico.
Actores	Aplicación.
Propósito	Permitirle al actor poder mostrar u ocultar el Modelo Geométrico.
Referencias	RF5

Tabla 7: CU Mostrar Modelo Geométrico.

Nombre del caso de uso	Gestionar Secuencia de Imágenes.
Actores	Aplicación.
Propósito	Permitirle al actor poder inicializar, actualizar y finalizar una secuencia de imágenes.
Referencias	RF3, RF8, RF11

Tabla 8: CU Gestionar Secuencia de Imágenes.

3.5 Descripción de los casos de uso del sistema

3.5.1 CU Gestionar Impostor

Nombre del caso de uso	Gestionar Impostor.
Actores	Aplicación.
Propósito	Permitirle al actor poder inicializar o finalizar un Impostor.
Resumen El caso de uso se inicia cuando el actor manda a inicializar o finalizar un Impostor y finaliza cuando se ha iniciado o finalizado un Impostor.	
Referencias	RF1, RF9
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
a) Si se desea inicializar un Impostor, ir a la sección:	

Inicializar. **b)** Si se desea finalizar un Impostor, ir a la sección: Finalizar.	
Sección: Inicializar	
Acción del actor	**Respuesta del sistema**
1. Se manda a inicializar el Impostor.	
	1.1 Se toman los datos e inicializa todos los atributos del Impostor.
Sección: Finalizar	
2. Se manda a finalizar el Impostor.	
	2.1 Se finaliza la visualización del Impostor y se libera la memoria ocupada.
Precondiciones	Debe haberse inicializado previamente un Modelo Geométrico.
Pos condiciones	Se inicializa un Impostor o se finaliza un Impostor.

Tabla 9: Descripción del CU Gestionar impostor.

3.5.2 CU Actualizar Impostor

Nombre del caso de uso	Actualizar Impostor.
Actores	Aplicación.
Propósito	Permitirle al actor poder actualizar el estado del Impostor.
Resumen El caso de uso se inicia cuando el actor manda a actualizar el Impostor y finaliza cuando el Impostor es actualizado.	

Referencias	RF6
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1. Se manda a actualizar el Impostor.	
	1.1 Se actualiza el estado del Impostor actualizando todos sus atributos.
Precondiciones	El Impostor debe haberse iniciado.
Pos condiciones	Se actualiza el Impostor.

Tabla 10: Descripción del CU Actualizar Impostor.

3.5.3 CU Mostrar Impostor

Nombre del caso de uso	Mostrar impostor.
Actores	Aplicación.
Propósito	Permitirle al actor poder mostrar u ocultar el Impostor.
Resumen El caso de uso se inicia cuando el actor manda a mostrar u ocultar el Impostor y finaliza cuando se muestra u oculta el Impostor.	
Referencias	RF4
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1. Se manda a mostrar el Impostor.	
	1.1 Se muestra el Impostor si está oculto.
2. Se manda a ocultar el Impostor.	

	2.1 Se oculta el Impostor si está visible.
Precondiciones	El Impostor debe haberse Inicializado.
Pos condiciones	Se muestra u oculta el Impostor

Tabla 11: Descripción del CU Mostrar Impostor.

3.5.4 CU Gestionar Modelo Geométrico

Nombre del caso de uso	Gestionar Modelo Geométrico.
Actores	Aplicación.
Propósito	Permitirle al actor poder inicializar y finalizar el Modelo Geométrico.
Resumen El caso de uso se inicia cuando el actor manda a inicializar o finalizar un Modelo Geométrico y finaliza cuando se inicializa o finaliza el Modelo Geométrico.	
Referencias	RF2, RF10
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
a) Si se desea inicializar un Modelo Geométrico, ir a la sección: Inicializar. **b)** Si se desea finalizar un Modelo Geométrico, ir a la sección: finalizar.	
Sección: Inicializar	
Acción del actor	**Respuesta del sistema**
1. Se manda a inicializar el Modelo Geométrico.	
	1.1 Se toman los datos e inicializa el Modelo

	Geométrico inicializando todos sus atributos.
Sección: finalizar	
2. Se manda a finalizar el Modelo Geométrico.	
	2.1 Se finaliza el Modelo Geométrico y libera la memoria ocupada.
Precondiciones	-
Pos condiciones	Se inicializa o finaliza el Modelo Geométrico.

Tabla 12: Descripción del CU Gestionar Modelo Geométrico.

3.5.5 CU Actualizar Modelo Geométrico

Nombre del caso de uso	Actualizar Modelo Geométrico
Actores	Aplicación.
Propósito	Permitirle al actor poder actualizar el estado del Modelo Geométrico.
Resumen El caso de uso se inicia cuando el actor manda a actualizar el Modelo Geométrico y finaliza cuando se actualiza el Modelo Geométrico.	
Referencias	RF7
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1. Se manda a actualizar el Modelo Geométrico.	
	1.1 Se actualiza el estado del Modelo Geométrico actualizando sus atributos.
Precondiciones	Debe haberse iniciado previamente el Modelo Geométrico
Pos condiciones	Se actualiza el Modelo Geométrico.

Tabla 13: Descripción del CU Actualizar Modelo Geométrico.

3.5.6 CU Mostrar Modelo Geométrico

Nombre del caso de uso	Mostrar Modelo Geométrico.
Actores	Aplicación.
Propósito	Permitirle al actor poder mostrar u ocultar el Modelo Geométrico.
Resumen El caso de uso se inicia cuando el actor manda a mostrar u ocultar el Modelo Geométrico y finaliza cuando se muestra u oculta el Modelo Geométrico.	
Referencias	RF5
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
1. Se manda a mostrar el Modelo Geométrico.	
	1.1 Se muestra el Modelo Geométrico si está oculto.
2. Se manda a ocultar el Modelo Geométrico.	
	2.1 Se oculta el Modelo Geométrico si está visible.
Precondiciones	El Modelo Geométrico debe haberse Inicializado.
Pos condiciones	Se muestra u oculta el Modelo Geométrico.

