UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5

“Módulo de detección de polígonos en colisión entre objetos en un Sistema de Realidad Virtual”

Trabajo para optar por el Título de Ingeniería en Ciencias Informáticas

Autores: Juan Carlos Suárez Fernández

Rainer Sierra Peña

Tutores: Lic. Juan Manuel Medero Martínez

Ing. Daily Ibarreche Delgado

Ciudad de la Habana

Junio del 2008

“El verdadero precio de todas las cosas, es el esfuerzo y la molestia que supone adquirirlas."

Adam Smith

Declaración de Autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos al Proyecto “Herramientas de Desarrollo para Sistemas de Realidad Virtual”, de la Facultad 5 de la Universidad de las Ciencias Informáticas, a hacer uso del mismo en su beneficio.

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

Autores:

Juan Carlos Suárez Fernández

_____

Rainer Sierra Peña

_____

Tutores:

Lic. Juan Manuel Medero Martínez

_____

Ing. Daily Ibarreche Delgado

_____

Datos de Contacto

Nombre y Apellidos: Juan Manuel Medero Martínez

Edad: 26 años

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Licenciado en Ciencias de la Computación

Categoría Docente: Profesor Instructor

E-mail: juanm@uci.cu

Graduado de la UCLV, con un año de experiencia en el tema de la Gráfica Computacional y líder de un proyecto de Realidad Virtual en la Universidad de las Ciencias Informáticas.

Datos de Contacto

Nombre y Apellidos: Daily Ibarreche Delgado

Edad: 23 años

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informáticas

Categoría Docente: Profesor Adiestrado

E-mail: dibarreche@uci.cu

Graduada de la UCI, con un año de experiencia en el tema de la Gráfica Computacional y Analista de un proyecto de Juegos 3D en la Universidad de las Ciencias Informáticas.

Dedicatoria

A mi novia Daily, eslabón clave de amor, comprensión y apoyo.

A mi papá y a Grichel, por los momentos de consejos y de sustento.

A mis suegros Roberto y Daimy por ser ellos y por siempre estar ahí.

A mi mamá por darme vida.

A Dios por ser el gran ayudador, el más fiel de todos los amigos y Padre eterno.

A Randy y Jeniley, para que lleguen a resultados iguales o mejor.

A Toniel y Deiny, por sus abrazos y besos cuando llego y cuando me voy.

A mi abuelo Suárez, ejemplo de esfuerzo y sencillez.

A mi abuela, a mi tío Ramón.

A Javier, Deisita y familia por mostrarme la verdad.

A los profesores que desde pequeño me inculcaron superación.

A mi perseverancia de aprender en mi juventud.

A todos los amigos pasados, a todos los golpes del pasado, a todos aquellos que de alguna forma, sembraron en mí alguna semilla de amor, de fe, de fuerza y valentía para enfrentar la vida y sus retos.

Juan Carlos.

A mis padres por todo el amor, el cariño, comprensión que me han brindado a lo largo de mi vida.

A la Revolución que me dio la oportunidad de realizarme como profesional.

A Dios por estar siempre a mi lado.

Rainer

Agradecimientos

De Juan Carlos

A mi novia por estar siempre presente con amor y mucho consuelo.

A mi tutor, que siempre me dio confianza, tareas y herramientas de apoyo.

A mi mamá, a Walter y Randy por creer en mí.

A mi papá, a Grichel, a Jeniley, a mi abuela, a mi tío y aquellos que estuvieron al tanto de mí mientras pudieron.

A mis suegros por ayudarme desinteresadamente y siempre confiar en mí.

A Walter (Felicidades papá), por la ayuda con los cálculos matemáticos necesarios para mi resultado.

A Dayamí por armarse de paciencia en ocasiones.

A la UCI por dejarme estar dentro de sus aulas y ser uno más de sus estudiantes.

De Rainer

A cada uno de mis familiares que siempre me dieron confianza y fuerza para enfrentar cualquier reto.

A todos los profesores que he tenido a lo largo de mi vida que de una forma u otra han tenido que brindarme sus conocimientos desinteresadamente para poder llegar a donde estoy hoy.

A mi novia Magdalena por su paciencia a lo largo de estos cinco años y no dejar nunca que muera el amor.

A mis amigos del barrio que me enseñaron muchas cosas que han sido imprescindibles en la vida.

A Yamilka por todo el cariño, afecto y comprensión que me dio a lo largo de estos años en la UCI.

A Lester por mantenerse siempre a mi lado y nunca perder la confianza en mí.

A mis amigos en la UCI por estar a mi lado en todo momento en que los necesité.

A mis compañeros de grupo a lo largo de estos cinco años

Resumen

En los últimos años, debido al gran avance de la computación, los Sistemas de Realidad Virtual (SRV) se han visto enriquecidos con sensaciones del mundo real a través de estímulos visuales, auditivos y sensoriales que afectan al usuario de forma interactiva durante un Tiempo Real simulado.

Lograr una eficiente y rápida Detección de Colisiones en entornos virtuales, constituye un problema al que se le debe prestar una significativa atención en la computación gráfica.

Este trabajo está centrado en detectar las colisiones entre objetos y a partir de ahí obtener él o los polígonos de intersección entre los mismos. Para esto, se realizó una investigación que permite conocer el estado del arte a nivel mundial y las principales técnicas usadas para este fin y concluir con la creación de un módulo de software multiplataforma, usando para ello el Proceso Unificado de Software e implementado en C++ y acoplado a la Herramienta de Desarrollo de Realidad Virtual (SceneToolkit).

Se eligió la técnica de subdivisión de objetos mediante un volumen envolvente jerárquico, seleccionando para esto el volumen conocido como: Caja alineada a los ejes de coordenadas o Axis-aligned bounding boxes (AABB). Esta técnica proporciona una noción específica de la zona de colisión de los objetos, logrando un chequeo de colisión entre zona y no entre objetos, por lo que se alcanza una alta eficiencia en la detección de colisión entre objetos.

Contenido

Introducción

En el mundo actual de grandes transformaciones de toda índole, la expansión de las nuevas tecnologías de la información ha hecho necesario que se desarrolle a grandes escalas la rama de la informática. La Informática Gráfica ha venido destacándose durante las dos últimas décadas como la rama más sensacional de la Informática. Un concepto a destacar en la Informática Gráfica es la “Realidad Virtual”, donde se logran efectos de interactividad y realismo, alcanzando cambios en la forma de percibir el tiempo, el espacio e incluso la propia identidad, ocasionando sensaciones virtuales de la realidad.

El término Realidad Virtual se relaciona al tratamiento de ambientes tridimensionales e interactivos orientados a la visualización de objetos 3D generados por computadoras, mostrando situaciones del mundo real en mundos virtuales.

En la actualidad, la Realidad Virtual (RV) es utilizada en cuantiosos sistemas del mundo, como por ejemplo en la empresa norteamericana VPL Research (Visual Programming Language), quien ha sido la desarrolladora, vinculada a la NASA, de una arquitectura primordial para lograr la implementación de diversos laboratorios virtuales. (16)

Otras esferas han centrado sus esfuerzos en la creación y utilización constante de laboratorios virtuales con el fin de llevar a cabo diversas funciones visuales útiles para la comprobación de datos y estimación de errores. Algunas de ellas son la Educación, la Exploración Espacial, la Arquitectura, el Turismo, la Medicina, las Telecomunicaciones, entre otras. (14)

Hoy en día la Informática Gráfica ha tenido que desarrollar innumerables algoritmos para mantener activamente el concepto de RV en cada software, sistema o videojuego realizado, entre los cuales se encuentra el proceso de dibujado de escenas virtuales, el proceso de simulación de terrenos virtuales y el proceso de detección de las colisiones o choques dinámicos efectuados arbitraria e irregularmente.

Una Colisión no es más que la ocupación de una parte del espacio por dos o más objetos en un mismo intervalo de tiempo, mientras que la Detección de la misma se basa en la referencia al cálculo de las intersecciones entre dichos objetos, más bien,

si se agrupan mutuamente ambas definiciones para ser utilizado en ambientes tridimensionales, es entonces cuando nace lo conocido por “Detección de Colisiones”. (1)

La Detección de Colisión es de gran necesidad para entornos virtuales, pues a través de toda la información obtenida después de existir una colisión, el sistema logrará ejecutar acciones de dibujado, de sonido y deformaciones, brindando al usuario mayor realismo de las escenas virtuales y mejor similitud de las acciones de la vida real y cotidiana.

Cuba ha tenido la necesidad de alcanzar cierto desarrollo científico en el uso de las nuevas tecnologías relacionadas con la información y las comunicaciones en general, así como en la importante rama, Informática Gráfica. Como muestra de lo antes expuesto, Cuba cuenta con diferentes espacios virtuales orientados al trabajo colaborativo, de enseñanza y desarrollo, entre las comunidades científicas y académicas basadas en la organización existente y proyección económico-social para el futuro próximo.

La Universidad de las Ciencias Informáticas (UCI), ha ocupado un importante lugar en los avances alcanzados en la Informática Gráfica, teniendo entre sus líneas de desarrollo e investigación la Realidad Virtual. Existen varios proyectos enfocados al tema de los gráficos generados por computadoras como son: “Simulador Quirúrgico” y “Herramientas de Desarrollo para Sistemas de Realidad Virtual” (**HDSRV**), entre otros. La HDSRV se encarga de crear e implementar herramientas genéricas con el fin de ser utilizadas en proyectos virtuales.

El proyecto antes mencionado cuenta con una biblioteca básica denominada SceneToolkit, donde se han desarrollado escasas funcionalidades para lograr una eficiente Detección de Colisiones, que no se corresponden con el realismo que necesitan los novedosos simuladores y los juegos 3D que utilizan dicha herramienta, por ejemplo el simulador KHEIPROS del proyecto Simulador Quirúrgico.

Puesto que, los entornos virtuales requieren de una avanzada técnica y modelación para Detectar las Colisiones existentes entre diversos objetos (simples y complejos) a tiempo real, constituye una importante necesidad la creación de un Módulo que Detecte Colisiones y que devuelva los polígonos en colisión para lograr una eficaz respuesta de los sistemas de RV.

Cómo Problema Científico de este trabajo se plantea la siguiente interrogante: ¿Cómo implementar un Módulo, que posibilite una eficiente Detección de Polígonos en Colisión entre Objetos, para la biblioteca SceneToolkit del proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual de la Universidad de las Ciencias Informáticas? Teniendo como Objeto de Estudio el proceso de Detección de Colisiones para Sistemas de Realidad Virtual, mientras que el Campo de Acción es delimitado por el proceso de obtención de polígonos en colisión entre objetos en un entorno virtual.

El Objetivo General de esta investigación es crear un Módulo, que permita Detectar los Polígonos en Colisión entre Objetos de un Entorno Virtual, para la biblioteca SceneToolkit del proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual.

Este trabajo defiende la siguiente idea: el Módulo de Detección de Polígonos en Colisión entre Objetos, integrado a la SceneToolkit, brindará eficientes respuestas e importante información sobre las colisiones o choques efectuados en escenas virtuales.

Para dar cumplimiento al objetivo propuesto es necesario definir un grupo de Tareas Investigativas referenciadas a continuación:

- Investigar y analizar las bibliotecas de Detección de Colisiones existentes en el mundo.
- Buscar y analizar los algoritmos y métodos eficientes para la Detección de Colisiones entre Objetos en un Entorno Virtual.
- Seleccionar los algoritmos y métodos eficientes para la Detección de Colisiones entre Objetos.
- Investigar, analizar y seleccionar técnicas y métodos para obtener los polígonos en Colisión entre Objetos en una escena virtual.
- Diseñar el Módulo de Detección de Polígonos en Colisiones para Objetos.
- Estudiar las características de la biblioteca existente: SceneToolkit.
- Implementar Módulo de Detección de Polígonos en Colisiones para Objetos.
- Acoplar el Módulo de Detección Colisiones a la biblioteca SceneToolkit.

Con el cumplimiento satisfactorio de las tareas investigativas se pretende, obtener un Módulo que Detecte los Polígonos en Colisión entre Objetos de una escena virtual con el cual se podrán obtener resultados visibles muy reales. Dicho Módulo

será acoplado a la biblioteca SceneToolkit, permitiendo una constante actualización, un fácil uso por parte de sus usuarios y un nivel de flexibilidad avanzado.

1

Capítulo 1 Fundamentación Teórica

Introducción

Los Sistemas de Realidad Virtual (**SRV**) son sistemas interactivos que permiten sintetizar un mundo tridimensional ficticio, creando una ilusión de realidad. La Realidad Virtual es una técnica inmersiva a 360 grados, la cual permite movimientos hacia arriba o hacia abajo, realizar acercamientos o alejamientos, provocándole un control absoluto al usuario. En concepto, los SRV consisten en simular todas las posibles percepciones de una persona, como los gráficos para la vista, sonido, tacto e incluso sensaciones de aceleración o movimiento. Todas estas sensaciones diferentes deben ser presentadas al usuario de forma que se sienta inmerso en el universo generado por el ordenador, hasta el punto de dejar de percibir la realidad.

Los SRV presentan características propias que no se pueden pasar por alto, como es el caso del comportamiento de cada objeto enfocado a la detección de choques entre ellos, es decir, la detección de colisiones. Un SRV que contenga una eficiente detección de colisiones, brinda mayor sensación y ofrece un realismo bastante similar al mundo exterior.

En este Capítulo se lleva a cabo un estudio de las principales características de los Sistemas de Realidad Virtual, los conceptos fundamentales y las principales técnicas, así como los algoritmos más usados en la actualidad para la detección de colisiones en escenas virtuales. También se estudian las características de la biblioteca SceneToolkit, para lograr un mejor entendimiento entre lo que existe y lo que se requiere implementar.

1.1 Sistemas de Realidad Virtual

1.1.1 Sistemas de Realidad Virtual (SRV).

Desde la pasada década, crear entornos gráficos generados por computadoras u ordenadores, se ha hecho una necesidad para el mundo de la informática programada. Los Sistemas de Realidad Virtual se han visto enriquecidos con sensaciones del mundo real a través de estímulos visuales, auditivos y sensoriales que afectan al usuario de forma interactiva durante un tiempo real simulado.

En esencia lo que se conoce como "Realidad Virtual" es todo aquello que se relaciona con imágenes diseñadas para tres dimensiones, generadas por ordenadores propiciando una interacción bastante real de cada usuario con dichos ambientes gráficos. Para ocasionar todo este campo de acción-reacción es necesario un complejo sistema electrónico para enviar y recibir señales con información sobre la actuación del usuario en el también llamado Mundo Virtual.

Examinando con mayor detalle sus características, se definen dos tipos de Realidad Virtual, la Inmersiva y la No inmersiva.

Los métodos Inmersivos de Realidad Virtual con frecuencia se ligan a un ambiente tridimensional creado por computadora, el cual se manipula a través de cascos, guantes u otros dispositivos que capturan la posición y rotación de diferentes partes del cuerpo humano (ver fig. 1.1).

Fig. 1.1 Dispositivos de la Realidad Virtual inmersiva

La Realidad Virtual No inmersiva utiliza medios como el que actualmente ofrece Internet, en el cual se puede interactuar en tiempo real con diferentes personas en

espacios y ambientes que en realidad no existen, sin la necesidad de dispositivos adicionales a la computadora (ver fig. 1.2).

Fig. 1.2 Realidad Virtual no inmersiva (Juego Online)

El concepto de Realidad Virtual (**RV**) cada vez se hace más difícil y controversial, puesto que a cada momento se le suman más atributos a los conceptos ya existentes y citados por grandes personalidades de la Informática Gráfica. A continuación se citan varias interpretaciones de diversos autores.

«La RV es la simulación de medios ambientes y de los mecanismos sensoriales del hombre por computadora, de tal manera que se busca proporcionar al usuario la sensación de inmersión y la capacidad de interacción con medios ambientes artificiales». (Dr. Homero Ríos Figueroa).

"Realidad virtual: un sistema de computación usado para crear un mundo artificial donde el usuario tiene la impresión de estar en ese mundo y la habilidad de navegar y manipular objetos en él". (Manetta C. y R. Blade (1995)).

"La realidad virtual te permite explorar un mundo generado por computadoras a través de tu presencia en él". (Hodder y Stoughton(s/a)).

"*Un Sistema de Realidad Virtual es aquel que da al usuario la experiencia de estar inmerso en un entorno sintético". (H.Fuchs).*

"La realidad virtual se caracteriza por ofrecer la ilusión de participación en un entorno sintético en vez de una observación externa del mismo. La realidad virtual es una experiencia inmersiva y multisensorial". (M. A. Gigante).

"Un entorno virtual se caracteriza por ser interactivo, con un procesamiento especial de imagen, sonido y tacto, para convencer al usuario que se encuentra inmerso en un espacio sintético". (S.R.Ellis). (15)

1.1.2 Aplicaciones en Tiempo Real

Existe un concepto muy esencial que está presente en todos los SRV como una característica principal que no debe ser violada y mucho menos ignorada, se le conoce como Tiempo Real (**TR**).

El concepto de Tiempo Real viene dado por el procesamiento digital de señales. Un sistema de TR es aquel que es capaz de procesar una muestra de señal antes de ingresar al sistema la siguiente muestra.

Los primeros sistemas de Tiempo Real se desarrollaron para la telefonía digital, utilizando hardware de propósito específico. Con el paso del tiempo y la mejora constante en la velocidad de los microprocesadores, la implementación de sistemas de TR se ha convertido en un campo de la informática donde se han presentado condiciones y problemas enfocados al disco duro de una computadora. (28)

Este concepto se puede aplicar a distintas ramas de la Informática y proviene de la Ingeniería de Control. En general la caracterización como TR se refiere a la existencia de determinadas restricciones sobre el comportamiento temporal de nuestro sistema. Dependiendo del tipo de aplicación, esas limitaciones serán de una clase u otra. El concepto de Tiempo Real, por tanto, puede aplicarse de forma más o menos estricta según lo duras que sean las restricciones impuestas. (29)

Un ejemplo de ejecución en Tiempo Real es la Realidad Virtual, esto expresa que lo que sucede, se calcula exactamente en ese momento, permitiendo una libertad y realismo, que no puede dar la animación. Significa que la computadora hace todos los cálculos necesarios y presenta una imagen nueva cada 30 frames por segundo (**fps**) como mínimo, haciendo que los requerimientos computacionales sean grandes. (30)

Estos mundos virtuales están compuestos por varios conceptos gráficos, de sonidos y sensoriales, además de constar con un gran modelo matemático que describe un espacio tridimensional, donde están contenidos objetos que pueden representar cualquier cuerpo, desde una simple entidad geométrica, como son los cubos o las esfera (ver fig. 1.3), hasta un cuerpo complejo, como son los modelos de estructura físico-genéticas (ver fig. 1.4).

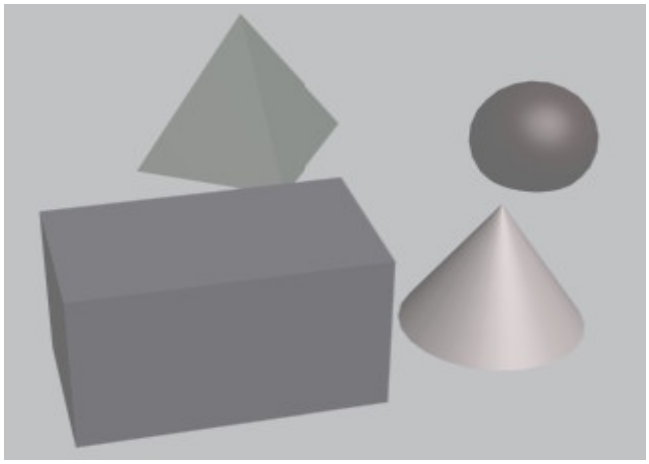

Fig. 1.3 Ejemplo de escena con cuerpos simples

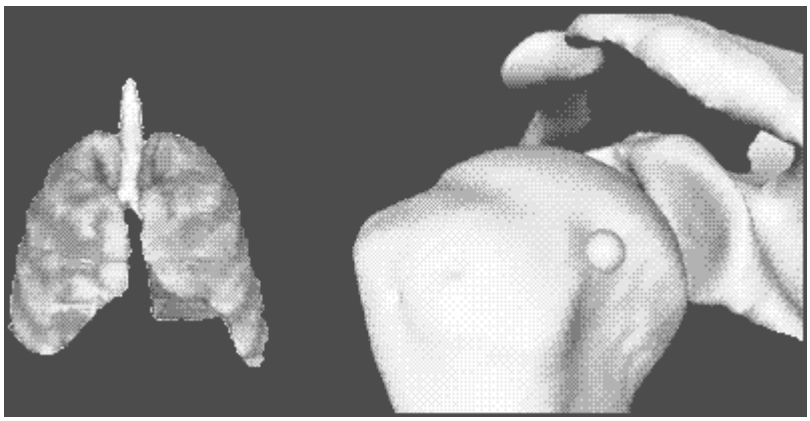

Fig. 1.4 Ejemplo de escena con objetos complejos

Para mejor entendimiento: Objetos Primitivos o Simples hace referencia a objetos geométrico-matemáticos regulares que no varían su forma en el tiempo como por ejemplo una esfera, una pirámide, un cubo, entre otros y Objetos Complejos hace alusión a todas aquellas figuras que no se pueden representar mediante objetos primitivos, es decir, son todos los objetos que tienen forma irregular y solamente pueden ser representados en el mundo de la Informática Grafica a través de sistemas triangulares en forma de mallas.

Con el desarrollo incremental de la Informática Gráfica y de la programación de escenas virtuales se considera una falta de dinamismo construir mundos estáticos, es decir, mundos donde los objetos no tengan vida propia ni comportamientos que propicien una interacción entre ellos mismos o con el usuario. Por tanto, es todo un hecho diseñar y programar escenas donde cada objeto se comporte según sus propiedades físico-matemáticas de manera que pueda relacionarse con el usuario mediante la definición de una serie de sensores de posición, de contacto ó de colisión, entre otras.

