UNIVERSIDAD DE LAS CIENCIAS INFORMATICAS

Facultad 5

Perfil "Realidad Virtual"

Módulo Para la Detección de Colisiones en Entornos Virtuales

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informática.

Autor: Alberto González González
Yuriel Calvo Lias

Tutor: Ing. Fernando Jiménez López

Asesor: MSc. Pedro Carlos Pérez Martinto

Ciudad de la Habana
Junio 2007

Declaración de Autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Alberto González González y
Yuriel Calvo Lias

_____

Fernando Jiménez López

_____

Dedicatoria

A toda mi familia y amigos.

Alberto

A toda mi familia, amigos y a mi novia.

Yuriel

Agradecimientos

A la revolución por habernos dado la posibilidad de cumplir nuestros sueños y estudiar esta especialidad y a todas aquellas personas que de una forma u otra contribuyeron al desarrollo de este trabajo.

Resumen

Los Sistemas de Realidad Virtual (SRV), se han expandido considerablemente a diferentes sectores de la sociedad en los últimos años. Esta tecnología, ofreciendo prestaciones de una manera más intuitiva, segura y económica, puede encontrarse en aplicaciones de la defensa, la medicina, la educación y el entretenimiento.

Toda la tecnología de Realidad Virtual (RV), se apoya en un núcleo central, que se encarga de accionar los diferentes módulos para el funcionamiento del sistema. Este núcleo, también denominado “motor de simulación”, cuenta entre sus módulos básicos con un módulo de detección de colisiones en tiempo real. Este proyecto aborda el diseño de una biblioteca de clases para la detección de colisiones entre objetos dinámicos, implementada en C++.

Para alcanzar esta meta, se estudian bibliotecas existentes, y se trabaja en la investigación y desarrollo de algoritmos eficientes para la detección de colisiones en tiempo real. Se exponen además, los métodos utilizados para dar respuesta a nuestra investigación. La herramienta de software resultante de esta investigación, permitirá mayor eficiencia al representar las detecciones de colisiones y la reducción de recursos en la PC a la hora de hacer los cálculos necesarios para dar una respuesta al sistema.

Autores: Alberto González González
Yuriel Calvo Lias

Contenido

Introducción

Desde que surge la Realidad Virtual todas las personas que se han dedicado al trabajo en esta disciplina de la ciencia han tratado de acercar lo más posible los entornos virtuales realizados en computadoras al mundo real en el que se desarrolla la vida cotidiana. Para esto se busca perfeccionar cada día más la interacción existente entre los objetos que conforman el entorno virtual como son personas, animales, automóviles entre otros. Las colisiones entre los objetos que se encuentran en movimiento en estos entornos es una tarea difícil a la cual un gran número de personas les han dedicado mucho tiempo de trabajo.

Estos sistemas de detección de colisiones son muy utilizados en los distintos tipos de EV (entornos virtuales) con objetos dinámicos que buscan la simulación de la realidad.

El motor de visualización del proyecto de simuladores existente en nuestra universidad todavía no cuenta con algoritmos óptimos para la detección de colisiones entre objetos dinámicos en entornos virtuales. Además, esto conlleva a que la simulación carezca de realismo.

Analizando detenidamente la anterior situación problémica se ha definido el siguiente problema científico, **¿cómo crear un módulo para la detección de colisiones que permita resolver este problema de una forma eficiente y rápida?**

El objeto de estudio de este trabajo consiste en el **proceso de detección de colisiones en Entornos Virtuales** (EV).

El campo de acción de nuestro trabajo comprende la **detección de colisiones entre los objetos que se encuentran en movimiento en entornos virtuales** y la posibilidad que existe de que estos objetos puedan colisionar en un momento dado.

El objetivo general de este proyecto es implementar un **módulo que permita detectar todas las colisiones que puedan existir entre objetos dinámicos en un Entorno Virtual (EV).**

Se plantean un grupo de tareas que permitan satisfacer el objetivo planteado anteriormente, las cuales se muestran a continuación:

- Estudio de las características básicas de los módulos de colisiones ya existentes.
- Estudio de algoritmos, tecnologías y tendencias actuales utilizadas en la detección de colisiones.
- Desarrollo de soluciones técnicas para alcanzar los objetivos propuestos.
- Análisis y diseño de una biblioteca de clases que dé solución a los problemas propuestos.
- Implementar una biblioteca de clases que dé solución a los problemas propuestos.

Como resultados de este trabajo se pretende obtener un módulo de detección de colisiones de objetos dinámicos en EV, con soporte de diferentes situaciones que aparecen en el mundo real.

1 Fundamentación Teórica

Introducción

La meta principal en el desarrollo de aplicaciones que representan entornos virtuales es simular lo más fielmente posible algunos aspectos del mundo real.

En el mundo real dos objetos no pueden ocupar el mismo punto en el mismo momento. Por lo que se considera una tarea importante encontrar un mecanismo capaz de detectar las configuraciones de interpenetración de objetos a las que se les llama colisiones.

Determinar si dos objetos colisionan o no es un problema puramente matemático. Por lo que las aplicaciones interactivas requieren de un procesamiento eficiente de estas pruebas de detección de colisión.

Los módulos de detección de colisiones exigen ante todo una rápida interacción con el usuario, por lo que se debe ser cuidadoso a la hora de seleccionar los algoritmos y métodos sobre los que se basarán dichos sistemas.

En el presente capítulo se hace un análisis de las características básicas de las librerías de detección de colisiones, así como de los principales algoritmos, tecnologías y tendencias que se están empleando en el mundo para el trabajo con los componentes mencionados. Se analizan además las diferencias y funcionalidades de algunos motores de Sistemas de Realidad Virtual (SRV), y los algoritmos y estructuras de clases que soportan sus funcionamientos.

1.1 Librerías de colisiones estudiadas

En el mundo se han hecho varios e importantes descubrimientos sobre detección de colisiones y paquetes de test de proximidad, con los que se han desarrollado varias librerías de detección de colisiones. A continuación se hará una breve descripción de estas librerías de detección de colisiones.

1.1.1 V-COLLIDE

V-Collide es una librería implementada en C++, para determinar cuál de los objetos en movimiento en una escena virtual están en contacto potencial. La arquitectura de V-Collide, al igual que en el caso del I-Collide que se verá mas adelante, se divide en dos niveles:

- En el primer nivel se determina qué pares de objetos se solapan en la escena. Para ello se emplea el mismo método que en el caso del I-Collide, la aproximación por cajas de inclusión axis aligned bounding boxes AABB. [4]
- En el segundo nivel se estudia la colisión de los pares de objetos seleccionados en la fase anterior. Para ello, el sistema calcula el árbol jerárquico de cajas de inclusión orientadas (OBB) mediante el mismo **algoritmo** que en la librería RAPID que se verá mas adelante. El sistema primero determina cuál de las cajas de inclusión del último nivel están en contacto y posteriormente estudia el solapamiento de los triángulos correspondientes. [4]

Por tanto, al contrario que en RAPID, V-Collide determina qué parejas de objetos de la escena virtual colisionan pero no la distancia entre ellos (como es el caso de I-Collide). Sin embargo, tiene mejores prestaciones que I-Collide en el caso de objetos no convexos. [4]

1.1.2 I-COLLIDE

I-Collide es capaz de determinar con gran rapidez y exactitud las colisiones presentes en un entorno virtual de N elementos en movimiento rígido (rotación y traslación).

I-Collide está implementada en ANSI-C y emplea la coherencia temporal para conseguir una mayor eficiencia. Nótese que la existencia de coherencia temporal (el estado de la simulación no cambia significativamente entre dos pasos de simulación consecutivos) implica coherencia geométrica (el tiempo de simulación es tan pequeño que hace que los objetos no se muevan largas distancias entre dos pasos de simulación). [5]

El algoritmo implementado en I-Collide consta de dos partes:

- Determinar los pares de objetos que se pueden solapar en la escena. Para ello emplea intersección de cajas de inclusión AABB. El resultado de este paso es una lista con los pares de objetos de la escena que pueden solaparse. [5]
- Determinar cuáles son las primitivas (puntos, líneas o caras) de cada uno de los objetos de la pareja más cercana. Para ello, la librería tiene implementado el algoritmo de Lin-Canny. Aunque este algoritmo es sólo para objetos convexos, I-Collide lo adapta mediante el uso de un árbol jerárquico, de tal manera que los objetos no convexos los tratará como un árbol donde todos los elementos pueden ser convexos o no convexos excepto en el último nivel que tienen que ser convexos. Al suponer coherencia temporal, se considera que los dos elementos más cercanos son la pareja de primitivas resultado del paso de simulación anterior. Si no es así, el algoritmo considera que las primitivas más cercanas están cerca, a su vez, de las primitivas anteriores y por tanto estudia la distancia en las primitivas vecinas. [5]

1.1.3 RAPID

La librería RAPID es la implementación en C++ del método de detección de colisiones presentado por Gottschalk, Manocha y Lin, método descrito para pares de objetos rígidos. El **algoritmo** es capaz de determinar de manera eficiente qué polígonos de dichos objetos se solapan. [13]

Para la búsqueda de la colisión RAPID precalcula árboles jerárquicos de cajas orientadas (cajas con alineamiento no paralelo OBB) de los dos objetos de la escena bajo estudio. Posteriormente estudia el solapamiento de cada uno de los volúmenes de inclusión. El **algoritmo** presentado es capaz de calcular todos los contactos entre geometrías muy complejas permitiendo interactividad. [13]

1.1.4 SWIFT

SWIFT es una librería de detección de colisiones, cálculo de distancias y determinación de contactos de objetos poligonales tridimensionales en movimiento rígido. Ha sido implementada en C++. [6]

Soporta geometrías cerradas y convexas (aunque la versión SWIFT ++ sí que soporta modelos poliédricos no-convexos). Para el cálculo de distancias hace uso de una jerarquía de modelos multirresolución de los objetos implicados que, junto a la utilización de un algoritmo derivado en las regiones de Voronoi, hace que la librería tenga gran velocidad de cómputo. [6]

También hace uso de cajas de inclusión para una primera aproximación en el cálculo de las colisiones. [6]

Esta librería es más rápida que otras librerías como I-Collide y V-Clip y además, presenta buenas prestaciones incluso cuando no hay coherencia temporal. [6]

1.1.5 SOLID

La librería Solid sólo se considera aquí en su versión 2.0, también llamada free SOLID. Esta librería, implementada en C++, sirve para la detección de colisiones entre objetos convexos y/o formas geométricas básicas (cilindros, cubos, esferas, etc.) bajo movimientos rígidos y deformaciones y en especial para entornos virtuales VRML (Virtual Reality Modeling Language). Esta librería tiene implementados los siguientes algoritmos:

- El algoritmo GJK (Gilbert, Johnson y Keerthi), implementado por Van den Bergen. Sirve para el cálculo de distancias entre objetos convexos (este algoritmo ha sido mejorado por Cameron para calcular distancias de penetraciones entre objetos. [11]
- Algoritmo para la detección de colisiones para modelos complejos deformables mediante el uso de árboles de cajas de inclusión con lados paralelos propuesto por Gino Van Den Bergen. Una de las principales ventajas del mismo reside en la gran rapidez en la actualización del árbol jerárquico de los objetos. [11]

Al trabajar con objetos convexos, Solid hace uso de la librería QHull, la cual permite calcular la superficie convexa de un conjunto de puntos. [11]

1.1.6 V-CLIP

EL algoritmo de V-CLIP (Voronoi Clip) es un algoritmo para la detección de colisiones entre objetos poliédricos convexos [12].

La librería V-Clip es una implementación en C++ del algoritmo V-Clip, realizada por MERL (Mitsubishi Electric Research Lab) que además facilita la manipulación de geometrías. Es capaz de calcular los puntos más cercanos de dos poliedros y la distancia entre los mismos en tiempo constante. Si existe penetración, V-Clip devuelve la profundidad de la misma. [14]

1.1.7 Principales Características

A continuación se muestra un resumen de las principales características de las librerías anteriores mediante una tabla.

Tabla 1 Tabla Resumen

Librería	Tipo de Modelos	Método o Algoritmos Utilizado	Entrada	Resultado
I-COLLIDE	Modelos (o conjunto de modelos) convexos	Lin-Canny	Conjunto de elementos en una escena virtual	Distancia entre cada uno de los pares de modelos de la escena
V-COLLIDE	Modelos triangulares	OBB	Conjunto de Objetos en una escena virtual	Par de objetos de la escena que se solapan
RAPID	Modelos triangulares	OBB	Par de modelos	Triángulos de los modelos que se solapan
			Conjunto de Modelos	Par de modelos que se solapan
SWIFT	Geometrías cerradas y convexas(*)	Lin-Canny	Par de modelos	-Distancia entre modelos (Aproximada o Exacta) -Primitivas más cercanas
SOLID	Objetos Convexos	Algoritmo GJK AABB	Par de objetos	-Punto en común de los objetos -Puntos más cercanos en el paso de simulación anterior
V-CLIP	Modelos Convexos	V-clip	Par de poliedros	Puntos más cercanos y la distancia entre ellos

1.2 Estrategias y Técnicas Actuales para la detección de colisiones

A continuación se expone el estudio realizado a determinadas técnicas y algoritmos que pudieran ser utilizados en este proyecto, dadas las tendencias para el desarrollo a nivel mundial de sistemas similares para el que se está concibiendo en este trabajo.