Tabla 14: Descripción del CU Mostrar Modelo Geométrico.

3.5.7 CU Gestionar Secuencia de Imágenes

Nombre del caso de uso	Gestionar Secuencia de Imágenes.
Actores	Aplicación.
Propósito	Permitirle al actor poder inicializar, actualizar y finalizar la Secuencia de Imágenes del Impostor.

Resumen	
El caso de uso se inicia cuando el actor manda a inicializar, actualizar o finalizar una secuencia de imágenes y finaliza cuando se inicializa, se actualiza o finaliza la Secuencia de Imágenes.	
Referencias	RF3, RF8, RF11
Curso normal de los eventos:	
Acción del actor	**Respuesta del sistema**
a) Si se desea inicializar la Secuencia de imágenes, ir a la sección: Inicializar. **b)** Si se desea actualizar la Secuencia de Imágenes, ir a la sección: ctualizar. **c)** Si se desea finalizar la Secuencia de Imágenes, ir a la sección: Finalizar.	
Sección: Inicializar	
Acción del actor	**Respuesta del sistema**
1. Se manda a inicializar la Secuencia de Imágenes.	
	1.1 Se toman los datos e inicializa la Secuencia de Imágenes.
Sección: Actualizar	
2. Se manda a actualizar la Secuencia de Imágenes.	
	2.1 Se actualiza la Secuencia de Imágenes de acuerdo al ángulo de visualización.

Sección: Finalizar	
3. Se manda a finalizar la Secuencia de Imágenes.	
	3.1 Se finaliza la Secuencia de Imágenes liberando el espacio de memoria ocupado.
Precondiciones	Debe haberse inicializado previamente un impostor
Pos condiciones	Se inicializa, actualiza o finaliza la Secuencia de Imágenes.

Tabla 15: Descripción del CU Gestionar Secuencia de Imágenes.

Conclusiones del capítulo

En este capítulo se puntualizaron los requisitos funcionales y no funcionales, se definieron y describieron los casos de uso del sistema de acuerdo a lo que espera el cliente que realice el módulo. De esta manera se puede proceder con el diseño del mismo.

Capítulo 4 Diseño e implementación del sistema

Introducción

En el presente capítulo se detalla todo lo relacionado con el módulo a desarrollar, mediante el diagrama de clases, los diagramas de secuencia y el diagrama de componentes. Además se describen cada una de las clases de dicho módulo.

4.1 Estándares de codificación

A continuación se muestran los estándares de codificación a emplear para el desarrollo del módulo, respetándose los establecidos oficialmente en el proyecto Entrenadores Aduaneros.

Nombre de los ficheros:

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

Name.cpp

Clases:

Class Name

Argumentos:

Los nombres de las variables en caso de ser argumentos de algún método, se les antepone el prefijo "arg".

Ej. void setVisible (bool argVisible);

Declaración de las variables:

Las instancias de tipo creados llevaran el prefijo "p".

Ej. int pID;

Métodos:

Los nombres de los métodos comenzarán con minúscula, en caso de tener más de un nombre el que sigue comienza con mayúscula y unido al primero.

Ej. void actualizarImpostor ();

Constructor y destructor:

Class Name ();

~Class Name ();

4.2 Diagramas de clases del diseño

4.2.1 Diagrama de clases del diseño

A continuación se presenta un diagrama de paquetes donde se muestra la relación que tiene el módulo para la visualización de multitudes con el motor gráfico Ogre.

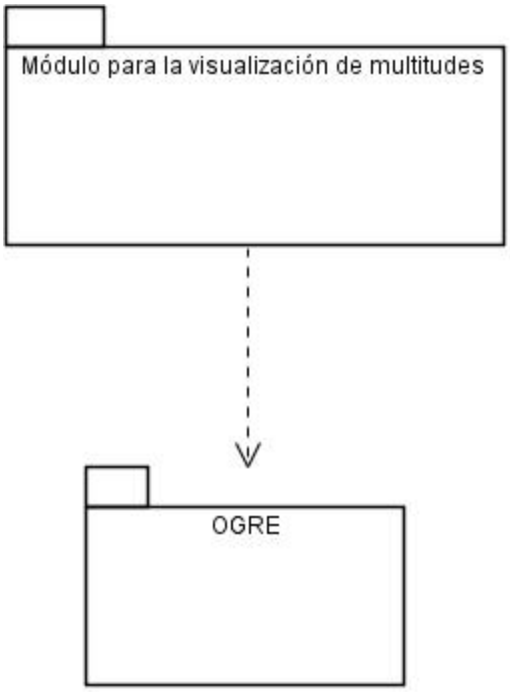

Figura 16: Diagrama de clases del diseño por paquetes.

4.2.2 Diagrama de clases del diseño

En el presente diagrama se muestra la relación entre las clases del módulo de visualización de multitudes y las clases del motor gráfico Ogre que usa dicho módulo.