1.2 Detección de Colisiones

1.2.1 Conceptos fundamentales

Las escenas dinámicas pueden estar conformadas por objetos simples ó primitivos y por objetos complejos, guardando relaciones unos con otros. Los objetos interactúan entre ellos según sus propiedades ó a causa de una interacción con una fuerza externa a él. Cada objeto tiene la libertad de moverse y vagar por las escenas virtuales según le corresponda y esto trae consigo situaciones de choques contra otros objetos o contra él mismo. Estas situaciones que se propician deben ser captadas y es obligatorio darle la respuesta necesaria a cada caso de colisión según las características de la problemática.

El término de Detección de Colisión se divide en tres partes: Consultas de Colisión que indica si dos o más objetos colisionan mediante un valor lógico, Determinación de Colisión el cual hace referencia al cálculo de las intersecciones entre objetos y la Respuesta de Colisión que señala las acciones a realizar en respuesta a la colisión entre objetos (ver fig. 1.5). (1)

Fig. 1.5 Detección, Determinación, Respuesta de Colisión.

Todos los objetos de una escena pueden tener una colisión en un momento dado, por tanto los objetos primitivos también llevan consigo esta característica. La Detección de Colisión entre objetos simples se lleva a cabo mediante la implementación de varios conceptos y métodos matemáticos relacionados con la figura geométrica que presente dicho estado de colisión. Todo este proceso de detectar colisiones en figuras geométricas se soluciona haciendo uso de los conceptos de distancia punto a punto, distancia segmento a punto, distancia segmento a segmento, distancia plano segmento, y distancia punto a plano, entre

otras. En fin, la Detección de Colisión se ajusta al tipo de figura y realiza los cálculos y conceptos necesarios predefinidos por cada figura geométrica. (31)

1.2.2 Fases de la Detección de Colisiones

En ambientes interactivos la mayoría de las veces comprobar la Detección de Colisión entre objetos se vuelve compleja y altamente costosa. Por una parte, nos encontramos con el problema del número de objetos presentes en la aplicación. En el peor de los casos, un algoritmo de fuerza bruta debe comprobar cada uno de los objetos con todos los demás, necesitando un tiempo O (N^2). En el algoritmo, este tiempo es O($N \cdot M+N^2$), es decir, se deben comprobar todos los pares de objetos en movimiento y todos los pares formados por un objeto estático y otro en movimiento. Caso similar ocurre cuando se tiene que chequear el punto exacto de colisión entre dos objetos que se debe comprobar en qué punto las mallas que forman ambos objetos colisionan. Para reducir este tiempo cuadrático debemos disminuir el número de pares de objetos sobre los que realizar la Detección de Colisión y después que se tengan estos pares de objetos obtener el área en la cual estos colisionan. Para ello, esta tarea se divide en tres fases: Fase Amplia, Estrecha y Exacta (ver fig. 1.6).

En la Fase Amplia se descartan rápidamente pares de objetos sobre los que no se realizará la comprobación de colisión. De esta forma sólo ciertos pares de objetos pasan a la Fase Estrecha, que trata de determinar si se produce o no colisión entre dichos pares de objetos y la zona de colisión de dichos objetos en caso que ocurra, para de ahí pasar a la Fase Exacta la cual se encarga de verificar si existe colisión entre los polígonos que se encuentran en la zona de colisión previamente detectada en la fase anterior. (18)

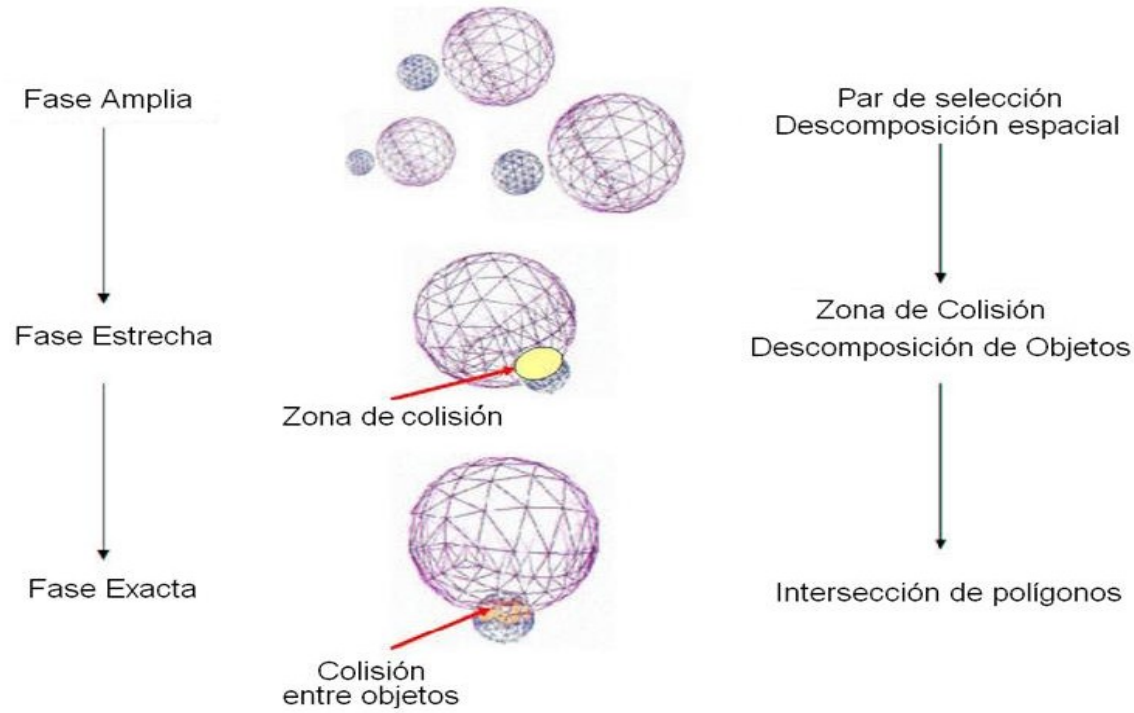

Fig. 1.6 Fases de Colisiones.

Fase Amplia también llamada fase de poda o colisión dinámica, consistente en descartar el máximo número de parejas que se saben demasiado lejanos como para colisionar y seleccionar los pares de objetos que probablemente colisionaran. Aquellos pares no descartados en esta fase son a los que se le realizarán los test de colisión estática en la denominada fase estrecha. Los algoritmos usados en esta fase se basan principalmente en la descomposición del espacio, entre los que se encuentran: el KD-Tree, BSP, Octrees o Quadtrees, División Uniforme o Rejilla Regular, que subdividen el entorno en una jerarquía espacial (ver fig. 1.7). Los objetos de la escena son agrupados jerárquicamente según las regiones en que ellos se encuentran. (1)

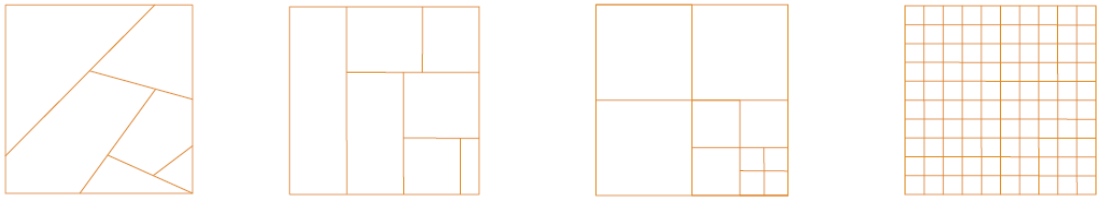

Fig. 1.7 Algoritmos de Detección de Colisión utilizados en la Fase Amplia

Fase Estrecha también llamada fase detallada, colisión detallada, colisión de bajo nivel o colisión estática es donde se seleccionan las regiones de colisión de dos objetos. Esta fase no computa exactamente las entidades en colisión pero da una noción de la región dentro de cada objeto donde una colisión podría existir. Esta región se llama la zona de colisión y los algoritmos en esta fase son principalmente basados en la descomposición del objeto. Cada objeto se subdivide en una jerarquía envolvente.

Existen métodos y estructuras frecuentes a emplear durante este tipo de jerarquía como son los Axis-Aligned Bounding Boxes (AABB) y el uso de envolventes esféricos. Ambos métodos son los menos costosos y suelen ser bastante eficientes. Algunos de los algoritmos más costosos son el oriented boundig box (OBB) y el DOP que reducen el margen de errores al mínimo. (18)

La Fase Exacta obtiene las entidades exactas de colisión. Los algoritmos verifican la intersección entre dos polígonos o primitivas que son los que conforman el modelo geométrico que forman al objeto. Los pares de primitivas más utilizados por esta fase son: esfera/esfera, caja/caja y triángulo/triángulo. (18)

1.2.3 Técnicas usadas en la Detección de Colisiones.

1.2.3.1 Descomposición espacial

La descomposición espacial es la técnica más utilizada actualmente para descartar objetos que a priori no van a colisionar en la fase amplia. Permite realizar un chequeo de colisión entre objetos (o partes) sólo si ocupan la misma zona del espacio, ello se realiza mediante una subdivisión jerárquica del espacio que permite la adaptación a las densidades locales del modelo. Generalmente en regiones en las que hay un gran número de objetos, se divide más que en las que hay menos. En entornos interactivos se subdividen celdas cuando entran nuevos objetos y se agrupan cuando salen.

A continuación se muestran algunos algoritmos utilizados en la descomposición espacial:

Rejilla Regular o **División Uniforme** (19) es una partición del espacio en celdas rectangulares y uniformes, es decir, todas las celdas tienen el mismo tamaño. Cada objeto se asocia con las celdas en las que se encuentra (ver fig. 8). Para que dos objetos colisionen deben ocupar una misma celda. Sólo debe comprobarse la colisión entre objetos que ocupen la misma celda.

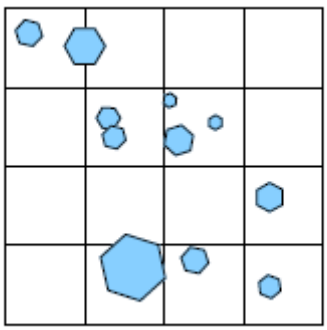

Fig. 1.8 Rejilla Regular o División Uniforme.

Un **Octree** (17) (18) es una partición jerárquica del espacio mediante vóxeles alineados con los ejes. Cada nodo padre se divide en ocho hijos. El nodo raíz suele representar el volumen del espacio completo a representar (la escena). Este volumen se divide en ocho sub-volúmenes tomando la mitad del volumen en los ejes

x, y, z. Estos ocho volúmenes se dividen recursivamente de la misma forma (ver fig. 1.9). El criterio de parada puede consistir en alcanzar una profundidad máxima en el árbol generado o un volumen mínimo para un nodo.

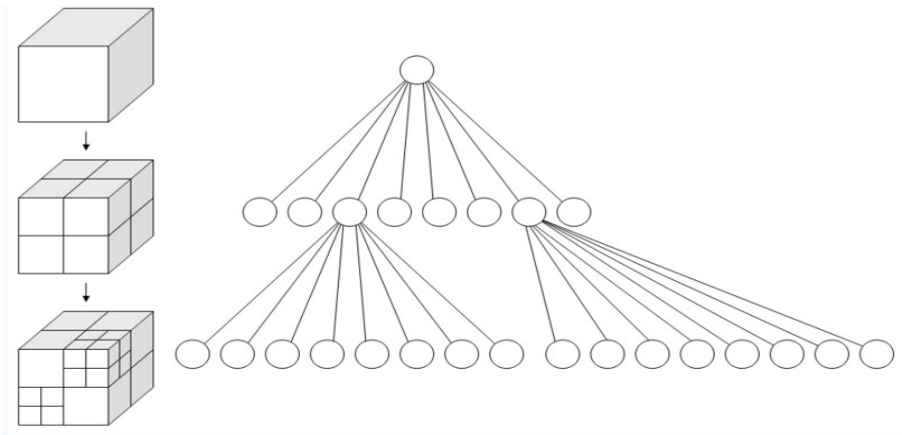

Fig. 1.9 Representación de un Octree

En un **K-d tree** (20) cada región es dividida en dos partes por un plano alineado con los ejes. Un K-d tree puede realizar divisiones uniformes del espacio o variables (existen también Octrees con divisiones variables). Las divisiones variables, aunque necesitan mayor almacenamiento para la estructura, se adaptan mejor a los objetos. A continuación se muestra en la figura la construcción de un K-d tree con divisiones variables.

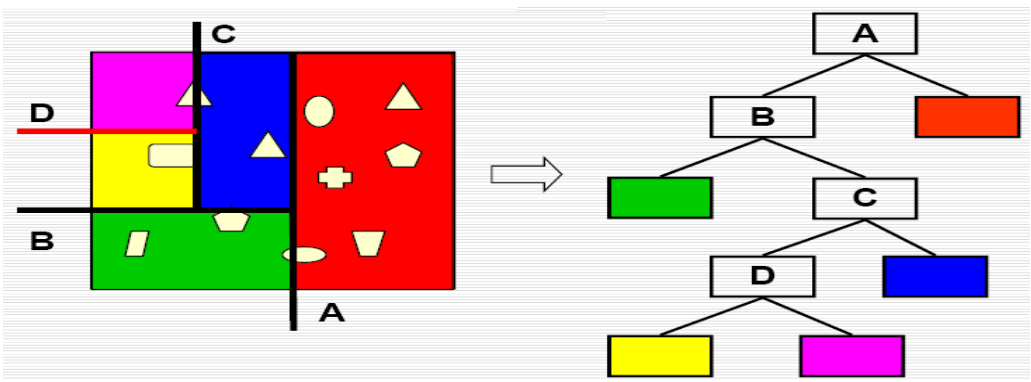

Fig. 1.10 Representación del algoritmo k-d tree.

Un árbol de **Partición Binaria** del **Espacio (BSP)** (21) es una estructura jerárquica que subdivide el espacio en celdas convexas. Cada nodo interno del árbol divide la región convexa asociada con el nodo en dos regiones, utilizando para ello un plano con orientación y posición arbitraria. Es análogo a un k-d tree variable sin la restricción de que los planos estén orientados ortogonalmente con los ejes. Las

hojas de dicho árbol representan a un objeto, mientras que los nodos no hojas representan planos ejes separadores como se muestra en la siguiente figura.

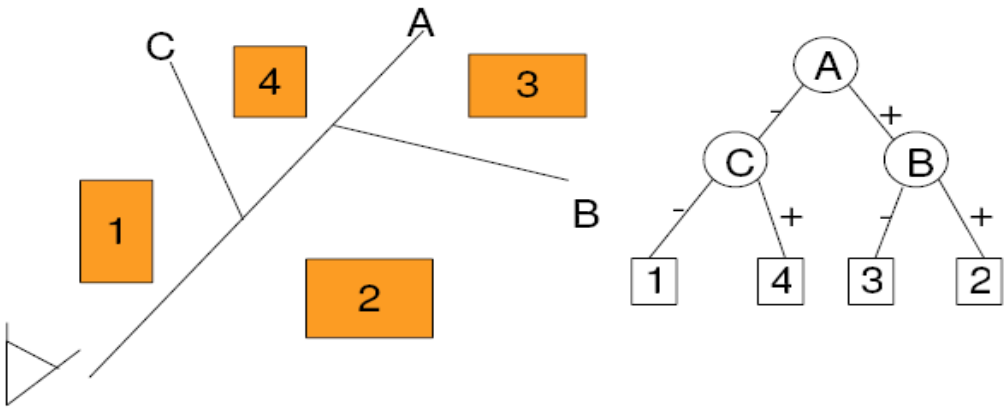

Fig. 1.11 Representación del algoritmo BSP-tree.

1.2.3.2 Descomposición de Objetos

Volúmenes de Inclusión ó volumen envolvente

Las pruebas de colisión entre objetos complejos generalmente se vuelven muy costosas y difíciles de comprobar. Suele utilizarse volúmenes de inclusión de forma que, si colisionan dichos volúmenes se realiza una prueba de colisión detallada entre objetos y así se reduce el costo de colisión.
La técnica de volúmenes de inclusión es muy rápida y de fácil comprobación cuando dos objetos no entran en colisión.

Requerimientos que deben cumplir los volúmenes de inclusión (1):

1. El volumen envolvente debe ajustarse al objeto tanto como sea posible.
2. El cálculo de colisión entre volúmenes envolventes debe ser poco costoso en comparación con las pruebas de colisión entre objetos.
3. El cálculo del volumen envolvente debe ser poco costoso sobre todo si este debe recalcularse con cierta regularidad.
4. El volumen envolvente debe poder rotarse, transformarse fácilmente.
5. El volumen envolvente debe representarse utilizando poca cantidad de memoria.

Los volúmenes de inclusión más representativos son los de esferas, cajas alineadas a los ejes de coordenadas (AABB), las cajas orientadas al objeto (OBB), y los *k*-DOPs, entre otros (ver fig. 1.12).

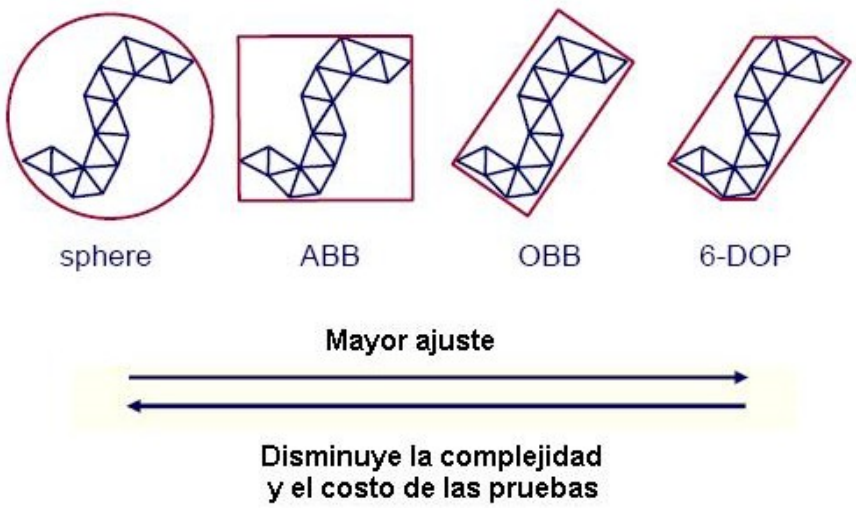

Fig. 1.12 Ejemplo de volúmenes de inclusión.

Esferas de inclusión

Conocida también como BS (Bounding Sphere) (3), consiste en comprobar colisiones entre esferas que contienen objetos. Es el volumen de inclusión más simple, pues requiere de poca memoria para su almacenamiento, está definido por un punto y un radio, (ver fig. 1.13). Una Esfera de Inclusión se beneficia de un chequeo de colisión poco costoso ya que es rápido de calcular, simplemente debe compararse la distancia entre los centros con la suma de los radios de ambas esferas. Si la distancia es mayor que la suma de los radios no existe contacto entre ellas; en caso contrario, sí existe.

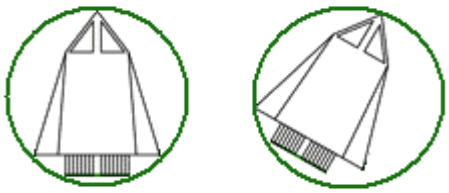

Fig. 1.13 Volumen envolvente esférico.

Cajas Alineadas a los ejes de coordenadas

Comúnmente conocida como AABB (**Axis Aligned Bounding Boxes**) (26) es una caja rectangular cuyas caras están orientadas de manera que sus normales son paralelas a los ejes coordenados (ver fig. 1.14). La detección de colisiones se hace de forma muy sencilla: existe intersección si y sólo si las proyecciones de las cajas sobre los ejes de coordenadas se solapan. Este tipo de volumen tiene buenas prestaciones en objetos estáticos, o que únicamente se trasladan o escalan, sin alterar su rotación, o que son rotados en raras ocasiones. (11)

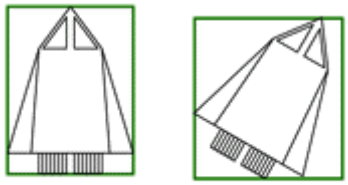

Fig. 1.14 Volumen envolvente AABB.

Cajas Orientadas

Conocida como OBB **(Oriented Bounding Boxes)** (2) este tipo de volúmenes de inclusión es representado por una caja rectangular con una orientación arbitraria aunque suelen utilizarse determinadas direcciones prefijadas (ver fig. 1.15). Al contrario del AABB, el OBB tiene un chequeo de colisión bastante costoso. Debido a que el tiempo de cálculo es relativamente grande, este tipo de volumen 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. (6)

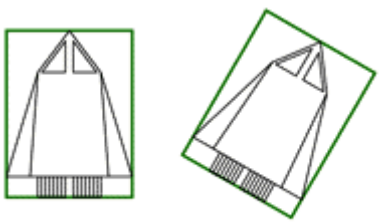

Fig. 1.15 Volumen envolvente OBB.