1.2.1 Volúmenes de Inclusión

Un volumen de inclusión es una forma primitiva que encierra el modelo y simplifica la cantidad de polígonos y tiene las siguientes propiedades:

- Debe ajustarse lo más estrechamente posible al modelo.
- Los volúmenes de inclusión deben tener una cantidad de polígonos lo mas pequeña posible.
- Debe poder ser representado con una cantidad relativamente pequeña de memoria.
- Los volúmenes más simples tienen cálculos más rápidos pero pueden necesitar más operaciones que los volúmenes complejos.

1.2.1.1 Esferas de Inclusión:

Las esferas son el más simple y uno de los tipos de volúmenes de inclusión más comúnmente usados. Una esfera puede ser guardada utilizando sólo 4 escalares. Testear una intersección entre una pareja de esferas necesita solamente 11 operaciones primitivas. [3]

Para muchas formas, las esferas no son el tipo de forma más ajustado, sin embargo su simplicidad y el hecho de no variar con las rotaciones hacen que sea un volumen de inclusión popular en entornos dinámicos. [3]

1.2.1.2 AABB (axis aligned bounding boxes)

El sistema de referencia del bounding box tiene sus ejes paralelos a las aristas de la caja. Dependiendo de la dirección de esas aristas el volumen de la caja será mayor o menor, con lo que se ajustará más o menos al objeto real, puede elegir como ejes los mismos que tiene el mundo completo, dando lugar a lo que se llaman cajas fronteras alineadas a los ejes (axis-aligned bounding boxes o AABB). [7]

El cálculo de un AABB es muy sencillo. Una vez que se tienen todos los vértices del objeto en coordenadas del sistema de referencia del mundo, se recorren y se consigue el mínimo y máximo valor que toman en cada uno de los ejes; además la forma de almacenarlos es mucho más compacta, pues guardando únicamente dos vértices con los valores mínimos y máximos se pueden averiguar las coordenadas de los ocho vértices que lo componen. Determinar si un AABB es visible desde una posición determinada es más rápido de calcular que uno orientado. Cuando el objeto al que envuelven se traslada por el mundo, se puede aplicar la misma transformación de traslación a los dos vértices que almacenan el volumen, para que siga siendo válido. Lo mismo ocurre si el objeto es escalado. Sin embargo, este procedimiento no puede aplicarse a la hora de las rotaciones, pues al aplicar la rotación a los dos vértices no puede asegurarse que el AABB obtenido con los nuevos vértices delimite al objeto. Por eso, este tipo de bounding boxes es sólo adecuado para objetos estáticos, o que únicamente se trasladan o escalan, sin alterar su rotación, o que son rotados en raras ocasiones. [7]

Otro punto en contra de los AABB es que por lo general no son óptimos, de forma que se puede encontrar un OBB que tenga menos volumen que él, ajustándose mejor al objeto, lo que proporciona una probabilidad más alta de que el objeto sea descartado utilizando el volumen delimitador. [7]

1.2.1.3 OBB (Oriented Bounding Boxes)

El cálculo del OBB óptimo es bastante más complicado que el del AABB (es básicamente un problema de minimización). El método propuesto actualmente utiliza métodos estadísticos. Debido a que el tiempo de cálculo es relativamente grande, se suele obtener al principio del programa, cuando se carga cada uno de los objetos, o directamente se cargan desde el fichero donde están los modelos. [7]

Son más eficientes que los AABB porque aparte de su ajuste al objeto que envuelven, no se ven afectados por traslaciones, escalados o rotaciones de los objetos, es decir, el OBB no necesita ser recalculado. Lo único que se necesita es aplicar la misma transformación al sistema de referencia del volumen y a la longitud de sus aristas (si la transformación es el escalado). Eso aporta un factor decisivo a la hora de la elección del tipo de delimitador elegido para objetos dinámicos. Sin embargo, comprobar si un OBB es visible requiere algo más de tiempo que el cálculo con un AABB, sin embargo la detección de colisiones es más compleja y por tanto su coste computacional es mayor, el coste del test de intersección de un OBB es de unas primitivas. Para reducirlo se usa un AABB que englobe al OBB para reducir un gran número de primitivas en el caso de una gran densidad de objetos. [7]

1.2.2 Partición del Espacio.

Es la subdivisión del espacio en regiones convexas llamadas celdas. Se mantiene una lista de referencias a los objetos que son parcialmente contenidos en la celda.

Se pueden desechar rápidamente la comprobación de intersección de muchos pares viendo si están o no en la misma celda, además envía sólo los elementos visibles desde el punto de vista. Dentro de los esquemas más usados aparecen los siguientes:

- Uso de Rejillas Regulares
- Uso de Octrees
- Árboles BSP

1.2.2.1 Uso de Rejillas Regulares:

Se divide el modelo en una malla de voxels rectangulares uniformes. Sólo necesita almacenar la posición y el tamaño de la caja envolvente del modelo.

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

Su principal debilidad está dada en que las rejillas regulares manejan áreas densas y escasas con la misma subdivisión. Esto implica que no se pueden desechar las partes grandes del modelo en un alto nivel de la jerarquía.

Considere una escena de una cuidad. Aquí una celda encierra una tienda por lo cual debe contener muchas geometrías mientras que otra celda contiene solo la porción de una calle o sea una pequeña geometría. Se debe tratar de subdividir el espacio en celdas que tengan la mayor cantidad posible de geometrías, porque su manipulación es muy costosa. [8]

1.2.2.2 Uso de Octrees:

A diferencia de las rejillas regulares, el **octrees** (primero introducido por Andrew Glassner, 1984) es un método jerárquico para la subdivisión espacial. Para construir un octrees al principio se coloca un voxel a encerrar la geometría de toda la escena. Después recurrentemente subdividimos cada voxel que encierra un área muy grande en ocho subvoxels del igual tamaño. Cada polígono es adjudicado al voxel más pequeño que encierra el polígono completamente. La colocación de los polígonos también puede ser hecha como la de las rejillas regulares. [8]

En los árboles octales, el estado de cada nodo ya no es binario. Cada nodo del árbol consiste en un código triario y ocho punteros hacia ocho hijos numerados del 0 al 7. Si el código es negro (lleno), la parte del espacio representada por el nodo está totalmente dentro del objeto, siendo el nodo una hoja del árbol. Si el código es blanco (vacío), la

parte del espacio no tiene objeto y el nodo es también una hoja. La tercera posibilidad es que el código sea gris (parcial) y corresponde al caso en que la parte del espacio correspondiente está parcialmente llena intersectando al objeto. [2]

En este último caso, los ocho punteros apuntan a ocho hijos que resultan de realizar una nueva subdivisión en octantes. [2]

Como en la enumeración exhaustiva, los octrees tienen una precisión limitada por las representaciones aproximadas. El modelo es siempre válido, no ambiguo y único. En general, el número de nodos necesarios para la representación de un sólido es proporcional al área del objeto y se reducen notablemente las necesidades de almacenamiento respecto al empleo de voxels. El modelo soporta operaciones cerradas como traslaciones y rotaciones. [2]

Las representaciones mediante quadtrees y octrees tienen actualmente un gran desarrollo, empleándose frecuentemente en visión artificial o robótica. Con el fin de reducir espacio de almacenamiento, se utilizan variantes como la subdivisión binaria, donde un nodo gris se divide en dos mitades (en vez de ocho), realizándose subdivisiones sucesivas en las direcciones (X, Y, Z). Otras variantes tratan de compactar aún más el espacio requerido, como la denominada **subdivisión lineal del espacio**. [2]

El Octrees no es muy conveniente para escenas dinámicas cuando la actualización de la estructura de datos no es tan trivial como en caso de una rejilla regular, sobre todo si se quiere reconstruir el octree parcialmente (muchos objetos dinámicos en la escena podrían hacer el viejo obsoleto octrees después de algún tiempo). Pero el octrees tiene una ventaja: si un voxel no es visible (o no participa en controles de colisión) entonces todos sus hijos no son visibles tampoco, entonces se puede usar el octrees para entresacar partes no visibles y grandes de la escena de una forma muy rápida y óptima. [8]

1.2.2.3 BSP- Tree

En un árbol de particionamiento binario, se divide el espacio en cada nodo utilizando un plano de particionamiento libremente orientado, ubica las geometrías en el espacio al que pertenece y este proceso se aplica de forma recursiva. Es una alternativa al particionamiento ortogonal y es útil para representar objetos cóncavos. [3]

Henry Fuchs en 1980 lo desarrolló para solventar el problema de las superficies ocultas en las escenas estáticas. Lo que lo hace diferente a los otros algoritmos escritos de subdivisión del espacio, es que aquí la subdivisión que se usa en un plano puede hacerse sin alinearlo a las coordenadas del pivote. El volumen que envuelve la escena inicialmente es recursivamente dividido escogiendo un polígono y utilizando un plano como plano de subdivisión. La subdivisión continúa hasta todos los polígonos que se han agregado al árbol. Todos los polígonos que cortan el plano son divididos, generando un árbol-BSP así también pueden aumentar la cantidad de polígonos en la escena. [8]

El árbol-BSP no es recomendable para escenas dinámicas, aunque la versión *areanode* sí puede ser eficiente en este tipo de escenas, también nótese como la subdivisión del plano afecta el árbol-BSP. Construir el árbol también es una tarea que consume tiempo. Aunque el Árbol-BSP requiere de escenas estáticas para ser más eficiente, muchos juegos de computadoras hechos recientemente apuestan por este algoritmo de los 80. (Quake arena III, Unreal Tournament, Half-life, Doom III, entre otros). [8]

1.2.3 Poda.

La tarea de detectar una colisión es secuencial y sigue 3 etapas. La idea del módulo de poda es discriminar casos para no tener que seguir haciendo más pasos.

Para los problemas donde varios cuerpos se están moviendo al mismo tiempo (tal como bolas de billar en una mesa) un paso de poda preliminar sirve para reducir el número de pares de objetos que se necesita considerar para la colisión. Si uno tiene n

objetos entonces hay n(n-1)/2 pares de objetos que tienen la posibilidad de colisionar, tal que uno puede iterar los otros pares y esto genera un algoritmo de O (n2).

Por supuesto en problema para los objetos grandes muy cerca unos de otros, no es fácil encontrar una línea que separe los objetos en dos sistemas que no se intersecten. Esto se puede resolver un poco, haciendo un algoritmo recursivo, y obtener un programa que trabaje más rápidamente para ciertos datos cuando n es grande.

En el peor de los casos, se observa si todos los objetos ocupan el mismo punto en espacio, entonces necesariamente habrá n(n- 1)/2 = O(n2) pares a comprobar. Si se esta interesados en escribir algoritmos de salida sensible donde el tiempo de ejecución esté acorde con el tamaño de la salida. En nuestro caso, éstos son los algoritmos que funcionarán más rápidamente cuando el número real de colisiones es pequeño. [16]

1.2.4 V-Clip o clip de Voronoi.

El V-Clip (o clip de Voronoi) es un algoritmo basado en los algoritmos Lin-Canny y GJK (Gilbert, Jonson, Keerthi) para la detección del par de características más cercano entre los poliedros convexos usando representaciones del límite. Usando la topología del poliedro convexo, el algoritmo se mueve de características a características hasta que encuentra que los poliedros se intersectan o hasta que encuentra el par de características con la distancia mínima de separación. [12]

El resultado de este acercamiento es que el algoritmo es más robusto que Lin-Canny (particularmente en la detección de poliedros que se intersectan), y ofrece mejor funcionamiento que el algoritmo GJK. También ofrece una mejora general en robustez numérica. Porque el algoritmo trabaja de un punto de partida dado en cada cuerpo, se satisfacen particularmente las pruebas sucesivas de la separación de los cuerpos que se mueven en torno a uno. [12]

Las pruebas del funcionamiento proporcionadas se basan en diversos niveles de la coherencia entre los objetos. El algoritmo se restringe a los objetos convexos, así que los objetos no convexos se deben descomponer en elementos convexos u otros

acercamientos tales como octrees o cajas de limitación pueden ser más apropiados. El algoritmo V-Clip podría ser una herramienta útil en aplicaciones de simulación implicados en la animación u otras áreas donde sea necesario detectar la proximidad o la colisión de objetos móviles. [12]

1.2.5 Algoritmo GJK.

El algoritmo desarrollado por Gilbert, Jonson and Keethi (GJK) tiene sus orígenes en la programación matemática y usa un procedimiento de descenso de distancia entre polígonos multidimencionales. La teoría general del algoritmo es generar una secuencia de polígonos cuyos vértices son un subconjunto de los vértices originales tal que los puntos cercanos del polígono multidimencional convexo a un origen dado convergen a la solución. Como declaró Mirtich, la distancia entre el origen y la cara más cercana del poliedro es igual a la distancia original entre los poliedros. Cuando los polígonos convexos colisionan, el algoritmo devuelve el grado de penetración. [9]

Un subalgoritmo para evaluar la distancia entre los polígonos elementales contribuye a la eficacia global del algoritmo. Se declara que el algoritmo termina después de un número finito de pasos cuando la entrada es un polígono compacto y convexo. Sin embargo, un procedimiento de reserva puede ser invocado para finalizar el algoritmo porque los errores numéricos pueden hacer que el subalgoritmo falle. Pueden tomarse varios pasos reducir los errores en el algoritmo y llevar a una aplicación más eficaz. Primero, el origen del sistema puede ser movido para estar en la línea que une el centroide de los vértices. También, una inicialización especial basada en el conocimiento de un estado puede usarse en lugar de la sola inicialización del punto. El paso más probable para determinar el punto cercano, basado en el paso anterior, puede ponerse al principio de la lista. [9]

Este algoritmo se probó seleccionando pares de polígonos de una familia de doce. El número de vértices fue de 2 a 100, para que las geometrías no fueran sumamente complejas. Los ejemplos estudiaron los casos dónde los pares de polígonos estaban separados, solo tocándose e intersectándose. Un par dado de poliedros fue alterado con traslaciones y rotaciones arbitrarias para generar un tamaño de muestra grande.

