Universidad de las Ciencias Informáticas

FACULTAD 5

ENTORNOS VIRTUALES

Algoritmo para detectar Colisiones en un Mundo Virtual

Trabajo de Diploma para optar por el título de
Ingeniero en ciencias Informáticas

Autor

Rolan Fuentes Socorro

Tutor

Ing. Yusleidy Guelmes León

Ciudad de la Habana, Cuba

Julio 2007

DECLARACIÓN DE AUTORÍA

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

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

Rolan Fuentes Socorro Ing. **Yusleidy Guelmes León**

_____ _____

Firma del Autor Firma del Tutor

DATOS DE CONTACTO

Ing. Yusleidy Guelmes León

Graduada en Ingeniería Informática de la CUJAE y la Universidad de Cienfuegos, en Julio del 2005, desde su graduación se desempeña como profesora en la disciplina Ingeniería y Gestión de Software de la Facultad 5 de la UCI.

yguelmes@uci.cu

AGRADECIMIENTOS

Agradecer a mis padres y hermana, por apoyarme y confiar en mí.

A mis amigos y familiares que de una forma u otra se preocuparon por mi desarrollo como profesional.

A mi tutora por guiarme y apoyarme durante el proceso de desarrollo del trabajo.

A todos aquellos que de una forma u otro me ayudaron desinteresadamente en el desarrollo del trabajo.

DEDICATORIA

A mis padres, Rolando Fuentes Lantigua y Marlenis Socorro Rodríguez.

Mi hermana Marlen Fuentes Socorro.

RESUMEN

En la facultad 5 de la Universidad de las Ciencias Informáticas en conjunto con la empresa SIMPRO se desarrollan una serie de simuladores para el entrenamiento de los estudiantes en escuelas militares de nuestro país. En dichos simuladores como en otros softwares (juegos virtuales, etc.) no se cuenta con un algoritmo que permita el manejo de colisiones entre los objetos que interactúan en el mundo virtual, por lo que es objetivo de este trabajo encaminar la investigación en ese sentido, abordando los principales conceptos para entender el tema de colisiones desde el punto de vista físico y computacional, realizando una breve descripción y valoración en cuanto a ventajas y desventajas de varios modelos y técnicas existentes para resolver computacionalmente el tema de las colisiones. Se propone un algoritmo que permite detectar colisiones entre dos objetos en un instante de tiempo basado en una representación jerárquica de esferas a través del Octree. Para probar dicho algoritmo se diseñó una sencilla aplicación que permite verificar con datos de entrada la respuesta si los objetos colisionan. El algoritmo propuesto puede ser utilizado o adaptado a cualquier sistema que necesite hacer un tratamiento de colisiones y es independiente del formato o la estructura que tengan los objetos que interactúan. Se programó en el lenguaje C++ con entorno de desarrollo C++ Builder, versión 6.0.

Capítulo 3: Implementación y Prueba

Se brinda una explicación del funcionamiento del algoritmo, además se realizan pruebas para dos objetos cualesquiera partiendo que estos tengan su representación como árbol de esfera, para chequear colisión en un instante de tiempo.

PALABRAS CLAVE

Algoritmo, colisiones, Octree, árboles de esferas, volúmenes inclusión.

TABLA DE CONTENIDOS

INTRODUCCIÓN

En nuestro país contamos con poca experiencia en el desarrollo de simuladores ya que esta área de la informática es característica de los países desarrollados. Nuestro país se esta introduciendo poco a poco en esta rama de la informática.

Nuestra Universidad ha establecido relaciones con la empresa SIMPRO perteneciente al MINFAR, dicha institución se dedica al desarrollo de simuladores virtuales, con el objetivo de utilizarlos en los entrenamientos de las escuelas militares, debido a que el costo de estas prácticas a veces asciende a cientos de miles de dólares por lo caro del equipamiento y las municiones. Con el empleo de simuladores se disminuirían los gastos y no hay riesgo de vidas humanas. La escuela en coordinación con esta institución ha establecido un local donde se está desarrollando el simulador virtual para realizar las prácticas de tiro.

Situación Problémica

En nuestra universidad y específicamente en la facultad 5 en coordinación con la empresa Simpro se tiene previsto el desarrollo de una serie de simuladores para el entrenamiento en las escuelas militares de todo el país y perfeccionar los ya existentes. En dichos simuladores se presenta una problemática similar, no se manejan las colisiones entre los objetos, o esto se hace de manera muy rústica; por lo general un objeto en movimiento no coincide con cualquiera otro en el mundo virtual en el que se mueve, lo que le quita realismo al entorno virtual y por tanto afecta la eficacia del entrenamiento.

En el mundo se han desarrollado gran número de modelos y algoritmos para la detección de colisiones entre objetos pero seleccionar el ideal para cada caso en que se vaya a emplear requiere de un proceso de estudio y valoración, así como su adaptación a las condiciones concretas del sistema informático y las herramientas con que se desarrolla.

Por lo tanto para realizar la investigación en dicho tema se plantea como **Problema Científico** la inexistencia de un algoritmo implementado para resolver las colisiones entre objetos del mundo virtual de los simuladores que se desarrollan en nuestra universidad.

Objeto de Estudio

Simulación Virtual.

Objetivo

Implementar un algoritmo que pueda ser utilizado en la detección de colisiones entre objetos en un mundo virtual para los simuladores que se desarrollan en nuestra universidad.

Campo de Acción

Algoritmos y técnicas para detección de colisiones entre objetos en un entorno virtual.

Idea a defender

El desarrollo de un algoritmo que permita tratar las colisiones entre objetos virtuales dotará a los simuladores que se desarrollan en nuestra universidad de mayor realismo y calidad para su explotación.

Tareas de investigación

- Analizar y Sintetizar bibliografía actualizada sobre el tema de colisiones.
- Estudiar y valorar los modelos, técnicas o algoritmos existentes más utilizados en el manejo de colisiones.
- Determinar un modelo para el manejo de colisiones en los simuladores que se encuentran en desarrollo en la universidad.
- Implementar un algoritmo que permita determinar las colisiones de acuerdo al modelo seleccionado.
- Probar el algoritmo.

El presente trabajo consta de tres capítulos:

Capítulo 1: Fundamentación Teórica

Se abordan los principales conceptos para entender el tema de colisiones desde el punto de vista físico y computacional, se realiza una breve descripción y valoración en cuanto a ventajas y desventajas de varios modelos y técnicas existentes para resolver computacionalmente el tema de las colisiones, seleccionando el que se considera más apropiado.

Capítulo 2: Descripción de la Técnica propuesta para la solución

Se da una explicación de cómo se puede formar un árbol de esferas a través del Octree, con el objetivo de un mejor entendimiento del funcionamiento del algoritmo.

Capítulo 3: Implementación y Prueba

Se brinda una explicación del funcionamiento del algoritmo, además se realizan pruebas para dos objetos cualesquiera partiendo que estos tengan su representación como árbol de esfera, para chequear colisión en un instante de tiempo.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

El tema colisiones puede resultar de cierta forma complejo, por lo cual se considera necesario hacer referencia a ciertos conocimientos previos que sirva de base para adentrarse en el tema, a través de los conceptos que a continuación se exponen.

Un **simulador** es un dispositivo que permite la simulación de un sistema, reproduciendo su comportamiento como si fuera real. Los simuladores reproducen sensaciones que en realidad no están sucediendo. Pretende reproducir tanto las sensaciones físicas (velocidad, aceleración, percepción del entorno) como el comportamiento de los equipos de la máquina que se pretende simular [Wikipedia, 2007]. Donde las colisiones juegan un rol fundamental brindando mayor realismo, permitiendo mayor interacción entre los cuerpos al determinar que pudiera pasar luego de la colisión.

Existen varios tipos de simuladores que ayudan a la formación y/o entrenamiento de técnicos, especialistas, ingenieros etc. Dentro de ellos encontramos:

Simulador de Carretera: es uno de los más populares, permite conducir autos, motocicletas, camiones etc. Ejemplo: rFactor, GTR, GT Legends.

Simulador de Vuelo: permite dominar el mundo de la aviación y pilotar aviones, helicópteros… Ejemplos: Microsoft Flight Simulator.

Simulador de Tiro: permite la ejecución de tiro, ya sea de infantería, armamento pesado etc.

- **Relámpago - CITEFA:** permite la ejecución de tiro con armas antitanque, en espacios reducidos.
- **FATS - Firearms:** permite el tiro con armas portátiles, entrenar a varios tiradores simultáneamente.

Los simuladores tienden por lo general a tener elevados costos para su producción, entre más sofisticados sean y más se acerquen a la realidad, aumentará más su coste.

Ventajas

- Son los medios de ayuda más adecuados para alcanzar altos niveles de instrucción.
- Disminuyen los costos en cuestión de práctica real.
- Contribuyen a la preservación del medio ambiente.

- Reducen los riesgos de accidentes y sus consecuencias.
- Los sistemas de tipo salón, permiten el adiestramiento y/o entrenamiento en espacios reducidos.
- Los sistemas de tipo duelo, ofrecen un mayor realismo en el entrenamiento y/o adiestramiento.

Desventajas

- No reemplaza a la ejecución de ejercitaciones en el terreno o la práctica laboral.
- No reemplaza a la ejecución de ejercitaciones de tiro con munición de guerra, por más sofisticado y perfecto que sea el sistema de simulación empleado.
- Nunca reemplaza a la realidad, por más que la simulación logre excelencia.
- Crea una atmósfera lúdica e ilusoria, inclusive cuando se utilizan los sistemas tipo duelo. [Ponzi]

En los simuladores de entornos reales como los que se desarrollan en nuestra universidad generalmente hay que hacer un fuerte trabajo para reflejar la interacción física entre los objetos visibles, por ejemplo los choques, que también en varios contextos se les denomina colisiones o intersecciones.

1.1 Tipos de colisiones

Una **colisión** ocurre cuando dos o más cuerpos se encuentran en contacto. Existen dos tipos de colisiones, la elástica e inelástica.

En una **colisión elástica** los objetos rebotan uno de otro sin pérdida en el total de la energía cinética del sistema. El total de la energía cinética antes y después de la colisión es la misma. Un ejemplo casi perfecto de una colisión elástica es el choque de unas bolas de billar.

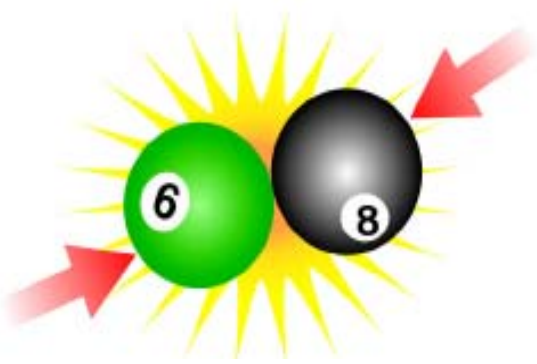


Fig.1.

En una **colisión inelástica** los objetos cambian de figura, es decir se deforman, o quedan unidos, y el total de la energía cinética del sistema disminuye. La energía no se destruye, solamente se transforma en otra forma diferente a la energía cinética, por ejemplo un cambio permanente en la forma. Un huevo al estrellarse en el piso es un ejemplo de colisión inelástica. Otro es dos vehículos chocando. En ambos casos parte de la energía cinética causa cambios en las figuras de los objetos. [Franco G, 2006]

Fig.2.

Fig.3.

El tema de las colisiones es muy tratado en el mundo de la computación y la informática, para lo cual se han desarrollado múltiples herramientas, técnicas y concepto.

1.1.1 Detección de colisiones a través del tiempo

Si se conoce la posición de los cuerpos a través del tiempo, puede calcularse el punto de contacto y el momento en que dicho contacto se dio, de forma exacta o con una gran precisión. Pero el costo computacional regularmente resulta ser excesivamente alto. Si la geometría de los objetos y su movimiento son complejos, la detección de colisiones exacta puede resultar inviable aún para un conjunto pequeño de objetos. Por ello existen otras aproximaciones menos precisas pero más baratas computacionalmente.

Se puede emplear como base un sistema discreto que proporcione la posición de los objetos en instantes dados de tiempo y estimar la posición de los cuerpos entre esos instantes. Esto puede aún ser costoso si los objetos presentan variaciones en su velocidad durante el período y si rotan, lo que puede simplificarse considerando que entre cada muestra el objeto se mueve con velocidad uniforme y sin rotaciones. Incluso

puede calcularse el sólido que se forma por extrusión de los objetos desde una posición, hasta la posición siguiente y calcular si existe contacto entre los sólidos formados, este enfoque es más simple debido a que no toma en consideración los efectos de la velocidad, pero igualmente las simplificaciones pueden conducir a un sistema excesivamente impreciso, dependiendo de la aplicación en la que se le use.