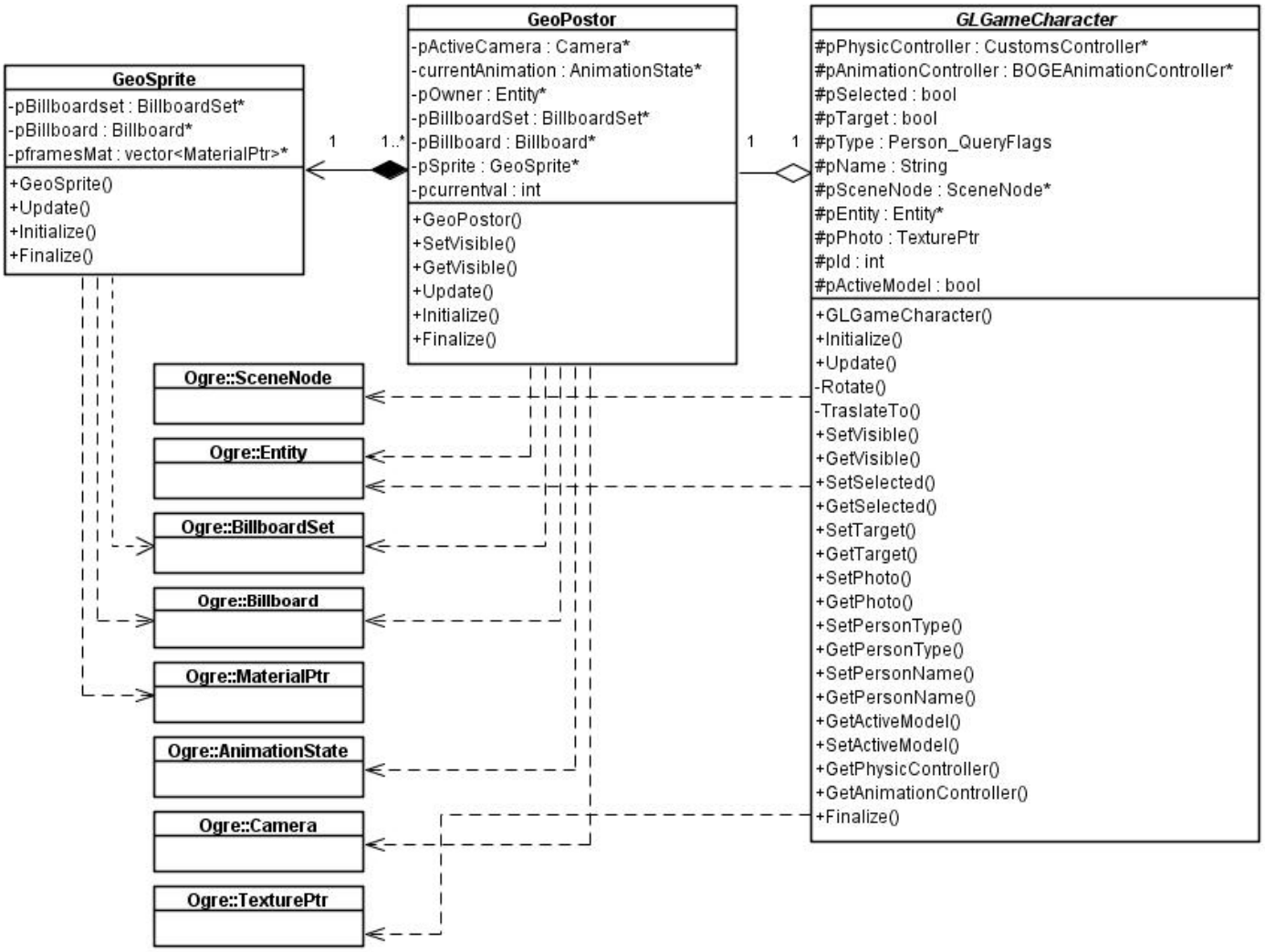

Figura 17: Diagrama de clases del diseño.

El módulo de visualización, al tener fuerte dependencia con el motor gráfico, utiliza las potencialidades que pueden brindar las clases del motor con las que se relaciona. Una de estas clases permite crear entidades, las cuales tienen un nombre que las identifican en cada momento y pueden ser adjuntadas a

nodos, los cuales pueden ser trasladados, rotados o escalados. Se tiene una clase *Camera* para navegar por la escena y realizar el cambio entre el impostor y el modelo geométrico. El llamado *Billboard* no es más que un plano que se orienta con respecto a la cámara de forma dinámica, los cuales serán usados para representar los impostores y el *BillboardSet* para asignarle el material al plano. Las clases *MaterialPtr* y *TexturePtr* permiten asignarle un material al modelo y crear una textura respectivamente. Además se cuenta con la clase *AnimationState* para capturar en todo momento el *frame* de animación a representar por el impostor.

4.3 Descripción de las clases de diseño

En este apéndice se especifican las características de las clases del módulo de visualización, además se describen los atributos y métodos que las conforman, dándose una panorámica de lo que cada uno realiza. Estas clases se relacionan entre sí y tienen una fuerte relación con las clases del motor gráfico Ogre especificadas anteriormente.

Nombre: *GeoPostor*	
Tipo de clase: Controladora	
Atributo	**Tipo**
pActiveCamera	*Ogre::Camera**
c*urrentAnimation*	*Ogre::AnimationState**
pOwer	*Ogre::Entity**
pBillboardSet	*Ogre::BillboardSet**
pBillboard	*Ogre::Billboard**
pSprite	*GeoSprite**
pcurrentval	*int*
Para cada responsabilidad	
Nombre:	*SetVisible(bool argParam)*
Descripción:	Muestra u oculta el Impostor.
Nombre:	*GetVisible()*
Descripción:	Devuelve true si el impostor esta visible y false si está oculto

Nombre:	*Update(Ogre::Real timeSinceLastFrame,Ogre::Vector3 argLookVector)*
Descripción:	Actualiza el impostor.
Nombre:	*Initialize(Ogre::Entity * argEntity,Ogre::Camera *argcam, Ogre::AnimationState*arganim)*
Descripción:	Inicializa un impostor.
Nombre:	*Finalize()*
Descripción:	Finaliza un impostor.