Después de estas pruebas se concluyó que el tiempo de ejecución en el CPU (Unidad Central de Procesamiento) estuvo en el rango de 10 a 30 milisegundos (ms). [9]

1.2.6 Algoritmo de Lin-Canny

Es uno de los algoritmos más representativos, sirve para objetos poliédricos y convexos. Las colisiones se detectan en base a vértices, aristas o caras. El algoritmo de Lin-Canny mantiene un par con los elementos más cercanos (vértices, aristas, o caras) entre dos poliedros convexos y la distancia entre esas características para detectar la colisión. [3]

El método hace uso de la coherencia ya que las características más cercanas entre objetos cambian poco en el tiempo. [11]

Aunque este algoritmo es sólo para objetos convexos, algunas librerías lo adaptan mediante el uso de un árbol jerárquico, de tal manera que los objetos no convexos los tratará como un árbol donde todos los elementos pueden ser convexos o no convexos excepto en el último nivel donde tienen que ser convexos. [11]

Al suponer coherencia temporal, se considera que los dos elementos más cercanos son la pareja de primitivas resultado del paso de simulación anterior. Si no es así, el algoritmo considera que las primitivas más cercanas están cerca, a su vez, de las primitivas anteriores y por tanto estudia la distancia en las primitivas vecinas. [11]

1.3 Lenguajes de programación

En esta sección se analizará el lenguaje de programación a utilizar.

Existen principalmente tres lenguajes que se utilizan para desarrollar aplicaciones gráficas profesionales en 3D: Lenguaje Ensamblador, C y C++, por ser los que con más velocidad ejecutan el código (menor costo de ejecución del programa). A éstos se ha unido recientemente el Java como una opción para el desarrollo de este tipo de aplicaciones.

La decisión de implementar este proyecto mediante el lenguaje C++, esta dada en que el motor de visualización existente en nuestro proyecto, para el cual se esta desarrollando este módulo esta hecho sobre el lenguaje C++. Además del potencial de este lenguaje a la de trabajar con gráficos.

Si se estudian las características de este lenguaje, se puede apreciar lo acertado de la elección, dado que C++ es un lenguaje de programación de propósito general, especialmente indicado para la programación de sistemas por su flexibilidad y su potencia. Es uno de los más utilizados en las comunidades de desarrolladores de *software*, incluyendo la programación gráfica. [10.1]

C++ es la evolución de C adaptada a la programación orientada a objetos. Tiene algunas cuestiones más pulidas como un control más estricto en el manejo de tipos de datos, y otras características que ayudan a la programación libre de errores. [10.1]

En general puede llegar a ser un lenguaje tan rápido como C (el más rápido después del Lenguaje Ensamblador), sin embargo, si se maneja herencia múltiple, funciones virtuales y polimorfismo en forma inadecuada, o se accede mucho en niveles de profundidad en la llamada a objetos (Objeto1.Objeto2.Objeto3.Objeto4…), puede llegar a hacerse un poco más lento, lo cual no es conveniente para una aplicación en tiempo real. [10.1]

Conclusiones

Durante el desarrollo de este capitulo se ha hecho un estudio detallado y especifico de algunas de la librerías de detección de colisiones existentes a nivel mundial, además de una profunda investigación de las principales técnicas y algoritmos utilizados por estas librerías para lograr un manejo adecuado y muy óptimo de las colisiones en los entornos virtuales.

2 Soluciones Técnicas

Introducción

En este capítulo se proponen las soluciones técnicas para lograr un procedimiento eficiente para la detección de colisiones en Entornos Virtuales y se explicarán las ventajas que traerá consigo el uso de estas.

2.1 Volumen de Inclusión a Utilizar.

Para el desarrollo de este proyecto se han estudiado varios volúmenes de inclusión con el objetivo de minimizar la complejidad de los cuerpos en el momento de analizar las colisiones. El volumen de inclusión que se ha decidido utilizar es el *"Oriented Bounding Boxes (OBB)",* se eligió esta variedad ya que la mayoría de los objetos dinámicos que pueblan las escenas urbanas son geometrías irregulares por lo cual el OBB es mas eficiente para lograr un encierre ajustado al objeto y al espacio temporal que puede ocupar este.

El manejo de este volumen de inclusión es muy sencillo si partimos desde el punto de vista que se puede crear a la hora de cargar el objeto con solo realizar algunas operaciones matemáticas y adicionarle cierta cantidad de vértices a la matriz del objeto, evitando tener que realizar un gran número de operaciones matemáticas para crearlo después de cargado el objeto. Además se ajuste muy bien al objeto que envuelve, no se ven afectados por traslaciones, escalados o rotaciones de los objetos, es decir, el OBB no necesita ser recalculado.

2.2 Particiones del espacio para encerrar la escena

El espacio se puede dividir con varias estructuras de datos como son la *"Rejilla Regular",* el *"BSP-Tree",* el *"Octrees"* entre otros. Para realizar este trabajo se ha decidido utilizar la "Rejilla Regular" para la división del espacio debido a que después que se ajusta la rejilla a la escena, en cada una de las celdas de dicha rejilla habrá una lista donde sólo necesita estar almacenada la posición y el tamaño de la caja envolvente del modelo o modelos que se encuentren contenido en la celda, cuando el objeto sea movido de posición si cambia de celda solo habrá que eliminarlo de la lista donde se encontraba y colocarlo en la lista perteneciente a la celda a la que fue trasladado. Esto facilita el manejo de los modelos dinámicos y ahorra una gran cantidad de tiempo y recursos, lo cual facilitaría el trabajo a la hora de reducir el espacio donde se harán los test de colisión mediante el método de la *"Poda".* Más adelante se abordará con más detalles el trabajo con este método.

2.3 Método para reducir el área a comprobar la existencia de colisiones

En una aplicación de Realidad Virtual en tiempo real lo más importante a la hora de comprobar la existencia de colisiones es la velocidad en que esto se realiza para poder crear en el usuario una mayor sensación de realismo. Para ello es necesario discriminar pares de objetos a la hora de realizar el test de colisiones para disminuir la cantidad de pasos a realizar y así disminuir el tiempo de procesamientos. Para ello se utiliza el método de la Poda con algunas modificaciones que permitan resolver nuestro problema de una forma óptima. Por ejemplo en la poda para reducir el área donde se realizará el test lo que hacen es dividir el entorno en 4 partes básicamente y solo se realizán test en el cuadrante donde se encuentre situado el objeto a comprobar y por tanto en el caso de que el objeto este en el limite de un cuadrante puede existir una colisión justo delante nuestro y no ser detectada, además de que el área pudiera resultar extremadamente grande por lo que se pudieran hacer algunos test de colisión que resultarían innecesarios en algún momento. Para solucionar este problema se ha decidido realizar un boundaring temporal con forma de una circunferencia tomando como centro nuestra posición y un radio acorde con nuestro campo de visión, todas las celdas que pertenezcan a dicha circunferencia serán almacenadas para crear el boundaring temporal y luego a los objetos que estén contenidos en esas celdas será a los que se le realice el test de colisión.

Procedimiento Matemático.

Atributos:

Dir en Y = (0,1,0)

Left en X = (1,0,0)

Loc = Location (localización del objeto)

Rad = Radio

Vectores $\mathbf{P_1}$, $\mathbf{P_2}$, **P3** y $\mathbf{P_4}$ =?

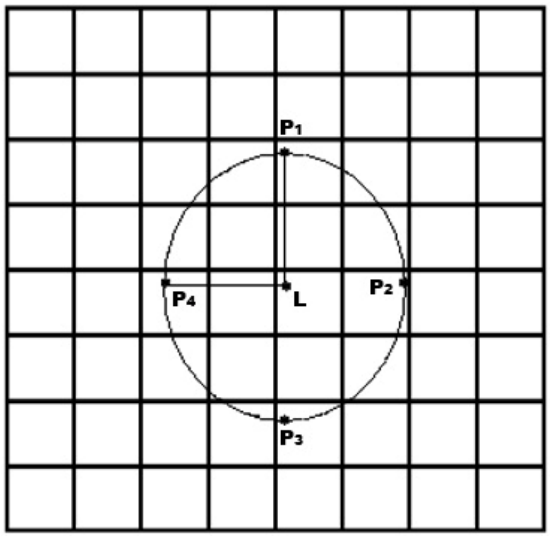

Fig. 1 Rejilla con nuestra ubicación y el área donde se hará el test de comprobación.

P_1 = Dir en Y * Rad + Loc

P_3 = **- P_1**

P_4 = Left en X * Rad + Loc

P_2 = **- P_4**

2.4 Método para determinar si dos cajas se intersectan

Como se explicaba en el en epígrafe 2.1 se ha decidido utilizar el OBB *"Oriented Bounding Boxes"* como volumen de inclusión por lo que se caracterízaran todos los objetos dinámicos a tratar como cajas. Debido a esto se necesita un modelo o método que permita conocer con exactitud y rapidez si dos cajas se intersectan. Para ello se tomaran los 4 vértices de la base de una de las 2 cajas y se trata como polígonos y mediante un algoritmo matemático comprobamos si alguno de los vértices de la base de la otra caja se encuentra en contacto con la base de la primer caja y viceversa. De existir contacto alguno existirá colisión entre los 2 objetos en cuestión.

Conclusiones

A lo largo de este capítulo se ha realizado una propuesta de las técnicas que posibilitarán darle solución a nuestro problema de la forma mas eficiente teniendo en cuenta el tipo de aplicación para la cual esta destinado este módulo.

3 Descripción de las Propuestas

Introducción

En este capítulo se comienza a tener la visión del sistema a desarrollar. Aquí se inicia la concepción práctica del producto a elaborar, sobre la base de las dificultades, necesidades y características organizacionales del cliente, conociendo ya las técnicas a utilizar descritas en el capítulo anterior.

3.1 Reglas de negocio

Las celdas de la rejilla deben tener un tamaño relativamente grande para lograr que la rejilla no sea excesivamente grande. La rejilla no puede eliminarse o modificarse hasta que no termine la ejecución de la aplicación.

El boundaring del objeto debe crearse al iniciarse la aplicación cuando se carguen los objetos dinámicos para un mejor manejo de los mismos y debe quedar lo más ajustado posible para obtener un resultado óptimo y de alta calidad, el objeto debe de cargarse orientado al eje x para que no provoque la afectación de los boundaring cuando sean rotados o trasladados posteriormente.

A cada objeto se le debe de crear un boundaring de proximidad. Este boundaring debe eliminarse y crearse uno nuevo siempre que el objeto sea trasladado de su posición actual. Si el objeto no se ha movido o no se le ha realizado el test de proximidad el boundaring no puede afectarse.

El módulo debe ser acoplable a la arquitectura del SceneToolkit (STK).

3.2 Modelo de Dominio

Dada la relación entre los conceptos fundamentales se obtiene el siguiente modelo de dominio. Esto es un pequeño acercamiento a la solución del problema.