1.1.2 Detección de colisiones en un instante de tiempo

El análisis sobre detección de colisiones se emplea para determinar si dos objetos se encuentran en contacto en un instante dado de tiempo, es decir que no importan sus posiciones anteriores o la dirección en la que se estén moviendo. Este es un enfoque típico en sistemas discretos, donde cada cierto tiempo se muestrea la posición de todos los objetos y se determina si alguno de ellos se encuentra en contacto y se realiza alguna acción en consecuencia. Sin embargo, este enfoque aproximado puede conducir a errores importantes como una colisión no detectada. Inicialmente el problema puede resolverse aumentando la frecuencia de muestreo del sistema, sin embargo es posible que aún a frecuencias altas existan colisiones no detectadas. [Moctezuma 2006]

1.2 Detección de colisiones

La **detección de colisiones** es un conjunto de algoritmos matemáticos que nos permiten simular el comportamiento de los objetos en la realidad [Varela C]. Reporta de manera automática una intersección entre dos objetos cualesquiera, así como el momento y el lugar exacto donde se dio dicho contacto. El problema de la detección de colisiones aparece en múltiples aplicaciones tales como el diseño asistido por computador, juegos, la robótica y la automática, la computación grafica, la animación y la realidad virtual. La importancia de la detección de colisiones es que permite el diseño basado en la simulación, análisis en la ingeniería, planificación de movimientos en la robótica, animación basada en modelos físicos, navegación en modelos virtuales, proporciona mayor realismo a las aplicaciones, mayor interacción entre los objetos, posibilitando dar respuestas ante la detección, evita el efecto fantasma, el cual se produce cuando dos objetos (que aparentan ser sólidos) se atraviesan como si no se produjese contacto entre ellos etc.

Diferentes tipos de subproblemas aparecen en la detección de colisiones. El más general es sencillamente determinar si dos objetos se tocan; ó si los objetos están compuestos por partes, cuales son las partes que se tocan. A veces necesitamos saber cuál es la distancia de separación entre dos objetos, otras

veces, cuando los objetos están en movimiento y conocemos su localización en el espacio, nos preguntamos si va haber una colisión entre ellos.

La detección de colisiones forma parte esencial de cualquier algoritmo de planificación de movimiento de objetos. En este caso la detección de colisiones sirve para verificar si hay ó no colisiones entre los objetos del entorno. En la planificación del movimiento de un objeto y en muchas otras aplicaciones, la detección de colisiones es uno de los componentes que más tiempo consume. Ello es debido a lo costoso que resulta verificar intersecciones entre objetos de geometría arbitraria. El costo computacional de un algoritmo de detección de colisión depende de varios factores (la forma de los objetos, las pruebas y las veces que se aplican), aunque el equilibrio entre rapidez y robustez dependerán de las exigencias de la aplicación. [Giraldo Orozco]

Cuando se prueban un conjunto de objetos por colisiones entre ellos, es necesario probar cada uno contra todos los demás, tarea que tiene una complejidad computacional O (n^2), por ello los sistemas de detección de colisiones buscan simplificar cada una de las pruebas que se llevan a cabo y disminuir el número de pruebas realizadas. Para este fin la detección de colisiones es dividida en tres etapas, la lejana, cercana y respuesta.

La etapa lejana consiste en una serie de operaciones sencillas que eliminan prematuramente la posibilidad de que dos objetos estén en contacto. Algunas técnicas utilizadas son:

- Dividir el espacio en celdas, Octree, BSP Tree etc.: permiten eliminar rápidamente la posibilidad de colisión entre objetos si estos no se encuentran en el mismo cuadrante.
- Trazar un rayo en dirección de movimiento del objeto: para probar si se encuentra en camino de colisión hacia otros objetos.

La etapa cercana consiste en realizar los cálculos para determinar si existe colisión entre un par de objetos y posible respuesta a esta. Aquí podemos encontrar:

- Fuerza Bruta.
- Volúmenes de limitación.

  [Moctezuma 2006]

La etapa de respuesta consiste en determinar posibles interacciones (movimientos, deformaciones…) de los objetos un vez que se comprueba la etapa cercana.

Existen una gran variedad de técnicas en las ciencias de la computación que han sido propuestas para abordar el problema de la detección de colisiones. Dentro de ellas figura:

Las estructuras de representación

Volúmenes de inclusión, dentro de los que encontramos:

- Esferas.
- Cilindros.
- Cajas.

Representaciones jerarquizadas de volúmenes envolventes.

- Árboles de Esferas.
- Árboles de Cajas.

Partición del espacio

- Octree.
- BSP Tree.

Técnicas de intercepción

- Test múltiple de intersecciones.
- Intersección de volúmenes barridos.
- Test de intersección tetradimensional.

A continuación se abordarán algunas de estas técnicas con más detalle.

Fuerza Bruta (brute force).

Consiste en comprobar todos los pares de polígonos posibles para determinar si hay o no colisión entre ellos (Fig.4). Es posible detectar tanto las colisiones entre objetos distintos como las de un objeto consigo

mismo. Requiere de una gran cantidad de cálculo, incrementando le costo computacional. [Muñoz Moreno, et al.]

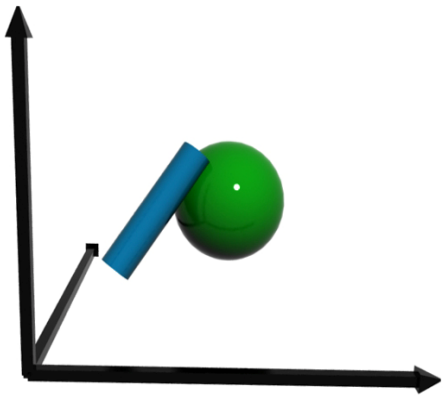


Fig.4.

1.3 Volúmenes de inclusión (Boundary volumes)

Se definen volúmenes con geometría simple que encierra al objeto de interés (Fig. 5). Debe ajustarse lo más estrechamente posible al modelo. El coste de cómputo de un volumen de inclusión debe ser bajo. Debe poder ser representado con una cantidad relativamente pequeña de memoria. Luego se determina si hay colisión entre los volúmenes. En el caso en que la haya, se aplica el algoritmo de fuerza bruta a los polígonos contenidos en el volumen. Es muy rápido y sencillo discriminar cuándo dos objetos no entran en colisión. No permite un ajuste fuerte para ciertos objetos. [Muñoz, et al.] [Roa López]

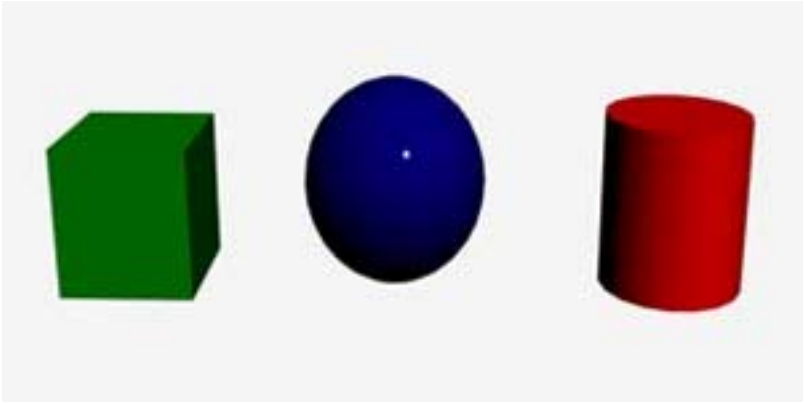


Fig.5.

1.3.1 Esferas de inclusión

Los objetos se inscriben en esferas ajustadas (Fig.6). La intersección entre dos esferas es muy sencilla y rápida de calcular, simplemente debe compararse la distancia entre los centros con la suma de los radios de ambas esferas y si la distancia es mayor que la suma de los radios no existe contacto entre ellas.

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 \leq (r_1 + r_2)^2$$

La esfera delimitadora puede calcularse de varias formas. Dado un conjunto de puntos que forman un objeto, hacer que el centro de la esfera sea la media de todos los puntos, y el radio la distancia desde éste hasta el vértice más alejado del objeto. Sin embargo, la esfera resultante no es óptima. En ocasiones se puede encontrar una esfera delimitadora con un volumen menor. El algoritmo que obtiene la esfera óptima se basa en un trabajo de Emo Welzl 1991 [E, 1991]; tiene tiempo esperado lineal, pero en el peor caso tiene un rendimiento polinomial, por lo que el cálculo no debería realizarse durante la ejecución de aplicaciones en tiempo real, sino tenerlas precalculadas. La invarianza de la esfera frente a rotaciones hace que sea rápido y sencillo el proceso de actualización de las mismas cuando el objeto que representan se mueve; por lo que la mayoría de los movimientos a analizar son traslacionales. Una de las dificultades de la esfera es que cuando los objetos tienen formas complejas, esta técnica produce muchos falsos positivos, es decir, aunque haya intersección entre estas no existe tal entre los objetos. [Gómez, et al., 2002] [Muñoz, et al.] [Pérez F, 1995]

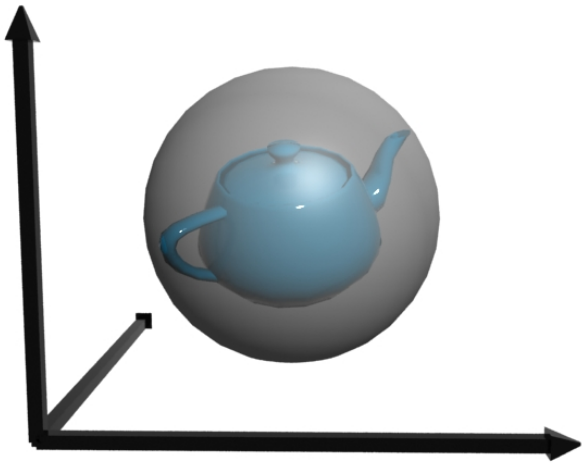


Fig.6.

1.3.2 Cajas alineadas a los ejes (AABB, Axis Aligned Bounding Boxes)

Los volúmenes contenedores son cubos alineados con los ejes de coordenadas (Fig.7 y 8). La detección de colisiones entre cajas es relativamente sencillo, se hace mediante el algoritmo propuesto por Cohen [J.D, et al., 1995)]: existe intersección si y sólo si las proyecciones de las cajas sobre los ejes de coordenadas se solapan. Su almacenamiento es más compacto, guardando únicamente dos vértices con los valores mínimo y máximo, se pueden averiguar las coordenadas de los 8 vértices que lo componen. Resulta ineficiente cuando los objetos tienen formas complejas o están orientados arbitrariamente, por lo que el ajuste no es fuerte. Es necesario recalcular la caja contenedoras cuando se produce alguna deformación. No es recomendable aplicar la rotación porque los dos vértices mínimo y máximo obtenidos luego del movimiento no pueden asegurar que el AABB obtenido delimite al objeto, por lo que se tendría que recalcular la caja. Por lo que es recomendable solo usarlos para objetos estáticos o que los movimientos sean de traslación o escalados. [Gómez, et al., 2002] [Muñoz, et al.]

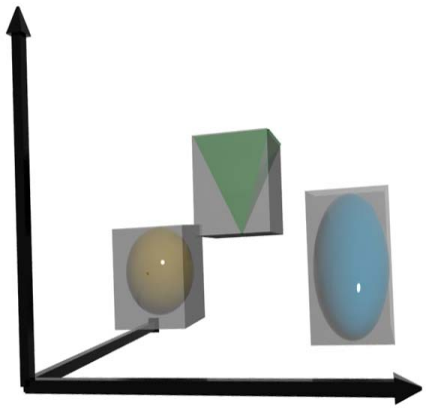


Fig.7.

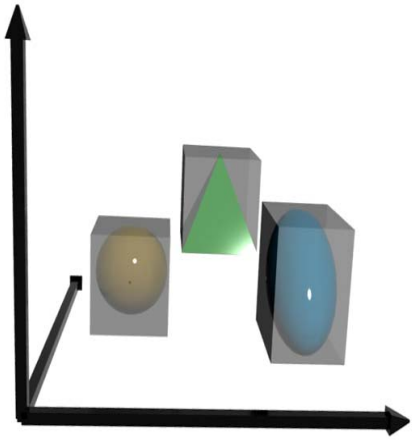


Fig.8.

1.3.3 Cajas orientadas (OBB, Oriented Bounding Boxes)

Las cajas contenedoras se orientan según los objetos (Fig.9). El cálculo de la intersección entre cajas se hace aplicando el teorema de los ejes de separación [S, et al., 1996]; en el teorema se define eje de separación entre dos objetos convexos como aquella recta tal que las proyecciones ortogonales de los objetos sobre ella no se solapan. Resulta más eficiente que AABB cuando los objetos están orientados arbitrariamente. Tiene un mejor ajuste en comparación con las esferas y AABB, no se ven afectados por traslaciones, escalados o rotaciones de los objetos, es decir, el OBB no necesita ser recalculado. Lo único que se necesita es aplicar la misma transformación al sistema de referencia del volumen y a la longitud de sus aristas. La detección de colisiones es más compleja que en AABB y por lo tanto su coste computacional es mayor, su construcción resulta más costosa, por lo que se recomienda obtenerlo al principio del programa, cuando se carga cada uno de los objetos, o directamente se cargan desde el fichero donde están los modelos. [Gómez, et al., 2002] [Muñoz, et al.]

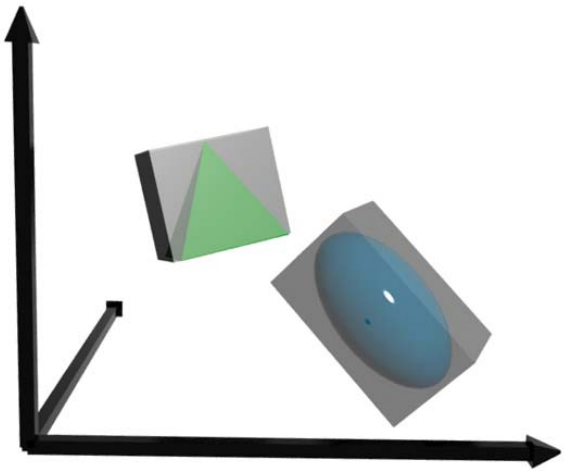


Fig.9.