Tabla 16: Descripción de la clase *GeoPostor.*

Nombre: *GeoSprite*	
Tipo de clase: Controladora	
Atributo	**Tipo**
pBillboardSet	*Ogre::BillboardSet**
pBillboard	*Ogre::Billboard**
pframesMat	*std::vector<Ogre::MaterialPtr> **
Para cada responsabilidad	
Nombre:	*Update(Ogre::Real argAngle,Ogre::Real argAnimationTime,int argSing)*
Descripción:	Actualiza la secuencia de imágenes dependiendo del punto de vista.
Nombre:	*Initialize(std::string argMeshName,int argframes,std::string argExtension,Ogre::BillboardSet *argBill,Ogre::Billboard *argBillb)*
Descripción:	Inicializa secuencia de imágenes.
Nombre:	*Finalize()*
Descripción:	Finaliza secuencia de imágenes.

Tabla 17: Descripción de la clase *GeoSprite*.

Nombre: *GLGameCharacter*	
Tipo de clase: Controladora	
Atributo	Tipo
pPhysicController	*CustomsController**

pAnimationController	*BOGEAnimationController **
pSelect	*bool*
pTarget	*bool*
pType	*Person_QueryFlags*
pName	*string*
pSceneNode	*Ogre::sceneNode**
pEntity	*Ogre::Entity**
pPhoto	*Ogre::TexturePtr*
pId	*int*
pActiveModel	*bool*
Para cada responsabilidad	
Nombre:	*Initialize(Ogre::SceneManager * argGameScene, Ogre::String argMeshName)*
Descripción:	Inicializa el modelo geométrico.
Nombre:	*Update(Ogre::Real argTimesteps, Ogre::Vector3 argDesiredPos, Ogre::Vector3 argDesiredLookVector, eAnimationType argTargetAnimation, Ogre::String argNameAnimation)*
Descripción:	Actualiza el modelo geométrico.
Nombre:	*Finalize()*
Descripción:	Deja de pintar el modelo geométrico.
Nombre:	*Rotate(Ogre::Vector3 * argAxisOrientation, Ogre::Vector3 * argNewDirection)*
Descripción:	Rota el modelo geométrico.
Nombre:	*TraslateTo(Ogre.Vector3::argDestination)*
Descripción:	Traslada el modelo Geométrico
Nombre:	*SetVisible(bool argParam)*
Descripción:	Muestra u oculta el Impostor.
Nombre:	*GetVisible()*
Descripción:	Devuelve true si el impostor esta visible y false si está oculto

Tabla 18: Descripción de la clase GLGameCharacter.

4.4 Diagramas de secuencia

A continuación se muestran las imágenes correspondientes a los diagramas de secuencia donde se muestran 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.

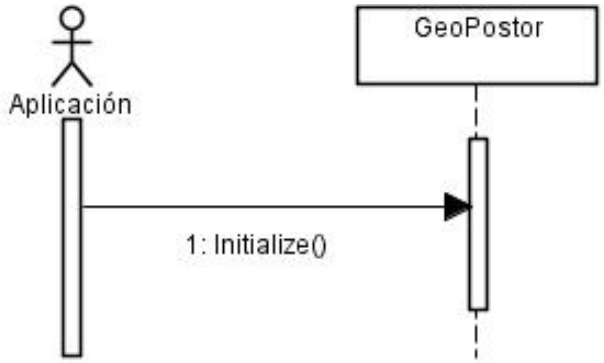

Figura 18: Diagrama de Secuencia “Inicializar Impostor”.

En el presente diagrama de secuencia se muestra como la aplicación inicializa un impostor mediante el método *Initialize*, al cual se le pasa como parámetros los siguientes argumentos: (*Ogre::Entity* argEntity, Ogre::Camera* argCam, Ogre::AnimationState* argAnim).*

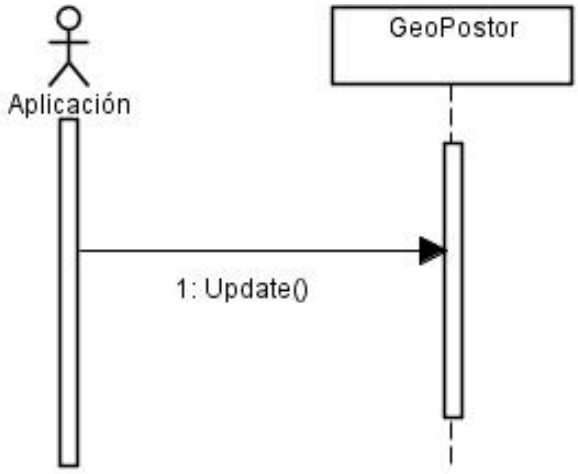

Figura 19: Diagrama de Secuencia "Actualizar Impostor".

En el presente diagrama de secuencia se muestra como la aplicación actualiza un impostor mediante el método *Update*, al cual le pasa como parámetros los siguientes argumentos: *(Ogre::Real argTimeSinceLastFrame, Ogre::Vector3 argLookVector).*

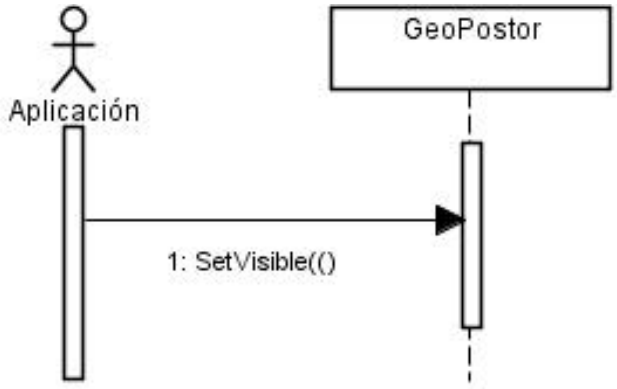

Figura 20: Diagrama de Secuencia " Mostrar Impostor".