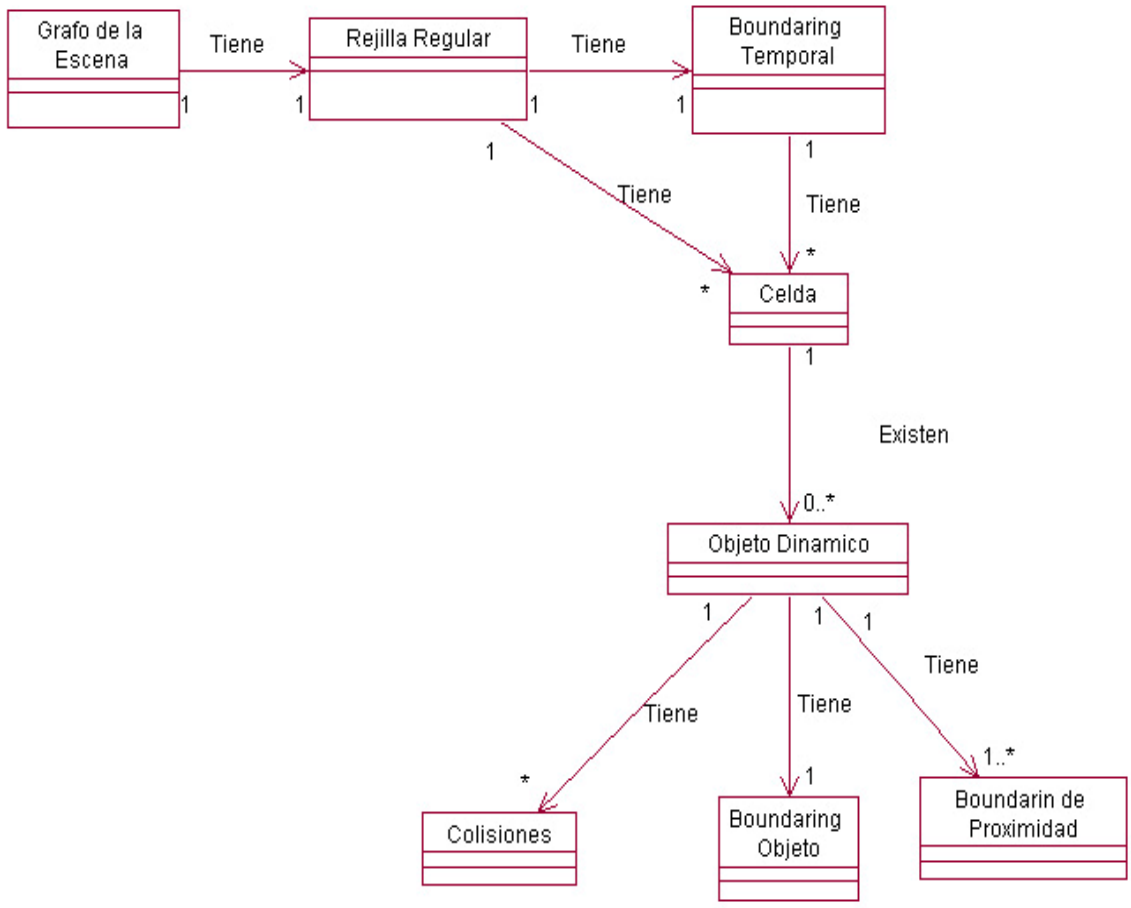

Fig. 2 Modelos de dominio

3.3 Glosario de Términos

Grafo de Escena: estructura jerárquica donde se almacenan y organizan los objetos de la escena para el control de su información y determinación de la visibilidad, donde las hojas contienen los objetos dibujables de la escena.

Celda: Espacio Regular en forma cuadrada en la cual se almacena la información de los objetos dinámicos.

Rejilla Regular: Estructura no jerárquica compuesta por celdas en forma matricial. (Epígrafe 1.2.2.1).

Boundaring Temporal: Es un conjunto de celdas de la rejilla que delimita un área de la misma.

Boundaring de Proximidad: área que rodea el objeto con radio x mediante la cual se puede saber si 2 objetos están próximos uno del otro.

Boundaring Objeto: Es un volumen de inclusión es una forma primitiva que encierra el modelo.

Objetos Dinámicos: Objetos con movimiento.

Colisión: Acción que ocurre cuando 2 objetos dinámicos se intersecan.

3.4 Captura de requisitos

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

3.4.1 Requerimientos Funcionales

1- Crear Rejilla Regular.

1.1 Determina el tamaño que debe tener las celdas.

1.2 Crea un número de celdas igual a una cantidad de [filas x columnas].

2-Eliminar Rejilla Regular

2.1 Elimina todas las celdas que la conforman.

3- Crear Boundaring de Temporal.

3.1 Calcula el área que tendrá el boundaring.

3.2 Determina las celdas que pertenecen a dicha área.

3.3 Crea lista de celdas que conformará el boundaring.

3.4 Devolver lista de celdas

4- Eliminar Boundaring Temporal.

4.1 Limpiar lista de celdas conformará el boundaring.

4.2 Destruir el boundaring.

5- Crear boundaring del objeto.

5.1 Determinar posición del objeto.

5.2 Determinar límite frontal, trasero, lateral derecho e izquierdo.

5.3 Calcular a posición donde se deben colocar los vértices para formar la caja frontera más ajustada al objeto.

5.4 Crear boundaring del objeto.

6- Actualizar boundaring del objeto.

6.1 Se actualizan los vértices del boundaring.

6.2 Se realizan las operaciones matemáticas correspondientes entre los vértices del boundaring con la matriz de orientación y el vector de traslación.

6.3- Se actualiza el boundaring.

7- Eliminar boundaring del objeto.

7.1 Eliminar los vértices que conforman el boundaring.

7.2 Destruir el boundaring.

8- Comprobar colisiones entre dos objetos.

8.1 Suma la distancia de los radios de proximidad de los 2 objetos a testear.

8.2 Calcula la distancia existente entre los objetos a comprobar.

8.3 Compara la distancia existente entre los objetos con la suma de las distancias de proximidad de cada uno de los objetos.

8.4 Si la distancia entre los objetos es menor que la suma de las distancias de proximidad entonces hay proximidad.

8.5 Determina si existe penetración entre los boundaring.

8.6 Si existe penetración existe colisión y se devuelve verdadero.

8.7 Si no existe penetración entonces no existe colisión y se devuelve falso.

9- Detectar Proximidad.

9.1 Determina el radio de proximidad de cada objeto.

9.2 Suma la distancia de los radios de proximidad de los 2 objetos a testear.

9.3 Calcula la distancia existente entre los objetos a comprobar.

9.4 Compara la distancia existente entre los objetos con la suma de las distancias de proximidad de cada uno de los objetos.

9.5 Si la distancia entre los objetos es mayor que la suma de las distancias de proximidad entonces no hay proximidad, sino pasa a determinar si existe colisión.

10- Detectar colisión.

10.1 Determinar los objetos a testear.

10.2 Determinar el mínimo valor de x, y, z entre los vértices de cada boundaring y los compara.

10.3 Determinar el máximo valor de x, y, z entre los vértices de cada boundaring y los compara.

10.4 Determina si existe penetración entre los boundaring.

10.5 Si existe penetración existe colisión.

11- Crear Celda.

11.1 Crear Celda.

12- Buscar Objetos.

12.1 Devuelve la lista de objetos dinámicos.

13- Insertar Objeto.

13.1 Inserta el objeto en su lista correspondiente.

14- Eliminar Objeto.

14.1 Elimina el objeto de su lista correspondiente.

3.4.2 Requerimientos no Funcionales

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: En una versión inicial deberá ser compatible con la plataforma Windows.

Legales: Se regirá por las normas ISO 9000.

Software: Sistema operativo Windows.

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

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

Usabilidad: La aplicación debe ser implementada con el objetivo de ser reutilizado por otra aplicación similar.

3.5 Modelo de Casos de Uso del Sistema

En esta sección se reconocen los posibles actores del sistema a desarrollar y se conciben, a través de la agrupación de los requisitos funcionales que fueron definidos anteriormente, los posibles resultados de valor que le pueda brindar a sus actores, o lo que es lo mismo, 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.

Actor de sistema

Tabla 2 Actor del Sistema

Actor	Justificación
Programador	Es el que se beneficiará con las funcionalidades que brinda el modulo a desarrollar, a grosso modo: servicios del boundaring, detección de colisiones.

Casos de uso del Sistema

1- Manejar Rejilla.
2- Manipular Boundaring Temporal.
3- Manipular Boundaring de Objeto.
4- Manipular Colisiones.
5- Detectar Colisión.

El primer ciclo de desarrollo tiene como objetivo detectar las colisiones existentes entre objetos dinámicos o la posibilidad de que estos objetos puedan colisionar en un determinado instante de tiempo, para lo cual se decidió desarrollar las funcionalidades que brindan los casos de uso: Manipular Colisiones, Boundaring Temporal, Manejar Rejilla, Boundaring de Objeto, Boundaring Proximidad.

3.5.1 Diagrama de Casos de Uso del Sistema.

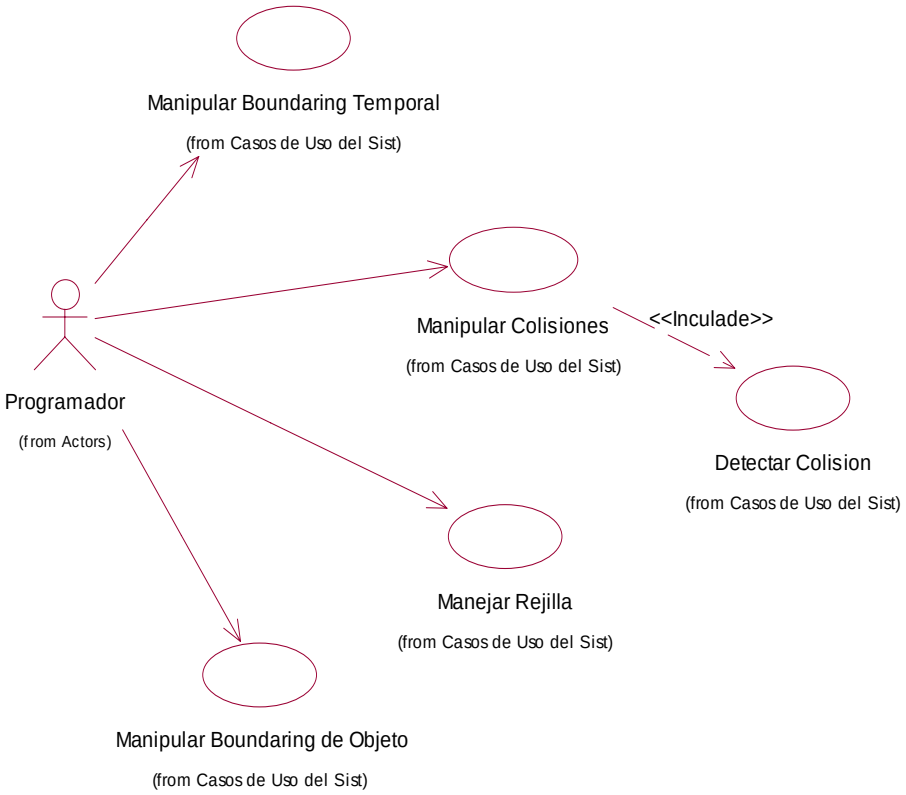

Fig. 3 Diagrama de Casos de Uso de Sistema

Descripción de los casos de Uso del sistema

Nombre del caso de uso	Manejar Rejilla
Actores	Programador
Propósito	Realizar todas las operaciones sobre una Rejilla Regular.
Resumen: El Caso de Uso se inicia cuando el Programador recibe las coordenadas de los vectores inicial y final del entorno que ha sido cargado. Al ocurrir esto se procede a calcular el tamaño de las celdas y a crear la rejilla, luego se ubican todos los boundaring de objeto en las celdas a la que corresponden según su posición en el entorno. Este caso de uso termina cuando se cierra la aplicación que se destruye la rejilla.	
Referencias	R- 3.1,3.2
Curso normal de los eventos	
Sección "Crear Rejilla Regular"	
Acción del actor	**Respuesta del sistema**
1. Solicita crear una Rejilla Regular con una cantidad específica de columnas y filas.	1.1 Determina el tamaño que debe tener las celdas. 1.2 Se calcula la posición de los vértices de la primera celda por las filas 1.3 Se calcula la posición de los vértices de cada celda por la columnas 1.4 Se crean todas las celdas que conforman la rejilla
Sección "Reubicar Objetos"	
Acción del actor	**Respuesta del sistema**
1. Solicita obtener todos los objetos dinámicos	1.1 Recorre la lista de bounaring de objetos de la escena 1.2 Verifica a que celda pertenece dada boundaring de objeto 1.3 Inserte el boundaring en la lista de la celda a la que corresponde
Sección "Eliminar Rejilla"	
Acción del actor	**Respuesta del sistema**
1.Solicita eliminar una Rejilla Regular	1.1 Se elimina la lista de objetos de cada celda 1.2 Elimina todas las celdas que la conforman