1.4 Representaciones jerarquizadas de volúmenes envolventes

Se define un volumen que engloba a todo el objeto, se pasa a niveles inferiores colocando más volúmenes que se van a aproximando al cuerpo en si. Una vez terminado queda conformado lo que llamaremos un árbol. En las hojas e hijos que serian los nodos del árbol, queda almacenada la información del cuerpo. Los volúmenes de los hijos pueden solaparse. Se hace con el objetivo de disminuir los errores por colisión que pueden ocurrir cuando se tiene una inclusión del objeto. Cuando se hacen pruebas por colisión se pueden eliminar posibles áreas donde no exista posibilidad de colisión disminuyendo el coste computacional. El problema se reduce a sucesivas comprobaciones entre volúmenes sencillos. La detección de colisiones se reduce a búsquedas en árbol, en las que el tiempo de ejecución generalmente es del orden O (Log N), donde N es el número de nodos. Da respuestas rápidas de los test de intersección. Aumenta el número de pruebas por colisión debido a que hay una gran cantidad de nodos hijos. Si se usan cajas, la actualización de estas en cada nivel de la jerarquía puede tener un alto coste computacional cuando se producen deformaciones del objeto. La construcción de los árboles pueden tener altos costes. [Muñoz, et al.]

1.4.1 Representación jerárquica de objetos mediante esferas

Se utilizan esferas como volúmenes envolventes (Fig.10). Para su construcción las técnicas mas usadas son la jerarquía mediante Octree (se explicará más adelante) y medial-axis surfaces [P.M, 1996]. Esta ultima se basa en calcular los ejes que recorren al objeto, considerando como eje a una superficie que es equidistante a dos caras del objeto. Las esferas son colocadas en función de los ejes con diferentes niveles de precisión para conseguir la jerarquía. Proporciona más ajuste pero obtener los ejes es muy costoso, por lo que en realidad se calcula una aproximación. Por lo tanto es más costoso hacer su construcción debido a la complejidad de los cálculos. Los cálculos por intersecciones es muy sencillo, pero la aproximación al objeto puede ser poco ajustada si el nivel de profundidad es pequeño. [Gómez, et al., 2002]

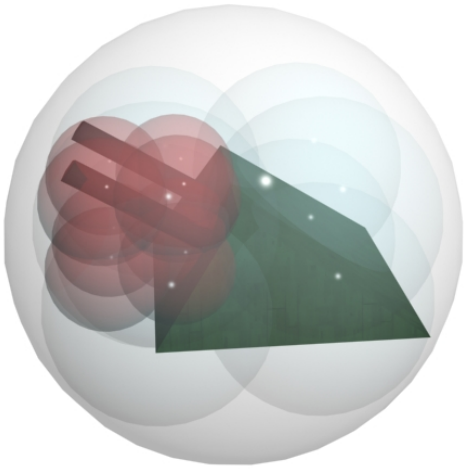


Fig.10.

1.4.2 Representación jerárquica de objetos mediante cajas orientadas (CO, OBB).

La construcción jerárquica de OBB de un objeto es dividiendo cada OBB en dos a través de su eje más largo (Fig.11). En cada paso, se calcula el OBB asociado a los polígonos que quedan en cada una de las mitades del OBB original. Permite ajustarse a la forma del objeto en pocos pasos. La construcción de la estructura es compleja. [Gómez, et al., 2002] [Muñoz, et al.]

Fig. 11.

1.4.3 Representación jerárquica de objetos mediante cajas alineadas a los ejes (CAE, AABB).

Se dividen los volúmenes AABB resultantes en cada paso a lo largo del mayor de los lados (Fig.12). 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. Al no tener en cuenta la orientación del objeto, pueden ser necesarias muchas subdivisiones para ajustarse a éste. [Gómez, et al., 2002]

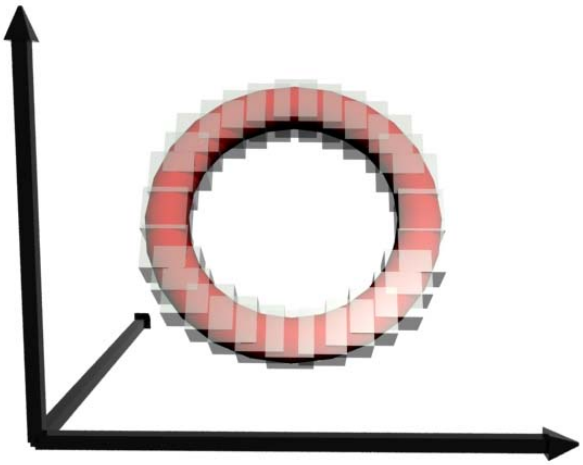


Fig. 12.

1.5 Técnicas de subdivisión del espacio

Se divide el espacio en cajas o por planos, se almacena en una lista la información de cada elemento de la partición. Puesto que se estructura el espacio en polígonos, el algoritmo de detección de colisiones es rápido. De hecho, el tiempo de ejecución teóricamente es constante. Además, no es necesario recalcular las cajas o planos cuando el objeto se deforma. Requiere gran cantidad de memoria para almacenar la tabla de la partición.

1.5.1 Octree

Es una representación de árboles octales, consiste en circunscribir el objeto o al mundo completo mediante un volumen (un AABB concretamente) y dividir éste en ocho volúmenes iguales (Fig. 13 y 14), que a su vez se dividirán en otros ocho que serán los hijos; este proceso se hace recursivamente permitiendo alcanzar cierto límite inferior [H, et al., 1998]. Se eliminan grandes áreas de testeo rápidamente y si un nodo no colisiona con otro, sus hijos tampoco lo harán. El modelo soporta operaciones cerradas como traslaciones, rotaciones y operaciones booleanas, la construcción de la estructura de volúmenes es sencilla, pero la aproximación cuando se aplica a un objeto puede que no tenga un ajuste fuerte.

Para el caso que un objeto tenga una trayectoria móvil en el mundo y existan otros almacenados en las hojas de la jerarquía, si este alcanza:

- Una celda vacía, entonces no hay colisión.
- Una celda llena, entonces hay colisión.
- Una celda con parte vacía y parte llena, entonces puede que sí o puede que no haya colisión, habría que averiguarlo.

[Muñoz, et al.] [Ortega, 2005]

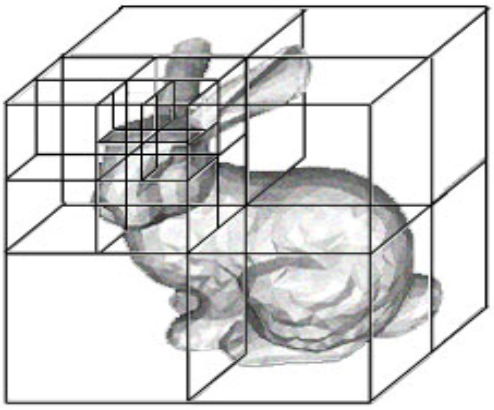


Fig. 13.

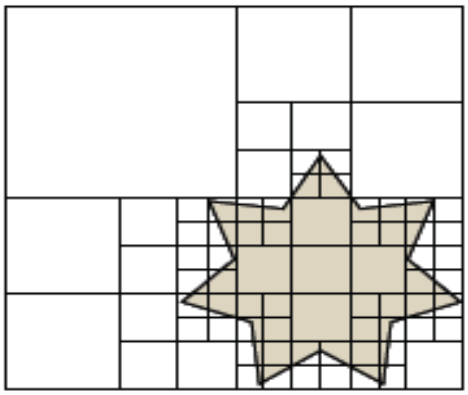


Fig. 14.

1.5.2 BSP Trees

En un árbol de particionamiento binario (BSP) (Fig. 15, 16 y 17), se divide el espacio en semiespacios, en cada nodo se utiliza un plano de particionamiento libremente orientado, ya sea a lo ejes o a los polígonos. Las hojas de dicho árbol representan a un objeto, mientras que los nodos no-hojas representan planos-ejes separadores. Un semiplano divide la subdivisión representada por el padre en dos partes, una a la izquierda y otra a la derecha (considerando el semieje con base en el semieje padre). Un caso crítico es cuando el semiplano separador intersecta un objeto de la escena, entonces:

- Se duplica colocándose tanto a izquierda como derecha del eje/plano.

Inconvenientes: Es menos costoso construir el árbol aunque lo será las pruebas por colisión.

- Se divide colocándose a izquierda y derecha cada uno de los trozos.

Inconvenientes: Más costoso construir el árbol, sobre todo para objetos complejos, aunque mejora las pruebas por colisión.

[Ortega, 2005] [Roa López]

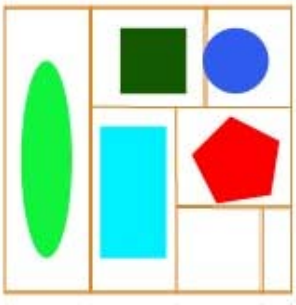


Fig. 15.

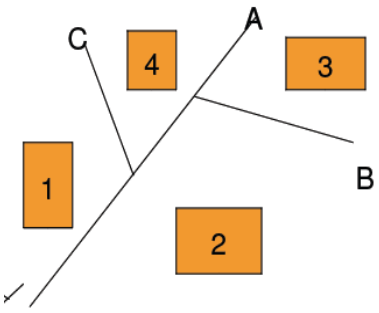


Fig. 16.

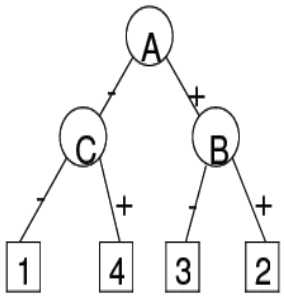


Fig. 17.

1.6 Técnicas de intersección

Las técnicas de intercepción son utilizadas para detectar colisiones. Es un grupo de pasos que tienen como meta optimizar la realización de pruebas por intersección, es decir, saber cuando es más óptimo realizar las pruebas, en que intervalos de tiempo o los posibles objetos que van a ser probados por colisión; reduciendo costos computacionales y tiempo en la simulación.

Se pueden presentar distintos problemas de complejidad atendiendo al número de objetos que van a ser probados por colisión, ejemplo:

- El caso más simple es aquel en el que sólo intervienen dos objetos.
- Se tiene cierto conjunto de objetos que desea probar si hay colisión entre ellos.
- El caso en el que existen varios conjuntos de objetos con un mismo color de manera que deseamos estudiar la intersección o colisión entre objetos de distinto color.

A continuación se detallan algunos de estos métodos.

1.6.1 Test múltiple de intersecciones

Este método consiste en reducir el problema de la detección de colisiones durante un movimiento a comprobar la ausencia de intersecciones geométricas para ciertos instantes {ti} $\in [t_0, t_f]$ del intervalo total de tiempo. Evidentemente, para que este método tenga alguna validez se ha de garantizar que el intervalo de tiempo Δti entre dos instantes de muestreo consecutivos es lo suficientemente pequeño para que no se

produzca una colisión entre las posiciones en ti+1 y ti que pasaría desapercibida. Por otro lado, si este ∆ti es demasiado pequeño el algoritmo sufre de un excesivo tiempo de computación.

Es por tanto necesario fijar cual debe ser el valor de ∆ti Existen diferentes estudios para hallar cual debe ser este valor, así Meyer [J, 1981] propone dividir el intervalo a estudiar en n partes iguales fijando el valor de n adecuadamente. Por otra parte Cameron [S, 1985], Culley y Kempf [R. K, et al., 1986] determinan el valor de ∆ti tomando una cota inferior de la distancia entre los objetos y una cota superior de sus velocidades relativas. [Pérez F, 1995]

1.6.2 Intersección de volúmenes barridos.

Este método consiste en generar el volumen barrido por un sólido tridimensional durante su movimiento. A cada objeto **B** que interviene le asociamos un nuevo sólido tridimensional **SVB** definido por su volumen barrido en algún momento del intervalo de tiempo considerado.

$$SVB = \{ F(t)B ; t \in [t_0, t_f] \};$$

Entonces, es condición suficiente para la ausencia de colisión entre dos objetos **B** y **C** en movimiento que no exista intersección entre sus correspondientes volúmenes barridos ***SVB*** ∩ ***SVC*** **=** ∅**.** Esta simplificación del problema, lógicamente, se consigue a costa de introducir algunas complicaciones adicionales. En primer lugar resulta en general difícil obtener una representación explícita del volumen barrido; en segundo lugar la existencia de intersección entre los volúmenes barridos por dos objetos no es una condición suficiente de colisión entre los mismos, pues evidentemente los objetos podrían haber ocupado el mismo espacio en instantes diferentes de tiempo, sin embargo la información temporal necesaria se ha perdido, pues no aparece en el proceso de generación de los volúmenes de barrido.

Para resolver el problema de obtener explícitamente el volumen barrido se han propuesto diversas soluciones:

- Restringir el conjunto de los sólidos que pueden intervenir, así como sus movimientos, de manera que podamos representar fácilmente el volumen barrido. Tal es el caso del sistema LARS [A, et al., 1983] que maneja esferas que pueden moverse siguiendo trayectorias rectilíneas o arcos de circunferencia, con lo cual generan cilindros y toros respectivamente.