En el presente diagrama de secuencia se muestra como la aplicación muestra u oculta un impostor mediante el método *SetVisible*, al cual le pasa como parámetros los siguientes argumentos: *(bool argParam).*

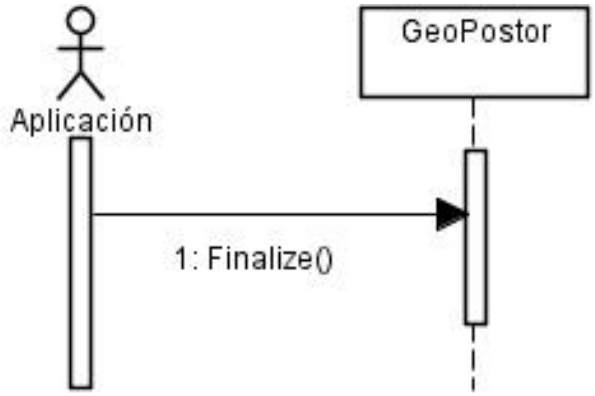

Figura 21: Diagrama de Secuencia: "Finalizar Impostor".

En el presente diagrama de secuencia se muestra como la aplicación finaliza un impostor mediante el método *Finalize*, al cual no se le pasa nada como parámetros.

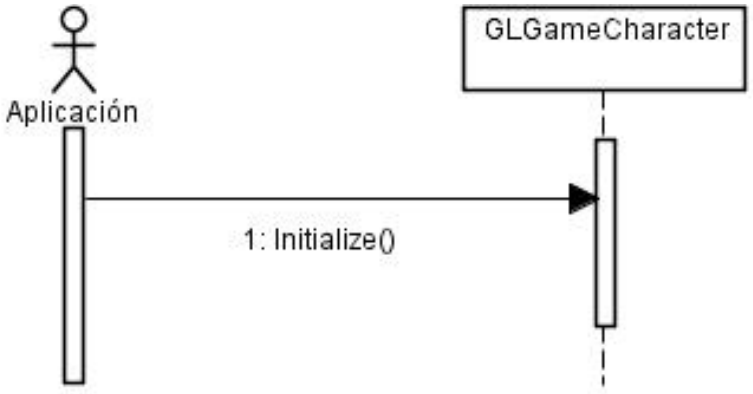

Figura 22: Diagrama de Secuencia "Inicializar Modelo Geométrico".

En el presente diagrama de secuencia se muestra como la aplicación inicializa un modelo geométrico mediante el método *Initialize*, al cual le pasa como parámetros los siguientes argumentos: *(Ogre::SceneManager * argGameScene, Ogre::String argMeshName).*

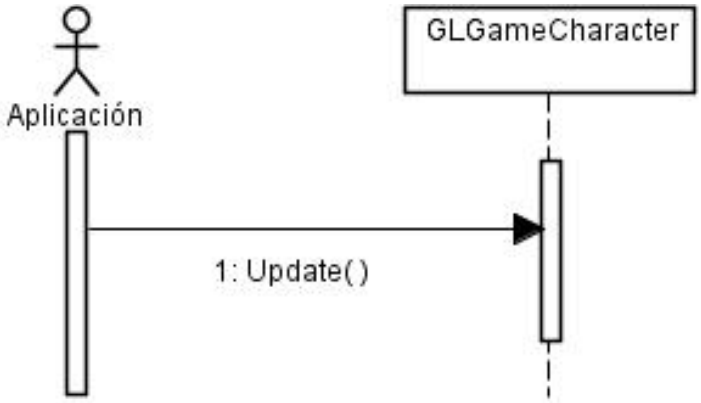

Figura 23: Diagrama de Secuencia "Actualizar Modelo Geométrico".

En el presente diagrama de secuencia se muestra como la aplicación actualiza un modelo geométrico mediante el método *Update*, al cual le pasa como parámetros los siguientes argumentos: *(Ogre::Real argTimesteps, Ogre::Vector3 argDesiredPos, Ogre::Vector3 argDesiredLookVector, eAnimationType argTargetAnimation, Ogre::String argNameAnimation).*

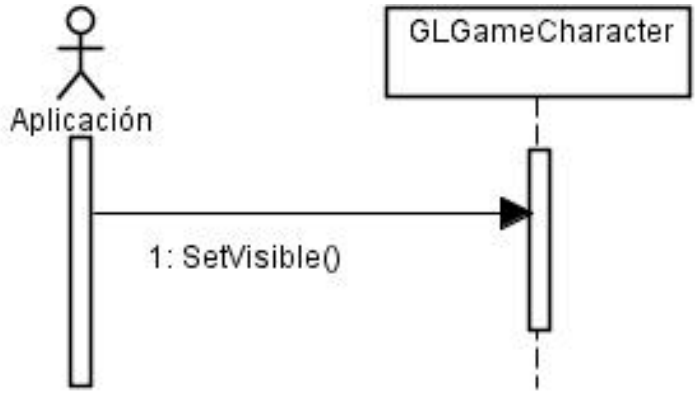

Figura 24: Diagrama de Secuencia "Mostrar Modelo Geométrico".

En el presente diagrama de secuencia se muestra como la aplicación muestra u oculta un modelo geométrico mediante el método *SetVisible*, al cual le pasa como parámetros los siguientes argumentos: (bool argParam).

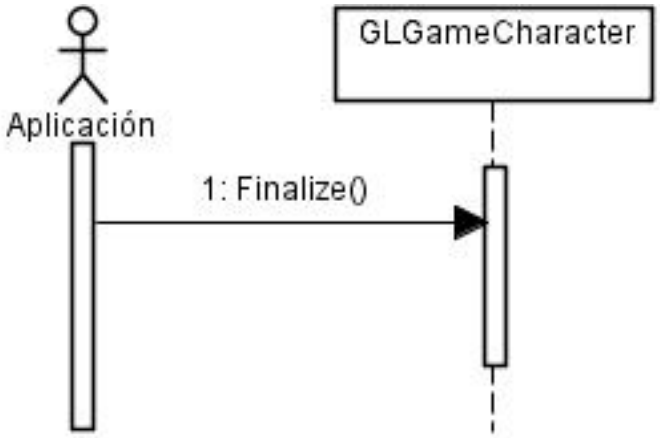

Figura 25: Diagrama de Secuencia "Finalizar Modelo Geométrico".