Politipo de orientación discreta

Es una generalización del AABB. Un **DOP,** conocido como **Discrete Orientation Polytipe** (4) es un politipo convexo que está construido tomando una serie de planos debidamente orientado hacia el infinito y avanzar en ellos hasta que chocan con el objeto. El DOP es entonces el politipo convexo resultante de la intersección de la mitad de los espacios delimitados por los planos. Un DOP construido por k planos es llamado *k*-DOP. Este volumen *k*-DOP es un politipo convexo definido por slabs cuyas normales pertenecen a un conjunto de direcciones predefinidas y comunes para todos los k-DOPs (ver fig. 1.16). Un **slabs** es la región infinita del espacio delimitada por dos planos paralelos. Un AABB es *6*-DOP pues está formado por 3 slabs. La actualización de un *k*-DOP es más costosa que la de un AABB debido a su mayor número de slabs, pero proporcional a *k planos*. El almacenamiento de un *k*-DOP depende también del número de slabs pero es poco costoso debido sobre todo a que estos slabs están restringidos a ciertas configuraciones. Este volumen tiene la ventaja adicional de que siempre se puede modificar el tipo de ajuste a un objeto cambiando el número de slabs a utilizar. (7) (9)

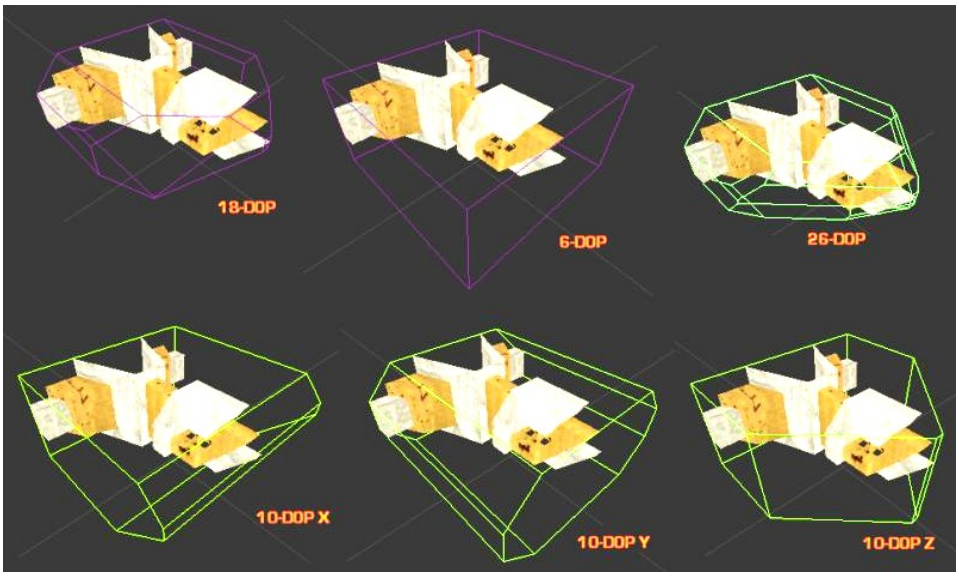

Fig. 1.16 Volumen envolvente k-DOPs.

Los volúmenes de inclusión o envolventes **(Boundary Volumes)** es una de las técnicas más utilizadas para conocer si existe colisión entre objetos en aplicaciones interactivas ya que con ellos se reduce el número de pruebas de colisión entre pares de objetos.

Volúmenes Envolventes Jerárquicos

Utilizar volúmenes de inclusión puede disminuir el tiempo de cálculo de colisión entre objetos. Sin embargo el tiempo de ejecución es el mismo aunque reducido por una constante, debido a que el número de parejas de objetos sobre el que se realiza este chequeo no varía. Se puede reducir la complejidad a logarítmica, llevando los volúmenes de inclusión a una jerarquía, constituyendo para ello, una jerarquía de volúmenes de inclusión.

El más general y versátil método para la detección de colisiones entre los modelos poligonales es el de jerarquías de volúmenes de inclusión. Conocido también como **Bounding Volume Hierarchies (BVH),** proporciona una manera rápida de realizar la detección de colisión entre modelos complejos. Este método consiste en construir una jerarquía de volumen de inclusión en forma de árbol para n objetos. En primer lugar se obtienen los volúmenes de inclusión de los objetos individuales y se colocan como nodos hoja del árbol que representa dicha jerarquía. Estos nodos son agrupados utilizando un nuevo volumen de inclusión de manera recursiva hasta obtener un volumen de inclusión que agrupe todos los objetos (representado por la raíz del árbol). Con esa jerarquía se comprueba la colisión entre los hijos de un nodo solo si se produce colisión entre los nodos padres, que son los primeros clasificados por el tipo de envolvente de volumen que se utiliza en cada nodo de una jerarquía de árbol.

Todo esto lleva implícito que para la actualización de las cajas de cada nivel de jerarquía cuando se produce deformación o rotación del objeto no es trivial y puede tener un alto costo computacional. (8)

En una jerarquía de volumen de inclusión debe cumplirse que (1):

1. La geometría contenida en los nodos de cualquier sub-árbol debería estar próxima entre sí.
2. Cada nodo de la jerarquía debe tener el mínimo de volumen posible.
3. Se debe considerar en primer lugar los nodos cercanos a la raíz de la jerarquía, pues eliminar uno de estos nodos es siempre mejor que hacerlo a mayor profundidad del árbol.
4. El solapamiento entre nodos del mismo nivel debe ser el mínimo.

5. La jerarquía debe ser equilibrada en cuanto a su estructura y su contenido.
6. En las aplicaciones de tiempo real, la determinación de colisión utilizando la jerarquía, no debe ser mucho peor que el caso medio, además la jerarquía debe generarse de forma automática sin la intervención de un usuario.

Las primitivas simples (Esferas, AABBs, etc.) cumplen muy bien el segundo requerimiento. Sin embargo en ocasiones no pueden ajustarse a algunos objetos. Algunas primitivas complejas (Elipsoide, OBBs, etc.) proporcionan un gran ajuste al objeto, pero el costo de la detección de colisiones de ellos es relativamente alto. Según el volumen de inclusión a utilizar cada programador debe tener en cuenta el costo computacional.

Estructuras Jerárquicas Esferas

La estructura que subyace es un árbol en donde el nodo raíz es una esfera que envuelve completamente a todo el objeto. Cada nodo del árbol es una representación. A medida que se desciende por el árbol las representaciones son más precisas. La idea es construir jerarquías de esferas en los que cada nivel de la jerarquía representa un mejor ajuste al objeto (ver fig. 1.17). La jerarquía de esferas se utiliza para dividir la superficie del objeto en varias regiones y a través de ellas poder dirigir la detección de la colisión hacia el área de la superficie del objeto donde probablemente existe intersección con el otro objeto. La meta principal de este tipo de algoritmo basado en árboles de esferas, es descartar rápidamente las posiciones libres de colisión dentro del ambiente y concentrar su trabajo en aquellas donde probablemente se presente una colisión y así reducir su tiempo de detección. (3)

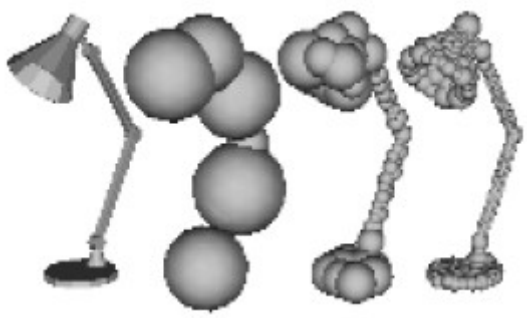

Fig. 1.17 Volumen envolvente jerárquico esférico.

Estructuras Jerárquicas AABB

En una jerarquía AABB, las cajas están alineadas a los ejes del modelo del sistema de coordenadas local y, por lo tanto, todas las casillas en un árbol tienen la misma orientación. Un árbol de AABB produce un fuerte ajuste al objeto. No presenta problema con el solapamiento o con los vecinos para ello crea pequeños árboles de cajas cada vez más ajustados a los objetos. Se basa solamente en la geometría del objeto, no le interesan las áreas adyacentes a este. No crea nodos vacíos, por lo que resulta muy rápido el chequeo de colisión (ver fig. 1.18). (10)

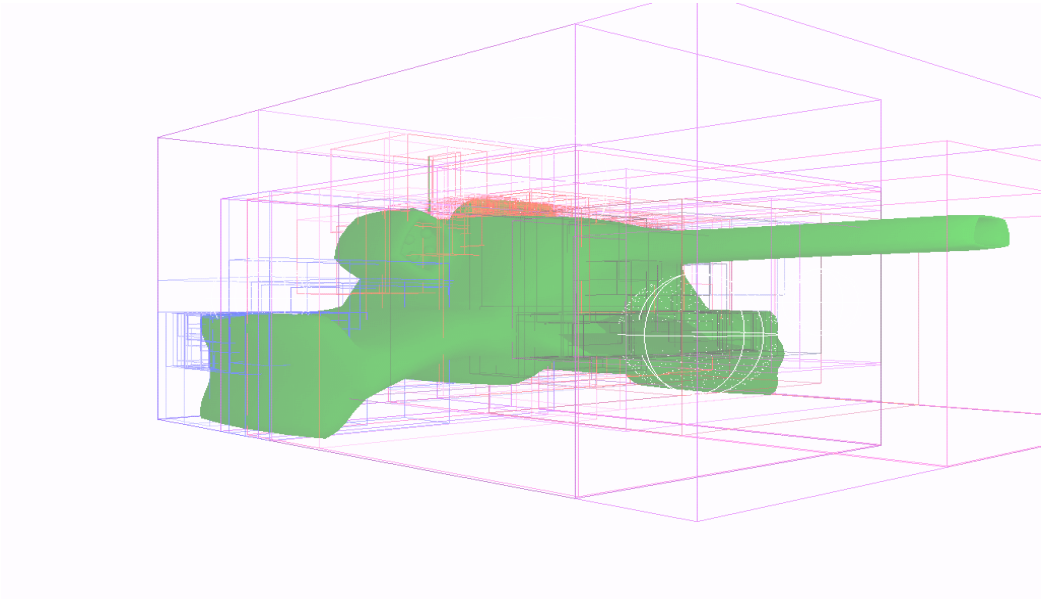

Fig. 1.18 Volumen envolvente jerárquico AABB.

Estructuras Jerárquicas OBB

La meta principal de una jerarquía OBB es encontrar el espacio más reducido entre el volumen y el modelo ó conjunto de vértices, lo que requiere un grado considerable de cálculo. Para construir un árbol OBB se requiere de dos componentes fundamentales: la colocación ajustada de un OBB alrededor de la colección de polígonos y la agrupación del OBB anidado dentro del árbol de jerarquía.

El árbol se construye de forma recursiva y cada caja OBB se divide según la orientación que varía en cada subdivisión (ver fig. 1.19). (5)

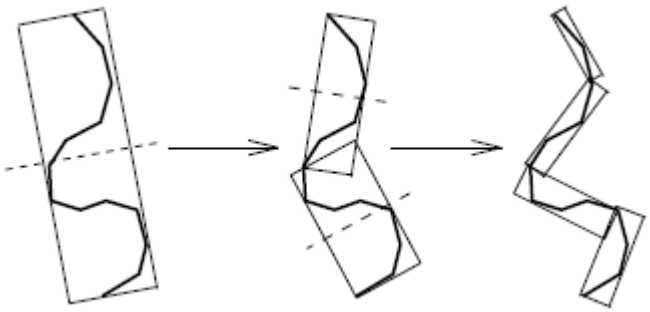

Fig. 1.19 Volumen envolvente jerárquico OBB.

1.2.3.3 Colisiones entre Objetos Simples o Primitivas

Esfera /Esfera

Principio: la intersección Esfera –Esfera es la más simple y rápida. Dando dos esferas A y B con radio r_A y r_B respectivamente, la intersección queda conformada de la siguiente forma:

$|p_A - p_B| <= r_A + r_B$ (13)

Triángulo/Triángulo.

Esta es quizás la más importante debido a que la mayoría de los objetos virtuales, están conformados por mallas de triángulos irregulares. Existen eficientes algoritmos para detectar las intersecciones entre triángulos.

Principio: La idea básica en estos algoritmos es determinar la existencia de una línea que intercepta los dos triángulos conformando dos segmentos (*Ia, Ib*). Si ambos segmentos se interceptan entonces ambos triángulos están colisionando En la siguiente figura se muestra lo antes explicado. (13)

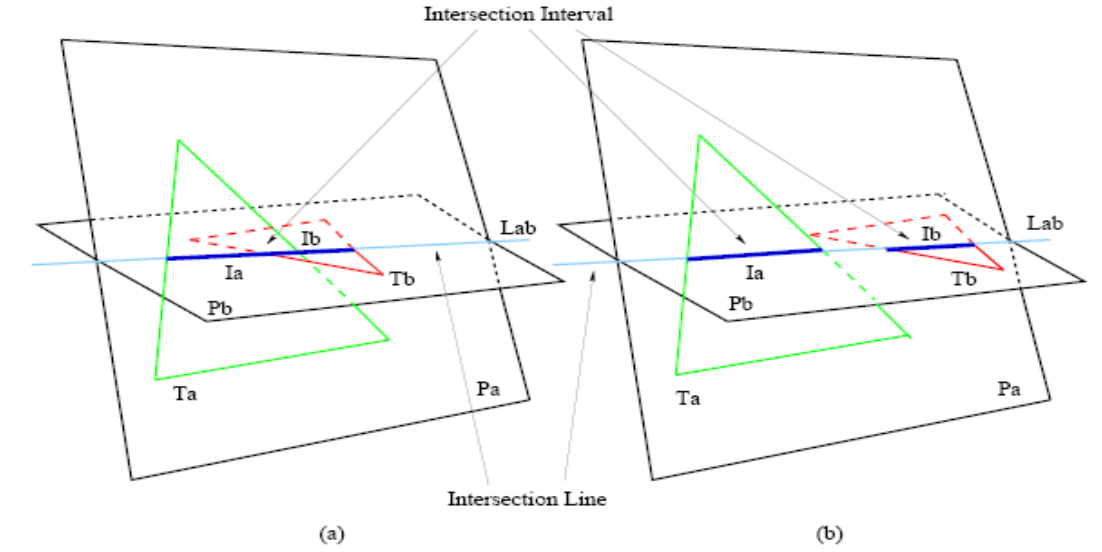

Fig. 1.20 Detección de colisiones entre triángulos.

Cubo/ Cubo

La intersección cubo/cubo es otro importante chequeo de colisión muy utilizado actualmente en la computación gráfica. La construcción de árboles jerárquicos usando cubo requiere este testeo.

Principio: La idea básica de este chequeo consiste en que si el punto máximo o mínimo de una caja se solapa con el punto máximo o mínimo de la otra o viceversa, entonces dichas cajas se interceptan. (13)

1.3 Bibliotecas para la detección de colisiones.

En la actualidad existen diferentes motores para la detección de colisiones en entornos 3D, donde cada uno presenta características y técnicas propias. Muchas de estas bibliotecas se han decantado por el desarrollo de software libre, por lo cual cuentan con actualizaciones bien detalladas y de fácil acceso por los desarrolladores. Dichas bibliotecas tienen muchas aplicaciones tanto en la rama de entretenimiento como en las ramas científicas y de educación.
A continuación se mencionan algunas de estas bibliotecas con sus principales características.

1.3.1 RAPID (Robust and Accurate Polygon Interference Detection)

Es una librería para la detección de colisión entre polígonos no estructurados. No realiza la fase de poda, de la que se encarga la librería VCOLLIDE que se verá a continuación. Es especialmente apropiada por su sencillez de uso y por su rapidez cuando hay pocos objetos en el entorno. (36)

1.3.2 V-COLLIDE

Es una librería construida para la poda previa en el sistema RAPID. Determina los pares de objetos que se encuentran potencialmente en contacto. Estos objetos son utilizados por la librería RAPID para la determinación de colisión. Permite entornos con un gran número de objetos, estáticos y dinámicos, así como la inserción o borrado interactivo de objetos. (37)

1.3.3 SOLID (Software Library for Interference Detection).

Esta librería, implementada en C++ y bajo licencia GNU, 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. 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 y para calcular distancias de penetraciones entre objetos.

• 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. Al trabajar con objetos convexos, SOLID hace uso de la librería QHull, la cual permite calcular la superficie convexa de un conjunto de puntos. (32)

1.3.4 ColDet (Free 3D Collision Detection Library)

Es una biblioteca de detección de colisión gratuita para polígonos genéricos, que proporciona una precisa detección entre objetos. Usa organización OBB para rápida detección; chequeos de intersección de triángulos adicional para un 100% de precisión. Baja petición, muestra el punto exacto de colisión, y también soporta la posibilidad de configuración para limitar la detección de colisión por tiempo. Esta biblioteca es un esfuerzo para proporcionar una detección de colisiones para poliedros genéricos. Su finalidad es principalmente para los juegos 3D, donde se necesita detectar colisiones exactas entre objetos. Proporciona un punto exacto de colisión, más el par de triángulos que chocó y apoya también el tiempo de ajuste, para limitar el tiempo de detección. (34)

1.3.5 OPCODE (Optimized Collision Detection)

Es una pequeña biblioteca de detección de colisiones. Trabaja sobre mallas (ya sean convexas o no convexas). Es una biblioteca de detección de colisión gratuita que muestra un menor uso de memoria por lo que las consultas de colisiones son más rápida que las estándares, para ello se basa en una estructura de volumen de jerarquía de árboles AABB. (33)

Conclusiones

En este Capítulo se definieron los principales conceptos que ayudarán a comprender el desarrollo del Módulo.

Como se pudo observar en el mismo se fundamentaron además, las principales técnicas, tecnologías y tendencias más utilizadas en la actualidad mundial en cuánto a estos temas, propiciando un mayor conocimiento para determinar cuáles son las mejores opciones para emplear en el diseño e implementación del Módulo.

2

Capítulo 2 Descripción de la Solución Propuesta

Introducción

En este Capítulo se presenta la descripción de la solución propuesta para el desarrollo del Módulo de Detección de Polígonos en Colisiones, colocando las fases de colisiones, detección y determinación de colisión y la descomposición de objetos y la comprobación exacta de los polígonos en colisión, como temas principales y necesarios para lograr mejoras eficientes en el realismo y el manejo de la detección de colisiones entre dos modelos conformados por mallas triangulares.

2.1 Lenguaje de Programación

En el mundo actual, existen tres lenguajes de programación que se destacan considerablemente en el desarrollo de aplicaciones gráficas, los cuales son el Lenguaje C, el Lenguaje Ensamblador y el Lenguaje C plus plus (C++). El Lenguaje C++, ha evolucionado en gran medida y tiene grandes resultados palpables en el acontecer gráfico generado por computadoras.

En las últimas décadas, se han unido recientemente los lenguajes de programación Java y C# para desarrollar este tipo de aplicaciones. (23) (14)

El Módulo de Detección de Polígonos en Colisión se implementó con el Lenguaje de programación C++ ya que la STK está desarrollada en este lenguaje y hasta la actualidad se han obtenido magníficos resultados.

C++ es un potente lenguaje de programación que surgió en la década de los años 80 para ampliar las ventajas, la flexibilidad y la eficiencia del Lenguaje C. Es la evolución del C adaptada a la filosofía de la **POO**, que ha desarrollado el manejo de tipos y estructuras de datos así como otras características que ayudan a que la programación se considere libre de fallas. (35)

El equipo que ha trabajado en el desarrollo de las funcionalidades de la herramienta tiene mayor experiencia en el Lenguaje C++, además de estar familiarizados con códigos fuentes estandarizados.

2.2 Detección y Determinación de colisión.

La detección de colisiones debe cumplir con tres conceptos fundamentales para alcanzar notorio realismo virtual. De estos conceptos mencionados en el capítulo anterior, el módulo desarrollado en este trabajo, solo cuenta con la Verificación de Colisión y con la Determinación de Colisión.

En el Capítulo anterior se mostraron algunas de las bibliotecas existentes en el mundo para detectar colisiones y una vía de solución a este trabajo hubiese sido utilizar alguna de estas bibliotecas. Sin embargo se llegó a la conclusión de implementar un Módulo de Detección de Polígonos en Colisión completamente desde las clases bases con su respectiva estructura de datos que se encarga de subdividir el o los objetos que se les realizará el chequeo de colisión.

Este trabajo se realizó de este modo, producto al significativo estudio sobre el estado del arte de las bibliotecas de Detección de Colisiones, existentes en el mundo actual, se determinó que la documentación era bien escasa y que se basan en código inextensible como para valerse de las mismas.

La biblioteca SceneToolkit (STK) está desarrollada con código estandarizado y específico, por lo cual hubo la necesidad de implementar el módulo de Detección de Colisiones con código flexible, reutilizable, extensible y estandarizado en C++ según las exigencias de la STK. Dicho módulo se adapta satisfactoriamente a las necesidades de la STK.

2.3 Fases implementadas en el módulo.

Como se ha explicado en el capítulo anterior, la detección de colisiones se enmarca en tres fases fundamentales como son la Fase Amplia, Estrecha y Exacta. Dicho Módulo ha desarrollado la detección de colisiones mediante la implementación de técnicas propias utilizadas por las Fases Estrecha y Exacta. La Fase Amplia no fue implementada debido a que no existe un entorno virtual complejo como para desarrollar los algoritmos y las estructuras específicas de dicha fase, por lo cual no constituyó un objetivo para este trabajo.

Hasta la actualidad, la **STK** y proyectos que la utilizan como “Simulador Quirúrgico” predeterminan cuáles son los objetos a chequear colisión del Entorno Virtual. Por tanto en estos momentos es más importante la detección exacta de polígonos pero luego se podrá extender a la Fase Amplia.

Entrada y salida de Fase Estrecha

1. Objetos de la escena posibles a colisionar (Entrada).
2. Divide la malla del objeto con profundidad 2^n donde N es la altura.
3. Devuelven las estructuras jerárquicas donde existe una posible colisión entre ambos objetos (Salida).

En las figuras 2.1, 2.2, 2.3, 2.4 se ejemplifica gráficamente cómo ocurren las iteraciones en la Fase Estrecha desde la entrada del objeto a subdividir hasta la creación de la estructura de colisión.

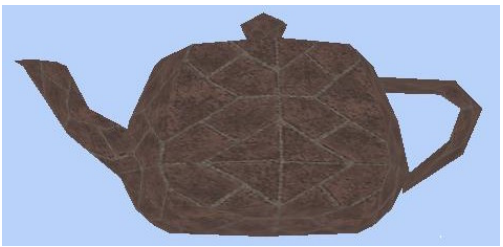

Fig. 2.1 Entrada del objeto a subdividir

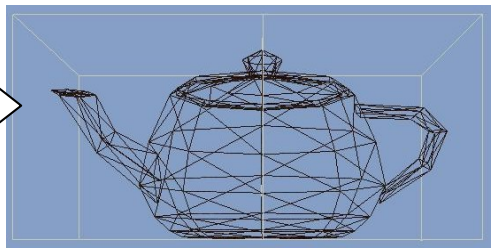

Fig. 2.2 Paso de subdivisión con altura 2

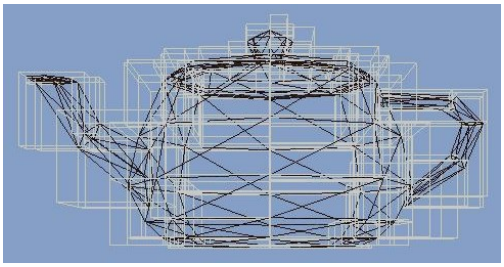

Fig. 2.3 Paso de subdivisión con altura 8

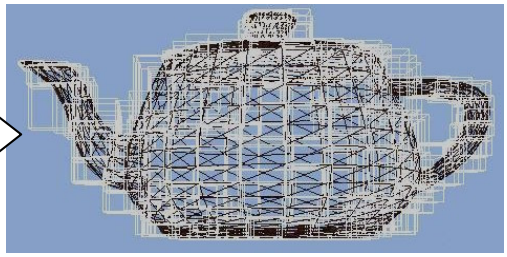

Fig. 2.4 Paso de subdivisión con altura 12

Entrada y salida de Fase Exacta

1. Estructuras Jerárquicas donde hay posible colisión entre ambos objetos (Entrada).
2. Se chequean los polígonos de cada área de posible colisión.
3. Se devuelven los polígonos solapados. (Salida)

En las figuras 2.5 y 2.6 se ejemplifica gráficamente cómo ocurren las iteraciones en la Fase Exacta desde la entrada de las estructuras de posible colisión hasta la intersección de polígonos de ambos modelos.