- Trabajar únicamente con una representación implícita del volumen barrido; así, por ejemplo, Boyse [J.W, 1979] presenta un método en el que detecta colisiones entre poliedros generando un barrido de cada elemento vértices, aristas y caras individualmente para después estudiar por separado las posibles intersecciones entre los barridos de estos elementos, sin intentar en ningún momento construir una representación explícita conjunta del volumen total barrido.
- Realizar aproximaciones convexas del volumen barrido y, únicamente cuando se produce intersección de los volúmenes barridos globales, se procede a subdividir la trayectoria y a realizar una aproximación convexa, que usualmente resultará más fina, del volumen barrido en cada tramo Foisy y Haybard [V, et al., 1993]. [Pérez F, 1995]

1.6.3 Test de intersección tetradimensional.

Este método plantea, quizá, el enfoque formalmente más atractivo, resolviendo el problema desde un punto de vista diferente, pués en lugar de trabajar en el espacio ordinario tridimensional añade una dimensión más, el tiempo para reducir la detección de colisiones en un espacio con cuatro dimensiones, las tres coordenadas espaciales y el tiempo.

Así, para cada objeto real definimos un conjunto de puntos en ese espacio tetradimensional que nos sirve para describir perfectamente el movimiento del objeto, ya que estará formado por las distintas posiciones que han ocupado los puntos del objeto para cada instante de tiempo $t \in [t_0, t_f]$. Es evidente que una condición necesaria y suficiente para la existencia de colisión entre dos objetos es que la intersección entre sus correspondientes conjuntos tetradimensionales sea no nula. En efecto, si dos puntos se superponen en el espacio de cuatro dimensiones tendrán sus cuatro coordenadas (x, y, z, t) iguales, lo cual significa que los objetos originales han compartido una cierta porción del espacio en el mismo tiempo, es decir han colisionado. La operación que permite obtener un conjunto tetradimensional a partir de un sólido **B** y de su movimiento, dado por la función de localización *F* (*t*), se conoce con el nombre de *extrusión*, de manera que la extrusión de **B** viene dada por

$$EXB = \{ (Q, t); Q \in F(t)B , t \in [t_0, t_f] \};$$

y la condición suficiente de colisión entre dos objetos **B** y **C** en movimiento es $\boldsymbol{EXB} \cap \boldsymbol{EXC} \neq \varnothing$. Aunque este marco formal resulta en principio atractivo, sus aplicaciones prácticas presentan lógicas dificultades,

pués en la definición anterior no se ha establecido cómo se construye la extrusión de un objeto a partir de su función de localización. Esta dificultad, unida al hecho de que las técnicas matemáticas necesarias son poco conocidas, han motivado que el test de intersección tetradimensional haya sido comparativamente poco usado frente a los métodos anteriores. [Pérez F, 1995]

1.7 Trazado de rayos.

Otra técnica que es utilizada en muchos algoritmos para la detección de colisiones es el trazado de rayos. Un rayo (usando una representación vectorial) se representa mediante un vector que denota el comienzo y un vector (usualmente normalizado) que muestra la dirección hacia donde se dirige el rayo. Es un objeto especial que nos puede ser de utilidad para saber si dos objetos se encuentran en rumbo de colisión y permite calcular la distancia entre estos. Esencialmente, un rayo empieza en el punto de inicio y sigue la dirección del vector. La ecuación del rayo es:

PointOnRay= Raystart + t*Raydirection.

Donde t es un float que toma valores del intervalo [0, infinito). Con 0 se obtiene el punto de inicio y sustituyendo los otros valores se consiguen los puntos correspondientes a lo largo del rayo. PointOnRay, Raystart, Raydirection son vectores 3D con valores (x, y, z). Se puede usar esta representación del rayo y calcular intersecciones con planos, cilindros, esferas cajas etc. [Sánchez]

1.8 Paquetes de colisión.

Para el tratamiento de las colisiones se han creado una variedad de librerías que están formadas por varias herramientas y algoritmos que le permiten detectar colisiones, entre ellas figuran:

1.8.1 SWIFT

SWIFT proporciona datos de proximidad tales como: detección de la intersección, cómputo exacto y cálculo de distancia usando una jerarquía de modelos multirresolución de los objetos implicados, junto a la utilización de un algoritmo derivado en las regiones de Voronoi, hace que la librería tenga gran velocidad de cómputo [S.A, et al., 2000], determinación del contacto de los objetos tridimensionales que experimenta el movimiento rígido. Los objetos permitidos son poliedros convexos u objetos compuestos construidos de pedazos convexos (aunque la versión SWIFT++ sí que soporta modelos poliédricos no-convexos).

Hace uso de cajas de inclusión para una primera aproximación en el cálculo de las colisiones. Esta librería es más rápida que otras librerías como I-Collide y V-Clip y además, presenta buenas prestaciones incluso cuando no hay coherencia temporal.

Se basa en tres fases:

- Descomposición del poliedro en superficies convexas.
- Creación de una jerarquía, donde cada nodo de la jerarquía es una malla convexa que acota a sus hijos.
- Chequeo de colisión entre cada par de hojas con el algoritmo de Lin y Canny. [M, et al., 1991]

1.8.2 I-COLLIDE

I-COLLIDE trabaja solamente para los modelos que son poliedros convexos. Aprovecha las características especiales de poliedros convexos para determinar con rapidez y exactitud las colisiones en un entorno de N elementos en moviendo rígido (rotación y traslación). También explota coherencia temporal para conseguir una mayor eficiencia, de modo que el tiempo de búsqueda de colisiones sean extremadamente rápidos cuando los modelos se mueven solamente en una cantidad relativamente pequeña entre los marcos.

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

- Determinar los pares de objetos que se pueden solapar en la escena. Para ello emplea intersección de cajas de inclusión AABB. El resultado de este paso es una lista con los pares de objetos de la escena que pueden solaparse.
- Determinar cuáles son las primitivas (puntos, líneas o caras) de cada uno de los objetos de la pareja más cercana. Para ello, la librería tiene implementado el algoritmo de Lin-Canny [M, et al., 1991]. Aunque este algoritmo es sólo para objetos convexos, I-Collide lo adapta mediante el uso de un árbol jerárquico, de tal manera que los objetos no convexos los tratará como un árbol donde todos los elementos pueden ser convexos o no convexos excepto en el último nivel que tienen que ser convexos.

El paquete SWIFT proporciona la misma funcionalidad que I-COLLIDE y más. Es también más rápido y más robusto.

1.8.3 V-CLIP

EL algoritmo de V-CLIP (Voronoi Clip) es un algoritmo para la detección de colisiones entre objetos poliédricos convexos. [B, 1998]

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

1.8.4 RAPID

De los paquetes, RAPID es el más pequeño y la más fácil de utilizar. Trabaja con grupo de polígonos, los modelos poligonales no requieren ninguna estructura topológica particular como formación de un acoplamiento o de un objeto cerrado. Implementada en C++ y utiliza el método de colisión desarrollado por Gottschalk, Manocha y Lin [S, et al., 1996].

El algoritmo es capaz de determinar de manera eficiente qué polígonos de dichos objetos se solapan. Para la búsqueda de la colisión RAPID precalcula árboles jerárquicos de cajas orientadas (cajas con alineamiento no paralelo OBB) de los dos objetos de la escena bajo estudio. Posteriormente estudia el solapamiento de cada uno de los volúmenes de inclusión. El algoritmo presentado es capaz de calcular todos los contactos entre geometrías muy complejas permitiendo interactividad.

1.8.5 V-COLLIDE

V-Collide [V, et al., 1993] fue desarrollada por el grupo GAMMA (Geometric algorithm for modeling, motion and animation) de la universidad de California del Norte (UNC) implementada en C++. Determina cuáles de los objetos en movimiento en una escena virtual están en contacto potencial, realiza una detección rápida y exacta de colisiones entre modelos poligonales triangulados. La librería está diseñada para operar en ambientes que contienen un gran número de objetos geométricos formados con mallas de triángulos. La arquitectura de V-Collide, al igual que en el caso del I-Collide, se divide en dos niveles:

- En el primer nivel se determina qué pares de objetos se solapan en la escena. Para ello se emplea el mismo método que en el caso del I-Collide, la aproximación por cajas de inclusión AABB.

- En el segundo nivel se estudia la colisión de los pares de objetos seleccionados en la fase anterior. Para ello, el sistema calcula el árbol jerárquico de cajas de inclusión orientadas (OBB) mediante el mismo algoritmo que en RAPID. El sistema primero determina cuál de las cajas de inclusión del último nivel están en contacto y posteriormente estudia el solapamiento de los triángulos correspondientes.

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

[Muñoz, et al.] [Gamma]

Estas librerías son muy eficientes y resuelven una gama amplia de colisiones entre objetos, permitiendo detectarlas con niveles altos de precisión. Sin embargo la mayoría de estas requieren de grandes costos computacionales y cálculos matemáticos, muchas no tienen disponibilidad para su uso, y cuando lo tienen son versiones más antiguas que se publican permitiendo su uso solo con fines educativos, por lo que tenemos que tener en cuenta los problemas de las licencias, compatibilidad con la herramienta que esta desarrollada en nuestro centro para las simulaciones. Cuando la herramienta es de carácter propietario, una vez adquirida se tiene que estar al tanto de nuevas actualizaciones, pagos constantes de licencia y si se desea extender sus funcionalidades, se tendría que poner en contacto con la firma para hacer la solicitud, lo cual implicaría nuevos costos. Por lo que es más factible implementar nuestro propio algoritmo, teniendo el código fuente lo que facilita realizar nuevas modificaciones o ser rehusable en otras aplicaciones.

1.9 Conclusiones

En este capitulo se analizaron diferentes conceptos que sirven de base al tema que se trata, entre ellos los tipos de colisiones en sus distintas clasificaciones y se determinó trabajar sobre colisiones en un instante de tiempo, o sea, solo detectar si ocurre la colisión sin tener en cuenta sus posiciones anteriores o la dirección en la que se estén moviendo ni lo que ocurre posteriormente (rebote o deformación) esto se propone para investigaciones posteriores. Se hizo mención a una serie de técnicas, entre las cuales encontramos los volúmenes de inclusión, para detectar la colisión se escogió el volumen esfera por la sencillez que proporciona en los cálculos, consume poco espacio de almacenamiento por tener pocos datos que la describen (solo coordenadas del centro y radio), no necesitan volver a ser calculadas ante giros del objeto; debido a que su ajuste no es fuerte, se analizaron una serie de técnicas para mejorar las representaciones de los objetos, estas son las representación jerarquizadas de volúmenes, dentro de ellas veremos los Árboles de Esferas que permiten un mejor ajuste al cuerpo. Se expusieron algunas de las técnicas utilizadas para el análisis de intersecciones entre los objetos, con el objetivo de tener un mejor entendimiento de cómo se pudieran tratar y que sirva de base para estudios posteriores. Se hizo mención a algunas de las librerías utilizadas en la detección de colisiones y se decidió no utilizar estas, pues es más conveniente implementar nuestro propio algoritmo, teniendo el código fuente lo que facilita realizar nuevas modificaciones o ser rehusable en otras aplicaciones.

CAPÍTULO 2: DESCRIPCIÓN DE LA TÉCNICA PROPUESTA PARA LA SOLUCIÓN

Para el tratamiento de las colisiones se ha elegido trabajar con las representaciones jerarquizadas y más específicamente la de las esferas. A continuación explicaremos como pude ser construido a partir de la técnica de Octree. Así garantizar un mejor entendimiento del funcionamiento del algoritmo y que sirva como punto de partida si se desea desarrollar.

2.1 Volumen Envolvente

Para analizar las colisiones se decidió trabajar con esferas por ser una de las primitivas más sencillas, brindando una serie de ventajas para detectar colisiones y/o intersección entre objetos. Una esfera queda perfectamente descrita mediante un único parámetro que la caracteriza por completo: su radio, a diferencia de otras figuras como cajas, cilindros, etc. y su almacenamiento en memoria es bajo.

El movimiento de una esfera se describe completamente mediante el movimiento de un único punto: su centro. Proporciona mayor rapidez cuando se va a actualizar una representación jerárquica de esferas cuando el objeto realiza movimientos de rotación o desplazamiento, bastará con rotar o desplazar los centros de todas las esferas del mismo modo que los vértices del objeto. [Gómez, et al., 2002]

Una vez que se tenga la representación envolvente, determinar colisión entre objetos, sería calcular si hay intersección, lo cual es muy sencillo, solo se tendría que determinar la distancia entre los centros de las esferas y si la suma de su radios es menor que esa distancia. [Pérez F, 1995]

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 \leq (r_1 + r_2)^2$$

Por lo que el costo computacional con respecto a los cálculos es bajo.

La técnica de volumen de limitación envolvente, es muy utilizada en la actualidad para tratar colisiones aunque presenta como inconveniente que se puede correr el riesgo de que si los objetos tienen formas complejas, puede que sus volúmenes envolventes no se ajusten lo suficiente a los objetos, lo que podría ocurrir que estos estén colisionando, pero sus cuerpos contenidos no estén en contacto, creándose un margen elevado de error, y más aún si el volumen de inclusión que se utiliza es la esfera (Fig. 18).