En el presente diagrama de secuencia se muestra como la aplicación finaliza un modelo geométrico mediante el método *Finalize*, al cual no se le pasa nada como parámetros.

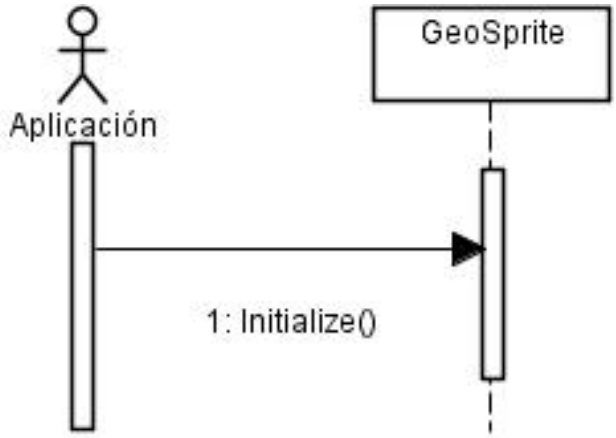

Figura 26: Diagrama de Secuencia "Inicializar Secuencia de Animación".

En el presente diagrama de secuencia se muestra como la aplicación inicializa una secuencia animada mediante el método *Initialize*, al cual le pasa como parámetros los siguientes argumentos: *(std::string argMeshName,int argframes,std::string argExtension,Ogre::BillboardSet *argBill,Ogre::Billboard *argBillb)*.

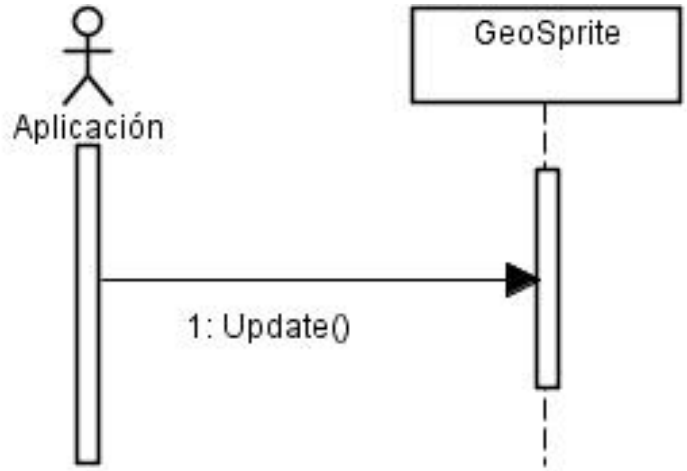

Figura 27: Diagrama de Secuencia "Actualizar Secuencia de Animación".

En el presente diagrama de secuencia se muestra como la aplicación actualiza una secuencia animada mediante el método *Update*, al cual le pasa como parámetros los siguientes argumentos: *(Ogre::Real argAngle,Ogre::Real argAnimationTime,int argSing)*.

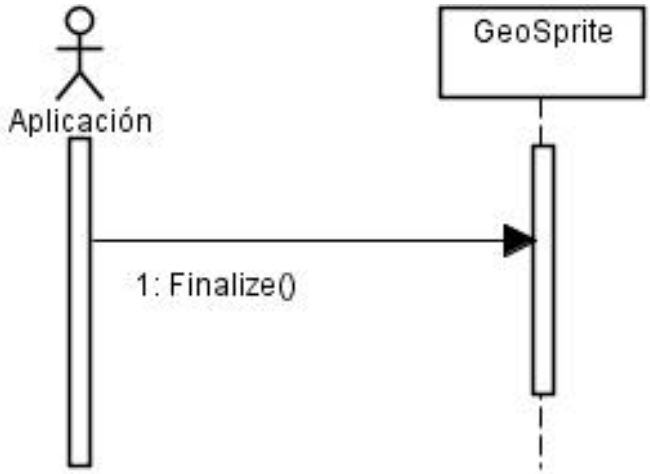

Figura 28: Diagrama de Secuencia "Finalizar Secuencia de Animación".

En el presente diagrama de secuencia se muestra como la aplicación finaliza una secuencia animada mediante el método *Finalize*, al cual no se le pasa nada como parámetros.

4.5 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++.