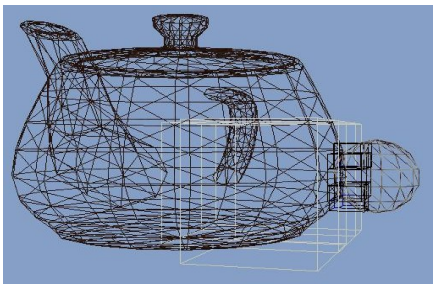

Fig. 2.5 Entrada de las áreas de posible colisión

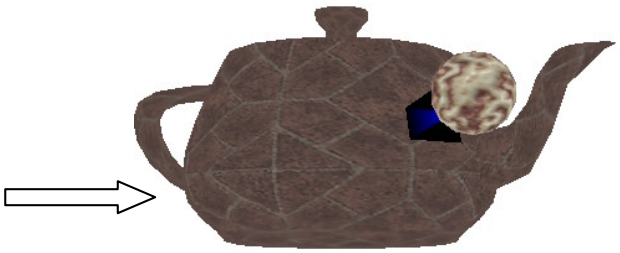

Fig. 2.6 Polígonos de intersección

Con la implementación de las fases de colisión anteriormente explicadas, el Módulo cuenta con un realismo eficiente y detallado para el manejo y detección de las colisiones entre objetos 3D generados por computadoras.

2.4 Intersección de polígonos

En la actualidad de la gráfica generada por computadoras, los efectos reales, de un entorno virtual, se logran a través de cálculos exactos obtenidos mediante métodos geométricos-matemáticos eficientemente implementados en una clase responsable. En la matemática geométrica realizada específicamente para lograr realismo en los entornos gráficos, es necesario resolver problemas básicos tanto de distancias entre objetos o estructuras matemáticas como problemas de cruzamientos de conceptos matemáticos (rectas, segmentos de rectas, planos, triángulos, etc.) o de puntos interiores o pertenecientes a un objeto matemático.

En la STK, existe una clase encargada de realizar las operaciones antes señaladas la cual no posee la totalidad de operaciones necesarias y las que tiene incluidas no están eficientemente desarrolladas, puesto que suponen datos necesarios que tienen que ser calculados previamente para dar la respuesta eficaz.

Dada la necesidad de incorporar algoritmos matemáticos a la STK, se diseñó y se implementó una clase Algoritmos Matemáticos manteniendo los conceptos de la filosofía de Programación Orientada a Objetos con funciones del lenguaje C++ estandarizados, según la codificación utilizada en la herramienta (ver fig. 2.7).

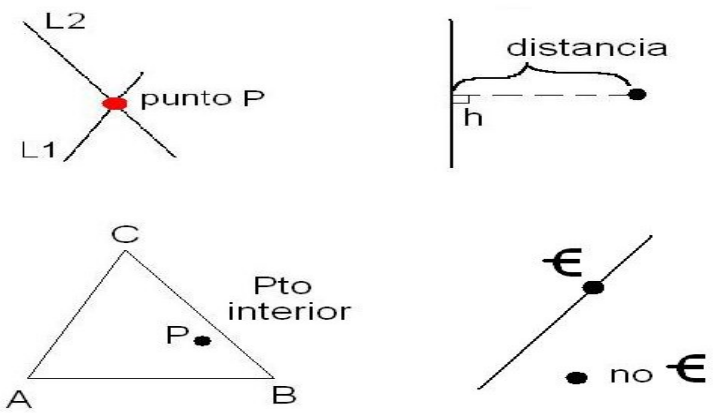

Fig. 2.7 Cálculos matemáticos básicos necesarios

La Informática Gráfica ha puesto todo su esfuerzo en la visualización de objetos conformados por mallas poligonales y la STK visualiza los objetos mediante ficheros

3DX, el cual se basa en la creación de una malla triangular del objeto, que a su vez contiene los vértices y los índices de los vértices de cada triángulo. Los triángulos se construyen ajustando un plano a tres puntos cercanos no colineales y se adosan sobre el objeto formando una malla con diferente nivel de detalle, en función de la complejidad del objeto. Cada objeto queda representado por un conjunto de superficies planas que se ajustan a un conjunto previo de puntos en forma de triángulos.

Todos los objetos virtuales están conformados por mallas de triángulos irregulares. Es muy importante, en el momento de manejar los gráficos, tener bien preciso toda la información de cada triángulo de la malla, desde la ubicación de cada vértice hasta la localización y dirección de los mismos.

Los objetos virtuales tienen la opción de moverse, de rotar, de transformar tanto su forma visual como la forma visual del objeto que interactúe con él. Todos los comportamientos antes dichos, se logran con una eficiente manipulación de cada triángulo del objeto, más bien, con cada vértice de cada triángulo de la malla, por tanto es muy necesario tener almacenada toda la información de cada triángulo o tener la opción de adquirir dicha información en tiempo de ejecución para el pintado.

La biblioteca SceneToolkit, trabaja con las mallas triangulares y centra todos sus cálculos en dicha malla para lograr efectos reales a la hora de pintar cada objeto (ver fig. 2.8 y fig. 2.9). La detección de colisiones entre objetos se logra con la utilización de la información de cada triángulo de la malla de los objetos que han cambiado su estado, su posición o su forma en el tiempo, las figuras a continuación dan muestra gráfica de un objeto y su malla.

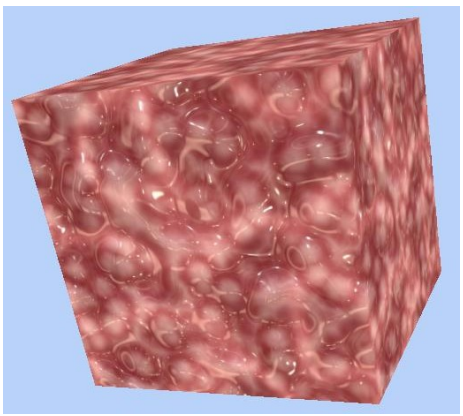

Fig. 2.8 Objeto 3D

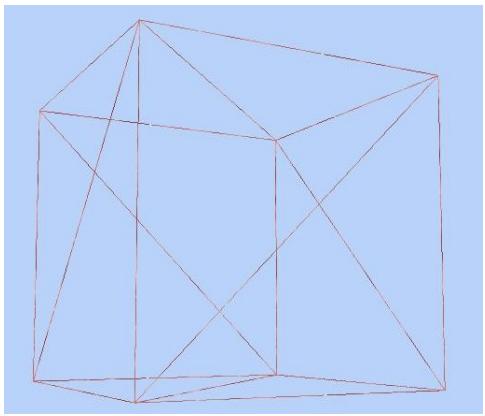

Fig. 2.9 Malla triangular del Objeto 3D

La STK, carecía de una **clase Triángulo** que controlara la situación real e instantánea de los triángulos de la malla, además de un conjunto de información y de operaciones propias necesarias para la detección de colisión del objeto procesado. Por lo antes señalado se llevó a cabo la implementación de dicha clase con los atributos y métodos correspondientes ajustados a las propiedades físicas y matemáticas de un triángulo manteniendo el diseño y la implementación dentro de la filosofía de Programación Orientada a Objetos, a través de funciones programadas puramente en el **lenguaje de desarrollo** C++.

2.5 Descomposición de objetos mediante el volumen envolvente jerárquico de AABB.

En el Capítulo anterior se estudiaron los principales algoritmos utilizados para la detección de colisiones en entornos virtuales en la Fase Estrecha. En el siguiente epígrafe se propone cuál de dichos algoritmos resulta más eficiente, para de esta forma dar cumplimiento a nuestro objetivo inicial.

2.5.1 Volumen Envolvente AABB.

La técnica de volumen envolvente, es muy utilizada en la actualidad en la Fase Estrecha, en el capítulo anterior se mencionan una serie de algoritmos que son muy utilizados en esta fase, entre ellos se ha elegido trabajar con el de: cajas alineadas a los ejes (AABB).

AABB es un simple y elegante volumen envolvente y es muy utilizado en todas las ramas de la computación gráfica. Presenta una rápida actualización, por lo que es muy utilizado en la detección de colisiones entre modelos deformables (ver fig. 2.10).

El cálculo 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 (ver fig. 2.11). (2)

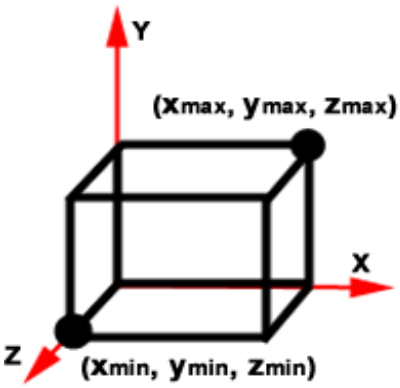

Fig. 2.10 Aspectos básicos del AABB.

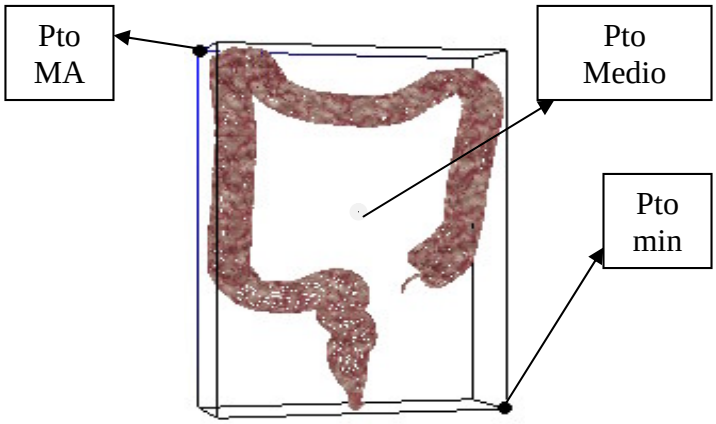

Fig. 2.11 AABB.

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 (ver fig. 2.12). 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 lo que tiene que reconstruirse (ver Fig. 2.13).

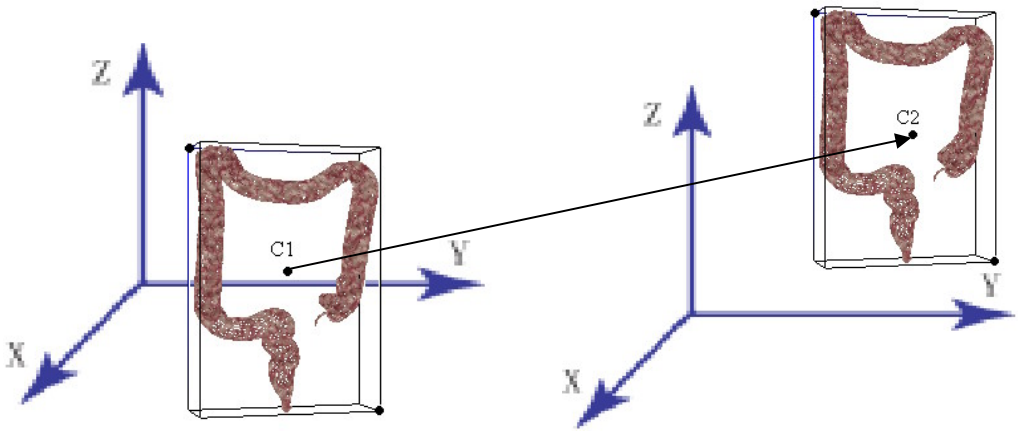

Fig. 2.12 Traslado de un AABB.

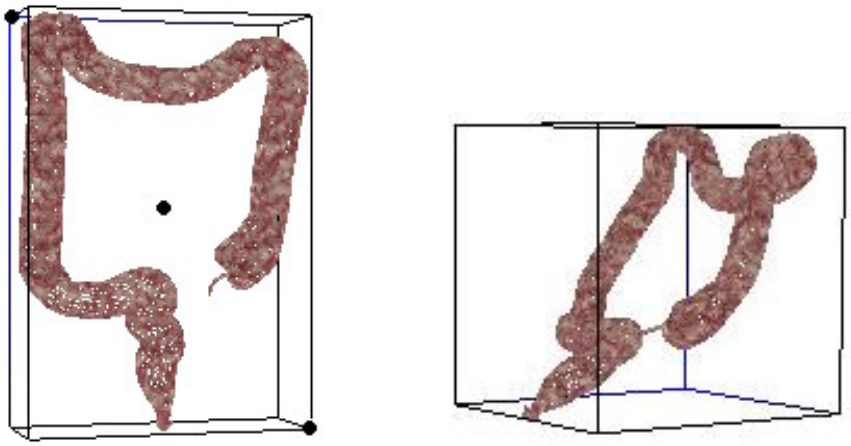

Fig. 2.13 Rotación de un objeto que se encuentra englobado en un volumen envolvente AABB

Este tipo de volúmenes de inclusión no produce un ajuste apropiado para muchos objetos. Además, actualizar este volumen de inclusión suele ser más costoso que otros cuando los objetos rotan. (24)

La prueba de colisión a este volumen es muy rápida y sencilla, consistiendo en la comparación directa de los valores de coordenadas. No así la de un OBB, que aunque presenta un mayor ajuste al objeto y una rápida actualización en las rotaciones, la prueba de colisión de este resulta menos ineficiente que la de un AABB**.** Cuando los objetos están orientados arbitrariamente, se hace necesario guardar mucha más información para representar este volumen bajo la forma del centro de la caja, tres longitudes y una matriz de rotación, por lo cual la detección de colisiones se hace más compleja. También es necesario recalcular las cajas contenedoras cuando se produce una deformación, por lo que resulta ineficiente para ambientes interactivos que presentan objetos deformables.

Todo lo contrario lo presentan los volúmenes esféricos envolventes, que aunque tiene la ventaja de ser un volumen invariante ante rotaciones y de necesitar poco espacio de almacenamiento, presenta como inconveniente que no se ajusta a determinados objetos, siendo difícil obtener la esfera que mejor ajuste a un objeto dado. Un ejemplo de ello sería que cuando los objetos son planos, o alargados y delgados, esta técnica produce muchos falsos positivos, es decir, aunque haya intersección entre las esferas no existe tal entre los objetos, por lo que resulta poco efectivo para objetos convexos. (22)

2.5.1.1 Pruebas de Colisión con AABB

Se implementaron diferentes algoritmos matemáticos asociados a este volumen para complementar de esta forma el siguiente paso que sería construir un árbol binario de AABB.

AABB/AABB

A continuación se muestran los pasos a seguir para el chequeo de colisión entre dos $AABB_s$, el cual se necesitan los puntos máximos y mínimos en las coordenadas **x, y, z** de ambos (ver fig. 2.14).

Entrada: Puntos mínimos y máximos de dos AABB

Salida: Existe colisión o no

1- Verifica si el punto mínimo en la coordenada x del primer AABB es mayor que el punto máximo en la coordenada x del segundo AABB o si el punto mínimo en la coordenada x del segundo AABB es mayor que el punto máximo en la coordenada del primer AABB.

2- Repetir para las coordenadas y, z.

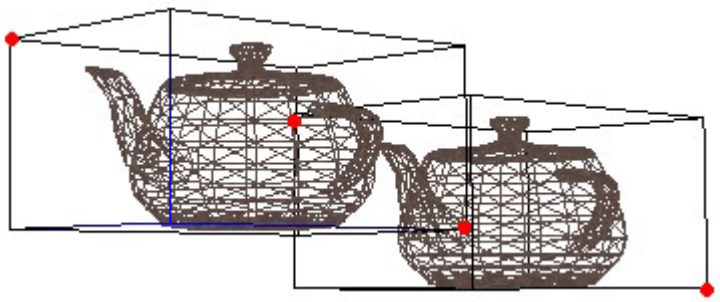

Fig. 2.14 Chequeo de colisión entre cajas AABB.

Esfera/AABB

A continuación se presenta los pasos que se llevan a cabo para el chequeo de intersección entre una esfera y un AABB. En el mismo se calcula la distancia del centro a los puntos máximo y mínimo en las coordenadas **x, y, z** del AABB (ver fig. 2.15).

Entrada: Centro y Radio de la esfera, punto mínimo y máximo del AABB a colisionar

Salida: Existe colisión o no

1- Si la coordenada x del centro de la esfera es menor que el punto mínimo en la coordenada x del AABB:
   a. Adicionar a la distancia entre la esfera y el AABB, la distancia que existe entre la coordenada x del centro de la esfera restado con el punto mínimo en la coordenada x del AABB al cuadrado.
2- Si la coordenada x del centro de la esfera es mayor que el punto máximo en la coordenada x del AABB:
   a. Adicionar a la distancia entre la esfera y el AABB, la distancia que existe entre la coordenada x del centro de la esfera sumado con el punto máximo en la coordenada x del AABB en x al cuadrado.
3- Repetir para las coordenadas y, z.
4- Verifica si la distancia entre la esfera y el AABB es menor ó igual que el radio de la esfera.

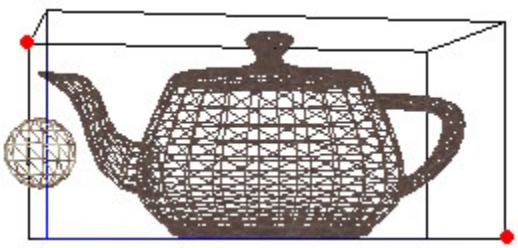

Fig. 2.15 Chequeo de colisión entre AABB/Esfera.

Segmento/AABB

En la figura 2.16 se muestra otro importante algoritmo de colisión como es el caso de la intersección de segmento – AABB. Al igual que el anterior algoritmo necesita los puntos mínimo y máximo del AABB además del punto inicial y final del segmento que va colisionar.

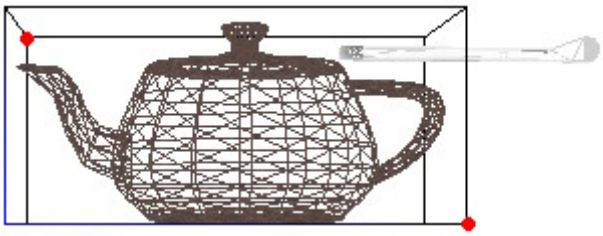

Fig. 2.16 Chequeo de colisión entre AABB/Segmento.

2.5.2 Volumen Envolvente Jerárquicos AABBtree

Anteriormente se conocieron algunas características fundamentales de un árbol de AABB. Otra característica de dicho algoritmo es que el costo de la actualización del árbol aumenta significativamente con el número de polígonos utilizados. Además, al no tener en cuenta la orientación del objeto, pueden ser necesarias muchas subdivisiones para ajustarse a éste. No obstante cuando se producen pequeñas deformaciones, no es necesario reconstruir todas las cajas, sino reajustar las cajas del nivel inferior de la jerarquía hasta un cierto nivel en el árbol, lo que le da una gran ventaja con respecto al OBB que debe reconstruir el árbol completo en caso de deformaciones.

Otro aspecto en contra del OBB es que aunque permite un mayor ajuste a la forma del objeto en pocos pasos, la construcción de la estructura es bastante compleja y mucho más costosa que la de un AABB. No así el caso de las estructuras jerárquicas esféricas que aunque presentan una rápida y mucha más sencilla construcción del árbol, presenta problemas con la mala aproximación a los objetos bastante irregulares, por lo que en ocasiones se debe construir un árbol de mucha más profundidad para lograr un mejor ajuste, lo que aumenta el costo computacional y al mismo tiempo el chequeo de colisión. (29)

2.5.2.1 Construir Árbol Binario AABB

El árbol AABB se construye de arriba hacia abajo, por subdivisión recursiva. En cada paso recursivo se calcula el punto medio promedio de la lista de triángulos que componen al objeto y que engloban el AABB, seguidamente se calcula el mayor eje de coordenadas del mismo y finalmente se decide con respecto a este eje y al punto medio promedio de triángulos, logrando de esta forma que los AABBs resultantes estén compuestos por una cantidad de triángulos proporcional. Este proceso continúa hasta que el árbol tenga la profundidad indicada o necesaria para lograr una eficiente detección de colisión. (25) (26) (27)

Para no tener triángulos repetidos en los AABB, se calcula el punto medio de los mismos y se incluye dentro del AABB correspondiente a este según el punto promedio de los triángulos y el eje mayor (ver fig. 2.17). De esta manera aumenta el

costo, pero queda compensado a la hora del chequeo de colisión, ya que no existirán triángulos repetidos dentro de los AABBs.

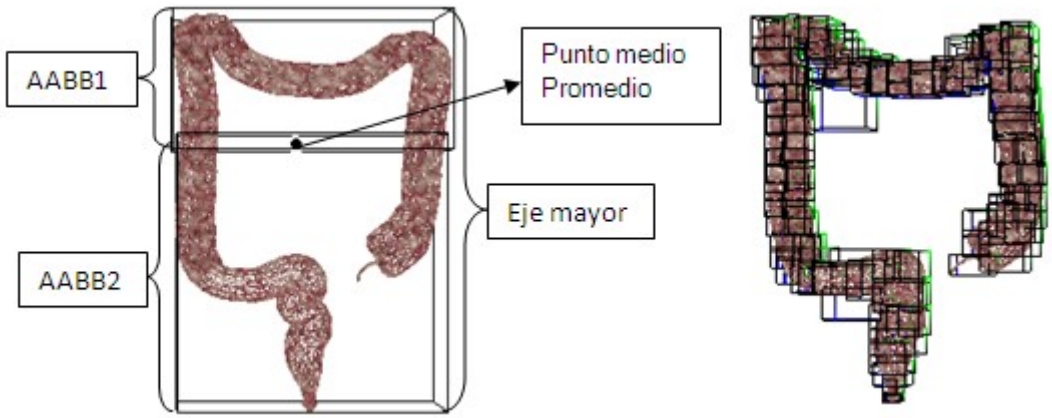

Fig. 2.17 Árbol AABB de profundidad 2 y profundidad 8 respectivamente.