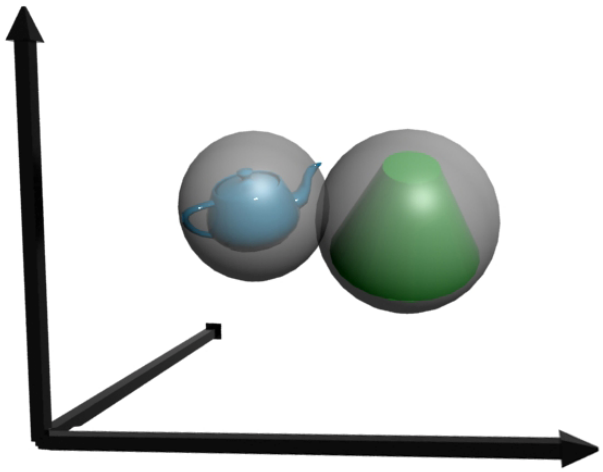
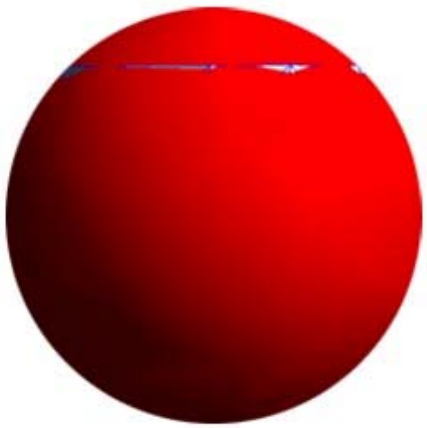
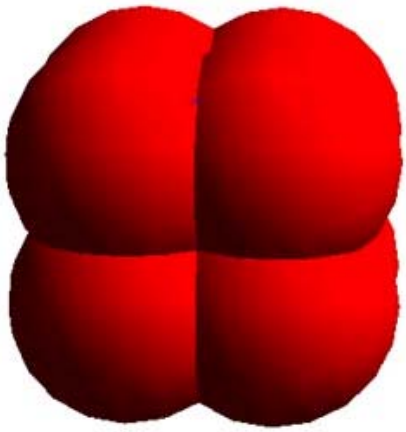


Fig. 18.

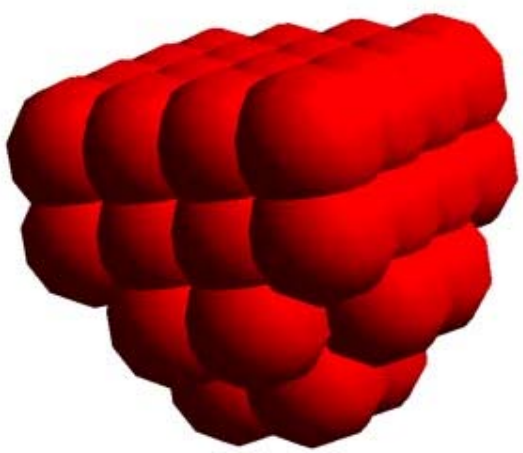
Este margen de error puede disminuir considerablemente si se aplica una representación jerárquica de esferas, la cual consiste en hacer una primera representación de un objeto compuesta por una única esfera y a medida que se pasa a niveles inferiores, se dispondrá de representaciones más precisas pero formadas por un mayor número de esferas. A esta estructura se denomina árbol de esfera (Sphere Tree). Los niveles inferiores de representación se ajustan de acuerdo a la exactitud que se desee obtener, de esta forma se podrá conseguir hacer una representación más exacta del cuerpo. Si se tiene mayor exactitud, aumentará el tiempo computacional requerido en los cálculos para la detección de la colisión, por lo que seria conveniente buscar un equilibrio entre ambos.



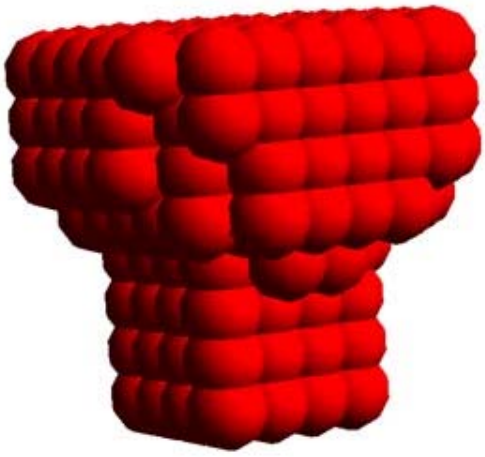
Nivel 1.



Nivel 2.



Nivel 3.



Nivel 4.

Representación a través de Octree.

2.2 Representación del objeto mediante jerarquía de esferas.

Para construcción del árbol de esfera se utilizó la técnica de Octree (Epígrafe. 1.5.1). Es un algoritmo rápido y simple para la construcción de árboles de esferas. Hay que considerar que casi siempre tendremos un margen de error y los Octree tienen una precisión limitada por las representaciones aproximadas.

Un objeto es un conjunto continuo de puntos, sin embargo no podemos listar todos los puntos que pertenecen a un sólido, ya que son infinitos. Podemos listar en pequeñas partes que sean contenidos dentro del sólido, sea de manera parcial o total. Cada elemento de la descomposición, se llama Voxel o elemento de volumen, el más común es el cubo, entre más pequeño se mejora la representación, sin embargo, n^3 celdas son necesarias para un objeto de n voxel en 3D. Los cubos son elementos que no se cubren, ni se extiende uno sobre otro, ni se pliegan o descansan de manera conjunta; son uniformes en forma, tamaño y orientación, haciendo una forma regular del espacio. Así, cada celda o cubo en 3D tiene seis caras, con doce orillas o aritas y ocho vértices, que comparten con sus vecinos, en el caso extremo, su grado de complejidad puede variar y generar reglas de validez. Cada elemento voxel puede ser

descrito en términos de sus esquinas, almacenando solamente sus coordenadas de una esquina de cada cubo. [Moreno, 2000]

2.2.1 Octree

Como ya se ha mencionado Octree es una herramienta que se utiliza para particionar el espacio. Se basa en el principio de "divide y vencerás" (divide and conquer), muy utilizado en diversos algoritmos de geometría computacional. Para el caso que explicaremos la herramienta se aplicará más específicamente al cuerpo. Inicialmente este se engloba en un caja ajustada, usando el principio del AABB (Epígrafe. 1.3.2), luego se divide utilizando planos a lo largo de los ejes X, Y, Z pasando por el centro, creándose 8 cajas hijas de igual tamaño mas pequeñas.

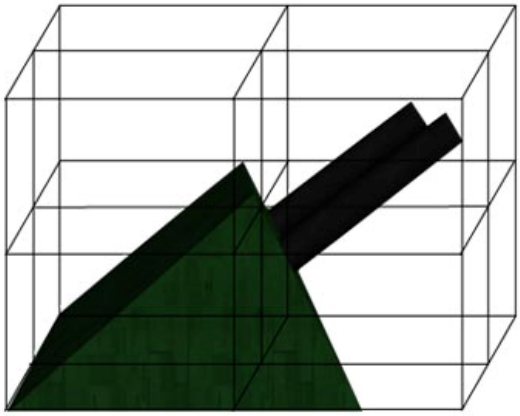


Fig.19.

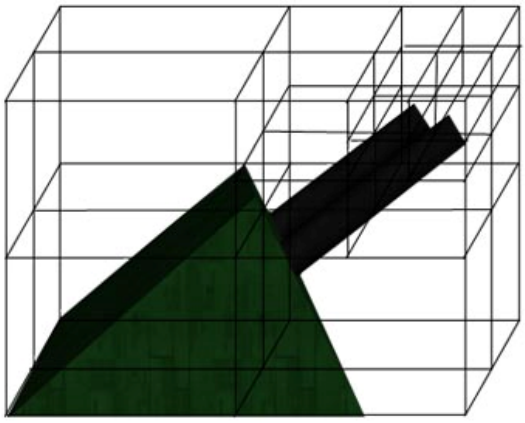


Fig.20.

Cada uno de los cubos puede estar lleno (totalmente dentro del objeto), vacío (ausencia del objeto) o parcialmente lleno (el cubo intersecta al objeto). Este proceso se repite para una profundidad arbitraria para cada cubo hijo creado que esté parcialmente lleno, atendiendo a cuanto se quiere aproximar al cuerpo o en dependencia de cuán exacta se desea que sea la detección de colisión, los cubos vacíos pasan a ser hojas. De esta forma cada nodo creado pude tener hasta 8 punteros hacia sus hijos.

A continuación mostramos la estructura de datos que representan al árbol.

La raíz del árbol que no es más que un AABB y un struct de tipo octree que contiene los posibles hijos del AABB.

```
struct raiz
{
        float Xmin, Ymin, Zmin;      // limites del objeto
        float Xmax, Ymax, Zmax;
        struct Octree *raiz;         // raiz del árbol
}
```

La struct octree que va a tener un estado y un arreglo del mismo.

```
struct Octree
{
        char estado;                 /* lleno, vacio, parcial*/
        struct Octree *oct [8];      /* hijos para el caso de parcial */
}
```

Se Procede a la construcción del Octree del objeto, que se le pasará como parámetro, el objeto ***obj**, el nodo inicial del octree ***o** y la máxima profundidad **cant** o la máxima recursión, además será necesario implementar algunos métodos auxiliares.

```
hacer_octree ( obj, o, cant )
{
        int i;
        switch (compara (obj, o))
        {
                case vacío: o -> estado = vacio;
                                break;
```

```
            case lleno: o -> estado = lleno;
                        break;
            case parcial: break;

            if (cant == 0)
                    o -> estado = parcial;
            else
            {
                    dividir ( o );
                    for ( int i=0; i < 8; i++)
                            hacer_octree( obj, o -> oct[i] , cant-1 );
            }
        }
}
```

Métodos:

compara: clasifica el nodo Octree contra el objeto.

dividir: divide al nodo en ocho octantes.

[Moreno, 2000]

Finalmente los nodos del Octree, son usados para crear el árbol jerárquico 8-ario de esferas, sustituyendo los cubos por esferas, de tal forma que el centro de una esfera son las coordenadas formadas por los puntos medios de las dimensiones x, y, z de cada cubo (Fig.21).

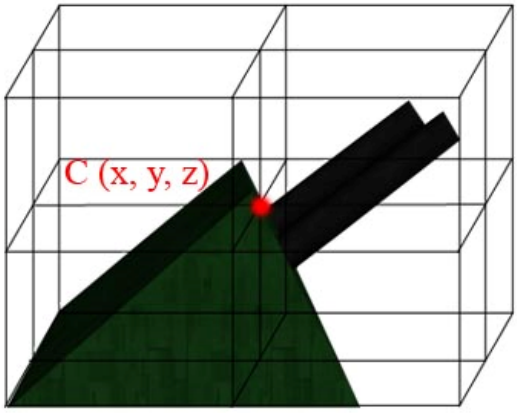
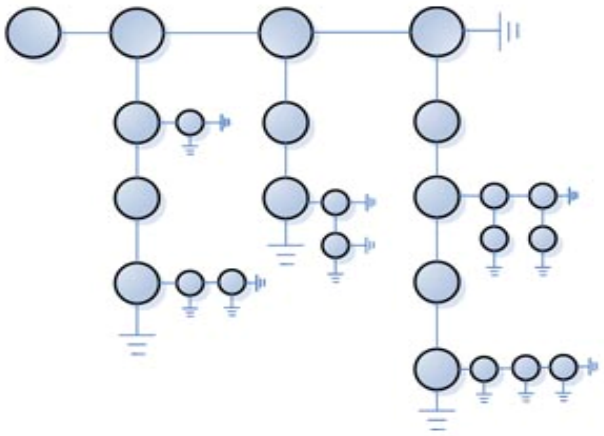


Fig.21

Así como su radio, que vendría dado por la hipotenusa del cubo entre 2, calculado también con las dimensiones x, y, z.

$$r = \frac{\sqrt{x^2 + y^2 + z^2}}{2}$$

El proceso sustitución por esferas se hace para cada nodo que esté lleno o que tenga parte del objeto, siguiendo el mismo patrón anterior.



Estructura de un árbol de esfera a partir de listas.

Nivel 1. Nivel 2. Nivel 3.

Se muestra como inicialmente se coloca la esfera raíz, luego sus primeros 8 hijos; repitiéndose el mismo proceso para una profundidad aleatoria.

Conclusiones

En este capítulo se analizó la técnica de Octree para lograr un mayor ajuste al contorno del objeto. El uso del cuerpo geométrico Esfera en lugar de Cajas en el árbol Octree proporcionará un considerable ahorro de memoria y tiempo en los cálculos de colisiones. La exactitud de la representación depende de la cantidad de niveles que contenga el árbol, pero con el aumento de niveles crece exponencialmente el número de nodos por lo cual es importante hallar un equilibrio entre la exactitud deseada en la representación y la cantidad de niveles mínimo del árbol que puedan dar una solución aceptable sin excesivos tiempos de procesamiento.

CAPÍTULO 3: IMPLEMENTACION Y PRUEBA

En este capítulo se realiza una propuesta para detectar colisiones entre árboles de esferas.

Una vez que se tengan los árboles construidos se pasa a verificar si existe colisión entre ellos, para esto se chequea si hay intersección en un instante de tiempo dado. O sea, se tienen dos árboles de esferas que representan dos objetos distintos, y se desea comprobar si hay colisión entre ellos, para lo cual bastará con tener la información de los radios y las coordenadas (x, y, z) de los centros de todas las esfera de cada árbol.