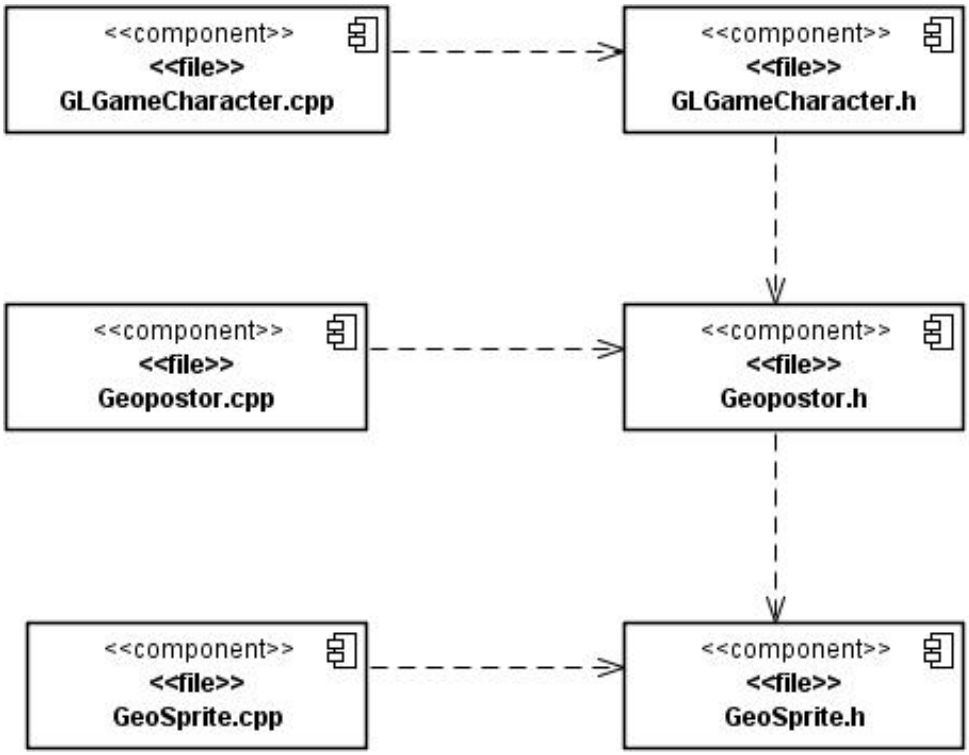

Figura 29: Diagrama de Componentes.

4.6 Resultados Obtenidos

En el presente epígrafe se muestran dos gráficas donde a partir de la visualización de 50-1000 modelos geométricos e impostores se reflejan los resultados obtenidos en cuadros por segundo contra cantidad de humanos virtuales.

En la Fig.30 puede apreciarse como al visualizar solamente modelos geométricos los cuadros por segundos en la aplicación disminuyen considerablemente a medida que va aumentado el número de modelos geométricos, manteniéndose una gran diferencia con respecto a la visualización de impostores, ejemplo de esto se puede ver como al visualizar en escena 1000 modelos geométricos los cuadros por segundo de la aplicación disminuyen hasta cerca de cero, y al visualizar 1000 impostores los cuadros de animación aún se mantienen por encima de 30, manteniéndose una buena visualización.

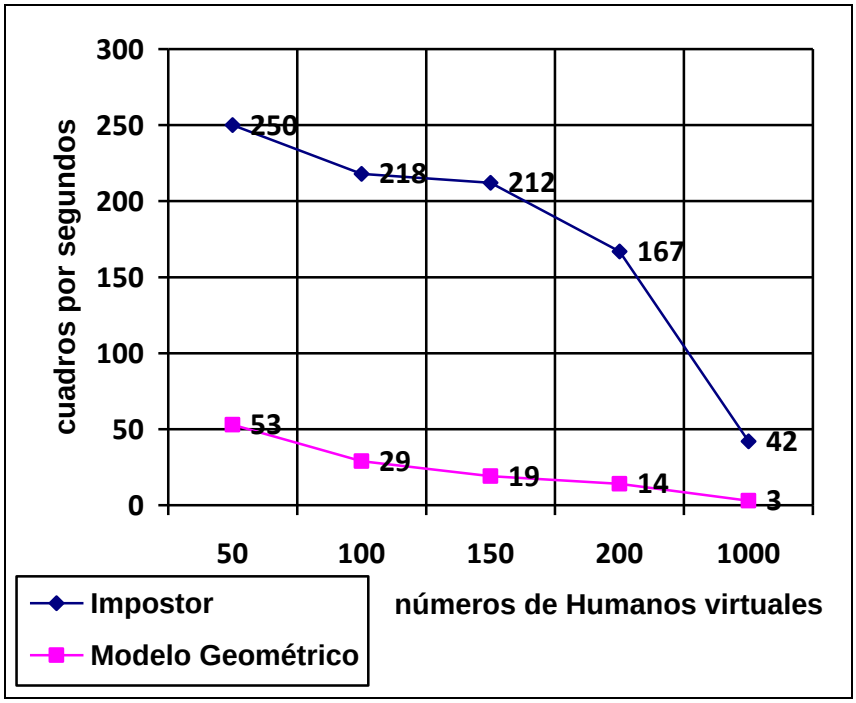

Figura 30: Impostor Vs Modelo geométrico.

En la siguiente figura se muestra el comportamiento de los cuadros de animación para la visualización de modelos geométricos y la combinación de impostores y modelos, donde se puede apreciar como al aumentar el número de humanos virtuales la combinación de ambas representaciones se mantiene por encima de los modelos geométricos.

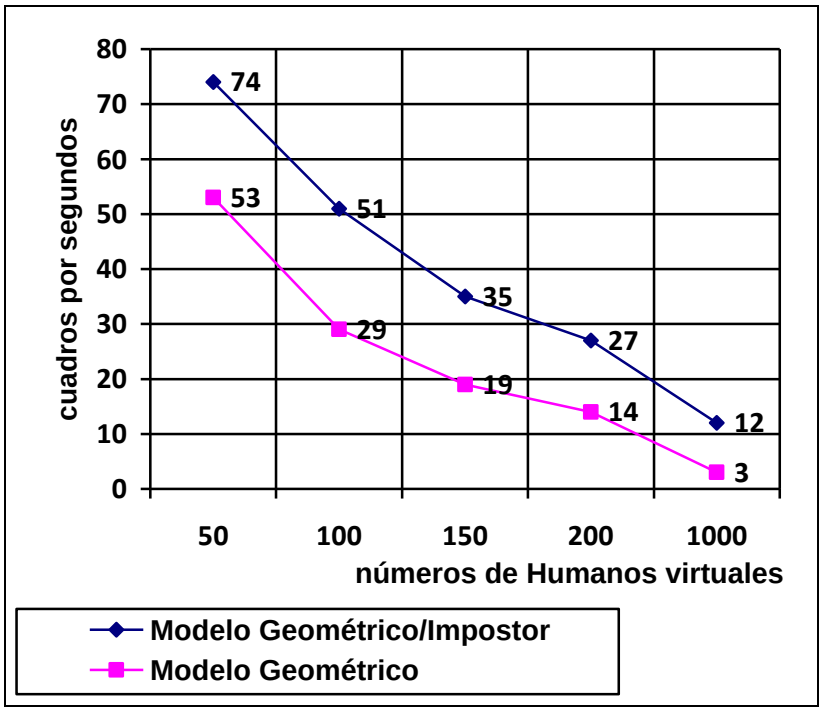

Figura 31: Modelo Geométrico Vs Impostores/Modelo Geométrico.