Nombre del caso de uso	Manipular Boundaring Temporal
Actores	Programador
Propósito	Realizar todas las operaciones con el boundaring temporal
Resumen: este caso de uso inicia cuando el programador activa el boundaring temporal, al ocurrir esto se procede a crear boundaring temporal de la forma mas optima, y terminaríamos esta caso de uso cunado se procede a eliminar boundaring temporal.	
Referencias	R- 2.1, 2.2, 2.3, 2.4
Curso normal de los eventos	
Sección "Crear Boundaring Temporal"	
Acción del actor	**Respuesta del sistema**
1- Solicita crear un boundaring	1.1- Calcula el área que tendrá el boundaring 1.2- Determina las celdas que pertenecen a dicha área 1.3- Crea lista de celdas que conformaran el boundaring 1.4- Devolver lista de celdas
Sección "Eliminar Boundaring Temporal"	
Acción del actor	**Respuesta del sistema**
1- Solicita eliminar boundaring	1.1- Limpiar lista de celdas conformaran el boundaring 1.2- Destruir el boundaring

Nombre del caso de uso	Manipular Boundaring de Objeto
Actores	Programador
Propósito	Realizar todas los operaciones del boundaring del objeto
Resumen: el caso de uso se inicia cuando el programador carga el objeto. Al ocurrir esto se crea el boundaring del objeto, luego cuando el objeto sufre alguna transformación se procede a actualizar el boundaring y terminamos cuando destruimos el boundaring.	
Referencias	R- 4.1,4.2

Curso normal de los eventos	
Sección "Crear boundaring del objeto "	
Acción del actor	**Respuesta del sistema**
1- Solicita crear Boundaring	1.1- Posición del objeto. 1.2- Determinar limite frontal. 1.3- Determinar limite trasero. 1.4- Determinar limite lateral derecho. 1.5- Determinar limite lateral izquierdo. 1.6- Calcular a posición donde se deben colocar los vértices para formar la caja frontera mas ajustado al objeto. 1.7- Crear boundaring del objeto
Sección "Actualizar boundaring del objeto "	
Acción del actor	**Respuesta del sistema**
1- Solicita actualizar Boundaring	1.1 Se actualizan los vértices del boundaring 1.2 Se realizan las operaciones matemáticas correspondientes entre los vértices del boundaring con la matriz de orientación y el vector de traslación. 1.3- Se actualiza el boundaring

Nombre del caso de uso	Detectar Colisión
Actores	Programador
Propósito	Realizar todas las operaciones de las colisiones
Resumen: el caso de uso se inicia al pasarle 2 objetos a comprobar, luego se le realiza un test de proximidad a los objetos de la lista y a los que se determine que están a una distancia relativamente pequeña se le s realiza un test de colisión.	
Referencias	R- 1.1, 1.3, 1.4, 1.5
Curso normal de los eventos	
Acción del actor	**Respuesta del sistema**
1- Solicita comprobar Colisión entre dos objetos	1.1- Suma la distancia de los radios de proximidad de los 2 objetos a testear. 1.2-Calcula la distancia existente entre los objetos a comprobar. 1.3 Compara la distancia existente entre los objetos con la suma de las distancias de proximidad de cada uno de los objetos. 1.4 Si la distancia entre los objetos es menor que la suma de las distancias de proximidad entonces hay proximidad. 1.5- Determina si existe penetración entre los boundaring. 1.6- Si existe penetración existe colisión y se devuelve verdadero. 1.7- Si no existe penetración entonces no existe colisión y se devuelve falso.

Nombre del caso de uso	Manipular Colisiones
Actores	Programador
Propósito	Realizar todas las operaciones de las colisiones
Resumen: el caso de uso se inicia cuando se activan las colisiones al ocurrir esto se determina la lista de objetos a comprobar, luego se le realiza un test de proximidad a los objetos de la lista y a los que se determine que están a una distancia relativamente pequeña se les realiza un test de colisión.	
Referencias	R- 1.1, 1.3, 1.4, 1.5
Curso normal de los eventos	
Acción del actor	**Respuesta del sistema**
1- Solicita comprobar Colisión para una lista de objetos	1.1- Verifica si se utilizara el boundaring. 1.2- Si se utiliza el boundaring. 1.2.1 Se recorrerá la lista de celdas pertenecientes al boundaring. 1.2.2 Se recorrerá la lista de objetos dinámicos pertenecientes a cada celda. 1.2.3 Comparar colisión entre los objetos de la lista (**Invocar CU Detectar Colisión**). 1.3 Si no se utiliza el boundaring 1.3.1 Se recorrerán todas las celdas de la rejilla. 1.3.2 Se recorrerá la lista de objetos dinámicos pertenecientes a cada celda. 1.3.3 Comparar colisión entre los objetos de la lista (**Invocar CU Detectar Colisión**).

Conclusiones

En el presente capítulo se definió qué es exactamente lo que espera el usuario con este sistema. Para ello quedaron establecidos los requisitos funcionales de éste, y descritos los casos de uso que le permitirán a su futuro usuario obtener los resultados esperados por el cliente.

4 Diseño del Sistema

Introducción

En este capítulo se verán los diagramas de clases de diseño del sistema propuesto como resultado de un refinamiento de las etapas anteriores, Se presentan además los diagramas de secuencia de la realización de los casos de uso, que intervendrán en el primer ciclo de desarrollo del proyecto. Además se verán sus responsabilidades y relaciones durante el desarrollo del sistema.

4.1 Diagramas de clases de diseño

Antes de pasar a los diagramas de clases, se aclararán 2 cuestiones importantes para la mejor comprensión:

1- La nomenclatura utilizada para los diagramas de clases se explica en el epígrafe “Estándares de codificación” del siguiente capítulo.
2- En la realización de este proyecto, y a petición del cliente, se generó el código con la herramienta utilizada para el proceso de desarrollo del sistema (Rational Rose Enterprise Edition, 2003), la cual brinda la posibilidad de generar los métodos de acceso a miembros de clases (“gets” y “sets”), constructores por defecto, constructores de copia y destructores, aun cuando no se hallan incluido en las especificaciones de las clases; por tanto, y para evitar su repetición, se omite en las clases los métodos antes mencionados. En algunos casos se incluyen algunos constructores, pero no son los “por defecto”.

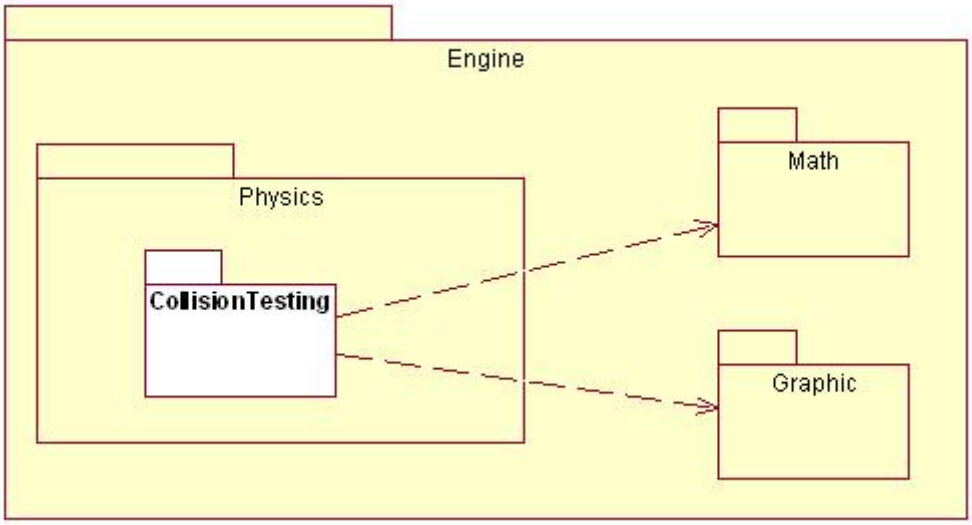

Fig. 4 Diagrama de paquetes de clases de diseño

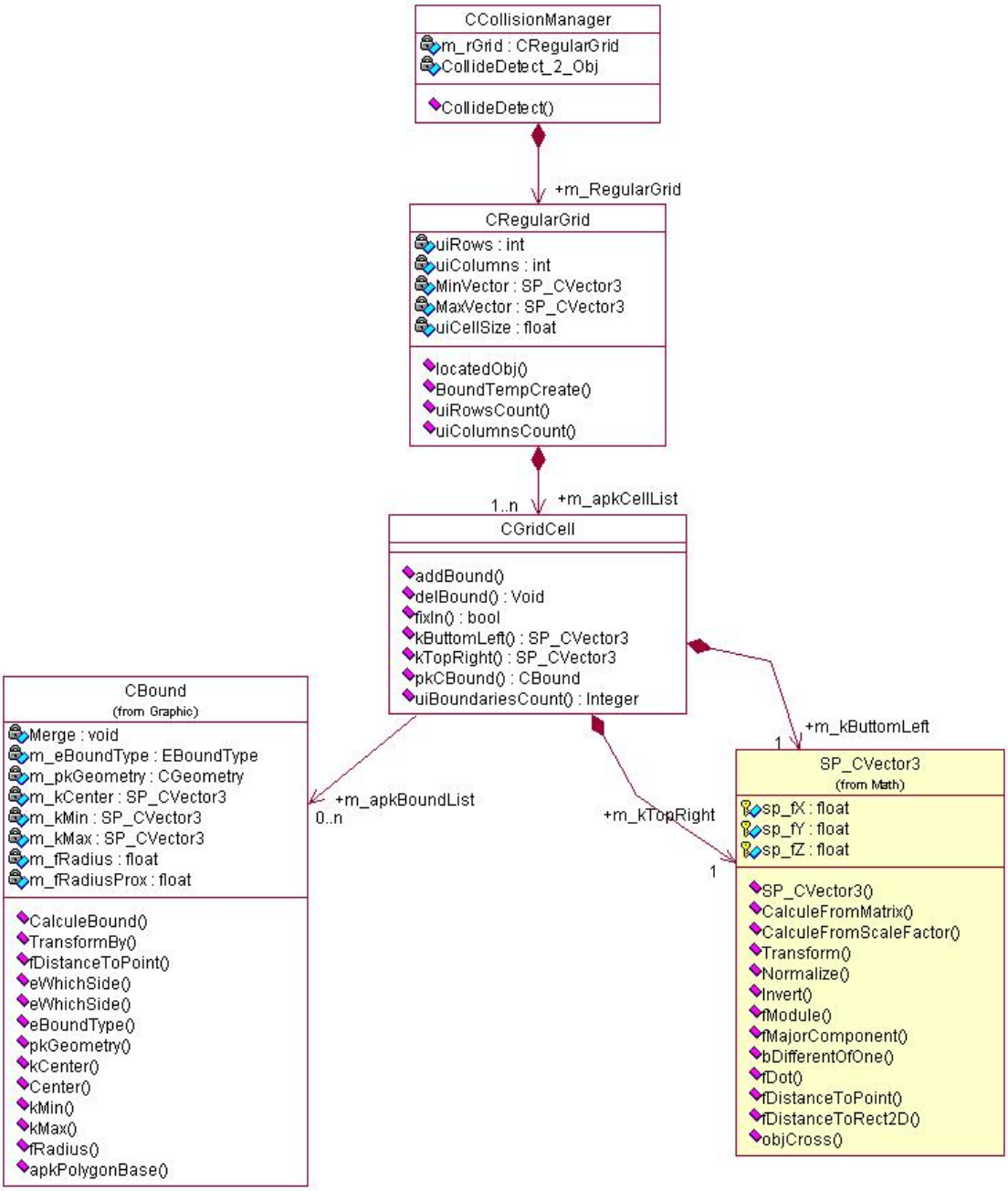

Fig. 5 Diagrama de clases de diseño "Manejar Colisión".

4.2 Diagramas de Secuencia

A continuación siguen los diagramas de secuencia del paquete "Colision Testing".

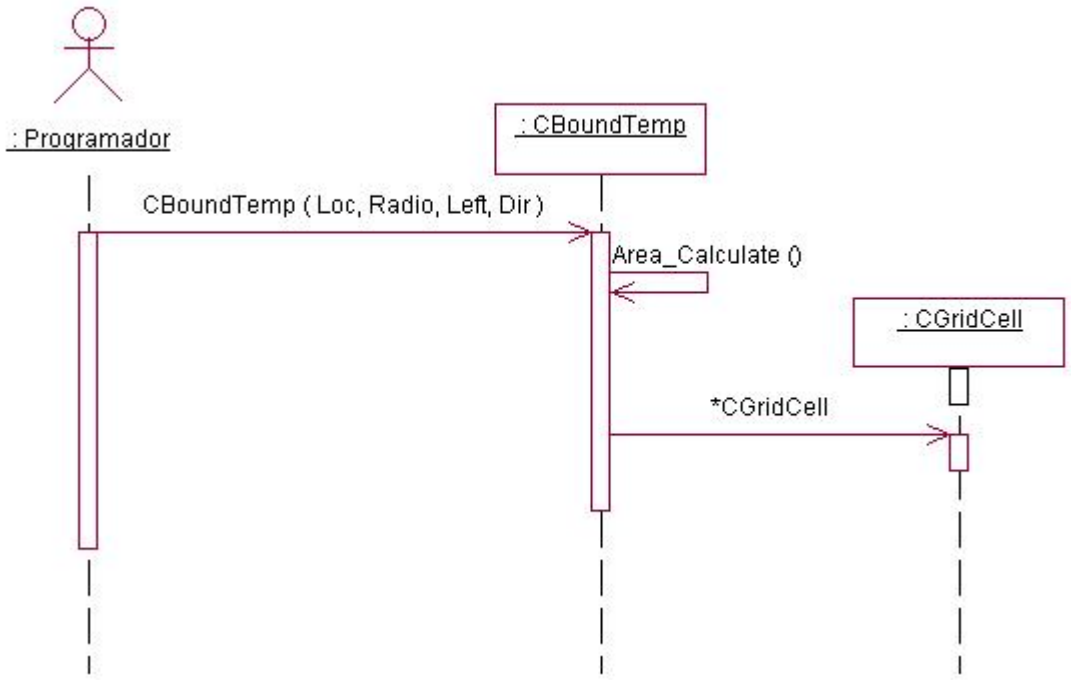

Fig. 6 Diagrama de secuencia "Crear Boundaring Temporal".