2.5.2.2 Pruebas de Colisión con un Árbol AABB

La prueba de intersección entre dos modelos se hace de forma recursiva a través de los pares de nodos. Cada nodo contiene un AABB. Si se solapan los nodos entonces se comprueba con sus hijos hasta llegar a las hojas y de esta forma devolver la menor cantidad de primitivas posibles que pueden estar colisionando (ver fig. 2.18).

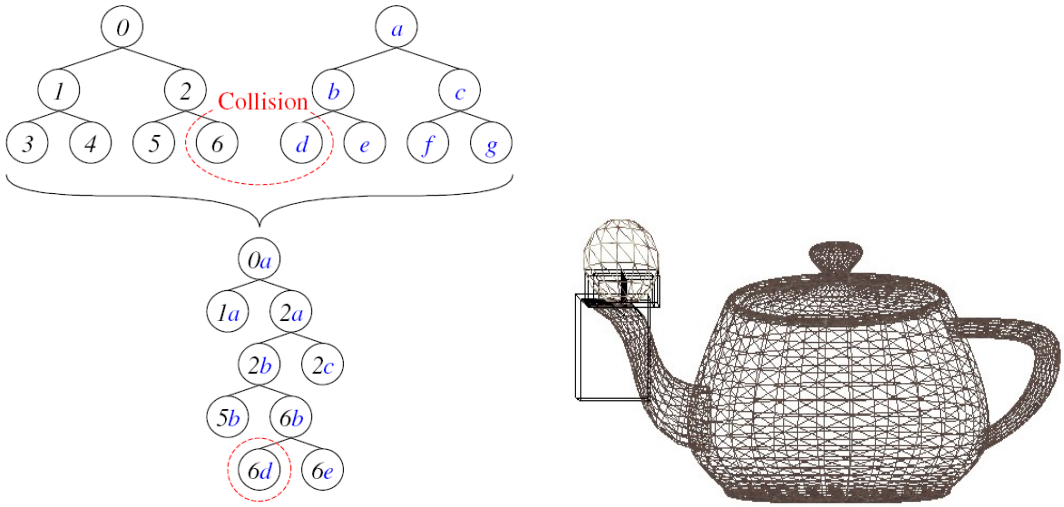

Fig. 2.18 Árbol de prueba de colisión entre AABBs.

Se desarrollaron además otras pruebas de colisiones muy utilizadas, como son: Esfera/AABB tree y Segmento/AABB tree. En la figura 2.19 se muestra el árbol de colisión del algoritmo de detección de colisión entre una esfera o un segmento y un AABB tree (ver fig. 2.20, 2.21).

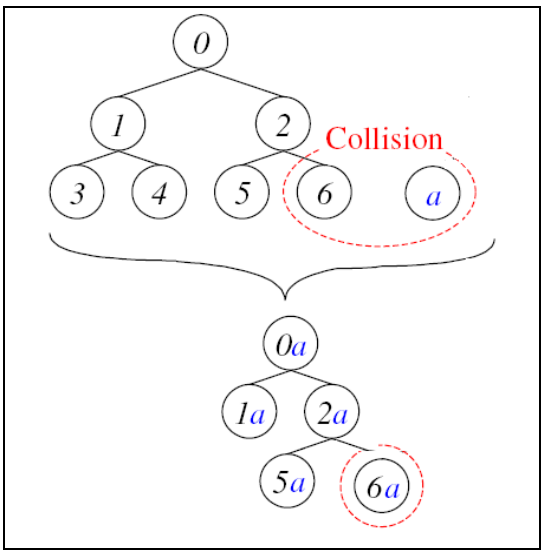

Fig. 2.19 Árbol de prueba de colisión entre un objeto y un AABB.

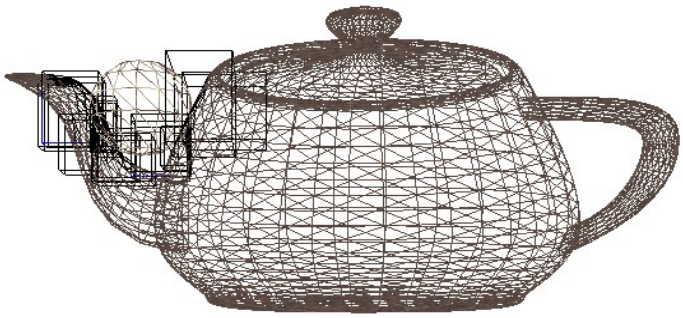

Fig. 2.20 Intersección entre una esfera y un AABB tree.

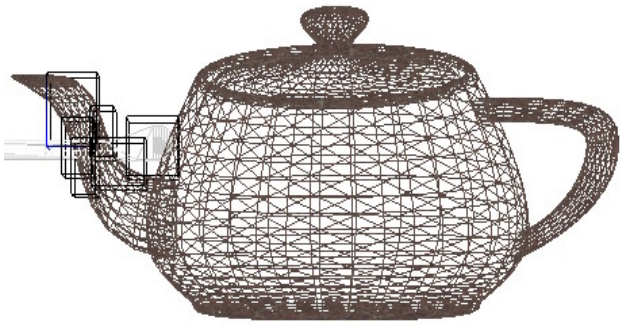

Fig. 2.21 Intersección entre un segmento y un AABB tree.

Conclusiones

En el presente Capítulo se especificaron las soluciones técnicas que se le dan al Módulo de Detección de Polígonos en Colisión entre objetos 3D y de esa forma diseñar una estructura de clases en correspondencia con las técnicas citadas y para la realización de los algoritmos planteados.

3

Capítulo 3 Diseño del Sistema

Introducción

En el presente Capítulo se mostrará una vista del Módulo que será implementado basado en las dificultades, necesidades y características de organización del cliente. Se matizan las reglas del negocio establecidas, se identifican él o los actores conjuntamente con los trabajadores, además de plantear los requerimientos funcionales y no funcionales con los cuales el sistema debe contar.

Además, se mostrarán representados los casos de uso del sistema acompañados de las respectivas relaciones a través del diagrama de casos de uso del sistema, modelado con el Lenguaje Unificado de Modelación (UML) y sus especificaciones en modo expandido.

3.1 Reglas del Negocio

Es importante saber identificar las reglas del negocio, pues a través de ellas, se describen políticas a cumplir y condiciones a satisfacer dentro del campo de acción del trabajo a realizar. Estas reglas deben ser evaluadas teniendo en cuenta lo sustancial que puede ser cada una para la investigación.

A continuación se exponen las reglas definidas para este dominio de trabajo:

- El sistema sólo soporta mallas triangulares.
- La aplicación debe cargar los dos objetos a los cuales se les harán las pruebas de colisión.
- Se debe indicar de qué tipo serán las pruebas de colisión.
- Las mallas triangulares de los objetos deben ser lo más regular posible.

3.2 Modelo de Dominio

El Módulo a desarrollar forma parte de la **STK** y por tanto, por sí solo no constituye un negocio, así que, según sus funcionalidades y prestaciones se llevará a cabo un Modelo del Dominio en el cual se mostrarán los conceptos fundamentales manejados por el sistema de desarrollo.

Dicho modelo del dominio proporciona una ayuda factible a las personas involucradas con el proyecto para comprender los términos empleados, realizar una correcta captura de requisitos y agrupar de manera óptima los conceptos necesarios que permiten identificar las clases que pasarán a ser parte del diseño (ver fig. 3.1).

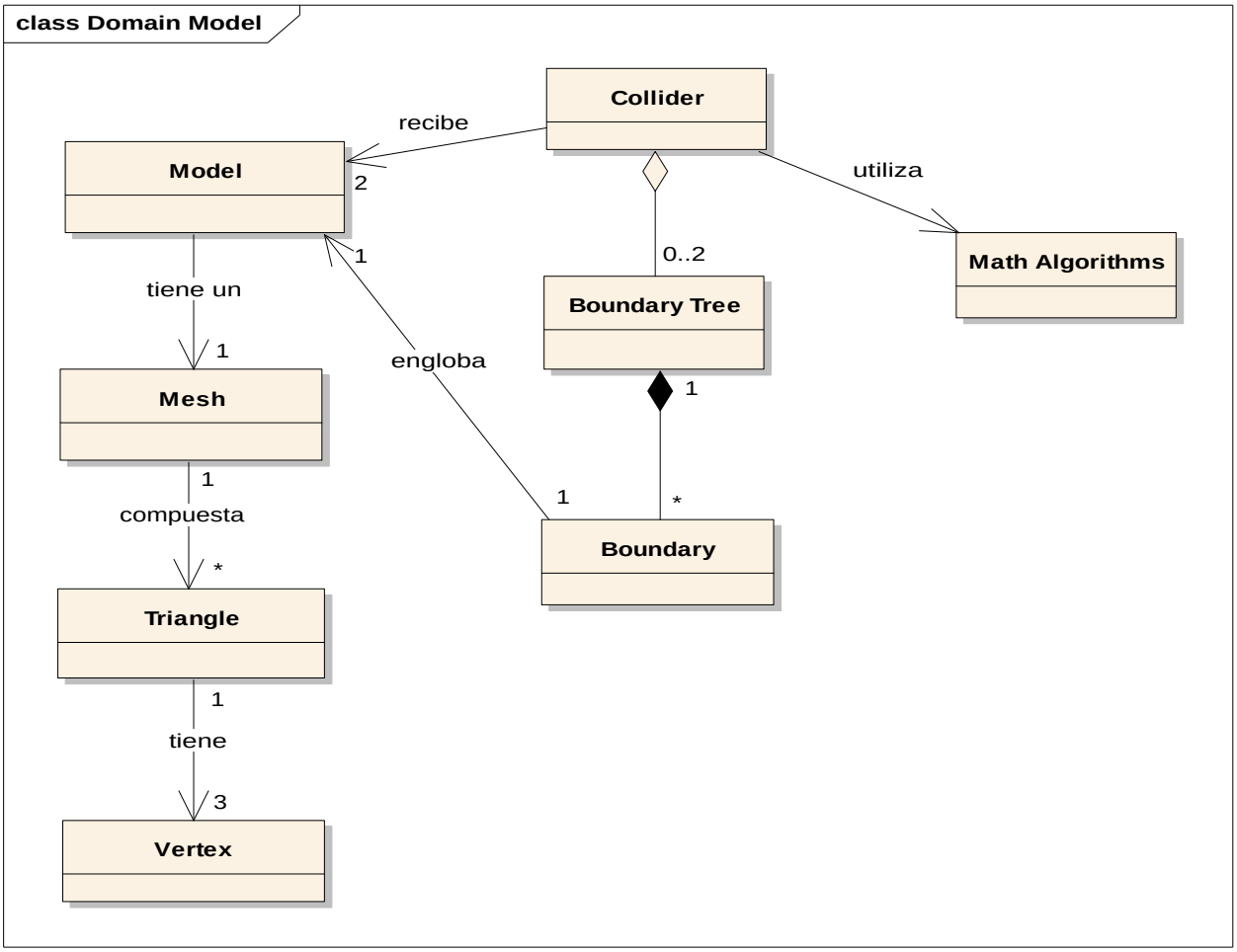

Fig. 3.1 Modelo del dominio.

3.3 Glosario de términos del modelo del dominio

Para facilitar la compresión del diagrama “Modelo del Dominio”, a continuación se presentan un conjunto de conceptos que conforman el glosario de los términos empleados.

CCollider: Encargado de llevar a cabo todo el proceso de detección y determinación de las colisiones entre modelos.
Model: Objeto de la escena al cual se le realizará la prueba de colisión.
Mesh: Forma de representar un modelo a partir de polígonos. Conjunto de vértices y aristas conectados.
Triangle: Almacena tres vértices de un triángulo y opera dicho triángulo según sus vértices.
Vertex: Encargado de almacenar las coordenadas x, y, z.
Boundary tree: Subdivide el modelo en partes más pequeñas de manera que sea más sencilla la prueba de colisión.
Boundary: Encapsula el modelo en un volumen de inclusión para reducir costo de colisión.
Math Algorithms: Concentra varios métodos capaces de resolver situaciones geométricas-matemáticas.

3.4 Levantamiento de Requisitos

Las capacidades o condiciones que el sistema debe cumplir y facilitan el entendimiento entre clientes y usuarios son conocidos como Requisitos. Estos requisitos deben ser levantados para llevar a cabo un desarrollo eficiente teniendo en cuenta los requisitos funcionales y no funcionales, además de las restricciones impuestas y las propiedades propias del sistema.

A continuación se muestran las funcionalidades e instrucciones a cumplir.

3.4.1 Requisitos Funcionales:

1- Crear colisionador.
   1.1- Crear lista de volúmenes de inclusión de modelos estáticos.
   1.2- Determinar objetos estáticos.
   1.3- Adicionar objetos estáticos a la lista.
2- Verificar colisiones entre modelos.
   2.1- Identificar tipo de colisión.
   2.2- Identificar objetos estáticos.
   2.3- Crear estructura envolvente a objetos no estáticos identificados.
   2.4- Chequear colisión entre modelos.
      2.4.1- Chequear colisión entre dos estructuras envolventes (**EE**).
      2.4.2- Chequear colisión entre dos estructuras envolventes jerárquicas. (**EEJ**).
      2.4.3- Chequear colisión entre **EE** y **EEJ**.
      2.4.4- Chequear colisión entre **EE** y **Esfera**.
      2.4.5- Chequear colisión entre **EE** y **Segmento.**
      2.4.6- Chequear colisión entre **EEJ** y **Esfera.**
      2.4.7- Chequear colisión entre **EEJ** y **Segmento.**
      2.4.8- Generar lista de triángulos de colisión.
      2.4.9- Chequear colisión entre **Segmento** y **Esfera.**
3- Crear estructuras de volúmenes de inclusión.
   3.1- Generar lista de triángulos de la malla del modelo.
   3.2- Determinar la cantidad de triángulos de la malla.
   3.3- Asignar la cantidad de triángulos.
   3.4- Determinar el id del volumen de inclusión.

3.5- Asignar el id del volumen de inclusión.

3.6- Hallar coordenadas X, Y, Z del punto máximo.

3.7- Hallar coordenadas X, Y, Z del punto mínimo.

3.8- Asignar punto máximo al volumen de inclusión.

3.9- Asignar punto mínimo al volumen de inclusión.

3.10- Calcular coordenadas X, Y, Z del punto medio de la malla.

3.11- Asignar el punto medio de la malla.

4- Chequear intersecciones entre objetos básicos.

4.1- Chequear intersección entre punto y segmento.

4.2- Chequear intersección entre punto y punto.

4.4- Chequear intersección entre segmento y segmento.

5- Crear estructura de Volumen de Inclusión Jerárquico.

5.1- Generar árbol binario de volumen de inclusión.

5.1.1- Crear dos listas de triángulos.

5.1.2- Calcular punto medio del volumen de inclusión.

5.1.3- Determinar el eje de coordenada mayor del volumen de inclusión.

5.1.4- Dividir el volumen de inclusión según criterio de división.

5.1.4.1- Llenar primera lista de triángulos.

5.1.4.2- Llenar segunda lista de triángulos.

3.4.2 Requisitos no Funcionales

Los requisitos que a continuación se muestran son las propiedades o las cualidades que dicho Módulo debe tener, estableciendo así los aspectos que regulan el comportamiento del Módulo de Detección de Colisiones.

Usabilidad:
Se necesita de programadores con alta calificación en el lenguaje C++ para los niveles apropiados de usabilidad y con elementales conocimientos acerca de la programación gráfica y sobre la Herramienta de Desarrollo para Sistemas de Realidad Virtual. Es necesario mantener la terminología de la implementación en el idioma extranjero inglés.

Rendimiento:
Dicho módulo tendrá buena velocidad de procesamiento ó de cálculo, además presentará eficiente tiempo de respuesta, de recuperación y disponibilidad como aplicación bajo las normas establecidas para el concepto de tiempo real.

Soporte:
Es compatible con las plataformas de Windows y Linux.

Hardware:
Se necesita un ordenador con 1GB de memoria RAM como mínimo, además, mantiene compatibilidad con tarjetas gráficas de la familia NVIDIA (Geforce 3, Geforce 4, FX 5200).

Diseño e implementación:
Será utilizada la biblioteca de clases STK donde se encuentran definidas gran número de funcionalidades básicas para el manejo de los gráficos por computadora, soporta las bibliotecas OpenGL y DirectX. El lenguaje de programación que se empleará para realizar las llamadas a la biblioteca STK será C/C++, manteniendo la filosofía de la Programación Orientada a Objetos.

Requerimientos de Seguridad:
Se garantizará el acceso a la información. Los dispositivos o mecanismos utilizados para lograr la seguridad, no ocultarán o retrasarán a los usuarios para obtener los

datos deseados en un momento dado, en fin, la información estará disponible y accesible en todo momento.

Requerimiento legal:

Dicho módulo se regirá en su totalidad por las normas ISO 9000.

3.5 Modelo Caso de Uso

A continuación se reconocerán él o los posibles actores del sistema a desarrollar además de crearse los Casos de Uso del sistema, en los cuales se dan los resultados de cuantía para un actor especificado. Sintonizado en el primer ciclo de desarrollo se seleccionan los casos de uso correspondientes y se les realizan las especificaciones textuales con el formato expandido.

3.5.1 Actores del Sistema

Los Actores son los roles que se llevan a cabo en algún momento por parte de un usuario o varios a la vez. No solamente son usuarios sino que también suelen ser sistemas con los que el sistema en proceso de modelado tiene interacción. A través de los actores el sistema es estimulado con eventos de entrada o con resultados producidos, los cuales serán captados.

Tabla 3.1 Actor del Sistema

Actores	Justificación
Sistema que va a hacer uso de la herramienta SceneToolKit: Sistema Usuario	Es el que se beneficiará con las funcionalidades que brinda el módulo de clases, a grosso modo: crear la estructura de datos correspondiente a la detección de colisión y determinar la colisión entre dos modelos determinando el nivel de realismo que desee según la altura del AABB Tree.

3.5.2 Descripciones de Casos de Uso del Sistema (CUS):

El comportamiento del sistema desde el punto de vista del usuario es descrito por los casos de uso. Mediante estos se establece un acuerdo entre clientes y desarrolladores sobre las condiciones y posibilidades que debe cumplir el módulo a implementar. Como característica bien aprovechada de los casos de uso, es que también suelen incluir funcionalidades de otros casos de uso manteniendo sus propios procedimientos.

Tabla 3.2 Caso de uso del sistema, Determinar colisión.

CU1	Determinar Colisión.
Actor	STK
Descripción	Determinar si dos objetos colisionan.
Referencia	R1, R1.1, R1.1.1, R1.1.2, R1.1.3, R2, R2.1, R2.2, R2.3, R2.4, R2.4.1, R2.4.2, R2.4.3, R2.4.4, R2.4.5, R2.4.6, R2.4.7, R2.4.8, R2.4.9, R4, R4.1, R4.2, R4.3

Tabla 3.3 Caso de uso del sistema, Construir AABB Tree.

CU2	Construir AABB Tree
Actor	CU1
Descripción	Se encarga de subdividir el objeto
Referencia	R3, R3.1, R3.2, R3.3, R3.4, R3.5, R3.6, R3.7, R3.8, R3.9, R3.10, R3.11, R5, R5.1, R5.1.1, R5.1.2, R5.1.3, R5.1.4, R5.1.4.1, R5.1.4.2

3.6 Diagrama de Caso de Uso del Sistema

El modelo de Caso de Uso describe la funcionalidad propuesta del nuevo sistema, en este caso, del Módulo de Detección de Polígonos en Colisión. Se describen aquellos aspectos del sistema que son relevantes al propósito del modelo y a un apropiado nivel de detalle, dando una visión general de las relaciones de las funcionalidades o procesos fundamentales.

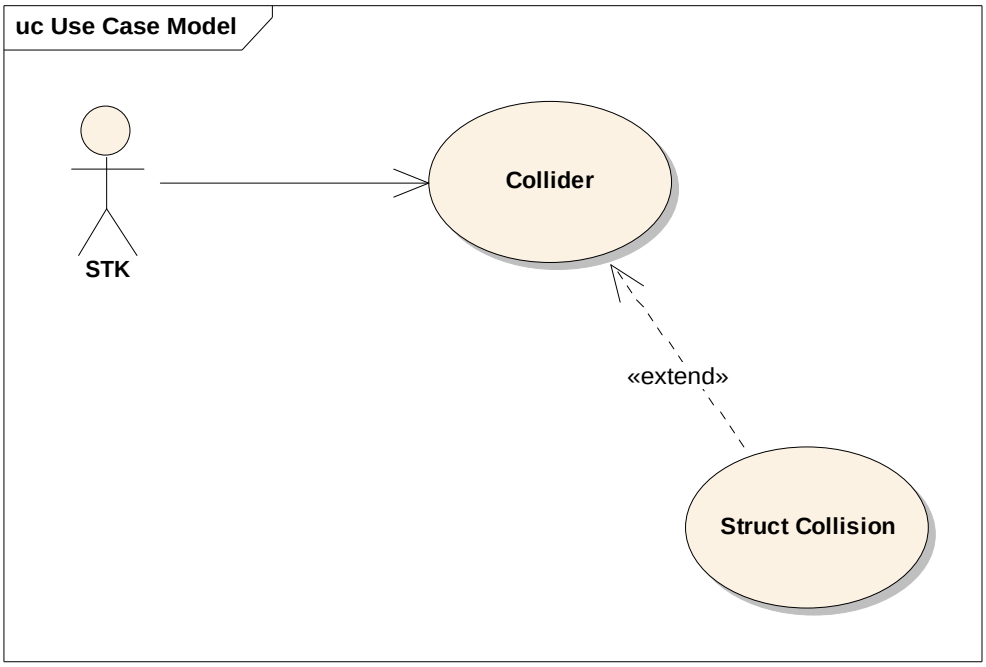

Fig. 3.2 Diagrama de casos de uso del sistema.