Conclusiones del capítulo

Durante el desarrollo de este capítulo se presentaron los artefactos para desarrollar el módulo de visualización de multitudes. Se definió una arquitectura flexible para acoplarlo al proyecto Entrenadores Aduaneros.

También se puntualizó como se harán físicas las clases de diseño, fortaleciendo el proceso de desarrollo para dar paso a la programación de los casos de uso.

Al finalizar este capítulo se tiene concebido de forma detallada el diseño completo del módulo.

Conclusiones generales

Con el desarrollo del presente trabajo se ha dado cumplimiento a los objetivos propuestos, obteniéndose como resultado un módulo para visualizar multitudes en tiempo real. Este módulo permite mejorar el rendimiento de la visualización de las escenas virtuales que se simulen, lográndose así una mayor calidad en dichas visualizaciones y en las aplicaciones como tal.

Se realizó el acople del módulo de visualización de multitudes al proyecto Entrenadores Aduaneros. Por tanto, se puede asegurar que se obtuvo un producto totalmente funcional.

Recomendaciones

Se recomienda realizar el cambio de textura a nivel de *hardware* gráfico.

.

Bibliografía

1. **Franco Tecchia, Yiorgos Chrysanthou.** *Real-Time Rendering of Densely Populated.* Londres,Italia : s.n.

2. **Andereck, Michael.** *Crowd Animation.* 2008.

3. **Carol O'Sullivan, Justine Cassell, Hannes Vilhjálmsson, Simon Dobbyn, Christopher Peters,William Leeson, Thanh Giang and John Dingliana.** *Crowd and Group Simulation with Levels of Detail for Geometry, Motion and Behaviour.* Irlanda : s.n.

4. **Chrysanthou, Yiorgos.** *Adding Virtual Characters to the Virtual Worlds.* Cyprus : s.n.

5. **A. Aubel, R. Boulic, and D. Thalmann.** *Real-time display of virtual humans:Levels of details and impostors. IEEE Transactions on Circuits and Systems.* 2000.

6. **Dobbyn, Simon.** *Hybrid Representations and Perceptual Metrics for Scalable Human Simulation.* Irlanda : s.n.

Referencias bibliográficas

1. http://icsp.lacoctelera.net. *http://icsp.lacoctelera.net.* [Online] [Cited: 03 15, 2010.] http://icsp.lacoctelera.net/post/2009/05/05/realidad-virtual.

2. http://www.dei.uc.edu.py. *http://www.dei.uc.edu.py.* [Online] [Cited: 02 20, 2010.] http://www.dei.uc.edu.py/tai99/vr/introduc.htm.

3. http://www.tdr.cesca.es. *http://www.tdr.cesca.es.* [Online] [Cited: 04 26, 2010.] http://www.tdr.cesca.es.TESIS_UC...TDR...02de10.FLGcap2.pdf.

4. **Franco Tecchia, C´eline Loscos, Yiorgos Chrysanthou.** *Image-Based Crowd Rendering.* London : s.n.

5. **Simon Dobbyn, John Hamill, Keith O'Conor, Carol O'Sullivan.** *Geopostors: A Real-Time Geometry / Impostor Crowd Rendering System.* Irlanda : s.n.

6. **Ladislav Kavan, Simon Dobbyn1, Steven Collins1, Jirı Zara, Carol O'Sullivan1.** *Polypostors: 2D Polygonal Impostors for 3D Crowds.* Praga : s.n.

7. **Joshua Barczak, Natalya Tatarchuk,Christoper Oat.** *GPUBasedSceneManagementLargeCrowds_SLIDES.* 2008.

8. **Katherine Gómez Cuello, Ariangna Garcés Gilart.** *Visualización de Sistemas de Realidad Virtual.* Ciudad de la Habana : s.n., 2007.

9. http://www.alsentia.com. *http://www.alsentia.com.* [Online] [Cited: 01 17, 2010.] http://www.alsentia.com/descargas/Descripcion_Tecnica.pdf.

10. http://www.taringa.net. *http://www.taringa.net.* [Online] [Cited: 01 23, 2010.] http://www.taringa.net/posts/animaciones/3535922/Massive-Software---Animacion-de-Multitudes!.html.

11. http://www.ogre3d.org. *http://www.ogre3d.org.* [Online] [Cited: 04 27, 2004.] http://www.ogre3d.org/about.

12. **Alexey Broche Medina, Yordankis Alonso Cardoso.** *Módulo de carga dinámica para la herramienta SceneToolkit.* Ciudad de la Habana : s.n., 2009.

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.

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.

POO: Programación Orientada a Objetos.

RV: Realidad Virtual.

LOD: Niveles de Detalle.

IBR: Imágenes Basadas en *Render*.

CEDIN: Centro de Desarrollo de Informática Industrial.

IMN: Imagen de Mapa Normal.

Glosario de términos

Glosario de términos del dominio

Impostor: Una foto que intenta suplantar al modelo real**.**

Modelo Geométrico: Humano virtual diseñado en 3D.

Modelo: Sinónimo de modelo geométrico.

Billboard: Plano orientado de forma dinámica hacia la cámara.

Glosario de términos generales

Fotogramas: Cada imagen estática que forma la secuencia animada.

Cuadro de animación: Igual a fotograma.

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

Realidad Virtual: Término que pretende describir la interacción de los seres humanos en mundos virtuales o simulados.

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

Simulación: Reproducción o recreación artificial de procesos que se dan en la realidad.

Índice de figuras y tablas

Índice de figuras

Índice de tablas