En este diagrama se muestra el flujo para crear un boundaring temporal. Para crear este boundaring se le debe pasar la posición del objeto, el radio de la dimensión que se desea que tenga, le boundaring y los vectores dirección y lateral izquierdo. Con esos datos se calcula el área que tendrá el Boundaring y se crea una lista con las celdas que pertenecen a dicha área.

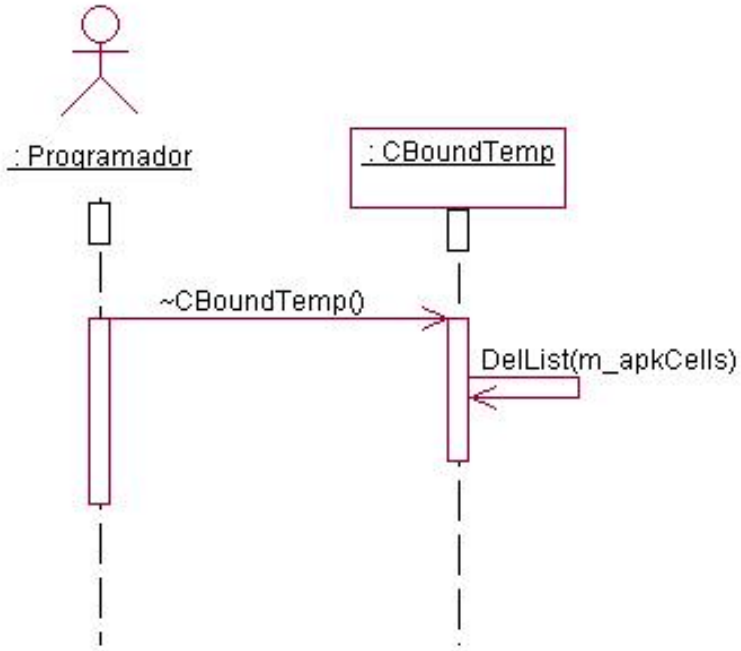

Fig. 7 Diagrama de secuencia "Eliminar Boundaring Temporal".

En este diagrama se muestra el flujo para eliminar boundaring temporal. Para ello solo se llama al destructor de la clase y se le pasa el objeto a destruir.

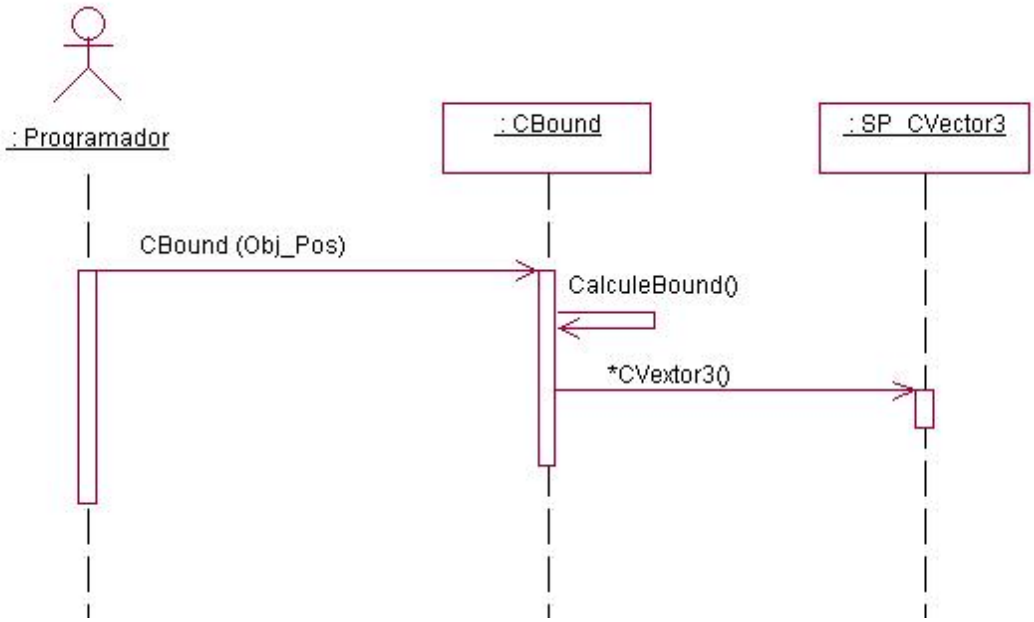

Fig. 8 Diagrama de secuencia "Crear Boundaring de Objeto".

En el diagrama anterior se muestra el flujo de secuencia para crear un boundaring de objeto, para crear el mismo se le debe pasar la posición del objeto al cual debemos crearle el boundaring, luego se le calculan los limites laterales, el frontal y el trasero y luego creamos un lista con los vectores que conformarían el boundaring.

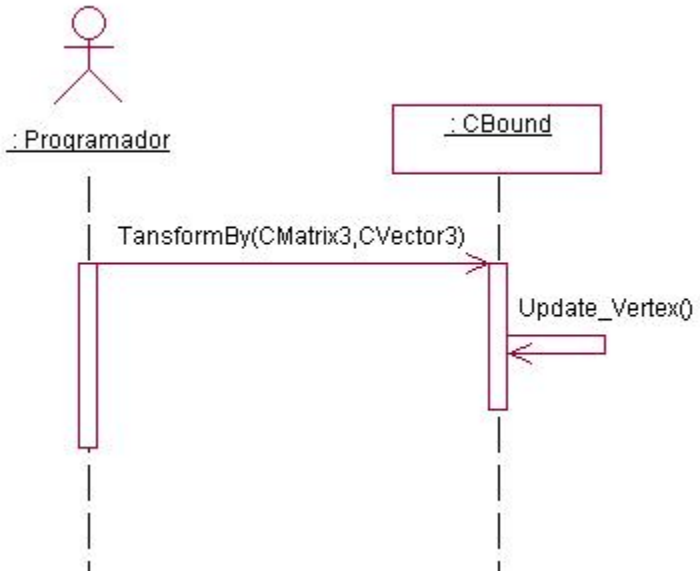

Fig. 9 Diagrama de secuencia "Actualizar Boundaring de Objeto".

En este diagrama se describe la secuencia de operaciones para actualizar un boundaring de objeto. Para iniciar se debe conocer la traslación y/o rotación que ha sufrido el objeto, luego se le aplican las modificaciones a cada vértice para que se modifiquen sus posiciones, de ser necesario, y quede actualizado el boundaring.

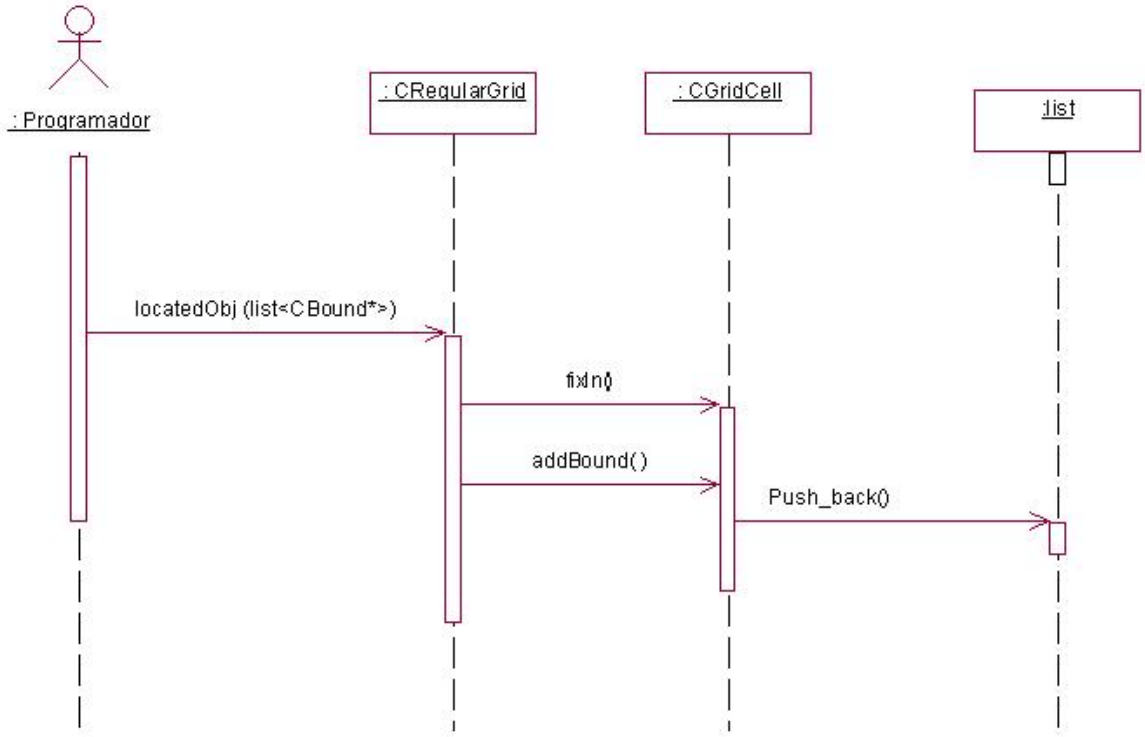

Fig. 11 Diagrama de secuencia "Ubicar Objeto".

En el diagrama anterior se muestra el flujo de secuencias a seguir para ubicar los boundaring de objeto un la rejilla regular. Para ello se recorre la lista de boundaring de toda la escena, se busca a que celda pertenece el objeto en cuestión y luego se inserta en la lista de dicha celda.

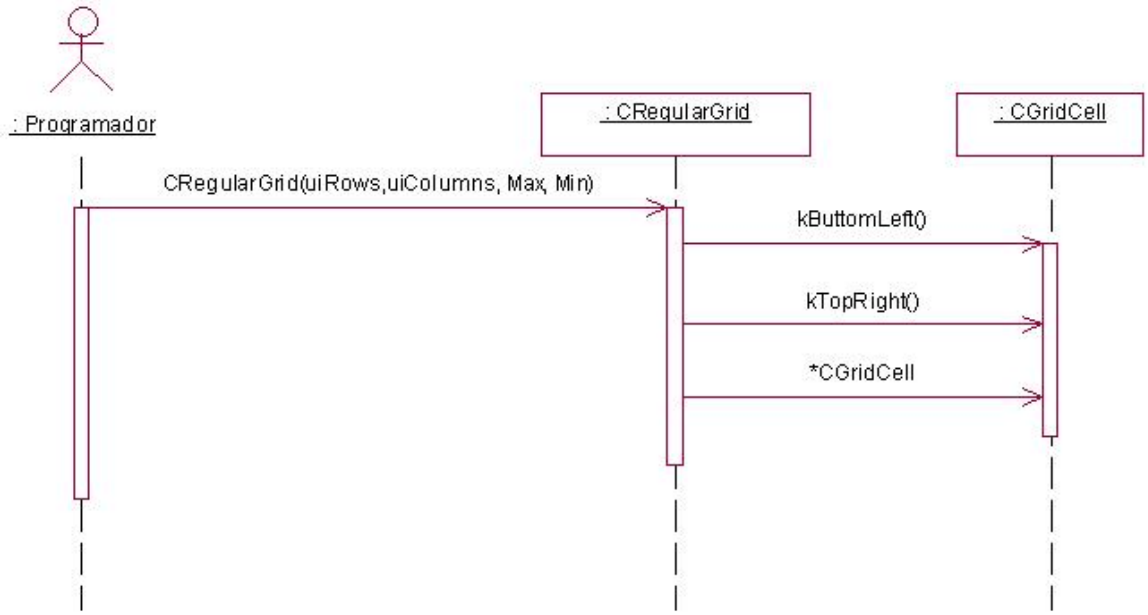

Fig. 12 Diagrama de secuencia "Crear Rejilla".