3.1 Implementación

Método de colisión.

1. Inicialmente comprobamos que la raíz **R** de un árbol **X** que contiene dentro un objeto está interceptando con la raíz de otro árbol **Y,** y la distancia entre ellos es menor que la suma de sus radios (ver epígrafe. 1.3.1), entonces: paso 2.
2. Se recorre el árbol **X** para ver si los nodos interceptan con la raíz de **Y** hasta llegar a las hojas, almacenando estas últimas en una lista, o sea, las hojas de **X** que colisionan con la raíz **Y** se almacenan en una lista **L.**
3. Se compara cada elemento de **L** contra el árbol **Y** hasta que exista una intersección entre dicha lista y una hoja de **Y**, entonces hubo una colisión entre los objetos en ese instante de tiempo.
4. Cuando se termine la lista, si no hubo colisión entre ningún par de hojas, entonces no hubo colisión en ese instante.

Como propuesta para la implementación del algoritmo, se desarrollaron métodos utilizando como lenguaje de programación C++ por ser un lenguaje estándar ampliamente utilizado en el mundo y en los proyectos que se desarrollan en la facultad 5 de Entornos Virtuales, la principal destinataria de los resultados de la presente investigación.

El método **Colision** es el controlador de colisiones entre árboles.

- Se le pasan como parámetro dos árboles de esferas.
- Devuelve verdadero o falso si estos colisionaron en un instante de tiempo determinado.

```
bool Colision (NTree* m_Arbol1, NTree* m_Arbol2)
{
        return m_Arbol1 -> Colision_Arbol (m_Arbol2);
 }
```

Para representar los árboles que se pasan como parámetros al método anterior se definió la clase NTree.

```
class NTree
{
   private:
      int CantHijos;
      Esfera      m_Esfera;  // Raíz del árbol
      vector<NTree*> m_lstEsferas; // Lista de hijo de NTree
   public:
      NTree();
      NTree (Esfera maxima);
      void set_Esfera (Esfera m_esf);
      Esfera get_Esfera();
      bool addEsfera (Esfera m_esf);
      bool Colision_Arbol (NTree* o_ArbolEsfera);
      void Colision_Esfera_Arbol (Esfera o_esf, vector<Esfera*> &h_intercepta);
      bool Colision_Real (Esfera o_esf);
};
```

A continuación se explicarán los métodos más importantes.

El método **Colision_Arbol** es el encargado de gestionar la colisión entre un árbol **X** y otro pasado por parámetro **Y**.

```
bool Colision_Arbol (NTree* o_ArbolEsfera)
{
    // Se comprueba que las raíces de los árboles estén colisionando
    if ( this -> get_Esfera().Interseccion (o_ArbolEsfera  -> get_Esfera() ) )
    {
        /* Se almacena en una lista (compuesta por hojas) todas las esferas de mi árbol X que colisionaron con la raíz del otro árbol Y.*/
        vector<Esfera*> m_hojas;
        this -> Colision_Esfera_Arbol (o_ArbolEsfera  -> get_Esfera(), m_hojas);
        // Se comprueba que la lista no esté vacía.
        if ( m_hojas.size() != 0 )
        {
            /* Se recorre la lista para buscar si algún nodo esfera de X colisionó con una hoja del otro árbol  Y.*/
            for ( int i = 0; i < m_hojas.size(); i++ )
                if ( o_ArbolEsfera -> Colision_Real (*m_hojas[i]) )
                    return true;
        }
    }
    return false;
```

```
}
```

El método **Colision_Esfera_Arbol** es el encargado de recorrer el árbol para hacer pruebas de colisión y devuelve la lista de esferas hojas (**h_intersectada**), pasándole como parámetro una esfera del otro árbol (**o_esf**).

```
void Colision_Esfera_Arbol (Esfera o_esf, vector<Esfera*> &h_intersectada)
{
    // Se chequea que no sea  hoja, es decir, que tenga hijos.
    if ( m_lstEsferas.size() != 0 )
    {
      // Se recorren los hijos  en busca de colisión.
      for ( int i = 0 ; i < m_lstEsferas.size(); i++ )
      {
        Esfera hijo = m_lstEsferas[i] -> get_Esfera();
        /* Si existe colisión con algún hijo, entonces se procede a buscar por los hijos de este; el proceso se repite recursivamente hasta llegar a las hojas.*/
        if ( hijo.Interseccion (&o_esf) )
          m_lstEsferas[i] -> Colision_Esfera_Arbol (o_esf, h_intersectada);
      }
    }
   // De ser hoja se chequea si esta colisiona con la esfera (o_esf) y se almacena en la lista.
    else if ( m_Esfera. Interseccion (&o_esf) )
    {
        h_intersectada.push_back (&m_Esfera);
```

```
    }
}
```

El método **Colision_Real** es le encargado de gestionar si un elemento de la lista de hojas colisionó contra otra esfera hoja del otro árbol, pasándole como parámetro un esfera de dicha lista.

// El método es parecido al anterior (**Colision_Esfera_Arbol**), lo único que varía es el resultado.

```
bool Colision_Real (Esfera o_esf)
{
    If (m_lstEsferas.size() != 0)
    {
      for ( int i = 0; i < m_lstEsferas.size(); i++ )
      {
         Esfera hijo = m_lstEsferas[i] -> get_Esfera();
         if ( hijo.Interseccion (&o_esf) )
           return m_lstEsferas[i] -> Colision_Real (o_esf);
      }
    }
    // Se comprueba que las dos esferas hojas estén colisionado.
    else if (m_Esfera.Interseccion (&o_esf) )
    {
          // Significa que los objetos colisionan
          return true;
    }
    return false;
```

```
}
```

El método de **Intersección** es el encargado de gestionar si una esfera **X** colisiona (Epígrafe. 1.3.1) con otra que es pasada por parámetro **Y**.

```
bool Interseccion (Esfera* o_esf)

{
    // Se calculan los valores de x, y, z.
    float X = o_esf -> get_vectorCentro().get_cx() -  vectorC.get_cx();
    float Y = o_esf -> get_vectorCentro().get_cy() -  vectorC.get_cy();
    float Z = o_esf -> get_vectorCentro().get_cz() -  vectorC.get_cz();
    /* Se calcula la distancia que hay entre los centros con los valores antes calculados y se suman los
    radios de la esfera X y Y.*/
    float distancia = sqrt (X*X + Y*Y + Z*Z);
    float sumaRadio = o_esf -> get_radio() + radio;
    // Se chequea que  la suma de sus radios sea mayor que la distancia.
    if ( sumaRadio >= distancia )
        return true;
    return false;
}
```

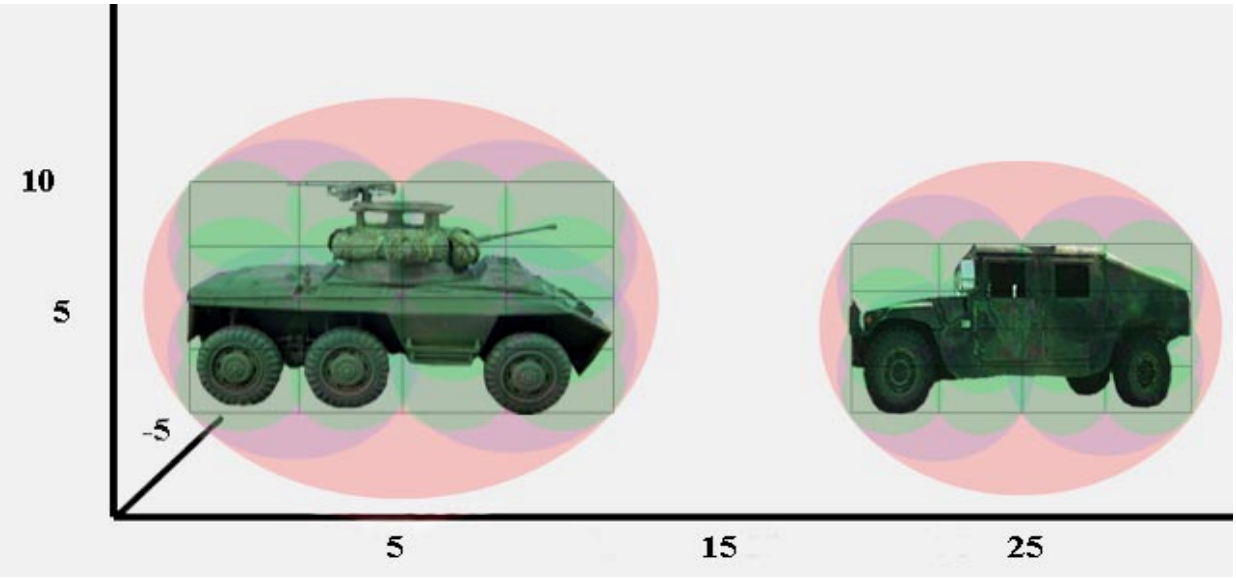
3.2 Prueba

A continuación se realizan pruebas a la aplicación para comprobar la efectividad del algoritmo ante situaciones posibles cuando los objetos colisionan y cuando no.

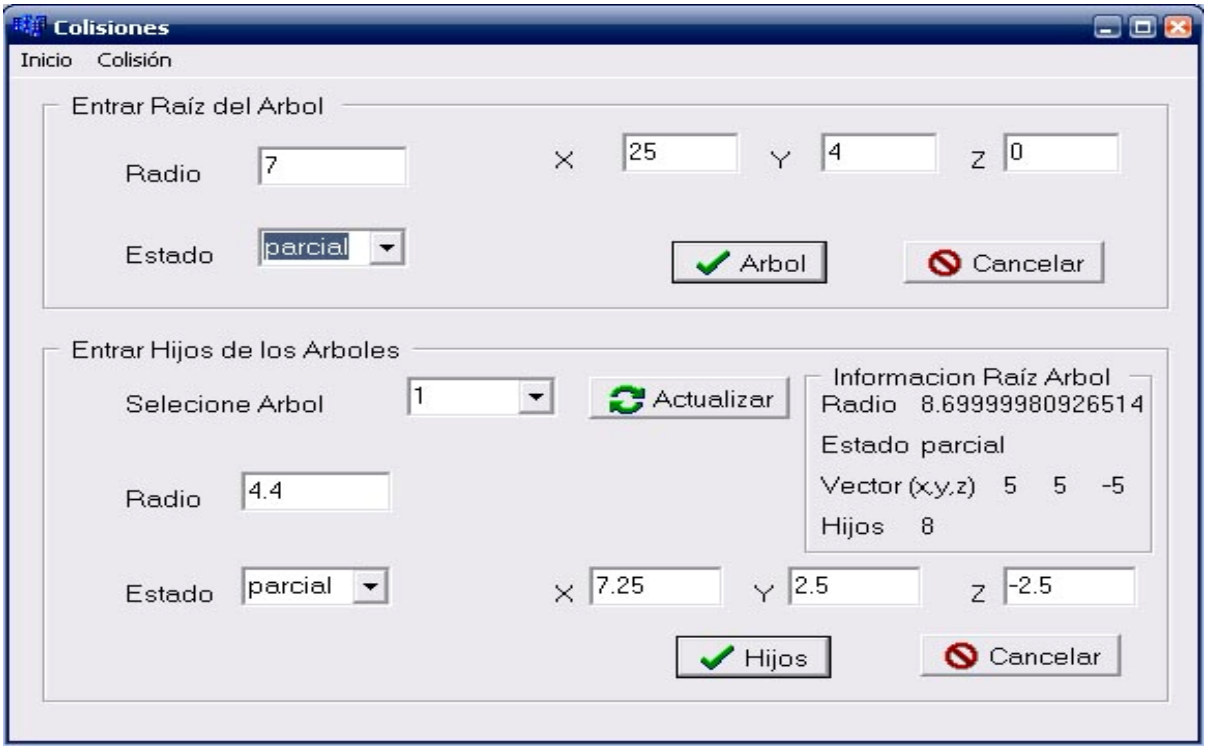
A este caso de prueba se le entraron los datos de árboles de esferas para dos objetos que no colisionan en ese instante de tiempo.

Tanque: C (5, 5, -5); r = 8.7. Este permanece estático. El resto de datos ver (anexo).