3.6.1 Expansión de Casos de Uso

Cada caso de uso tiene una descripción que describe la funcionalidad que se construirá en el sistema propuesto, las tablas presentadas a continuación forman parte de este formato expandido, donde se argumentan con mayor profundidad los flujos operacionales de cada caso de uso.

Tabla 3.4 CU Expandido, Determinar Colisión.

CU1	Determinar Colisión
Propósito	Determinar las coordenadas de colisión entre dos cuerpos
Actor	STK
Resumen: El caso de uso se inicia cuando el actor determina a qué modelo quiere hacerle la prueba de colisión.	
Referencias	R1, R1.1, R1.1.1, R1.1.2, R1.1.3, R2, R2.1, R2.2, R2.3, R2.4, R2.4.1, R2.4.2, R2.4.3, R2.4.4, R2.4.5, R2.4.6, R2.4.7, R2.4.8, R2.4.9, R4, R4.1, R4.2, R4.3
Precondiciones	Que estén cargados los dos modelos a los cuales se les realizará la prueba de colisión.
Poscondiciones	En caso de colisión: Determinar las coordenadas de colisión entre estos dos modelos.
Curso Normal de los Eventos	
Acción del actor	Respuesta del sistema
1- Elige los dos modelos a los cuales se les verifica si colisionan en algún momento y el tipo de colisión entre ellos.	1.1- Verifica el tipo de colisión de los modelos. 1.2- El sistema ejecuta algunas de las siguientes acciones: a) Si el tipo de colisión es entre modelos simples ir a la sección: "**Colisión Simple**" b) Si el tipo de colisión presenta algún modelos complejo ir a la sección: **"Colisión Compleja"**
Sección "Colisión Simple"	
Acción del actor	Respuesta del sistema

	1.3- Comienza el chequeo de colisión entre dos modelos simples. 1.4- En caso de que ocurra colisión se le informa al actor.
Sección “Colisión Compleja”	
	1.3- Manda a construir un AABB tree del o los modelos complejos ver CU2: **Construir AABB tree** 1.4- Comienza el chequeo de colisión entre dichos modelos. 1.5- En caso de que ocurra colisión se le informa al actor.
Flujo Alternativo	
Acción del actor	Respuesta del sistema
	5- No devuelve nada en caso de que no ocurra colisión.

Tabla 3.5 CU Expandido, Construir AABB Tree

Caso de uso		
CU2	Construir AABB tree	
Propósito	Se encarga de subdividir el objeto.	
Actor	CU1	
Resumen: El caso de uso se inicia cuando el actor determina qué modelo quiere subdividir.		
Referencias	R3, R3.1, R3.2, R3.3, R3.4, R3.5, R3.6, R3.7, R3.8, R3.9, R3.10, R3.11, R5, R5.1, R5.1.1, R5.1.2, R5.1.3, R5.1.4, R5.1.4.1, R5.1.4.2	
Precondiciones	Que esté cargado el objeto al cual se va a subdividir	
Poscondiciones	Objeto dividido en partes más pequeñas	
Curso Normal de los Eventos		
Acción del actor		Respuesta del sistema
1-Elige el modelo el cual se le va a construir un AABB tree y la profundidad de este.		
		2- Engloba al modelo mediante un AABB raíz.
		3- Localiza el punto medio promedio de la malla triangular que conforma el objeto y el eje mayor del AABB y construye
		4- Construye el árbol binario de manera recursiva hasta la profundidad requerida por el actor.

Conclusiones

Mediante la especificación de las soluciones técnicas que se le dieron a este proyecto en el presente Capítulo, se llevó a cabo una estructura de clases en total correspondencia con las técnicas citadas y para realizar la implementación de los algoritmos planteados.

Se ha propiciado el paso a la siguiente fase con el establecimiento de los requisitos funcionales y no funcionales donde se realizaron las descripciones detalladas de los casos de uso a implementar. Todo estuvo apoyado en el Modelo de Casos de Uso del Sistema, lo cual permite a los futuros usuarios conseguir los resultados esperados por el cliente.

4

Capítulo 4 Implementación del Sistema

Introducción

Para brindar una visión detallada de cómo se hará la implementación en este Capítulo se darán a conocer las características de la STK así como, el estándar de codificación que usarán los usuarios y programadores.

Se describirán los diagramas de clases del diseño dividido por paquetes para un mejor entendimiento de las clases y sus relaciones. Se presentaran los diagramas de secuencias de los casos de uso. Se convertirá el diseño de clases anteriormente mostrado en componentes físicos, mediante el diagrama de componentes, los cuales se convierten en ficheros .cpp correspondientes a la implementación en C++.

4.1 Características de la Scene Toolkit (STK)

Es una herramienta brindada por el Proyecto de **HDSRV** y presenta como objetivo básico, agrupar las funcionalidades comunes a cualquier Sistema de Realidad Virtual, de manera que se les facilite el trabajo a los programadores de juegos y simuladores a través de la reutilización de código.

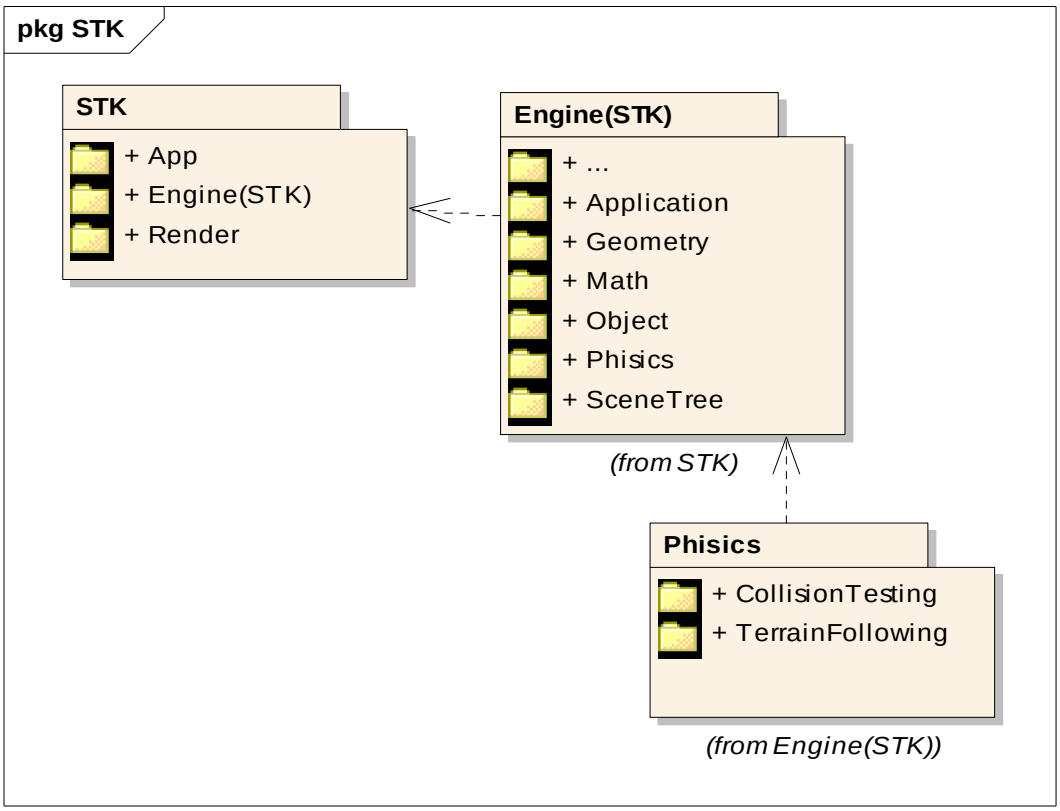

Fig. 4.1 SceneToolkit.

4.2 Estándares de codificación

El estándar de codificación es un patrón establecido para guiar a los programadores que pertenezcan a un proyecto de manera que los códigos usados sean entendidos por todos los que integran el mismo.

La HDSRV no está exenta de esto, pues también ha tenido que establecer un patrón para facilitar el entendimiento de la nomenclatura utilizada por cada uno de sus usuarios o de sus aportadores. Es una exigencia de los autores de la misma que cualquier módulo que se añada debe estar codificado siguiendo estos estándares.

Debido a que las palabras son simples se realizará el código en el idioma Inglés sin acentuación, respetando los estándares de codificación del lenguaje C++ como son el uso de espacios y líneas en blanco, la manera de identificación, etc.

Nombre de los ficheros:
Se nombrarán los ficheros .h y .cpp de la siguiente manera:
STKNameOfUnits.cpp
Se usará **STK** para identificar el nombre de la herramienta.

Constantes:
Las constantes se nombrarán con mayúsculas, utilizándose el "_" para separar las palabras: MY_CONST_ZERO = 0;

Tipos de datos:
Los tipos se nombrarán siguiendo el siguiente patrón:
Enumerados: enum **E**MyEnum {**ME**_VALUE, **ME**_OTHER_VALUE};
Indicando con "**E**" que es de tipo enumerado. Nótese que las primeras letras de las constantes de enumerados son las iniciales del nombre del enumerado. Véase otro ejemplo:
enum **E**Node**T**ype {**NT**_GEOMETRYNODE,…};
Estructuras: struct **S**MyStruct {…};
Indicando con "**S**" que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.
Clases: class **C**ClassName;

Indicando con "**C**" que es una clase

Declaración de variables:

Los nombres de las variables comenzarán 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 antepondrá el identificador "m_", si son globales se les antepondrá la letra "g", y en caso de ser argumentos de algún método, se les antepondrá el prefijo "arg_".

Tipos simples:

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
char **g**cGlobalVarName; //variable global, no se le antepone "m_"
short **s**Name;

Instancias de tipos creados:

EMyEnumerated **e**Name;
SMyStructure **k**Name;
CClassName **k**ObjectName;
CClassName* **pk**Name; //puntero a objeto
CClassName* **ak**Name; //arreglo de objetos
CClassName* **ak**Name; // variable miembro de clase
IMyInterface* **pI**Name; //puntero interfaces

Métodos:

En el caso de los métodos, se les antepondrá el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepondrá nada.
En el caso de los argumentos se les antepone el prefijo "arg_"

Constructor y destructor:

CClassName (bool **arg**_bVarName, float& **arg**_fVarName);

~**C**ClassName ();

Funciones:

bool bFunction1 (…);

int* **pi**Function2 (…);

CClassName* **pk**Function3 (…);

Procedimientos:

void Procedure4 (…);

Métodos de acceso a miembros:

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

int **i**MyVar; //variable

Obtención del valor:

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

Establecimiento del valor:

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

Obtención y establecimiento del valor:

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

4.3 Diseño del Sistema

Las clases del diseño fueron agrupadas en cinco paquetes fundamentales. De todos estos paquetes el SceneTree y el Math son paquetes propios de la STK los cuales mantienen sus relaciones entre clases sin modificación alguna.

Las clases correspondientes al Módulo Detección de Polígonos en Colisión se encuentran distribuidas entre el Collider (Colisionador), el Collision Bounding Hierarchy (Envolvente Jerárquico) y el Polygons Intersection (Intersección de Polígonos).

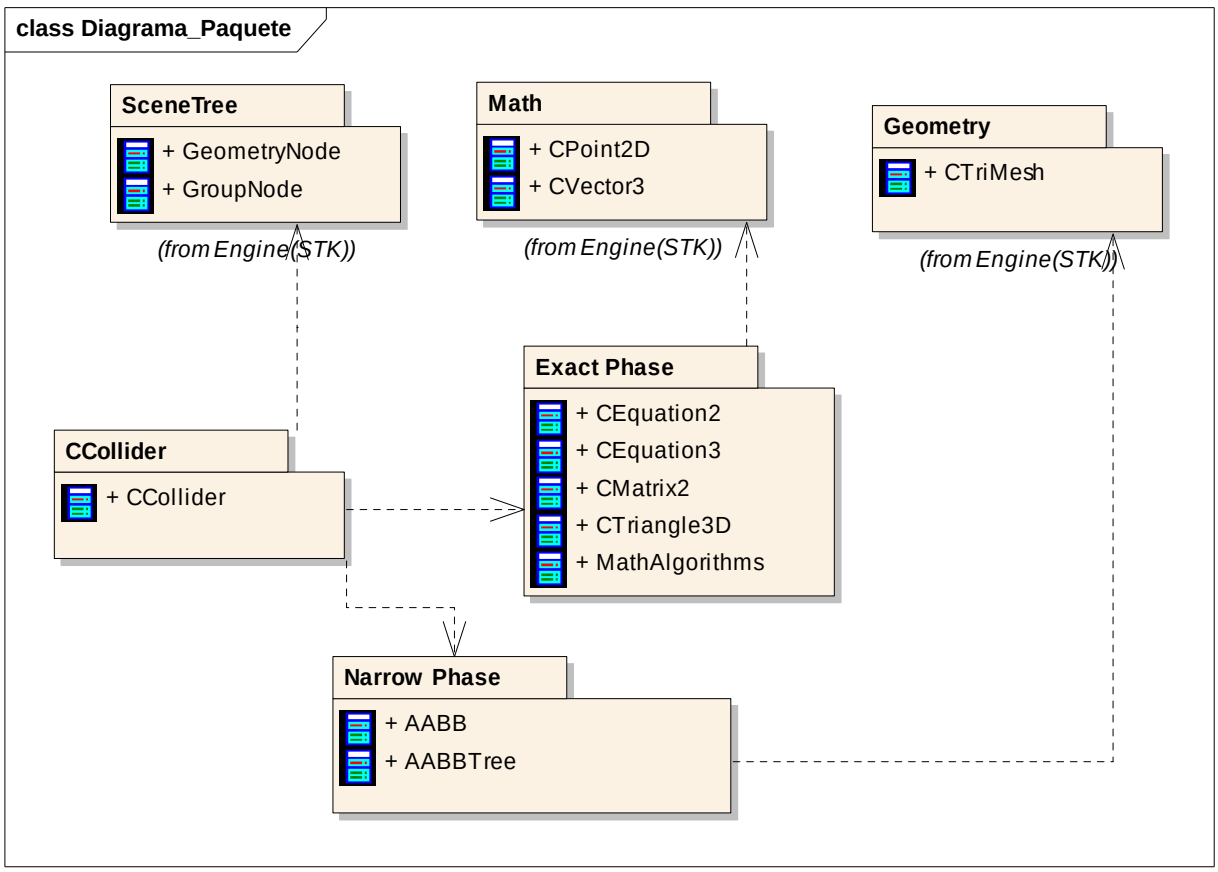

Fig. 4.2 Diagrama de Paquetes de Clases del Diseño.

4.3.1 Expansión de Paquetes

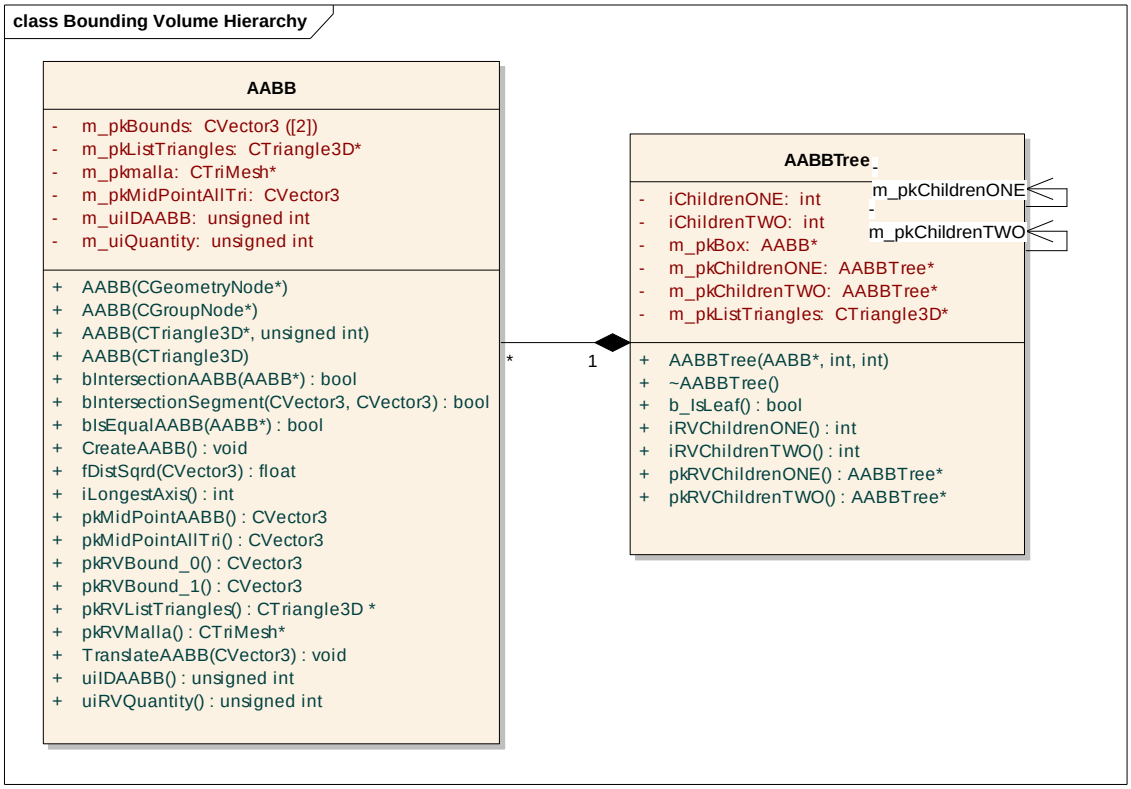

Fig. 4.3 Clases AABB y AABBTree.

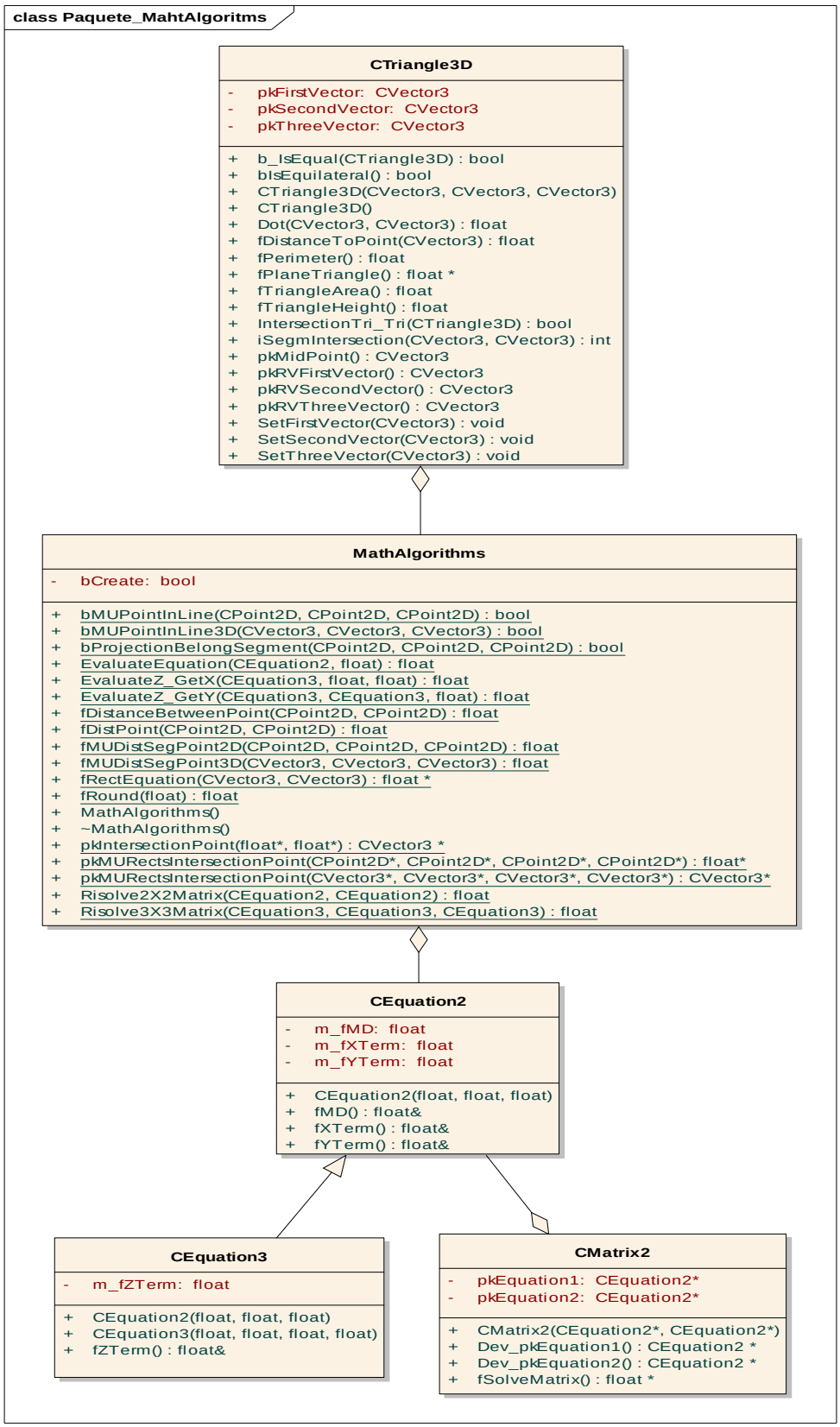

Fig. 4.4 Clases del paquete MathAlgoritms.