En este diagrama se describe la secuencia de operaciones que se realizan para crear la rejilla regular de la escena, este proceso se inicia pasándole como parámetro el número de filas y columnas que tendrá la rejilla, además de los vectores máximo y mínimo del entorno que ha sido cargado, luego se procede a calcular los vértices limite de las celda y posteriormente se crea la lista de celdas que conforman la rejilla

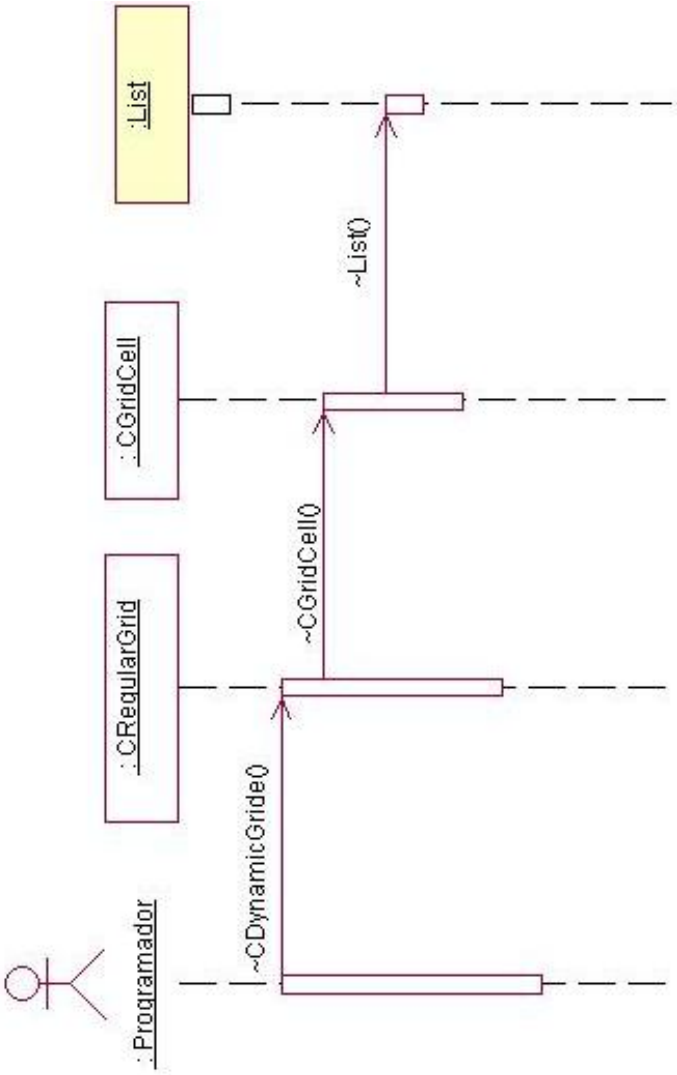

Fig. 13 Diagrama de secuencia "Eliminar Rejilla".

En el diagrama se muestra la secuencia de operaciones a seguir para eliminar una rejilla regular. Simplemente lo que se hace es que realizar una llamada al destructor de la rejilla, el mismo a su vez invoca el destructor de la celda y así sucesiva mente hasta destruir la lista de boundaring pertenecientes a cada una de las celdas.

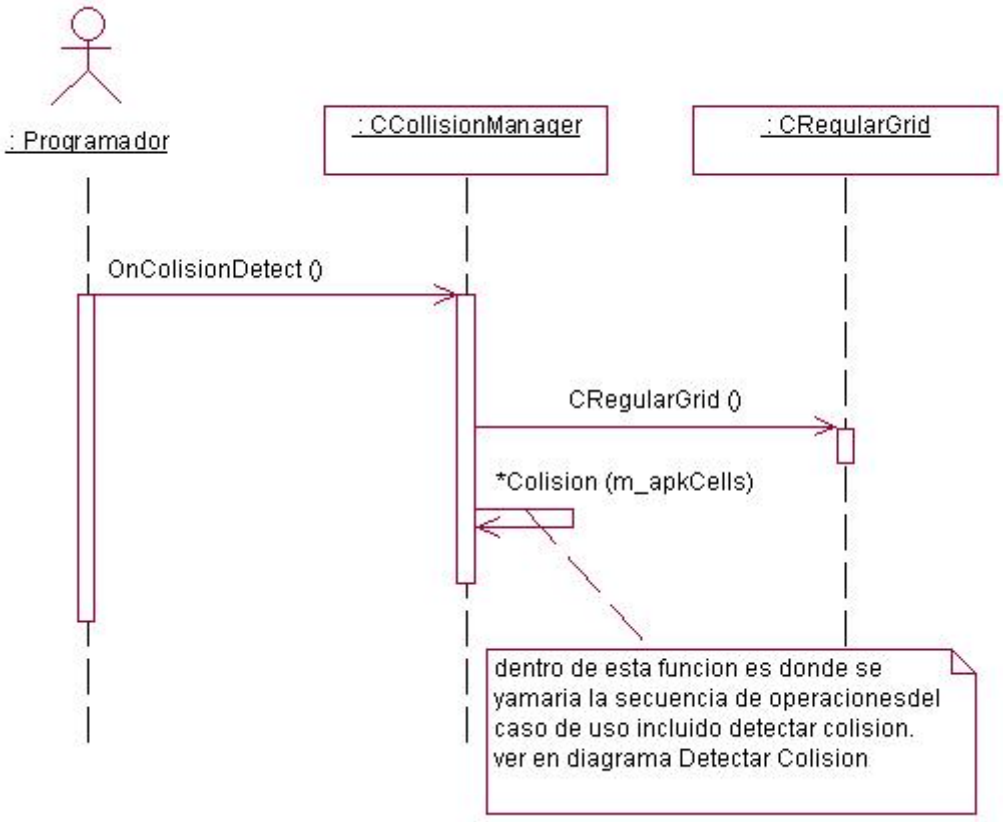

Fig. 14 Diagrama de secuencia "Manejar Colisión".

En este diagrama se muestra la secuencia de operaciones a seguir para detectar colisiones a una lista de objetos. Las operaciones se inician cuando se solicita testear las colisiones, luego se procede a recorrer la lista de celdas de la rejilla. Después se recorrerá la misma y se realizarán las operaciones que se muestran en el siguiente diagrama.

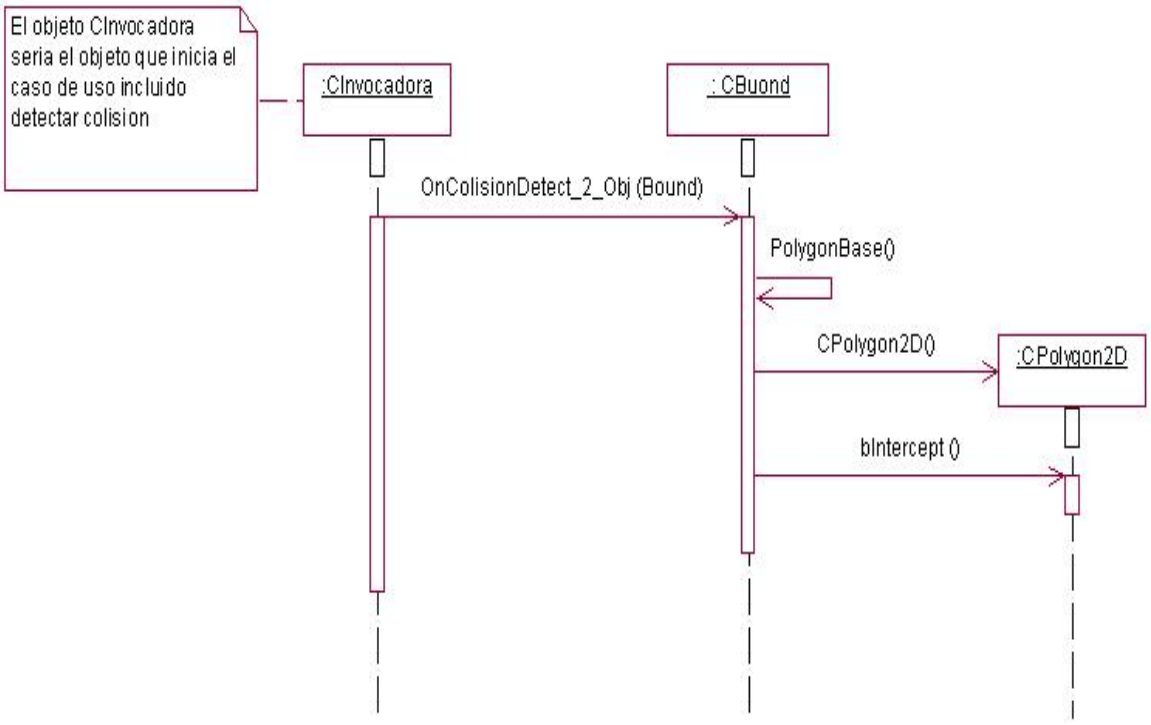

Fig. 15 Diagrama de secuencia "Detectar Colisión".

Como se decía anteriormente en este diagrama se verá la secuencia de operaciones para verificar si 2 objetos colisionan. Para esto se le pasarán como parámetro los 2 boundaring de los que se deseen testear y luego pasamos a verificar si ambos boundaring se intersecan.

Conclusiones

Al concluir este capítulo se tiene concebido detalladamente el diseño completo del sistema y la secuenciación de pasos traducida a mensajes entre clases de los primeros casos de usos a desarrollar, con lo que se puede pasar a la etapa de implementación del proyecto.

5 Implementación del Sistema

Introducción

Esta etapa del proyecto constituye el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .cpp correspondiente a la Implementación en C++. La implementación de este sistema se regirá por los estándares de codificación utilizados en el desarrollo de la herramienta SceneToolkit (STK), los cuales se muestran en el siguiente epígrafe.

5.1 Estándares de Codificación

El código seguirá el mismo estándar de la Herramienta. Este sigue algunos estándares propuestos por el grupo de desarrollo (respetando los estándares de codificación para C++ (identado, uso de espacios y líneas en blanco, etc.)). Está programado en inglés, debido a que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático. El conocimiento de los estándares seguidos para el desarrollo de la misma permitirá un mayor entendimiento del código, y es una exigencia de los autores de la misma que cualquier módulo que se añada debe estar codificado siguiendo estos estándares.

Nombre de los ficheros:

Los nombres de los ficheros .h y .cpp utilizan las iniciales del nombre de la herramienta (STK).

*ej:***STK**NameOfUnits.cpp

Constantes:

Las constantes se nombran con mayúsculas, utilizándose "_" para separar las palabras.

ej: const int MY_CONST_ZERO = 0;

Enumerados:

Para los enumerados se utiliza el indicador "**E**" en el nombre del tipo, y en las constantes se utilizan las iniciales del nombre del enumerado. Las constantes van en mayúsculas.

ej1: enum **E**MyEnum

{

ME_VALUE,

ME_OTHER_VALUE

};

ej2: enum **E**Node**T**ype

{

NT_GEOMETRYNODE,

NT_GROUPNODE...

};

Estructuras:

Se utiliza el indicador "**S**" para indicar que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.

ej: struct **S**MyStruct {…};

Clases:

Se utiliza el indicador "**C**" para indicar que es una clase. Ver más adelante la nomenclatura de las variables miembros.

ej: class **C**ClassName;

Interfaces:

Se utiliza el indicador "**I**" para indicar que es una interfaz.

ej: **I**MyInterfaceName

Listas e iteradores de la *std*:

Para los tipos de datos utilizados de la librería estándar de C++ (*vector*, *map*, *multimap*, etc.), se utiliza el indicador "**T**", con los sufijos **List**, **Map** y **MultiMap** según la estructura, así como el sufijo **Iter** para los iteradores. Además el nombre lleva el tipo de dato a almacenar en la estructura en cuestión:

*ej:*vector<CNode> **T**Node**List**;

TNodeList::iterator TNodeList**Iter**;

*ej:*map<> **T**Name**Map**;

TNameMap::iterator **T**name**MapIter**;

*ej:*multimap<> **T**Name**MultiMap**;

TNameMultiMap::iterator **T**Name**MultiMapIter**;

Declaración de variables:

Los nombres de las variables comienzan con un identificador 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 antepone el identificador "**m**_" (en minúscula), si son globales se les antepone el identificador "**g**_", y en caso de ser argumentos de algún método, se les antepone el prefijo "**arg**_".

Tipos simples:

ej:

bool **b**VarName;

int **i**Name;

unsigned int **ui**Name;

float **f**Name;

char **c**Name;

char* **ac**Name; // arreglo de caracteres

char* **pc**Name; // puntero a un char

char** **aac**Name; // bidimensional

char** **apc**Name; // arreglo de punteros

bool **m_b**MemberVarName; // variable miembro de una estructura o clase

char **g_c**GlobalVarName; // variable global

short **s**Name;

void* **pv**Name;

Instancias de tipos creados:

ej:

EMyEnumerated **e**Name;

SMyStructure **k**Name;

CClassName **k**ObjectName; // objeto de una clase

CClassName* **pk**Name; // puntero a objeto

CClassName* **ak**Name; // arreglo de objetos

CClassName* **m_ak**Name; // variable miembro de clase

IMyInterface* **pI**Name; // puntero interfaces

Métodos:

En el caso de los métodos, se les antepone el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepone nada. Los constructores y destructores, como lo exigen los compiladores, llevan el nombre de la clase.