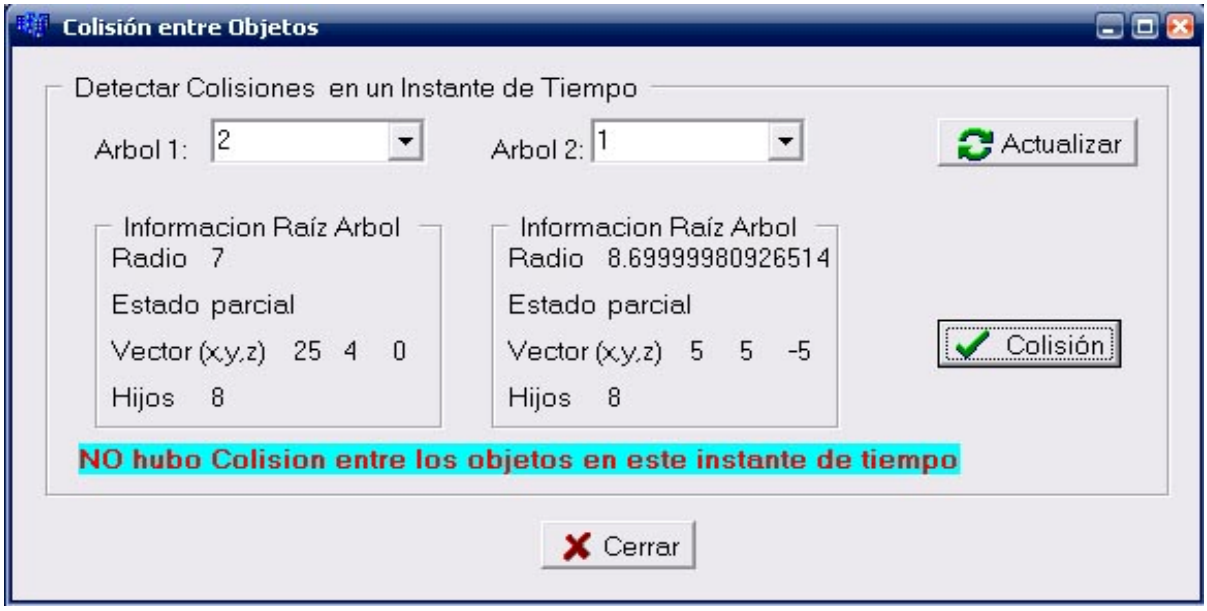
Jeep: C (25, 4, 0); r = 7. El resto de datos ver (anexo).



Objetos que no colisionan.



Entrada de datos.

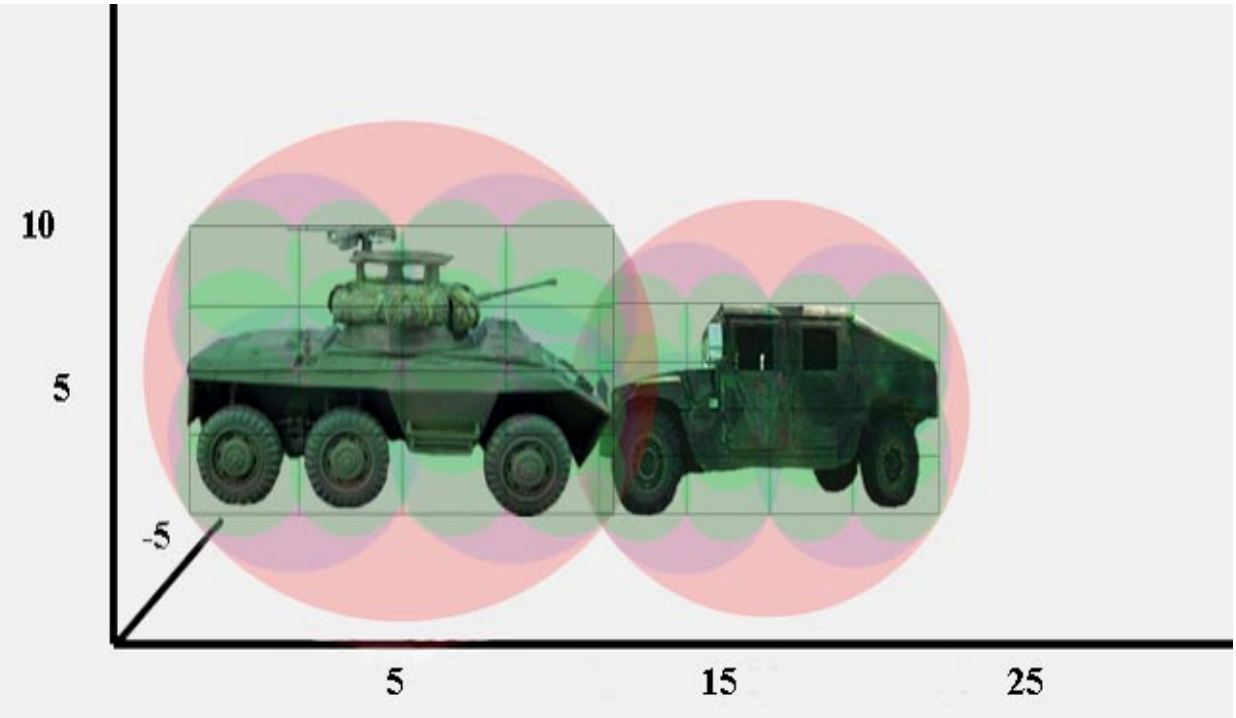


Respuesta para los valores entrados en este instante de tiempo.

A este caso de prueba se le entraron los datos de árboles de esferas para dos objetos que colisionan en ese instante de tiempo.

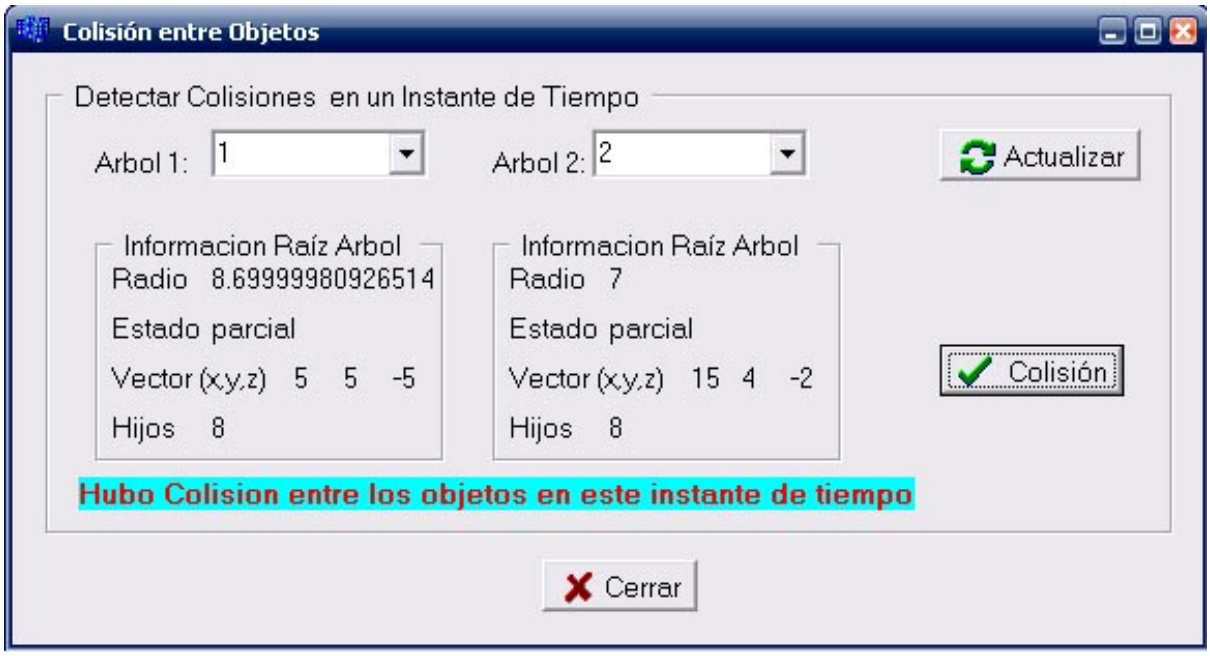
Tanque: C (5, 5, -5); r = 8.7. Este permanece estático. El resto de datos ver (anexo).

Jeep: C (16, 4, -2); r = 7. Aquí el jeep hizo un desplazamiento. El resto de datos ver (anexo).



Objetos que colisionan.

Entrada de datos.



Respuesta para los valores entrados en este instante de tiempo.

3.3 Conclusiones

En este capítulo se describe el algoritmo propuesto como solución para la detección de colisiones basado en la técnica de Octree con árboles de esferas. Se explican las clases y operaciones fundamentales que permitieron la implementación. Se realizaron pruebas para las posibles respuestas de la aplicación (para el caso en que dos objetos colisionan y par el caso en que no) con la que se comprobó la efectividad del algoritmo.

CONCLUSIONES

Al algoritmo obtenido de la investigación se le aplicaron algunas pruebas obteniendo buenos resultados. Cumpliendo con los requerimientos necesarios atendiendo a la técnica que fue aplicada para su análisis e implementación. Por lo que se puede concluir que:

- Se determinó usar el modelo de jerarquía de esferas a partir del la técnica Octree, para el manejo de colisiones en los simuladores que se encuentran en desarrollo en la universidad, ya que permite una mejor aproximación al contorno del objeto, especificar el nivel de detalle deseado y es muy sencillo realizar pruebas por colisión con esferas.
- Se implementó un algoritmo que permite detectar colisiones de acuerdo al modelo seleccionado.
- Se probó el algoritmo obteniendo satisfactoriamente los resultados esperados.

Por todo lo anterior expuesto se cumplieron los objetivos del trabajo, logrando implementar un algoritmo que podrá ser utilizado para el manejo de las colisiones en los Simuladores Virtuales que se desarrollan en la Facultad 5.

RECOMENDACIONES

- Implementar la conversión de objetos tridimensionales (con la estructura que tienen en el simulador al que pertenecen) a árboles de esferas utilizando la técnica de Octree.
- Realizar pruebas de rendimiento para analizar el consumo de tiempo y memoria del algoritmo para árboles con distintos niveles de profundidad
- Profundizar en técnicas que permitan minimizar el error que genera el uso de esferas en cuerpos difíciles de ajustar a estas.
- Continuar profundizando en el estudio del algoritmo y buscar variantes para su optimización
- Seleccionar e Implementar alguna técnica que permita el manejo de colisiones entre múltiples objetos interactuando en tiempo real, utilizando la solución propuesta en este trabajo para dos objetos en un instante de tiempo.
- Incluir la solución propuesta en uno de los simuladores que se desarrollan en la facultad para valorar su eficiencia y efectividad, así como su impacto en la visualización de los objetos que interactúan en el mundo virtual.

BIBLIOGRAFÍA

A, Penington, M.A, Balila and M.S, Bloor. 1983. *Geometric Modeling: A contribution towardsIntelligent robots.* Chicago : 13th International Symposium on Industrial Robotics, 1983.

B, Mirtich. 1998. *V-Clip: Fast and Robust Polyhedral Collision Detection.* s.l. : ACM Transactions on Graphics, 1998.

E, Welzl. 1991. *Smallest enclosing disks (balls and ellipsoids).* s.l. : Department of Computer Science, 1991.

Franco García, Ángel. 2006. Dinámica. *Dinámica de un sistema de partículas.* [Online] Octubre 3, 2006. [Cited: Enero 17, 2007.] http://www.sc.ehu.es/sbweb/fisica/dinamica/con_mlineal/dinamica/dinamica.htm.

Gamma, Team. Collision Detection. *Collision Detection.* [Online] [Cited: Abril 10, 2007.] http://www.cs.unc.edu/~geom/collide/.

Giraldo Orozco, William Joseph and Diego Fernando, Marín Sanabria. Escula de Ingeniería. *Escula de Ingeniería.* [Online] [Cited: Marzo 13, 2007.] http://www.univalle.edu.co/~enycompu/edicion20/revista19_1a.html.

Gómez Martín, Marco Antonio, Gómez Martín, Pedro Pablo and González Calero, Pedro A. 2002. *Agente pedagógico para enseñar la estructura de la JVM (JAVY).* 2002.

H, Sammer and R.E, Webber. 1998. *Hierarchical data structures and algorithms for computer graphics.* s.l. : Fundamentals, IEEE Computer Graphics and Aplications, 1998.

J, Meyer. 1981. *An emulation system for programable sensory robots.* s.l. : IBM Journal of Research & Developement, 1981.

J.D, Cohen, et al. 1995. *ICOLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments.* s.l. : ACM Int. 3D Graphics Conference, 1995.

J.W, Boyse. 1979. *Interference detection among solids and surfaces.* s.l. : Communications to the ACM 22, 1979.

M, Lin and J, Canny. 1991. *Efficient Collision Detection for Animation.* Cambridge, England : Proceedings of the Third Eurographics Workshop on Animation and Simulation, 1991.

Moctezuma Ramírez, Julio Guadalupe. 2006. *Manipulación Tridimensional de Objetos.* 2006.

Moreno, Chacón. 2000. Teoria de los Octree. *Teoria de los Octree.* [Online] Diciembre 2000. [Cited: Mayo 5, 2007.] http://catarina.udlap.mx/u_dl_a/tales/documentos/msp/chacon_m_d/capitulo2.pdf.

Muñoz Moreno, Emma and Rodríguez Bescós, Samuel. Detección de colisiones.Un problema clave en la simulación quirúrgica. *Tecnología de Simulación y Planificación.* [Online] [Cited: Mayo 10, 2007.] http://www.conganat.org/Seis/is/is48/IS48_23.pdf..

Ortega Alvarado, Lidia. Diciembre, 2005. *Visibilidad, Planificación de Trayectorias y Colisiones.* Diciembre, 2005.

P.M, Hubbard. 1996. *Approximating Polyhedra with Spheres for Time-Critical Collision Detection.* s.l. : ACM Transactions on Graphics, 1996.

Pérez Francisco, Miguel. Un enfoque práctico para la detección de colisiones entre objetos en movimiento. *Un enfoque práctico para la detección de colisiones entre objetos en movimiento.* [Online] [Cited: Abril 12, 2007.] http://www.dicc-cid.uji.es/InfTec/reports/rep01-10-95.ps.gz..