4.4 Diagrama de Diseño

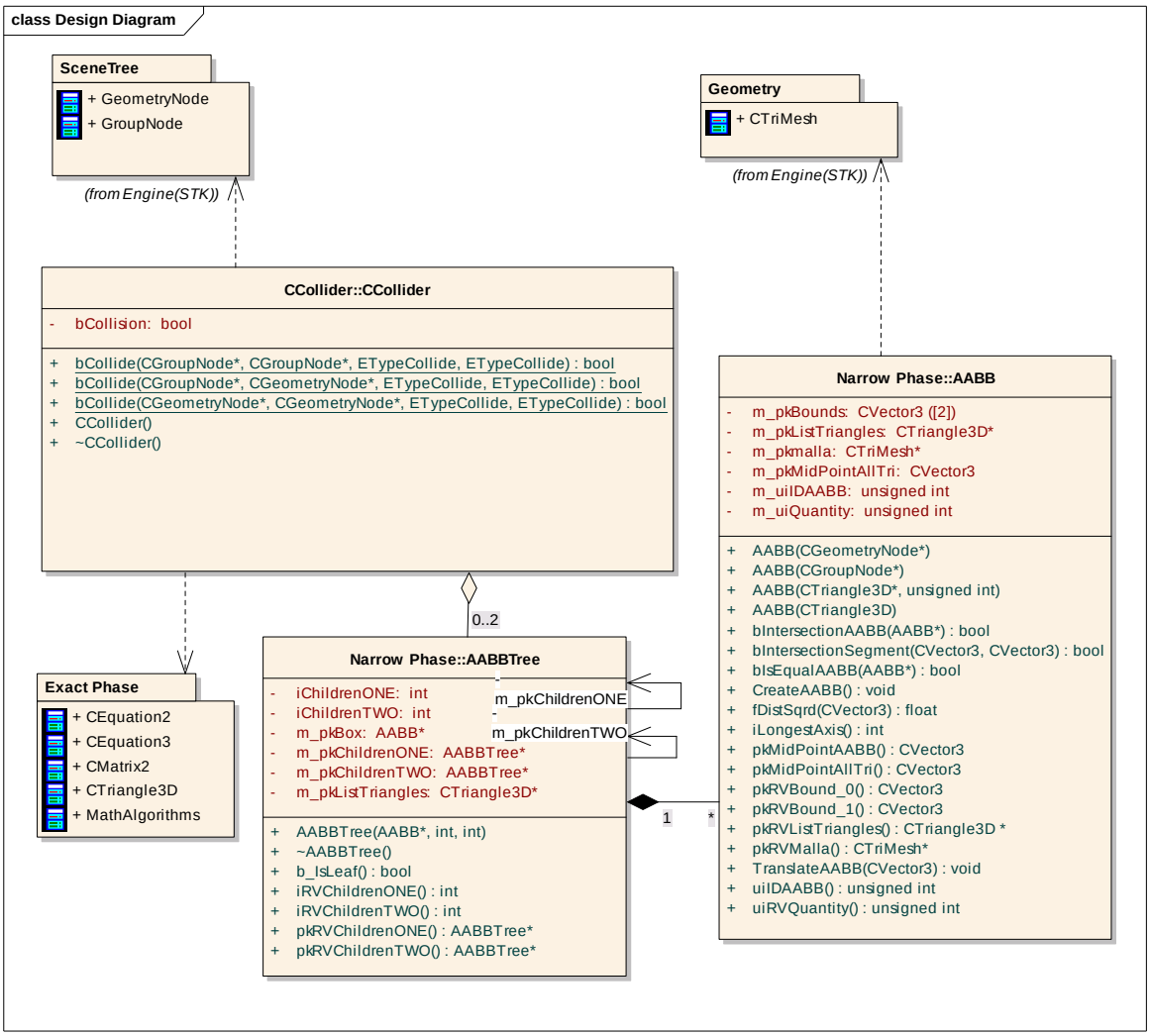

Fig. 4.5 Diagrama de clase del diseño.

4.4.1 Descripción de las Clases de Diseño

A continuación serán descritas las clases más importantes relacionadas a la implementación del Módulo sin detallar las clases ya implementadas por la STK.

Tabla 4.1 Descripción de la clase "Collider".

Nombre: Collider	
Tipo de clase: Controladora	
Atributo	Tipo
bCollision	bool
Para cada responsabilidad:	
Nombre:	CCollider()
Descripción:	Constructor de la clase
Nombre:	~CCollider()
Descripción:	Destructor de la clase
Nombre:	bCollide(CGroupNode* m_pkObj1, CGroupNode* m_pkObj2, ETypeCollide E_TYPE_OBJ1, ETypeCollide E_TYPE_OBJ2)
Descripción:	Función que se encarga de verificar si dos objetos de tipo CGroupNode colisionan
Nombre:	bCollide(CGroupNode* m_pkObj1, CGeometryNode* m_pkObj2, ETypeCollide E_TYPE_OBJ1, ETypeCollide E_TYPE_OBJ2)
Descripción:	Función que se encarga de verificar si dos objetos uno de tipo CGroupNode y el otro de CGeometryNode colisionan
Nombre:	bCollide(CGeometryNode* m_pkObj1, CGeometryNode* m_pkObj2, ETypeCollide E_TYPE_OBJ1, ETypeCollide E_TYPE_OBJ2);
Descripción:	Función que se encarga de verificar si dos objetos de tipo CGeometryNode colisionan

Tabla 4.2 Descripción de la clase "AABBtree".

Nombre: AABBtree	
Tipo de clase: Entidad	
Atributo	Tipo
iChildrenONE	int
iChildrenTWO	int
m_pkListTriangles	CTriangle3D *
m_pkChildrenONE	AABBTree *
m_pkChildrenTWO	AABBTree *
m_pkBox	AABB *
Para cada responsabilidad:	
Nombre:	AABBTree(AABB *arg_pkAABB, int arg_iDepth, int arg_maxDepth)
Descripción:	Construye un árbol binario de un AABB y una profundidad dado
Nombre:	~AABBTree()
Descripción:	Destructor del AABBtree
Nombre:	pkRVChildrenONE()
Descripción:	Retorna el AABBtree izquierdo del árbol
Nombre:	pkRVChildrenTWO()
Descripción:	Retorna el AABBtree derecho del árbol
Nombre:	iRVChildrenONE()
Descripción:	Retorna la cantidad de hijos izquierdo que tiene el árbol
Nombre:	iRVChildrenTWO()
Descripción:	Retorna la cantidad de hijos derecho que tiene el árbol
Nombre:	pkRVBox()
Descripción:	Retorna la raíz
Nombre:	b_IsLeaf()
Descripción:	Retorna si es hoja

Tabla 4.3 Descripción de la Clase "CAABB".

Nombre: CAABB	
Tipo de clase: Entidad	
Atributo	Tipo
m_pkBounds[2]	CVector3
m_uiQuantity	Int
m_pkListTriangles	CTriangle3D *
m_pkmalla	CTriMesh *
m_pkMidPointAllTri	CVector3
m_uiIDAABB	Int
Para cada responsabilidad:	
Nombre:	AABB(CGeometryNode* m_pkObj)
Descripción:	Función que se encarga de mandar un AABB en caso de que el objeto sea de tipo GeometryNode
Nombre:	AABB(CGroupNode* m_pkObj)
Descripción:	Función que se encarga de mandar un AABB en caso de que el objeto sea de tipo GroupNode
Nombre:	AABB(CTriangle3D *arg_pkListTriangles, unsigned int arg_uiQuantity)
Descripción:	Función que se encarga de mandar un AABB en caso de que se tenga la lista de triángulos que conforma el objeto y la cantidad.
Nombre:	AABB(CTriangle3D kTriangle);
Descripción:	Función que se encarga de mandar un AABB en caso de que se tenga un arreglo de triángulos que conforma el objeto
Nombre:	void CreateAABB()
Descripción:	Se encarga de crear el AABB
Nombre:	fDistSqrd(CVector3 arg_BoxMin)
Descripción:	Retorna la distancia entre un vector 3D y un AABB
Nombre:	iLongestAxis()
Descripción:	Retorna el eje mayor que presenta el AABB
Nombre:	TranslateAABB(CVector3 arg_pkVectTranslate)
Descripción:	Función que se encarga de traslada el AABB dado un vector de traslación

Nombre:	bIsEqualAABB(AABB *pkAABB)
Descripción:	Función que se encarga de calcular si dos AABBs son iguales
Nombre:	pkMidPointAABB()
Descripción:	Calcula el punto medio del AABB
Nombre:	pkMidPointAllTri()
Descripción:	Calcula el punto medio promedio de todos los triángulos que contiene el AABB
Nombre:	pkRVBound_0()
Descripción:	Retorna el punto mínimo del AABB
Nombre:	pkRVBound_1()
Descripción:	Retorna el punto máximo del AABB
Nombre:	pkRVListTriangles()
Descripción:	Retorna la lista de triangulo que engloba el AABB
Nombre:	uiRVQuantity()
Descripción:	Retorna la cantidad de triangulo que engloba el AABB
Nombre:	pkRVMalla()
Descripción:	Retorna la malla que engloba el AABB
Nombre:	uiIDAABB()
Descripción:	Retorna el identificador del AABB

Tabla 4.4 Descripción de la clase "CTriangle3D".

Nombre: CTriangle3D	
Tipo de clase: Entidad	
Atributo	Tipo
pkFirstVector	CVector3
pkSecondVector	CVector3
pkThreeVector	CVector3
Para cada responsabilidad:	
Nombre:	CTriangle3D(CVector3 arg_pkFirstVector, CVector3 arg_pkSecondVector, CVector3 arg_pkThreeVector)
Descripción:	Construye el triangulo dado los tres vectores que lo conforman
Nombre:	CTriangle3D()
Descripción:	Construye el triangulo por defecto
Nombre:	fTriangleArea ()
Descripción:	Función que se encarga de calcula el área del triangulo
Nombre:	fPerimeter ()
Descripción:	Función que se encarga de calcula el perímetro del triangulo
Nombre:	fTriangleHeight ()
Descripción:	Función que se encarga de calcula el peso del triangulo
Nombre:	bIsEquilateral ()
Descripción:	Función que se encarga de verificar si el triangulo es equilátero
Nombre:	fDistanceToPoint ()
Descripción:	Función que se encarga de calcula la distancia de un punto a un triangulo
Nombre:	b_IsEqual(CTriangle3D kTriangle)
Descripción:	Función que se encarga de verificar si dos triángulos son iguales
Nombre:	*fPlaneTriangle()
Descripción:	Función que se encarga de calcula el plano del triangulo
Nombre:	iSegmIntersection(CVector3 arg_pkVs, CVector3 arg_pkVe)
Descripción:	Función que se encarga de verificar si un segmento se intercepta con un triángulos
Nombre:	IntersectionTri_Tri(CTriangle3D arg_trianlge);
Descripción:	Función que se encarga de verificar si dos triángulos se

	interceptan
Nombre:	pkRVFirstVector ()
Descripción:	Función que se encarga de retornar el primer vector del triangulo
Nombre:	pkRVSecondVector ()
Descripción:	Función que se encarga de retornar el segundo vector del triangulo
Nombre:	pkRVThreeVector ()
Descripción:	Función que se encarga de retornar el tercer vector del triangulo
Nombre:	SetFirstVector(CVector3 arg_pkVectToSet)
Descripción:	Función que se encarga de actualizar el primer vector del triangulo
Nombre:	SetSecondVector(CVector3 arg_pkVectToSet)
Descripción:	Función que se encarga de actualizar el segundo vector del triangulo
Nombre:	SetThreeVector(CVector3 arg_pkVectToSet)
Descripción:	Función que se encarga de actualizar el tercer vector del triangulo

Tabla 4.5 Descripción de la clase "CMathAlgorithms".

Nombre: CMathAlgorithms	
Tipo de clase: Entidad	
Atributo	Tipo
bCreate	Bool
Para cada responsabilidad:	
Nombre:	MathAlgorithms()
Descripción:	Constructor de la clase
Nombre:	~MathAlgorithms()
Descripción:	Destructor de la clase
Nombre:	fRound(float num)
Descripción:	Redondea un numero a tres cifras significativas
Nombre:	fDistPoint(CPoint2D arg_pkPointONE, CPoint2D arg_pkPointTWO)
Descripción:	Método que se encarga de calcular la distancia entre dos puntos
Nombre:	fMUDistSegPoint3D(CVector3 arg_pkSegmentPointONE, CVector3 arg_pkSegmentPointTWO, CVector3 arg_pkPoint)
Descripción:	Método que se encarga de calcular la menor distancia entre un vector y un segmento
Nombre:	bMUPointInLine(CPoint2D arg_kPoint, CPoint2D arg_kRectPoint1, CPoint2D arg_kRectPoint2)
Descripción:	Método que se encarga de verificar si un punto está dentro de un segmento de recta 2D
Nombre:	bMUPointInLine3D(CVector3 arg_kPoint, CVector3 arg_kRectPoint1, CVector3 arg_kRectPoint2)
Descripción:	Método que se encarga de verificar si un punto está dentro de un segmento de recta 3D
Nombre:	* pkMURectsIntersectionPoint(CPoint2D *arg_kRectAPoint1, CPoint2D *arg_kRectAPoint2, CPoint2D *arg_kRectBPoint3, CPoint2D *arg_kRectBPoint4)
Descripción:	Método que se encarga de calcular el punto de intersección entre segmentos de rectas 2D
Nombre:	* pkMURectsIntersectionPoint(CPoint2D *arg_kRectAPoint1,

	CPoint2D *arg_kRectAPoint2, CPoint2D *arg_kRectBPoint3, CPoint2D *arg_kRectBPoint4)
Descripción:	Método que se encarga de calcular el punto de intersección entre segmentos de rectas 2D
Nombre:	* pkMURectsIntersectionPoint (CVector3 arg_kRectAPoint1, CVector3 *arg_kRectAPoint2, CVector3 *arg_kRectBPoint1, CVector3 *arg_kRectBPoint2)
Descripción:	Método que se encarga de calcular el punto de intersección entre segmentos de rectas 3D
Nombre:	bProjectionBelongSegment(CPoint2D arg_pkPoint1, CPoint2D arg_pkPoint2, CPoint2D arg_pkPoint)
Descripción:	Método que se encarga de verificar si la proyección un punto sobre un segmento 2D está dentro de dicho segmento 2D.
Nombre:	fMUDistSegPoint2D(CPoint2D arg_pkPoint1, CPoint2D arg_pkPoint2, CPoint2D arg_pkPoint)
Descripción:	Método que se encarga de calcular la menor distancia entre un punto 2D y un segmento de recta 2D
Nombre:	fDistanceBetweenPoint(CPoint2D arg_pkPoint1, CPoint2D arg_pkPoint2)
Descripción:	Método que se encarga de calcular la distancia entre dos puntos 2D
Nombre:	*fRectEquation(CVector3 arg_kPointONE, CVector3 arg_kPointTWO)
Descripción:	Método que se encarga de calcular la ecuación de la recta con dos puntos 3D
Nombre:	*pkIntersectionPoint(float *fRectEq, float *fPlaneEq)
Descripción:	Método que se encarga de calcular el punto de intersección entre segmentos de rectas 3D y un plano
Nombre:	Risolve2X2Matrix(CEquation2 arg_kEq1, CEquation2 arg_kEq2)
Descripción:	Método que se encarga calcular la variable “y” en un sistema de ecuación de 2 con 2
Nombre:	EvaluateEquation(CEquation2 arg_kEq2, float arg_fYEval)
Descripción:	Método que se encarga calcular la variable “x” en un sistema de ecuación de 2 con 2
Nombre:	Risolve3X3Matrix(CEquation3 arg_kEq1, CEquation3

	arg_kEq2, CEquation3 arg_kEq3)
Descripción:	Método que se encarga calcular la variable “z” en un sistema de ecuación de 3 con 3
Nombre:	EvaluateZ_GetY(CEquation3 arg_kEq1, CEquation3 arg_kEq2, float arg_fZEval)
Descripción:	Método que se encarga calcular la variable “y” en un sistema de ecuación de 3 con 3
Nombre:	EvaluateZ_GetX(CEquation3 arg_kEq1, float arg_fYEval, float arg_fZEval);
Descripción:	Método que se encarga calcular la variable “x” en un sistema de ecuación de 3 con 3

Tabla 4.6 Descripción de la clase “CEquation2”.

Nombre: CEquation2	
Tipo de clase: Entidad	
Atributo	Tipo
m_fXTerm	float
m_fYTerm	float
m_fMD	float
Para cada responsabilidad:	
Nombre:	CEquation2(float arg_fA, float arg_fB, float arg_fD)
Descripción:	Construye una ecuación de 2 incógnitas
Nombre:	& fXTerm()
Descripción:	Retorna el termino de la variable “x”
Nombre:	& fYTerm()
Descripción:	Retorna el termino de la variable “y”
Nombre:	& fMD()
Descripción:	Retorna el resultado de la ecuación

Tabla 4.7 Descripción de la clase "CEquation3".

Nombre:CEquation3		
Tipo de clase: Entidad		
Atributo		Tipo
m_fZTerm		float
Para cada responsabilidad:		
Nombre:	CEquation3(float arg_fA, float arg_fB, float arg_fC, float arg_fD)	
Descripción:	Construye una ecuación de 3 incógnitas	
Nombre:	& fZTerm()	
Descripción:	Retorna el termino de la variable "z"	

Tabla 4.8 Descripción de la clase "CMatrix2".

Nombre: CMatrix2	
Tipo de clase: Entidad	
Atributo	Tipo
pkEquation1	CEquation2 *
pkEquation2	CEquation2 *
Para cada responsabilidad:	
Nombre:	CMatrix2(CEquation2 *arg_pkEquation1, CEquation2 *arg_pkEquation2)
Descripción:	Construye una CMatrix2
Nombre:	*fSolveMatrix()
Descripción:	Retorna el termino de la variable "z"
Nombre:	fZTerm()
Descripción:	Retorna el termino de la variable "z"
Nombre:	*Dev_pkEquation1()
Descripción:	Retorna la ecuación número 1
Nombre:	*Dev_pkEquation2()
Descripción:	Retorna la ecuación número 2

4.5 Diagrama de secuencia

El Lenguaje Unificado de Modelado (UML), está provisto de un medio gráfico para representar la interacción entre los objetos a lo largo del tiempo mediante los diagramas de secuencia.

Los Diagramas de Secuencia muestran el flujo de mensajes de un objeto a otro, representando los métodos y los eventos soportados por un objeto o clase.

A continuación se muestran los Diagramas de Secuencias correspondientes a los casos de uso del Módulo de Detección de Polígonos en Colisión a implementar.

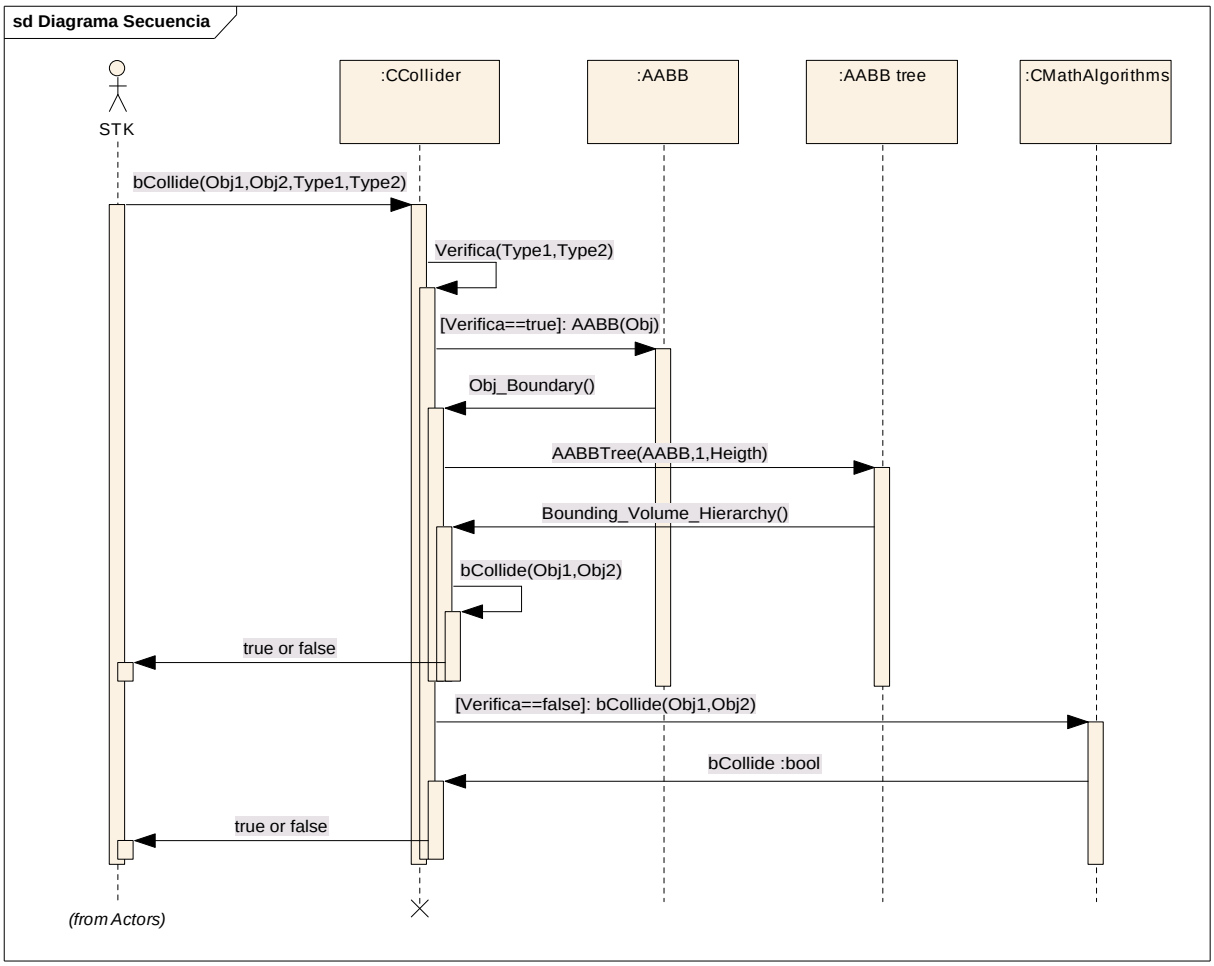

Fig. 4.6 Diagrama de secuencia "Chequear Colisión".

4.6 Diagrama de componentes

A continuación se muestra la relación entre los paquetes de componentes y la estrecha relación que existe entre las clases de la herramienta y el Módulo de Detección de colisiones.