Constructor y destructor:

*ej:*CClassName (bool arg_bVarName, float& arg_fVarName);

~CClassName ();

Funciones:

ej: bool bFunction1 (…);

int* piFunction2 (…);

CClassName* pkFunction3 (…);

Procedimientos:

ej: void Procedure4 (…);

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases no se nombrarán “Gets” ni “Sets”, sino como los demás métodos, pero **con el nombre** de la variable a la que se accede y sin el prefijo “**m**_”:

ej: Para la siguiente variable, los métodos de acceso serían:

int**m**_**i**MyVar; //variable

int **i**MyVar(); //”Get”

void MyVar(int arg_**i**MyVar) //”Set”

int& **i**MyVar(); //”Get” y ”Set”

5.2 Diagrama de Componentes

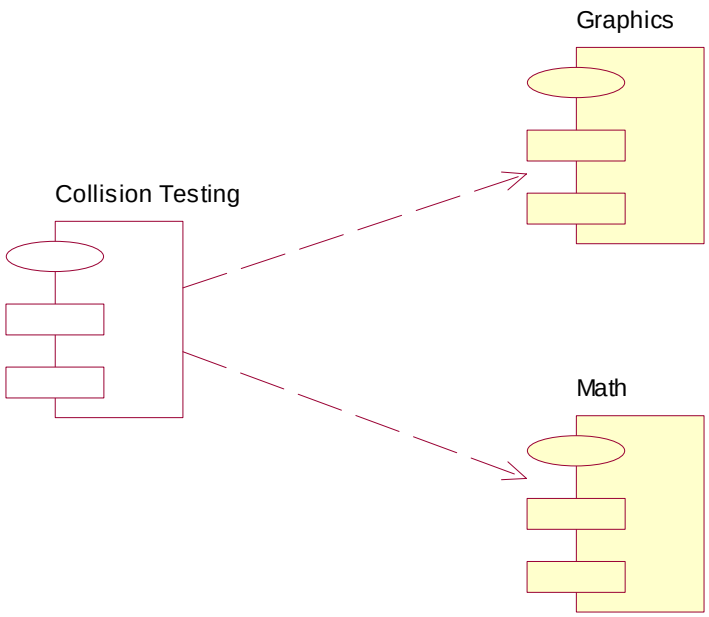

Fig. 16 Subpaquetes de componentes " Colisiones".

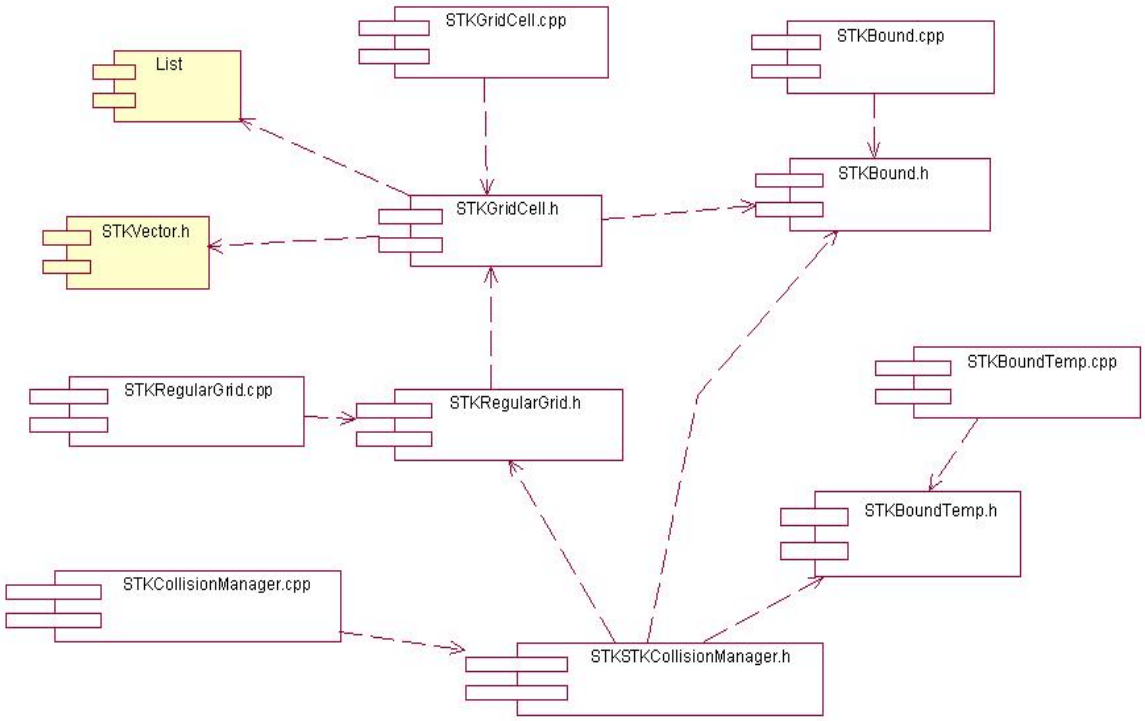

Fig. 17 Diagrama de componentes " Colision Testing".

Conclusiones

En este momento se encuentra todo preparado para pasar a la etapa de programación de los casos de uso desarrollados en el primer ciclo. Como posibilidad adicional que brinda la herramienta Rational Rose, ya es posible generar el código fuente de los componentes relacionados con los casos de uso a desarrollar en el primer ciclo.

Conclusiones

Para el cumplimiento de los objetivos de este proyecto, en concordancia con las exigencias hechas por la entidad cliente, se requirió primeramente hacer un estudio de las técnicas, tecnologías y tendencias actuales en relación con el tema de las colisiones. En el estudio se analizaron las características de algunas librerías y algoritmos para el manejo de estas, y las deficiencias entre ellas. Además, a partir de esa investigación se proponen las características técnicas de la solución.

Posteriormente se hizo la captura de los requisitos funcionales y no funcionales, y la agrupación de los primeros en los casos de uso del sistema. Se definió además una primera fase de desarrollo del proyecto, donde se realizarán solo una parte de los casos de uso para obtener la visualización de un modelo simple.

A continuación se transitó por las etapas de análisis, diseño e implementación utilizando los artefactos de RUP, donde surgieron y maduraron las clases, se realizaron los casos de uso a desarrollar en la primera fase y se creó el diagrama de componentes final que contendrá a las clases de la biblioteca.

Recomendaciones

Se recomiendan los siguientes aspectos al trabajo:

1- Utilizar otros tipos de volúmenes de inclusión además del OBB (oriented bounding boxes).

2- Optimizar los algoritmos de reubicación de objetos.

3- Implementar nuevos algoritmos de colisiones.

4- Estudiar estructuras de datos espaciales jerárquicas para particionar el entorno.

Referencias bibliográficas

Libros

[1]. Astle, Dave y HAWKING, Kevin.
OpenGL Game Programming.
Prima Tech Publishing. USA. 2001.

[1.1]. Capítulo 1: *The exploration begins: OpenGL and DirectX*.

[1.2]. Capítulo 2: *Using Windows with OpenGL.*

[2]. Colectivo de Autores
Modelos de Representación
Vigo 1996.

Libros Digitales

[3] Roa López, Francisco Javier.
Detección de Colisiones.
http://wwwdi.ujaen.es/~juanjo/DownloadIIG0506/Exposiciones/DeteccionColisiones.pdf. (2006)

[4] Hudson, Thomas C. y Lin, Ming C.
V-COLLIDE: Accelerated Collision Detection for VRML.
http://www.cs.unc.edu/~walk/papers/hudson/vcollide.pdf (2006)

[5] Chi, Diane M. y Kokkevis, Evangelos
Simulated Casualties and Medics for Emergency Training
http://www.cis.upenn.edu/~badler/medisim/mmvr.pdf (2006)

[6] Lin, Ming C
Accelerated Proximity Queries Between Convex Polyhedral By Multi-Level Voronoi Marching.
http://historical.ncstrl.org/litesite-data/uncch_cs/00-026.pdf (2006)

[7] Gómez Martín, Marco A. y González Calero, Pedro A**.**
JAVY: Agente pedagógico para enseñar la estructura de la JVM
http://gaia.fdi.ucm.es/people/marcao/publications/JAVYTechRep02.pdf (2006)

[8]Ville, Helin
Hierarchies for occlusion culling
http://www.tml.tkk.fi/Opinnot/T-111.500/2003/paperit/VilleHelin.pdf (2006)

[9] Cheng, Robert
Collision Detection
http://graphics.stanford.edu/courses/cs448b-00-winter/critiques/rcheng.pdf
(2006)

Sitio Web

[10]. VJuegos: Comunidad Iberoamericana de Desarrolladores de juegos.
http://www.vjuegos.org (2004)
[10.1]. *Análisis de Lenguajes de Programación*
http://www.vjuegos.org/modules.php?name=Content&pa=showpage&pid=4
(2006)

[11] Muñoz Moreno, Emma y Otros.

Detección de Colisiones, un Problema Clave en la Simulación.

http://72.14.209.104/search?q=cache:XHudkyx8C8oJ:www.conganat.org/Seis/is/is48/IS48_23.pdf+Algoritmo+de+Lin-Canny&hl=es&gl=cu&ct=clnk&cd=2. (2006)

[12] B. Mirtich.
Fast and Robust Polyhedral Collision Detection.
ACM Transactions on Graphics, Vol. 17, No. 3, pp. 177-208, July 1998.
http://portal.acm.org/citation.cfm?id=285857.285860&coll=GUIDE&dl=GUIDE&CFID=1103467&CFTOKEN=89358323#IndexTerms (2006)

[13] Gottschalk, Stefan y Manocha, Dinesh
A Hierarchical Structure for Rapid Interference Detection
http://www.siggraph.org/conferences/siggraph96/core/conference/papers/t1015.2.html (2006)

[14] Mitsubishi Electric Research Laboratories.
Collision Detection Algorithm.
http://www.merl.com/projects/vclip/V-Clip.html (2006)

[15] Basch, Julien y Otros.
Kinetic Collision Detection between Two Simple Polygons.
http://www.cs.brown.edu/cgc/cgc98/papers/submission14.ps. (2006)

[16] Wikipedia, the free enciclopedia
N-body pruning.
http://en.wikipedia.org/wiki/Collision_detection#n-body_pruning. (2006)

Apéndices

Glosario de abreviaturas

AABB: Axis Aligned Bounding Boxes (Cajas fronteras Alineadas a los Eje).

BSP: Binary Space Partition (Partición binaria del espacio).

CPU: Central Processing Unit (Unidad Central de Procesamiento).

CU: Use Case (Caso de Uso).

CU-GC: (Caso de Uso Gestionar Celda).

EV: Virtual Environment (Entornos Virtuales).

GJK: Gilbert, Johnson y Keerthi.

MERL: Mitsubishi Electric Research Lab (Laboratorio de investigación eléctrico de Mitsubishi).

OBB: Oriented Boundaring Boxes (Cajas de Fronteras Orientadas).

SP: Problemica Situation (Situación Problémica).

SRV: Sistemas de Realidad Virtual.

Glosarios de Términos:

A:

Aplicaciones interactivas: Herramientas computacionales en las que el usuario tiene un papel activo.

B:

Bounding Volume**:** Volumen frontera

Bounding Spheres**:** Esferas fronteras

Bounding Boxes**:** Cajas fronteras.

C:

Clusters: Conjunto contiguo de sectores que componen la unidad más pequeña de almacenamiento de un disco.

Celdas: Cuadrícula que forma parte de una Rejilla Regular.

Convexos: Se aplica a las superficies curvas redondeadas hacia el exterior.

Colisiones: Se atribuye cuando dos sistemas están intentando usar el mismo espacio al mismo tiempo.

E:

Entornos virtuales: Es un campo relacionado con la inteligencia artificial. Se trata de la simulación de mundos o entornos, denominados virtuales, en los que el hombre interacciona con la máquina en entornos artificiales semejantes a la vida real.

Escalares: Magnitud enteramente definida por su medida en función de una cierta unidad.

I:

Interacción: Acción que se ejerce recíprocamente entre dos o más grupos, personas u otros agentes.

O:

Octree: Estructura de datos, árbol de Partición del espacio.

Objetos poliédricos: Objeto limitado por superficies planas llamadas caras.

Objetos dinámicos: son los objetos que se mueven por el Grafo de trayectorias del entorno.

R:

Ray-tracing: Una técnica para simular efectos luminosos construidos en ambientes de gráficos en 3D.

Rejilla: Estructura de datos que divide el entrono en celdas

S:

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

Simulación: Se trata de la representación simplificada, mediante un modelo, de la realidad de un proceso.

V:

Voxels*:* Volumen de encierro en forma cúbica, en el caso del Octree y en la rejilla regular es el equivalente a una celda.