Ponzi, Luis Alberto. Sistemas de simulación y entrenamiento. *Sistemas de simulación y entrenamiento.* [Online] [Cited: Febrero 20, 2007.] http://www.rs.ejercito.mil.ar/Contenido/Nro649/Revista/sistemasimulacion.htm.

R. K, Culley and K. G, Kempf. 1986. *A collision detection algorithm based on velocity and distance bounds.* s.l. : IEEE International Conference on Robotics and Automation, 1986.

Roa López, Francisco. *Dtección de colisiones.* [Documento pdf]

S, Cameron. 1985. *A Study of the Clash Detection Problem in Robotics.* s.l. : IEEE Intl. Conf. on Robotics and Automation, 1985.

S, Gottschalk, M.C, Lin and D, Manocha. 1996. *OBBTree: A Hierarchical Structure for rapid Interference Detection.* s.l. : ACM SIGGRAPH, 1996.

S. Gottschalk, M. Lin y D. Manocha. 1996. *Hierarchical Structure for Rapid Interference Detection.* 1996.

S.A, Ehmann and M, Lin. 2000. *Accelerated Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching.* s.l. : International Conference on Intelligent Robots and Systems, 2000.

Sánchez González, Carlos y Santos Ares, Marisa. Detección de colisiones. *Tutorial de OpenGL.* [Online] [Cited: Abril 22, 2007.] http://rnasa.tic.udc.es/gc/trabajos%202002-03/tut_opengl_sombras_colisiones/colisiones.htm.

T, Hudson, et al. 1997. *Accelerated Collision Detection for VRML.* 1997.

V, Hayward and A, Foisy. 1993. *A safe swept volume method for collision detection.* s.l. : 6th International symposium on Robotic Research, 1993.

Varela C, Oswaldo. Detección de Colisiones. *Detección de Colisiones.* [Online] [Cited: Marzo 10, 2007.] http://www.geocities.com/oswaldovarela/colision.htm.

Wikipedia. 2007. Simulador. *Wikipedia.* [Online] may 7, 2007. [Cited: Febrero 20, 2007.] http://es.wikipedia.org/wiki/Simulador.

ANEXOS

El fichero de los casos de pruebas. Están compuestos por un primer número que lo encabeza, la cantidad total de árboles, en este caso es dos.

Ejemplo:

Los tres primeros representan las coordenadas del centro (x, y, z), (5, 5, -5); le sigue el radio r = 8.7, en este caso la raíz de un árbol, a continuación el estado, 1-parcial y 0-llena, y por ultimo la cantidad de hijo de 8 posibles. La raíz tiene 8 hijos de 8 posibles, unos de sus hijo (7.5, 2.5, -2.5) que este a su vez tiene 8 hijos. A continuación de cada están sus hijos.

5 5 -5 8.7 1
8 8
7.5 2.5 -2.5 4.34 1
8 8

Caso que no hay colisión.

2
5 5 -5 8.7 1 //raíz
8 8
7.5 2.5 -2.5 4.34 1
8 8
8.75 1.25 -1.25 2.2 1
0 8
8.75 1.25 -3.75 2.2 1
0 8
8.75 3.75 -1.25 2.2 0
0 8
8.75 3.75 -3.75 2.2 0
0 8
6.25 1.25 -1.75 2.2 1
0 8
6.25 1.25 -3.75 2.2 1
0 8
6.25 3.75 -1.25 2.2 0
0 8
6.25 3.75 -3.75 2.2 0
0 8
7.5 2.5 -7.5 4.34 1
8 8
8.75 1.25 -6.25 2.2 1
0 8
8.75 1.25 -8.75 2.2 1
0 8
8.75 3.75 -6.25 2.2 0
0 8
8.75 3.75 -8.75 2.2 0
0 8
6.25 1.25 -6.25 2.2 1
0 8
6.25 1.25 -8.75 2.2 1
0 8
6.25 3.75 -6.25 2.2 0
0 8
6.25 3.75 -8.75 2.2 0
0 8
7.5 7.5 -2.5 4.34 1
8 8
8.75 6.25 -1.25 2.2 1
0 8
8.75 6.25 -3.75 2.2 1
0 8
8.75 8.75 -1.25 2.2 0
0 8
8.75 8.75 -3.75 2.2 0
0 8
6.25 6.25 -1.25 2.2 1
0 8
6.25 6.25 -3.75 2.2 1
0 8
6.25 8.75 -1.25 2.2 0
0 8
6.25 8.75 -3.75 2.2 0
0 8
7.5 7.5 -7.5 4.34 1
8 8
8.75 6.25 -6.25 2.2 1
0 8
8.75 6.25 -8.75 2.2 1
0 8
8.75 8.75 -6.25 2.2 0
0 8
8.75 8.75 -8.75 2.2 0
0 8
6.25 6.25 -6.25 2.2 1
0 8
6.25 6.25 -8.75 2.2 1
0 8
6.25 8.75 -6.25 2.2 0
0 8
6.25 8.75 -8.75 2.2 0
0 8
2.5 2.5 -2.5 4.34 1
0 8
2.5 2.5 -7.5 4.34 1
0 8
2.5 7.5 -2.5 4.34 1
0 8
2.5 7.5 -7.5 4.34 1
0 8
25 4 0 7 1 //raíz
8 8
23 2 -2 3.49 1
8 8
22 1 -1 1.69 1
0 8
22 1 -3 1.69 1
0 8
24 1 -1 1.69 0
0 8
24 1 -3 1.69 0
0 8
22 3 -1 1.69 1

0 8
22 3 -3 1.69 1
0 8
24 3 -1 1.69 0
0 8
24 3 -3 1.69 0
0 8
23 2 2 3.49 1
8 8
22 1 1 1.69 1
0 8
22 1 3 1.69 1
0 8
24 1 1 1.69 0
0 8
24 1 3 1.69 0
0 8
22 3 1 1.69 1
0 8

22 3 3 1.69 1
0 8
24 3 1 1.69 0
0 8
24 3 3 1.69 0
0 8
23 6 -2 3.49 1
8 8
22 5 -1 1.69 1
0 8
22 5 -3 1.69 1
0 8
24 5 -1 1.69 0
0 8
24 5 -3 1.69 0
0 8
22 7 -1 1.69 1
0 8
22 7 -3 1.69 1
0 8

24 7 -1 1.69 0
0 8
24 7 -3 1.69 0
0 8
23 6 2 3.49 1
8 8
22 5 1 1.69 1
0 8
22 5 3 1.69 1
0 8
24 5 1 1.69 0
0 8
24 5 3 1.69 0
0 8
22 7 1 1.69 1
0 8
22 7 3 1.69 1
0 8

24 7 1 1.69 0
0 8
24 7 3 1.69 0
0 8
27 2 -2 3.49 1
0 8
27 2 2 3.49 1
0 8
27 6 -2 3.49 1
0 8
27 6 2 3.49 1
0 8

Caso que hay colisión.

2
5 5 -5 8.7 1
8 8
7.5 2.5 -2.5 4.34 1
8 8
8.75 1.25 -1.25 2.2 1
0 8
8.75 1.25 -3.75 2.2 1
0 8
8.75 3.75 -1.25 2.2 0
0 8
8.75 3.75 -3.75 2.2 0
0 8
6.25 1.25 -1.75 2.2 1
0 8
6.25 1.25 -3.75 2.2 1
0 8
6.25 3.75 -1.25 2.2 0
0 8
6.25 3.75 -3.75 2.2 0
0 8
7.5 2.5 -7.5 4.34 1
8 8
8.75 1.25 -6.25 2.2 1
0 8
8.75 1.25 -8.75 2.2 1
0 8
8.75 3.75 -6.25 2.2 0
0 8
8.75 3.75 -8.75 2.2 0
0 8
6.25 1.25 -6.25 2.2 1
0 8
6.25 1.25 -8.75 2.2 1
0 8
6.25 3.75 -6.25 2.2 0
0 8
6.25 3.75 -8.75 2.2 0
0 8
7.5 7.5 -2.5 4.34 1
8 8
8.75 6.25 -1.25 2.2 1
0 8
8.75 6.25 -3.75 2.2 1
0 8
8.75 8.75 -1.25 2.2 0
0 8
8.75 8.75 -3.75 2.2 0
0 8
6.25 6.25 -1.25 2.2 1
0 8
6.25 6.25 -3.75 2.2 1
0 8
6.25 8.75 -1.25 2.2 0
0 8
6.25 8.75 -3.75 2.2 0
0 8
7.5 7.5 -7.5 4.34 1
8 8
8.75 6.25 -6.25 2.2 1
0 8
8.75 6.25 -8.75 2.2 1
0 8
8.75 8.75 -6.25 2.2 0
0 8
8.75 8.75 -8.75 2.2 0
0 8
6.25 6.25 -6.25 2.2 1
0 8
6.25 6.25 -8.75 2.2 1
0 8
6.25 8.75 -6.25 2.2 0
0 8
6.25 8.75 -8.75 2.2 0
0 8
2.5 2.5 -2.5 4.34 1
0 8
2.5 2.5 -7.5 4.34 1
0 8
2.5 7.5 -2.5 4.34 1
0 8
2.5 7.5 -7.5 4.34 1
0 8
15 4 -2 7 1
8 8
13 2 -4 3.49 1
8 8
12 1 -3 1.69 1

0 8
12 1 -5 1.69 1
0 8
14 1 -3 1.69 0
0 8
14 1 -5 1.69 0
0 8
12 3 -3 1.69 1
0 8
12 3 -5 1.69 1
0 8
14 3 -3 1.69 0
0 8
14 3 -5 1.69 0
0 8
13 2 0 3.49 1
8 8
12 1 -1 1.69 1
0 8
12 1 1 1.69 1
0 8
14 1 -1 1.69 0
0 8
14 1 1 1.69 0
0 8
12 3 -1 1.69 1
0 8
12 3 1 1.69 1
0 8
14 3 -1 1.69 0
0 8
14 3 1 1.69 0
0 8
13 6 -4 3.49 1
8 8
12 5 -3 1.69 1
0 8
12 5 -5 1.69 1
0 8
14 5 -3 1.69 0
0 8
14 5 -5 1.69 0
0 8
12 7 -3 1.69 1
0 8
12 7 -5 1.69 1
0 8
14 7 -3 1.69 0
0 8
14 7 -5 1.69 0
0 8
13 6 0 3.49 1
8 8
12 5 -1 1.69 1
0 8
12 5 1 1.69 1
0 8
14 5 -1 1.69 0
0 8
14 5 1 1.69 0
0 8
12 7 -1 1.69 1
0 8
12 7 1 1.69 1
0 8
14 7 -1 1.69 0
0 8
14 7 1 1.69 0
0 8
17 2 -4 3.49 1
0 8
17 2 0 3.49 1
0 8
17 6 -4 3.49 1
0 8
17 6 0 3.49 1
0 8

GLOSARIO

Colisión: es cuando dos o más cuerpos se encuentran en contacto en tiempo y espacio.

Octree: es una técnica de particionamiento del espacio utilizando la representación de árboles octales. Consiste en circunscribir el objeto o al mundo completo, mediante un volumen (un AABB concretamente) y dividir éste en ocho volúmenes iguales, que a su vez se dividirán en otros ocho que serán los hijos; este proceso se hace recursivamente hasta alcanzar el nivel de profundidad deseado.

Arboles de esferas: es una representación jerárquica de esferas, consiste en hacer una primera representación de un objeto compuesta por una única esfera y a medida que se pasa a niveles inferiores, se dispondrá de representaciones más precisas pero formadas por un mayor número de esferas. Los niveles inferiores de representación se ajustan de acuerdo a la exactitud que se desee obtener, de esta forma podremos conseguir hacer una representación más exacta del cuerpo.

Volúmenes inclusión: se definen volúmenes con geometría simple (ej. esferas, cubos, cilindros...) que encierra al objeto de interés. Debe ajustarse lo más estrechamente posible al objeto (volúmenes envolventes).

Simulador: es un dispositivo que permite la simulación de un sistema, reproduciendo su comportamiento como si fuera real. Los simuladores reproducen sensaciones que en realidad no están sucediendo. Pretende reproducir tanto las sensaciones físicas (velocidad, aceleración, percepción del entorno) como el comportamiento de los equipos de la máquina que se pretende simular.

Sistema discreto: un sistema discreto es aquel en que el tiempo se mide en pequeños lapsos y no de forma continua.

Costo computacional: recursos requeridos durante el cálculo para resolver un problema. Los recursos comúnmente estudiados son el tiempo (mediante una aproximación al número de pasos de ejecución de un algoritmo para resolver un problema) y el espacio (mediante una aproximación a la cantidad de memoria utilizada para resolver un problema). Se pueden estudiar igualmente otros parámetros, tales como el número de procesadores necesarios para resolver el problema en paralelo.

Ajuste: Es la medida en la que un volumen envolvente se aproxima al objeto que envuelve.

Objeto Convexo: es aquel en el que se verifica que cualquier par de puntos ubicados en su interior determinan un segmento de recta también interior.

Objeto Cóncavo: lo opuesto de convexo**,** es aquel en el que algún par de puntos interiores no los une un segmento de recta interior al objeto.

Volumen barrido: es el volumen del espacio ocupado por una superficie cuando se desplaza a lo largo de una trayectoria, la construcción más simple es realizar traslaciones o revoluciones, pero se pueden igualmente construir sólidos realizando un barrido generalizado, escalando y rotando la superficie mientras se desplaza a lo largo de una trayectoria arbitraria.