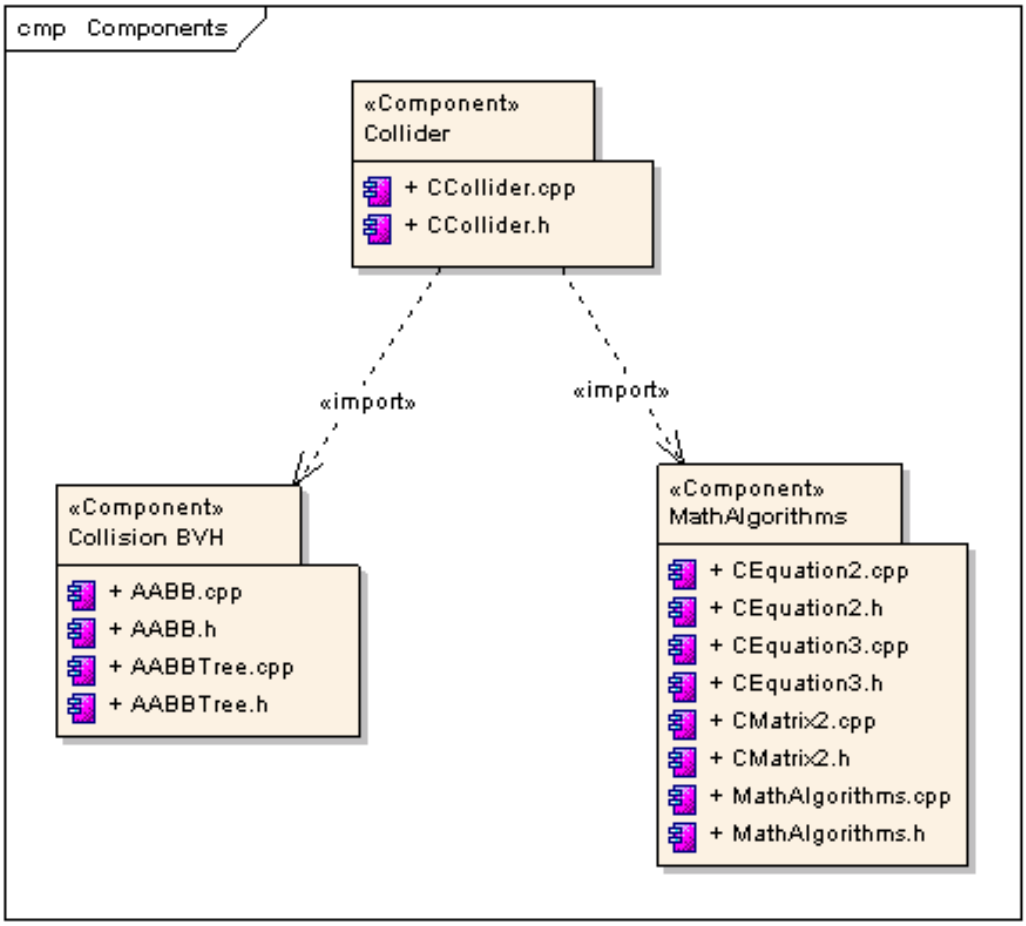

Fig. 4.7 Diagrama de relación entre paquetes de componentes.

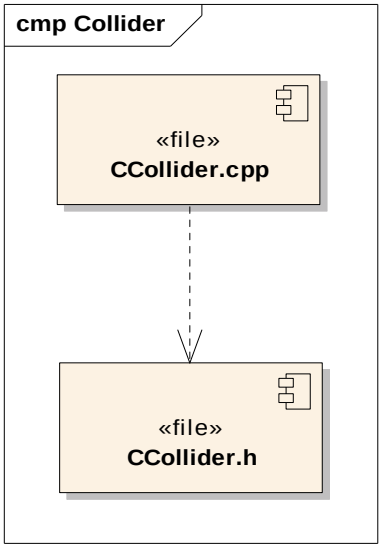

Fig. 4.8 Diagrama de componentes paquete Collider.

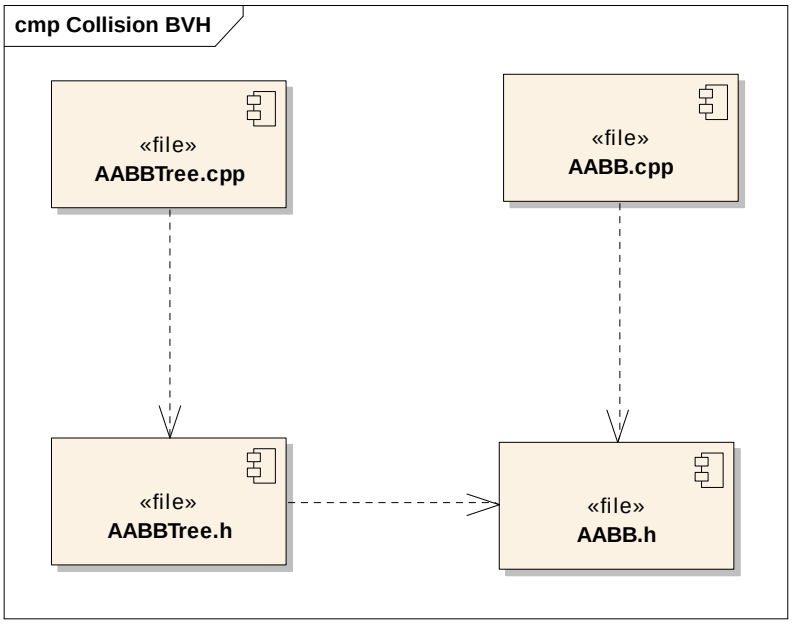

Fig. 4.9 Diagrama de componentes paquete Collision BVH.

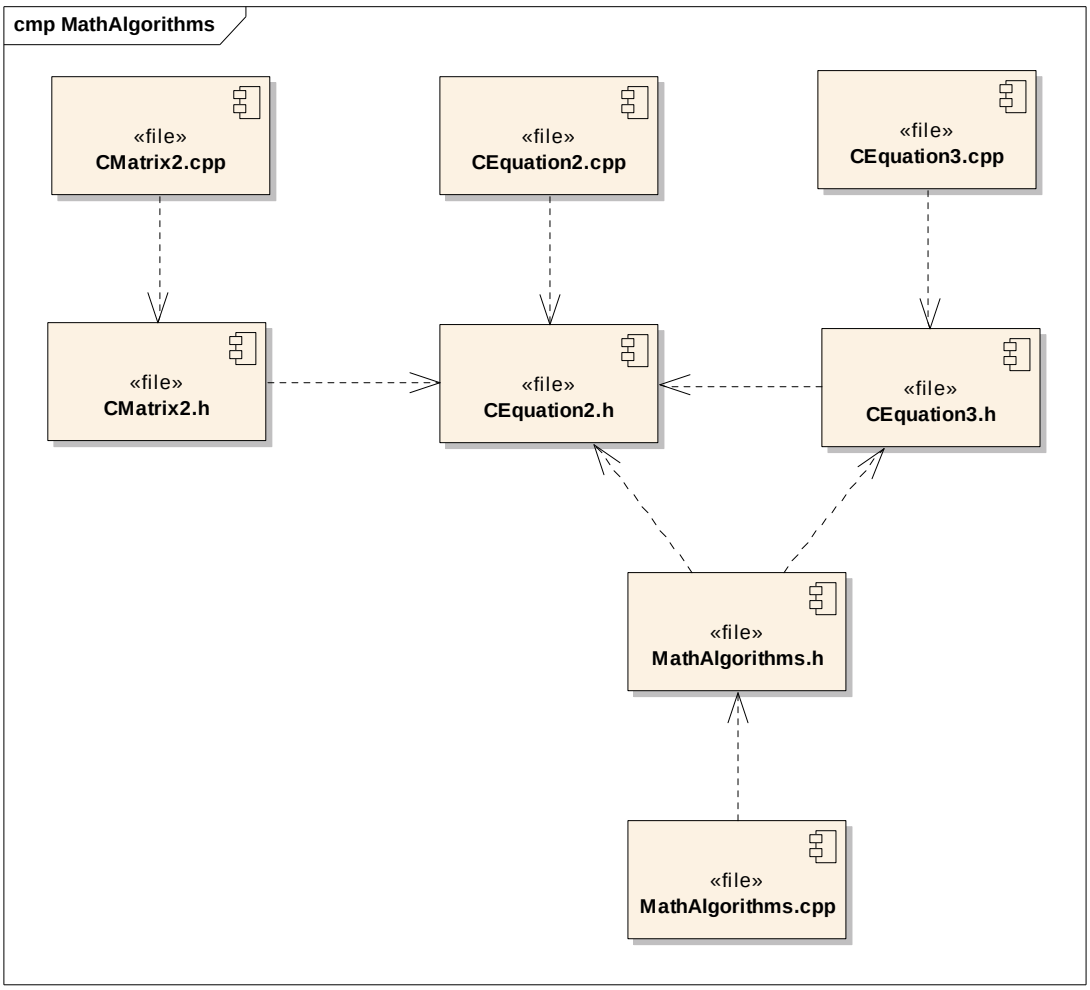

Fig. 4.10 Diagrama de componentes paquete MahtAlgorithms.

4.7 Resultados de implementación

Este epígrafe tiene como objetivo fundamental tomar casos de pruebas de creación del árbol binario AABB y de chequeos de colisión para obtener un tiempo resultante. El tiempo tomado será el peor tiempo ó tiempo crítico.

En la siguiente tabla se muestran los resultados obtenidos durante la creación del árbol AABB para una profundidad de 8 y para objetos conformados por malla de 1000 hasta 8000 polígonos.

Tiempo (ms)	Altura del árbol	Cantidad de polígonos
50	8	1000
250	8	8000
145	8	6000

Tabla 4.9 Resultados de creación del AABBTree

En la siguiente tabla se muestran los peores tiempos obtenidos durante las pruebas de colisión a las estructuras que presenta el Módulo donde el peor tiempo se evidencia en la detección de polígonos de colisión en el Malla/Malla con un tiempo aproximado de 1/8 de segundo.

Consultas de Colisión	Tiempo Crítico (milisegundos)	Altura del árbol	Cantidad de Polígonos
Árbol - Segmento	16	8	2300
Árbol - Esfera	16	8	2300
Árbol - Árbol	16	8	2470
Malla - Segmento	16	8	2300
Malla - Esfera	50	8	2300
Malla - Malla	126	10	5110

Tabla 4.10 Resultados de Consultas de Colisión.

Conclusiones

En este Capítulo se dio a conocer la conformación de un diseño completo del sistema, traducido en clases y sus relaciones, así como los diagramas de secuencias con sus mensajes, desarrollando las funcionalidades de los CU. Además, de quedar establecido el estándar a seguir para la programación de los casos de uso desarrollados y la relación entre los componentes a programar.

Conclusiones Generales

La revisión realizada sobre las técnicas, tecnologías y tendencias relacionadas con la Implementación del Módulo de Detección de Polígonos en Colisión, rindió los resultados esperados, permitiendo alcanzar los objetivos de este trabajo.

La técnica empleada para envolver las mallas triangulares, AABB conjuntamente con su estructura jerárquica (AABBTree), redujo el cálculo de las posibles áreas de colisiones y de la determinación de los polígonos de colisión eficientemente, logrando satisfacer los requisitos de la Realidad Virtual y cumpliendo con los parámetros de tiempo real. Dicho módulo se acopló eficientemente a la SceneToolkit manteniendo una flexibilidad y extensión total.

El Módulo funcional obtenido satisface los requisitos del cliente. Puede ser utilizado tanto en juegos como en simuladores para detectar colisiones. Este módulo integrado a la biblioteca SceneToolkit, propicia un mayor realismo en los sistemas de Realidad Virtual.

Recomendaciones

Por los resultados obtenidos se recomienda:

- Profundizar en el estudio de la primera fase de Colisión y su implementación para lograr un módulo más robusto.
- La implementación del método “hybrid” para recalcular un árbol de AABB en caso de deformación permitiría una mayor rapidez de existir colisión con objetos deformables.
- Estudio e implementación de la Coherencia Temporal y Espacial.
- Profundizar en la detección de colisiones mediante el GPU.

Referencias bibliográficas

Bibliografía consultada

1. **Delgado, Dr.Juan José Jiménez.** *Recubrimientos Simpliciales Detección de Colisiones mediante.* 2006.

2. **Eberly, David.** *Dynamic Collision Detection using Oriented Bounding Boxes.* 2002.

3. **Kavan, Ladislav, O'Sullivan, Carol and Zara, Jirı.** *Efficient Collision Detection for Spherical Blend Skinning.* 2005.

4. **Funfzig, Christoph and Fellner, Dieter W.** *Easy Realignment of k-DOP Bounding Volumes.* 2002.

5. **Gottschalk, S., D., M. C. Lin and Manocha.** *OBBTree: A Hierarchical Structure for Rapid Interference Detection.* 1996.

6. **Koziara, Tomasz and Bicanic, Nenad.** *Bounding box collision detection.* 2005.

7. **Hochgurtel, Stefan, et al.** *Technical Report TR-TI-2005-1- Collision Detection for k-DOPs using SAT with Error Bounded Fixed-Point Arithmetic.* 2005.

8. **Team Chicken Bomb.** *Collision and Interference Detection using Dynamic Hierarchical Bounding Volumes.* 2001.

9. **Zachmann, Gabriel.** *Rapid Collision Detection by Dynamically Aligned DOP-Trees.* 1996.

10. **Larssona, Thomas and Akenine-Moller, Tomas.** *A dynamic bounding volume hierarchy for generalized collision detection.* 2006.

11. **Tuft, David O.** *A SYSTEM FOR COLLISION DETECTION BETWEEN DEFORMABLE MODELS BUILT ON AXIS ALIGNED BOUNDING BOXES AND GPU BASED CULLING.* 2007.

12. **Larsson, Thomas and Akenine-Möllerz, Tomas.** *Collision Detection for Continuously Deforming Bodies.* 2001.

13. **Sundaraj, Kenneth.** *Real-Time Dynamic Simulation and 3D Interaction of Biological Tissue : Application to Medical Simulators.* 2004.

14. **Emilio, R. Escarpín.** *La realidad virtual, una tecnología educativa a nuestro alcance.* 2007.

15. **Inmaculada, Remolar.** *Aplicaciones de la Informática Gráfica.* 2006.

16. **Jhon, Friedich Walter and Stuart, Diesel Von.** *Virtual Reality, VPL Research.* 2007.

17. **Ayala, D., et al.** *Object representation by means of nonminimal division quadtrees and octrees.* 1985.

18. **Swan, J. E.** *II Octree-based collision detection with fast neighbor finding.* 1993.

19. **García-Alonso, A. and Serrano, N.** *Flaquer, J. Solving the Collision Detection Problem.* 1994.

20. **Held, M., Klosowski, J. and Mitchell, J.** *Evaluation of collision detection methods for virtual reality fly-throughs.* 1995.

21. **Naylor, B., Amanatides, J. and Thibault, W.** *Merging bsp trees yield polyhedral modeling results.* 1990.

22. **BERGEN, GINO VAN DEN.** *Efficient Collision Detection of Complex Deformable Models using AABB Trees.* 1998.

23. **Osorio, César Ignacio García and Antón, Luis Miguel Martín.** *Memoria del Proyecto Fin de Carrera Robot Recuperador Web SpiderBot 1.0 Ignacio Cruzado Nuño.* 2006.

24. **Akenine-Moller, Tomas and Larsson, Thomas.** *Strategies for Bounding Volume Hierarchy Updates for Ray Tracing of Deformable Models.* 2003.

Sitios Web

25. **Eberly, David.** http://www.cyberkreations.com. [Online] 2007.

26. **Gomez, Miguel.** http://www.gamasutra.com. *http://www.gamasutra.com.* [Online] 1999.

27. **Stokes, Conor.** http://www.flipcode.com. *http://www.flipcode.com.* [Online] 2002.

28. **wikipedia organization.** http://es.wikipedia.org/wiki/Tiempo_real. *http://es.wikipedia.org/wiki/Tiempo_real.* [Online] 2004.

29. **Depto. Informática y Electrónica. Universidad de Valencia.** http://informatica.uv.es/iiguia/AIG/web_teoria/tema6.pdf. *http://informatica.uv.es/iiguia/AIG/web_teoria/tema6.pdf.* [Online] 2006.º

30. **Sociedad Anonima de mexico.** http://www.galeon.com/ultimopunto/cat_tecno/tecno_vr.htm. *http://www.galeon.com/ultimopunto/cat_tecno/tecno_vr.htm.* [Online] 2001.

31. **Universidad de Argentina.** http://www.geocities.com/fabian.coello/Archivos . *http://www.geocities.com/fabian.coello/Archivos* . [Online] 1999.

32. **Bergen, Gino van den.** http://www.win.tue.nl/~gino/solid/. *http://www.win.tue.nl/~gino/solid/.* [Online] 1998.

33. **Terdiman, Pierre.** http://www.codercorner.com/Opcode.htm. *http://www.codercorner.com/Opcode.htm.* [Online] 2006.

34. **University of Freiburg, Germany.** http://coldet-free-3d-collision-detection-library.softonic.com/ColDet-Free3DCollision. *http://coldet-free-3d-collision-detection-library.softonic.com/ColDet-Free3DCollision.* [Online] 2003.º

35. **Herman.** http://www.vjuegos.org/modules.php?name=Content&pa=showpage&pid=4. *http://www.vjuegos.org/modules.php?name=Content&pa=showpage&pid=4.* [Online] 2006.

36. **Research Group on Modeling.** *http://www.cs.unc.edu/~geom/OBB/OBBT.html*. Retrieved from http://www.cs.unc.edu/~geom/OBB/OBBT.html. [Online] 1997.

37. **Research Group on Modeling.** *http://www.cs.unc.edu/~geom/V_COLLIDE/*. Retrieved from http://www.cs.unc.edu/~geom/V_COLLIDE/.[Online] 1998.

Apéndices

Glosario de abreviaturas

RV: Realidad Virtual

SRV: *Sistema de Realidad Virtual*

HDSRV: Herramientas de desarrollo para Sistemas de Realidad Virtual.

TR: Tiempo Real.

BSP: partición binaria del espacio.

BSPtree: Binary Space Partition tree.

BS: (Bounding sphere).Esferas de inclusión.

AABB: (Axis Aligned Bounding Boxes). Cajas Alineadas a los ejes de coordenadas.

OBB: (Oriented Bounding Boxes) Cajas orientadas al objeto.

DOP: Discrete Orientation Polytipe.

k-DOP: Un DOP construido por k planos.

POO: Programación Orientada a Objetos.

BVH: Bounding Volume Hierarchies.

Fps: Frames por segundos.

Ficheros 3DX:

CPU: Unidad Central de Procesamiento

VPL Research: (Visual Programming Language). Lenguaje Visual de Programación.

3D: Tres dimensiones

2D: Dos dimensiones

HW: Hardware.

Glosario de términos

-A-

Árboles de esferas: árbol en donde el nodo raíz es una esfera que envuelve completamente a todo el objeto.

Algoritmos: Conjunto finito de pasos o instrucciones que se deben seguir para realizar una determinada tarea.

Arista: Línea que resulta de la intersección de dos superficies, considerada por la parte exterior del ángulo que forman.

-C-

Convexa: Se dice de la línea o superficie curva cuya parte más prominente está del lado del que mira.

-D-

Detección de colisión: utilización de diferentes técnicas para la obtención de un choque efectuado por dos o más objetos de una escena virtual.

-E-

Escenas dinámicas: Escena donde todos los objetos del entorno se trasladan o rotan.

Estructura jerárquica: Cada nodo interno del árbol divide la región convexa asociada con el nodo en dos regiones, utilizando para ello un plano con orientación y posición arbitraria.

-F-

Frame*:* Ver cada una de las imágenes que componen una animación.

-I-

Informática gráfica: Es la creación, manipulación, análisis e interacción de las representaciones pictóricas de objetos y datos por medio de un ordenador.

-J-

Jerarquía espacial: Los objetos de la escena son agrupados jerárquicamente según las regiones en que ellos se encuentran.

-M-

Malla: Forma de representar un modelo a partir de polígonos. Colección de vértices, aristas y polígonos conectados de forma que cada arista es compartida como máximo por dos polígonos.

Métodos inmersivos de realidad virtual: Con frecuencia se ligan a un ambiente tridimensional creado por computadora el cual se manipula a través de cascos, guantes u otros dispositivos que capturan la posición y rotación de diferentes partes del cuerpo humano.

Modelos: Es un esquema teórico, habitualmente matemático, que representa el comportamiento y la evolución de un sistema definido mediante una serie de parámetros.

-O-

Objetos primitivos: objetos geométrico-matemáticos regulares y que no varían su forma en el tiempo como por ejemplo una esfera, una pirámide, un cubo, entre otros

Objetos complejos: objetos que tienen forma irregular y solamente pueden ser representados en el mundo de la informática grafica a través de sistemas triangulares en forma de mallas.

-P-

Pruebas de intersección geométrica: verificando si todos los polígonos que modelan la superficie de un objeto, intersectan a algún polígono del otro objeto, determinando así, si dos objetos chocan.

-R-

Realidad virtual no inmersiva: Utiliza medios como el que actualmente ofrece Internet en el cual se puede interactuar en tiempo real con diferentes personas en espacios y ambientes que en realidad no existen sin la necesidad de dispositivos adicionales a la computadora.

-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: Una simulación es la ejecución (habitualmente computerizada) de un modelo que reproduce el comportamiento de un sistema sometido a unas condiciones predeterminadas, posiblemente cambiantes en el tiempo.

-T-

Tiempo real: Se procesa cada señal digital ofrecida en un ambiente virtual antes de mostrarse la siguiente señal. Instantánea relación entre el soporte físico (hardware) y la programática (software) de la computadora.

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

-V-

Vértices: Punto en el que se unen los lados de un ángulo o las caras de un poliedro. Punto de una curva en el que la curvatura es máxima o mínima. Extremo o punto más alto de una cosa.

Índice de figuras y tablas

Índice de figuras

Índice de Tablas